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

Foteini Baldimtsi[1⋆], Aggelos Kiayias[2,3⋆⋆], Thomas Zacharias[2⋆⋆], and Bingsheng Zhang[4,5⋆⋆⋆]

[1] George Mason University, foteini@gmu.edu
[2] The University of Edinburgh, {akiayias,tzachari}@inf.ed.ac.uk
[3] IOHK
[4] Zhejiang University, bingsheng@zju.edu.cn
[5] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies

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 [57, 58, 32] to real-world implementations [12, 55, 43, 27, 26, 24] that can be used for a number of different security critical

⋆ Supported by NSF grant 1717067.
⋆⋆ Supported by Horizon 2020 project #780477 (PRIViLEDGE).
⋆⋆⋆ Supported by the Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang (Grant No. 2018R01005) and Zhejiang Key R&D Plan (Grant No. 2019C03133).

operations including auctions [12], e-voting [23, 41, 1], and privacy preserving statistics [48, 13]. An important paradigm for MPC that captures a large number of applications is the *client-server* model [30, 6, 25, 38, 49, 33] 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 [35]) 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 [27] and [26, 39] achieve very good performance for real world applications by utilizing an offline/online approach [5]), 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 [27]. 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 [56] in the context of *universally verifiable* MPC. The security analysis in [56] 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., [44] while the issue of removing setup assumptions such as the CRS or random oracles has been also recently considered [41, 40].

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 [29, 42] 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 [15], 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 [15] that 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) [17]. This model, which can been seen as an adaptation of the super-polynomial simulation concept [54] 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., by 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 [11, 52, 27, 26], 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. [37, 31], while we cannot utilize a subversion-sound NIZK either, cf. [7], 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. [14, 47], 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 [8] and randomness condensers [28]. 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 [50]. 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 [5] and utilized numerous times [27, 4]). In a nutshell, our construction follows the paradigm of SPDZ [27] 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 properties; 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 [53] 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 [9, 10], 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 $x \xleftarrow{\$} S$, 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 st_P, 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.

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 [8] 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 $I_q(f) = \min_{S \subseteq \{x_1,\ldots,x_m\}, |S| \leq q} I_S(f)$. We say that f is* (q, ε)-resilient *if $I_q(f) \leq \varepsilon$. In addition, for $0 < \tau < 1$, we say f is* τ-strongly resilient *if for all $1 \leq q \leq n$, $I_q(f) \leq \tau \cdot q$.*

We use the $(\Theta(\log^2 m/m))$-*strongly resilient* function defined in [50] (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 $\Omega(\log m/m^{O(1)})$ [36]. Hence, it is not possible to get a single bit string with $\varepsilon = m^{-\Omega(1)}$.

Publicly samplable adaptive one-way functions. Adaptive one-way functions (adaptive OWFs, or AOWFs for short) were formally introduced by Pandey *et al.* [53]. 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 $\mathsf{tag}' \neq \mathsf{tag}$. 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 [53]) 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 [53], 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 the full version of this paper [2], we provide an instantiation of a PS-AOWF 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 [16]. 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 [17]. 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 $\ell = \{1, \ldots, n\}$, $\mathsf{CVZK.Prv}_2(x, w, a, \langle c_1, \ldots, c_n\rangle, \mathsf{st}_P)$ 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 $|\mathcal{I}_{\mathsf{corr}}| \leq t_1$, 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 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 $|\mathcal{I}_{\mathsf{corr}}| \leq t_2$, 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 $|\mathcal{I}_{\mathsf{corr}}| \leq t_3$, 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 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 $\mathcal{H}$.

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

(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 from 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. Let $n = \lambda^\gamma$ for a constant $\gamma > 1$ and assume $\lambda \log \lambda$ divides n. Let f_{res} denote the $(\Theta(\log^2 m/m))$-strongly resilient function over m bits proposed

in [50]. 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 [50] 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 the full version of this paper [2].

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., [46, 19, 20, 34]). 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 $\mathcal{L} \in \mathbf{NP}$. 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) [46]. By the LS protocol, we know that there exists a fully input-delayed Σ-protocol for every **NP** language. In the full version of this paper [2], 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 addition, 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 [53].

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 $\mathsf{InD}.\Pi := (\mathsf{InD}.\mathsf{Prv}_1, \mathsf{InD}.\mathsf{Prv}_2, \mathsf{InD}.\mathsf{Verify})$ 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 [22, 20]: 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 $(d_1, \ldots, d_{\log \lambda})$, 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 $\mathsf{InD}.\mathsf{Sim}$ 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 [54], 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 $A := \big(\{a_i\}_{i\in[\log\lambda]}, \{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$ and the state $\mathsf{st}_P := \big(R, \{\mathsf{st}_i\}_{i\in[\log\lambda]}, \{\mathsf{st}^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$.
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 $\mathsf{st}_P := \big(R, \{\mathsf{st}_i\}_{i\in[\log\lambda]}, \{\mathsf{st}^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$.
   - 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 $Z := \big(E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$.
4. $\mathsf{CVZK.Verify}(x, A, C, Z)$:
   - Parse $A := \big(\{a_i\}_{i\in[\log\lambda]}, \{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$.
   - Parse $Z := \big(E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}\big)$.
   - 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 $\mathsf{InD.Verify}\big(x^*_{\ell,j}, a^*_{\ell,j}, r_\ell, z^*_{\ell,j}\big) = 1$.
   - 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 the full version of this paper [2].

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 $\{\mathcal{L}^*_{\mathsf{tag}_\ell}\}_{\ell\in[n]}$ 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 $\epsilon(\cdot)$[6], $\Sigma.\Pi$ 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 $(\mathsf{CVZK.Sim}_1, \mathsf{CVZK.Sim}_2^{\mathcal{H}_S})$ 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 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.

[6] The PS-AOWF family instantiated in [2] has perfect samplability, i.e. $\epsilon(\lambda) = 0$.

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. [18, 51, 41]). 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 $\kappa = O(\log\lambda)$, 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 2. 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 [17], 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 (we recall $\mathcal{G}_{\mathsf{BB}}$ and $\mathcal{F}_{\mathsf{sc}}$ in the full version [2]).

Ideal world setting. We formally describe the *ideal VMPC functionality* $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ that is defined w.r.t. to an *evaluation function* $f : X^n \longrightarrow Y$ and a *binary relation* $R \subseteq \mathsf{Img}[\mathsf{f}] \times \mathsf{Img}[f]$ over the image of f. The functionality $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$. 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, $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ an adversarial (hence, not necessarily meaningful) output value $\tilde{y}$ for the VMPC execution for U_ℓ. Then, $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$'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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ will consider $\tilde{y}$ only if (a) all servers are corrupted, or (b) an honest user's client is corrupted[7]. 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $\Pi = (\mathbf{Initialize}, \mathbf{Input}, \mathbf{Compute}, \mathbf{Verify})$ 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 [17], 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 $\mathsf{EXEC}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}^{\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})}(\lambda)$, and in the real world the bit $\mathsf{EXEC}_{\mathcal{A},\mathcal{Z},\mathcal{H}}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}(\lambda)$. Security is defined as follows:

Definition 6. *Let $f : X^n \longrightarrow Y$ be an evaluation function and $R \subseteq \mathsf{Img[f]} \times \mathsf{Img}[f]$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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*

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

Strength of our VMPC security model. Based on the description of $\mathcal{F}_{\mathsf{vmpc}}^{f,R}$, 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. Clearly, since the users communicate with the servers only via their clients, the user has to provide her input to the client which has to be trusted for privacy.

[7] 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.

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.

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 $(\textsc{Cast}, \mathsf{sid}, U_\ell, x_\ell)$ to Sim. Otherwise, it sends $(\textsc{Cast}, \mathsf{sid}, U_\ell, (x_\ell \stackrel{?}{=} \text{'}\mathsf{abstain}\text{'}))$ 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 $(\textsc{Result}, \mathsf{sid}, \tilde{y}, \tilde{v})$ to U_ℓ.

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

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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 the full version of this paper [2], 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, $\mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') = \left| \{ \ell \in [n] \mid x_\ell \neq x'_\ell \} \right|$.

We examine feasibility of realizing $\mathcal{F}_{\mathsf{vmpc}}^{f,R}$ 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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[8], 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 the full version of this paper [2].

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 $\mathsf{Img}[f]$.*

(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 $\min\{2^{-\kappa\delta} - \epsilon(\lambda), \gamma(\lambda)\}$.*

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

7 Constructing VMPC from CVZK

A number of efficient practical MPC protocols [11, 52, 27, 26] 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 a number of tools that are presented in the full version of this paper [2]: (i) a perfectly binding homomorphic commitment that is secure against helper-aided PPT adversaries, (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, (iii) a *Σ-protocol for Beaver triples*, and (iv) CVZK proofs that derive from compiling straight-line simulatable ZK proofs for **NP** languages 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 rooms and trusted mailing service. To provide intuition, we first present a construction for the single-server setting.

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 [2]) 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. 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 $\delta_\ell = x_\ell - r_\ell^{(b_\ell)}$[9]. 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 $c_\ell \leftarrow c_\ell^{(b_\ell)} \cdot \mathsf{DC.Com_{ck}}(\delta_\ell; \mathbf{0})$ 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 $\mathsf{DC.Com_{ck}}(x) \cdot \mathsf{DC.Com_{ck}}(y)$; (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 $C \cdot B^\alpha \cdot A^\beta \cdot \mathsf{DC.Com_{ck}}(\alpha \cdot \beta)$. 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, $\ell \in [n]$.

Security Analysis. 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 commitment scheme is computationally hiding under the adaptively secure DDH assumption (cf. [2] for a definition), 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. Assume that

[9] 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).

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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ 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 $(\hat{r}_\ell^{(0)}, \hat{r}_\ell^{(1)})$, where one component $\hat{r}_\ell^{(b^*)}$ is different than $r_\ell^{(b^*)}$ in the pair $(r_\ell^{(0)}, r_\ell^{(1)})$ committed to 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 $x_\ell^* = x_\ell + \left(\hat{r}_\ell^{(b^*)} - r_\ell^{(b^*)}\right)$ without being detected. Given that U_ℓ flips a fair coin, this attack has $1/2$ success probability.

This list of attacks 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 $2^{-\delta} + \mathsf{negl}(\lambda)$.

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 [27, 26]. However, all the underlying ZK proofs need to be compiled in CVZK. More specifically, we define an offline functionality $\mathcal{F}_{\mathsf{V.Offline}}$ to generate shared random Beaver triples and shared random values. The main differences between our $\mathcal{F}_{\mathsf{V.Offline}}$ 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. Due to space limitations, we provide the formal description of $\mathcal{F}_{\mathsf{V.Offline}}$ and its realization in the $\mathcal{H}$-EUC model in the full version of this paper [2]. Moreover, in [2], we formally present the multi-server VMPC scheme $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}$ in the $\{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}\}$-hybrid model along with a proof sketch of the following theorem.

Theorem 4. *Let $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}$ be our VMPC scheme 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 scheme $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}$ $\mathcal{H}$-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.*

Remark 3. 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 $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$.

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. [45, 41, 21]) 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, $Y = [a, b]$ 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 could count 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 $|\cdot|$. (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. $x_\ell = (x_{\ell,1}, \ldots, x_{\ell,m}) \in [a_1, b_1] \times \cdots \times [a_m, b_m]$, 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 $\mathbf{z}_1, \ldots, \mathbf{z}_t$ 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. J. Alwen, R. Ostrovsky, H. Zhou, and V. Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO*, 2015.
2. F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang. Crowd verifiable zero-knowledge and end-to-end verifiable multiparty computation. *IACR Cryptol. ePrint Arch.*, 2020:711, 2020.
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. D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
6. D. Beaver. Commodity-based cryptography (extended abstract). In *STOC*, 1997.
7. M. Bellare, G. Fuchsbauer, and A. Scafuro. NIZKs with an untrusted CRS: security in the face of parameter subversion. In *ASIACRYPT*, 2016.

8. M. Ben-Or and N. Linial. Collective coin flipping, robust voting schemes and minima of Banzhaf values. In *FOCS*, 1985.
9. J. Benaloh. Simple verifiable elections. USENIX EVT. USENIX Association, 2006.
10. J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT*, 2007.
11. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.
12. 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.
13. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.
14. M. Burmester and Y. Desmedt. Broadcast interactive proofs (extended abstract). In *EUROCRYPT*, 1991.
15. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
16. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, 2007.
17. R. Canetti, H. Lin, and R. Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, 2010.
18. D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE S&P*, 2004.
19. M. Ciampi, G. Persiano, A. Scafuro, L. Siniscalchi, and I. Visconti. Improved or-composition of sigma-protocols. In *TCC*, 2016.
20. M. Ciampi, G. Persiano, A. Scafuro, L. Siniscalchi, and I. Visconti. Online/offline OR composition of sigma protocols. In *EUROCRYPT*, 2016.
21. V. Cortier, D. Galindo, R. Küsters, J. Mueller, and T. Truderung. SoK: Verifiability notions for e-voting protocols. In *IEEE Security & Privacy*, 2016.
22. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.
23. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, 1997.
24. I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft. Confidential benchmarking based on multiparty computation. In *FC*, 2016.
25. 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.
26. 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.
27. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
28. Y. Dodis, T. Ristenpart, and S. P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In *TCC*, 2012.
29. C. Ellison. Ceremony design and analysis. IACR ePrint, Report 2007/399, 2007.
30. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *STOC*, 1994.
31. N. Fleischhacker, V. Goyal, and A. Jain. On the existence of three round zero-knowledge proofs. In *EUROCRYPT*, 2018.
32. 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.

33. S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.
34. C. Hazay and M. Venkitasubramaniam. On the power of secure two-party computation. In *CRYPTO*, 2016.
35. Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, 2010.
36. J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions (extended abstract). In *FOCS*, 1988.
37. Y. T. Kalai, G. N. Rothblum, and R. D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In *CRYPTO*, 2017.
38. S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *CCS*, 2012.
39. M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS*, 2016.
40. A. Kiayias, T. Zacharias, and B. Zhang. DEMOS-2: scalable E2E verifiable elections without random oracles. In *CCS*, 2015.
41. A. Kiayias, T. Zacharias, and B. Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT*, 2015.
42. A. Kiayias, T. Zacharias, and B. Zhang. Ceremonies for end-to-end verifiable elections. In *PKC*, 2017.
43. B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, 2012.
44. R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In *CCS*, 2010.
45. R. Küsters, T. Truderung, and A. Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Security & Privacy*, 2012.
46. D. Lapidot and A. Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.
47. M. Lepinski, S. Micali, and A. Shelat. Fair-zero knowledge. In *TCC*, 2005.
48. Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR ePrint 2008/197*, 2008.
49. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, 2012.
50. R. Meka. Explicit resilient functions matching Ajtai-Linial. In *SODA*, 2017.
51. C. A. Neff. Practical high certainty intent verification for encrypted votes. Votehere, Inc. whitepaper, 2004.
52. 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.
53. O. Pandey, R. Pass, and V. Vaikuntanathan. Adaptive one-way functions and applications. In *CRYPTO*, 2008.
54. R. Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, 2003.
55. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009.
56. B. Schoenmakers and M. Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *ACNS*, 2015.
57. A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982.
58. A. C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.