Multi-Client Verifiable Computation with Stronger Security Guarantees

S. Dov Gordon
Applied Communication Sciences
sgordon@appcomsci.com

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

Feng-Hao Liu
University of Maryland
fenghao@cs.umd.edu

Elaine Shi
University of Maryland
elaine@cs.umd.edu

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

Abstract

Choi et al. (TCC 2013) introduced the notion of *multi-client verifiable computation* (MVC) in which a set of clients outsource to an untrusted server the computation of a function f over their collective inputs in a sequence of time periods. In that work, the authors defined and realized multi-client verifiable computation satisfying soundness against a malicious server and privacy against the semi-honest corruption of a single client. Very recently, Goldwasser et al. (Eurocrypt 2014) provided an alternative solution relying on multi-input functional encryption.

Here we conduct a systematic study of MVC, with the goal of satisfying stronger security requirements. We begin by introducing a simulation-based notion of security that provides a unified way of defining soundness and privacy, and automatically captures several attacks not addressed in previous work. We then explore the feasibility of achieving this notion of security. Assuming no collusion between the server and the clients, we demonstrate a protocol for multi-client verifiable computation that achieves strong security in several respects. When server-client collusion is possible, we show (somewhat surprisingly) that simulation-based security cannot be achieved in general, even assuming semi-honest behavior.

Contents

1 Introduction

Protocols for *verifiable computation* (or *secure outsourcing*) allow computationally weak clients to delegate to a more powerful server the computation of a function f on a series of dynamically chosen inputs $x^{(1)}, x^{(2)}, \ldots$. The main desideratum is that, following a pre-processing stage whose complexity may depend on f, the work of the client per function evaluation should be significantly lower than the cost of computing the function itself. The initial proposal and construction of non-interactive verifiable computation [20] led to a long line of follow-up work [16, 1, 17, 9, 35, 27, 36, 18, 21, 34, 3, 26, 10, 7, 8, 33].

We are interested here in the *multi-client* setting introduced by Choi et al. [15]. Imagine that n clients wish to compute some function f over their joint inputs $\{(x_1^{(\mathsf{ssid})}, \ldots, x_n^{(\mathsf{ssid})})\}_{\mathsf{ssid}}$ for a series of subsessions identified by ssid. (One can view the ssid as encoding the current time period, though there are other possibilities as well.) As in earlier work, we assume no client-client communication, and focus on *non-interactive* solutions in which each evaluation of the function requires only a single round of communication between each client and the server.

In earlier works on multi-client verifiable computation [15, 24], the primary goal is to achieve security (soundness and privacy) against a *malicious server*, assuming that *clients behave honestly*. Soundness means that a malicious server should not be able to fool a client into accepting a wrong result; privacy means that clients' inputs should remain hidden from the server. (Choi et al. also considered privacy against clients, but while still assuming semi-honest client behavior.)

1.1 Our Contributions

In this paper, we conduct a systematic study of multi-client verifiable computation with stronger security guarantees. The primary question we address is security when *clients* may be malicious. These malicious clients may potentially be colluding with each other, or with the server.

Formal security modeling. We begin by introducing a simulation-based notion of security in the universal composability framework, which provides a unified way of defining soundness and privacy. As a technical advantage, it means that protocols satisfying the definition achieve a strong, simulation-based notion of security not considered in previous work. Our definition also automatically captures *adaptive soundness*[1] as well as *selective-failure*[2] attacks.

Impossibility when the server and clients collude. Ideally, one would like to achieve a strong notion of security where a subset of the clients may be corrupted, and may additionally be colluding with the server. Unfortunately, we show that simulation-secure MVC is impossible to realize (in general) when the server colludes with clients. This impossibility result holds even in the standalone setting, even when the server colludes with only a single, semi-honest client, and even in the presence of trusted setup assumptions such as PKI or a common reference string (CRS). Intuitively, this is due to a connection we establish between MVC and virtual black-box (VBB) obfuscation, which is already known to be impossible [4] (for general functions). More details can be found in Section 5.

[1] Adaptive soundness means that soundness should hold even if the inputs to the function are chosen adaptively following the pre-processing phase. See [5, 15] for further discussion.

[2] A selective-failure attack is one in which the server attempts to violate soundness by observing whether client(s) "fail" following a particular set of responses sent by the server. This attack is only applicable if client(s) continue interacting with the server even following such a failure.

Feasibility when the server and clients do not collude. In contrast to the above, we show positive results for the case when client-server collusion is assumed not to occur. We show a construction that achieves security (i.e., soundness and privacy) against either a malicious server, or an arbitrary set of malicious, colluding clients. Our construction achieves both adaptive soundness and security against selective-failure attacks.

Our construction relies only on *falsifiable* assumptions. While it is also possible to construct MVC schemes using the notion of *multi-input functional encryption* [24], that notion requires non-falsifiable assumptions or sub-exponential hardness assumptions [22]. Moreover, current constructions of multi-input functional encryption have prohibitively large overhead.

1.2 Techniques and New Primitives

When server-client collusion is not allowed, we take a two-step approach to achieving simulation-based security. As a stepping stone, we identify a new building block named multi-sender, attribute-based encryption (mABE) that may be of independent interest.

Our two-step approach for MVC. We start with a protocol that achieves simulation-based security against either (i) a malicious server or (ii) any coalition of semi-honest clients. Although this is also achieved by the protocol of Choi et al. [15]—even though not claimed explicitly there—our construction has the advantages of offering adaptive soundness based on standard assumptions as well as resilience to selective-failure attacks.

We then present a generic compiler that upgrades our intermediate solution (as well as the one by Choi et al. [15]) to handle an arbitrary subset of *malicious* clients. While we could rely on standard techniques here (such as having clients commit to random tapes during setup, and then asking each client to prove in zero knowledge that they behave honestly), we instead offer a compiler that does not require committed randomness, allowing us to reduce our setup assumptions to just a common reference string. We demonstrate that as long as the semi-honest protocol offers a sufficiently strong notion of privacy, our compiler ensures security against malicious corruption. This gives us a non-interactive, multi-client, verifiable-computation protocol secure against malicious adversaries, in the standard model and based on *falsifiable* assumptions.

A new building block: mABE. In the single-client setting, Parno et al. [36] showed a connection between attribute-based encryption (ABE) and verifiable computation (without input privacy). Later, Goldwasser et al. [26] showed (i) how to compile an ABE scheme to a private-index functional encryption scheme using fully homomorphic encryption (FHE), and (ii) that private-index functional encryption implies input-private publicly verifiable computation.

We conduct a parallel study in the multi-sender setting. The multi-sender counterpart (namely, mABE) is defined as follows. Each sender $\mathsf{P}_i \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ has an attribute value x_i and two input messages $(m_0^{(i)}, m_1^{(i)})$. A receiver can use a decryption key for function f_i to learn $\{m_b^{(i)}\}_{i=1}^n$ if and only if $b = f_i(x_1, \ldots, x_n)$. We show how to construct an mABE scheme secure against a malicious receiver or semi-honest senders. To do so, we first observe that the LWE-based ABE scheme by Gorbunov, Vaikuntanathan, and Wee [30] satisfies a special "local encoding" property. We use this observation to combine their scheme with the proxy-OT protocol proposed by Choi et al. [15].

Given an mABE scheme, we can apply the compiler of Goldwasser et al. [26] to transform it into an *attribute-hiding* mABE scheme (which can also be thought of as a multi-sender, private-index functional encryption scheme). Finally, just as single-sender private-index functional encryption

implies input-private verifiable computation, we show that attribute-hiding mABE implies multi-client verifiable computation with input privacy, secure against a malicious server or an arbitrary subset of semi-honest clients. We can then use the compiler described previously to obtain security against malicious clients, as long as there is no client-server collusion.

Sacrificing input privacy to handle client-server collusion. Since attribute-hiding mABE implies multi-client verifiable computation, it follows that attribute-hiding mABE is also impossible for general functions if sender-receiver collusion is allowed. However, it is still interesting to consider settings without input privacy (resp., attribute hiding). We show that any (non-attribute-hiding) mABE scheme that is secure with respect to some corruption pattern implies public-input MVC under the same corruption pattern. We also show that an mABE scheme secure with respect to client-server collusion, even in the standalone setting, implies extractable witness encryption [25]. So, building MVC protocols without input privacy via this approach would inherently require non-falsifiable assumptions.

1.3 Other Related Work

Various works have considered server-aided secure computation with the goal of eliminating client-to-client interaction [31, 32]. The protocols constructed in these works require multiple rounds of client-server interaction.

Earlier work by Shi et al. [37] studied multi-party outsourcing for specific functionalities such as summation and variance. They also describe various applications such as secure sensor network aggregation. In their model, the server learns the final outcome of the computation, and verifiability (i.e., soundness) is not an inherent part of the problem formulation.

2 Multi-Client Verifiable Computation

2.1 Definitions

We start by introducing the notion of non-interactive multi-client verifiable computation (MVC) that has the following structure: let κ be the security parameter, n be the number of clients $\mathsf{P}_1, \ldots, \mathsf{P}_n$ who are delegating some computation on some n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$ to a distinguished server Serv and would like to verify the correctness of their answers. Here we assume each client's input message space is $\mathcal{X}$, and output message space $\mathcal{Y}$, for some polynomial-length (in the security parameter) $|\mathcal{X}|$ and $|\mathcal{Y}|$.

Intuitively, MVC protocols have the following properties: (1) All participants are allowed to access to a certain initial setup $\mathcal{G}$ (e.g., PKI, CRS). (2) Then an offline stage follows; in the offline stage, each client sends a single message to the server Serv. (3) In the online stage, in a single time period (subsession), each client is only allowed to send an outgoing message to the server and then receive an incoming message from the server. In the whole paper, we assume that the clients cannot communicate with each other directly, and can only send a single round of message to the server per time period (subsession). Next, we give more details.

Definition 2.1 (Non-interactive multi-client verifiable computation) *Let κ be the security parameter, n be the number of clients and f be an n-ary function being computed. A non-interactive multi-client verifiable computation consists of n clients $\mathsf{P}_1 \ldots \mathsf{P}_n$ and a server* Serv *with the following structure:*

Setup stage: *All parties* P_i*'s,* $i \in [n]$ *and* Serv *have access to a setup* $\mathcal{G}$*, where party* P_i *obtains* $(\mathsf{pub}, \mathsf{sk}_i)$ *upon queries for some secret and public information.*

Offline stage: *Each client* P_i *sends a single message to the server. The server stores these as* $\hat{f}$*, an encoded version of* f*.*

Online stage: *This step is a query-response move: at each sub-session (or time period)* ssid*, upon receiving an input* (ssid, x_i) *for* $i \in [n]$*, the client* $\mathsf{P}_i(\mathsf{pub}, \mathsf{sk}_i, x_i)$ *computes some message* $(\hat{x}_i, \tau_i)$*. Then he sends* $\hat{x}_i$ *to the server and stores* τ_i *as a secret.*

The server Serv *carries out the computation on the messages received, and sends each client* P_i *for* $i \in [n]$ *an encoded output* $(\mathsf{ssid}, \hat{y}_i)$*.*

Each client computes and some output $y_i \cup \{\perp\}$ *based on* $(\mathsf{pub}, \mathsf{sk}_i, \hat{y}_i, \tau_i)$*, where* $\perp$ *means that he is not convinced with the outcome.*

Remark 2.2 *For the setup* $\mathcal{G}$*, we do not specify whether it is trusted in our definition. For our positive results, we want to minimize the requirements, and we showed that a self-registered PKI is enough for semi-honest client or malicious server corruptions. For the case of malicious clients corruptions, we further need an additional CRS. On the other hand, for our lower bound results, we rule out a large class of instantiations of* $\mathcal{G}$*, including the trusted PKI, CRS, shared secret randomness, and their combinations.*

Note that the trusted PKI is a setup where a trusted party generates public- and secret-key pairs for each user, and publishes the public keys to all users. The self-registered PKI is a weaker setup where each user generates their own key pairs, and registers the public keys with the setup so that the setup can publish the public keys to all users.

2.2 Security Definition

The security definition for non-interactive multi-client verifiable computation, MVC, turns out to be subtle. An MVC protocol cannot achieve the standard multi-party computation security, which requires that malicious clients have only one chance to provide their inputs, and cannot switch inputs later. In the non-interactive setting, if the server and some clients are simultaneously corrupted, then after gathering the transcripts of the honest clients, by definition the malicious clients can now select different inputs for themselves and learn the corresponding outputs. For example, consider $n = 2$. If client P_1 and the server are corrupted, then they effectively have access to oracle $f_1(*, x_2)$ where f_1 is the output of the first party, and x_2 is the honest input of P_2. The notation $*$ means that client P_1 can choose arbitrary inputs for itself and query this oracle a polynomial number of times. So our security definition would allow the adversary to learn $f_1(*, x_2)$ in the ideal world, and guarantees that this is the most that he can learn. On the other hand if interaction is allowed, it is well-understood that this issue can be avoided by standard techniques.

Based on this observation, we formally define the ideal functionality for *private* MVC in Figure 1 that captures the above issues, and soundness and privacy. The security of the protocol above follows the standard real/ideal paradigm [28, 29]. Here we only include the universal composability (UC) definition by Canetti [14, 13]. The standalone security definition can be found in [12, 23].

Definition 2.3 (Universal composability [14]) *A protocol* Π *securely realizes* $\mathcal{F}$ *if for any PPT adversary* $\mathcal{A}$ *in the real world, there exists a PPT simulator* $\mathcal{S}im$ *in the ideal world, so that no*

Multi-Client Private Verifiable Computation

The functionality is parameterized with an n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$. The functionality interacts with n clients P_i for $i \in [n]$, a distinguished server Serv, and the simulator $\mathcal{S}im$.

Initialization:
Upon receiving (Init) from client P_i, send $(\mathsf{Init}, \mathsf{P}_i)$ to notify the simulator $\mathcal{S}im$. Later, when $\mathcal{S}im$ returns $(\mathsf{Init}, \mathsf{P}_i)$, send a notification $(\mathsf{Init}, \mathsf{P}_i)$ to the server Serv.
Upon receiving (Init) from the server Serv, send $(\mathsf{Init}, \mathsf{Serv})$ to notify the simulator $\mathcal{S}im$.

Computation:
Upon receiving $(\mathsf{Input}, \mathsf{ssid}, x_i)$ from client P_i, send $(\mathsf{ssid}, \mathsf{P}_i)$ to notify $\mathcal{S}im$. Later, when $\mathcal{S}im$ returns $(\mathsf{ssid}, \mathsf{P}_i)$, store (ssid, x_i), and send a notification $(\mathsf{Input}, \mathsf{ssid}, \mathsf{P}_i)$ to server Serv.
Upon receiving $(\mathsf{Input}, \mathsf{ssid}, 1)$ from server Serv, retrieve (ssid, x_i) for all $i \in [n]$. If some (ssid, x_i) has not been stored yet, send $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to the server and all clients.

- **Server is not corrupted:** Compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$. Later when $\mathcal{S}im$ returns $(\mathsf{ssid}, \mathsf{P}_i, \phi)$, if $\phi = \mathsf{ok}$, send $(\mathsf{Output}, \mathsf{ssid}, y_i)$ to client P_i; if $\phi = \mathsf{fail}$, send $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to client P_i.
- **Server is corrupted:** Let $\mathcal{I} \subseteq [n]$ denote the set of indices corresponding to corrupted clients. Let $\overline{\mathcal{I}} := [n] \setminus \mathcal{I}$. Let $\mathbf{x}^*_{\mathcal{I}}$ denote the corrupted clients' inputs, $\mathbf{x}_{\overline{\mathcal{I}}}$ denote the remaining clients' inputs. Without loss of generality, we can renumber the clients such that $\mathcal{I} := \{1, 2, \ldots, |\mathcal{I}|\}$.

  The functionality provides to $\mathcal{S}im$ blackbox oracle access to the following oracle $O_{f,\mathcal{I}}$ where $\mathcal{S}im$ can choose inputs $\mathbf{x}^*_{\mathcal{I}}$ for corrupted clients to query:

  > Oracle $O_{f,\mathcal{I}}(\mathbf{x}^*_{\mathcal{I}})$:
  > Compute $(y_1, \ldots, y_n) \leftarrow f(\mathbf{x}^*_{\mathcal{I}}, \mathbf{x}_{\overline{\mathcal{I}}})$.
  > Output $\{y_i\}_{i \in \mathcal{I}}$ to $\mathcal{S}im$, and internally remember the last seen $\{y_i\}$ for $i \in \overline{\mathcal{I}}$.

  At any time (not necessarily simultaneously for all i), on receiving $(\mathsf{ssid}, \mathsf{P}_i, \phi)$ from $\mathcal{S}im$ for some $i \in \overline{\mathcal{I}}$, the functionality[a] sends to P_i $(\mathsf{Output}, \mathsf{ssid}, y_i)$ corresponding to the last seen y_i if $\phi = \mathsf{ok}$, otherwise it sends $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to P_i.

[a] Restricting to sending the last seen outputs does not lose generality, since the simulator can always repeat a previous query to the oracle $O_{f,\mathcal{I}}$.

Figure 1: Functionality $\mathcal{F}_{\mathsf{pVC}}$

PPT environment $\mathcal{Z}$ is able to tell the real world execution from the ideal world execution, i.e., $\mathsf{EXEC}_{\mathcal{A},\Pi,\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{S}im,\mathcal{F},\mathcal{Z}}$.

We can also define a notion of verifiable computation without input privacy. This is essentially the same definition, except that the server learns all the inputs of the clients. We present a formal description and provide a construction of this relaxed notion in Section C. In the following remarks we highlight and clarify a few properties of the stronger definition above:

Soundness against selective failure attacks: Our ideal functionality models a reactive functionality that has multiple sub-sessions after a pre-processing phase. Our definition implies soundness even if the server learns the decision bit of the clients since our security definition requires clients to report the outputs (and thus acceptance decisions) to the environment.

Communication model. We assume that the adversary controls the communication medium

between all parties. Our protocol later relies on PKI setup, and we can implement a secure channel with PKI. Therefore, while not explicitly stated, all our protocols are described assuming the secure channel ideal world.

Semi-honest vs. malicious corruption. Semi-honestly corrupted participants follow the protocol faithfully, but the adversary sees the internal states of all semi-honestly corrupted parties.

As mentioned above, due to the non-interactive nature, if the server and at least one client are simultaneously corrupted either in the malicious or semi-honest model, then our ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ implements a blackbox-access oracle which the simulator can query multiple times by specifying inputs for the malicious clients. For malicious corruption, the simulator can ask the ideal functionality to send outputs to different clients corresponding to different corrupted clients' inputs. For example, suppose P_1 and the server are maliciously corrupted, the simulator can ask the functionality to send $f_2(x_1, x_2, x_3)$ to P_2, and send $f_3(x_1', x_2, x_3)$ to P_3. For semi-honest corruption, the outputs sent back to the clients always correspond to inputs chosen by the environment.

Static corruption. We assume a static corruption model in this paper, where some protocol participants are corrupted at the beginning of protocol execution.

UC and stand-alone security. In the paper we use both the UC definition and standalone security definition. In the standalone security, the environment machine $\mathcal{Z}$ (i.e., the distinguisher) provides inputs to all protocol participants and the adversary at the beginning of protocol execution, and it receives outputs from these entities when the execution is complete. The environment and the adversary are not allowed to communicate during the protocol execution. Protocols secure in the standalone security model can be composed sequentially. On the other hand, in the UC framework, the environment and the adversary are always allowed to communicate. Protocols secure in the UC framework can be composted with arbitrary protocols. It is obvious that UC security implies stand-alone security.

Efficiency. An important feature of MVC is the *online* efficiency of the clients. Usually, we require the clients' computation time be much less than the complexity of the function f, so that over many online computations, the total cost of the clients will have low amortized cost. However, for private MVC, in some cases it is also interesting if the clients' computation time is similar to f, e.g. when the function f is simple. For example, it client P_1 and P_2 want to do a secure comparison over their inputs. The privacy requirement makes it interesting regardless of whether the clients' online computation time is smaller than the function being delegated. We do not specify a definition of efficiency but discuss it for each scheme individually.

3 Malicious Server and Semi-honest Client Corruptions

In this section and the following section, we will demonstrate constructions that achieve security against malicious adversaries, as long as there is no simultaneous server-client corruption.

Roadmap. As described in Section 1.2, our plan of action is: 1) define and obtain an mABE scheme; 2) use Goldwasser et al's compiler techniques [26] to achieve *attribute-hiding* mABE; and 3) show that attribute-hiding mABE implies private MVC.

All of the above primitives are proven secure under a malicious server or *semi-honestly* corrupted clients in this section. Then, in the following Section 4, we show a generic *compiler* based on non-interactive zero-knowledge proofs, such that any protocol secure against semi-honest corruption of

an arbitrary subset of clients, and additionally offering clients *perfect privacy* from one another, can be transformed into a protocol that is secure against either a malicious server or an arbitrary subset of malicious clients.

For convenience, in the remainder of the section, we focus on the case when only the first client P_1 learns output, and the remaining clients learn nothing. Based on this, we can obtain a protocol where every party learns outputs through simple parallel repetition.

3.1 Multi-Sender ABE

We define a multi-sender, two-outcome ABE scheme. Intuitively, the mABE functionality implements the following: consider n senders and a server. The first sender P_1 chooses two messages m_0 and m_1, and each P_i for $i \in [n]$ has an attribute x_i. The goal is for the server to m_b where $b = f(x_1, x_2, \ldots, x_n)$ while keeping m_{1-b} secret. We require the mABE scheme to be *non-interactive*, i.e., after an initial preprocessing phase in which the server learns an encoding of the function f, in each online phase, each sender sends a single message to the server, and the server can learn m_b.

We note that our mABE formulation can also be regarded as a generalization of the proxy oblivious transfer (POT) primitive proposed by Choi et al. [15]. We present the definition of POT in Appendix A. In other words, sender P_1 obliviously transfers one of m_0 and m_1 to the server, where which message is transferred is determined by a policy function f over all senders' attributes.

Figure 2 formally describes the mABE ideal functionality. We define mABE for the single-key setting, since our verifiable computation application is inherently single-key.

mABE Functionality

Functionality $\mathcal{F}^f_{\mathsf{mABE}}$ interacts with multiple senders $\mathsf{P}_1, \ldots, \mathsf{P}_n$, a server Serv, as well as a simulator $\mathcal{S}im$. The functionality is parameterized by a function $f : (\{0,1\}^\ell)^n \to \{0,1\}$.

- Upon receiving $(\mathsf{ssid}, m_0, m_1, x_1)$ from the sender P_1, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_1)$. Later, if $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_1)$, store $(\mathsf{ssid}, m_0, m_1, x_1)$, and notify Serv with $(\mathsf{ssid}, \mathsf{P}_1, x_1)$.
- Similarly, upon receiving (ssid, x_i) from other senders P_i for $i \in \{2, .., n\}$, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_i)$. Later when $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_i)$, if no (ssid, x_i) recorded yet, store it, and notify Serv with $(\mathsf{ssid}, \mathsf{P}_i, x_i)$.
- Upon receiving $(\mathsf{ssid}, 1)$ from Serv, if all $(\mathsf{ssid}, m_0, m_1, x_1)$, and (ssid, x_i) for $i \in \{2, .., n\}$ are recorded, return $(\mathsf{ssid}, m_{f(x_1,\ldots,x_n)})$ to Serv. Otherwise, if some tuple for ssid has not been recorded, return fail to Serv.

Figure 2: Functionality $\mathcal{F}^f_{\mathsf{mABE}}$

We now present our (non-interactive) protocol that realizes $\mathcal{F}^f_{\mathsf{mABE}}$ for any efficiently computable f. We use as building blocks a non-interactive POT protocol, and any two-outcome attribute-based encryption (ABE) scheme with a special structure where the attributes of ciphertexts can be encoded bit-by-bit. We formalize this local encoding property in the following. and observe that the ABE construction by Gorbunov, Vaikuntanathan, and Wee [30] satisfies this special property. Also, we remark that one can build a two-outcome ABE from a standard one, as shown by Goldwasser et al. [26]. Here we use ABE to denote the two-outcome ABE for simplicity.

Definition 3.1 (Two-outcome ABE with local encoding) *A two-outcome* attribute-based encryption scheme ABE *for a class of boolean functions* $\mathcal{F} = \{\mathcal{F}_\ell\}_{\ell\in\mathbb{N}}$ *from* $\{0,1\}^k \to \{0,1\}$*, is a tuple of polynomial time algorithms:* $\mathsf{ABE}.\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ *as follows:*

- $\mathsf{ABE.Setup}(1^k)$ *outputs a master public key* $\mathsf{mpk}_{\mathsf{ABE}}$ *and a master secret key* $\mathsf{msk}_{\mathsf{ABE}}$*.*
- $\mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, f)$ *On inputs* $\mathsf{msk}_{\mathsf{ABE}}$ *and a function* $f \in \mathcal{F}$*, output a function key* sk_f*.*
- $\mathsf{ABE.Enc}(\mathsf{mpk}_{\mathsf{ABE}}, x, m_0, m_1)$ *takes as input the master public key* $\mathsf{mpk}_{\mathsf{ABE}}$*, an attribute* $x \in \{0,1\}^\ell$ *for some* ℓ*, and two messages* m_0, m_1*, outputs a ciphertext* c*.*
- $\mathsf{ABE.Dec}(\mathsf{sk}_f, c)$ *takes as input a key* sk_f *and a ciphertext and outputs a message* m^**.*

Local encoding. We say that a two-outcome ABE scheme satisfies *local encoding* if the encryption algorithm $\mathsf{ABE.Enc}$ can be equivalently expressed as the following, where enc is a sub-algorithm:

1. select common randomness R;
2. for all $i \in [k]$, compute $\hat{x}[i] = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x[i]; R)$;
3. $\hat{m} = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m_0, m_1; R)$.

Finally, the ciphertext c can be written as $c := (\hat{x}[1], \hat{x}[2], \ldots, \hat{x}[k], \hat{m})$.

The correctness property guarantees that the decryptor can learn one of the messages m_b for $b = f(x)$, and the security guarantees that this is the only thing he can learn. We present the formal definitions in the appendix and also refer the readers to the work by Goldwasser et al. [26].

We present our construction of mABE in the $\mathcal{G}^{\mathsf{ABE}}$ setup model, where $\mathcal{G}^{\mathsf{ABE}}$ serves as a self-registered PKI which allows the sender to generate $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}}) \leftarrow \mathsf{ABE.KeyGen}(1^k)$, and register $\mathsf{mpk}_{\mathsf{ABE}}$. When queried by players other than the sender, it returns $\mathsf{mpk}_{\mathsf{ABE}}$.

Construction of mABE. Let $f : (\{0,1\}^\ell)^n \to \{0,1\}$ be a policy function, and without loss of generality, we let Serv denote the server, and let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ denote the senders. We make use of $(n-1)\cdot\ell$ instances of the functionality $\mathcal{F}_{\mathsf{POT}}$ indexed by (i,j) such that for $i \in \{2,\ldots,n\}$, all $j \in [\ell]$, in the (i,j)-th instance, P_1 plays the sender, P_i plays the chooser, and Serv plays the server. In the protocol below, we assume the existence of private channels; i.e. we assume that all parties encrypt their messages before sending them. This step is left implicit.[3] The parties act as follows:

- **Offline Stage**: Every party receives a function f as input. P_1 calls the setup $\mathcal{G}^{\mathsf{ABE}}$ to receive $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}})$, and computes some $\mathsf{sk}_f = \mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, f)$. He sends sk_f to the server. All the other clients runs an empty step.
- **Online Stage**:

– On input $(\mathsf{sid}, m_0, m_1, x_1)$, the sender P_1 does the following *in parallel*:

1. Sample a random string R. Compute $C = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m_0, m_1; R)$, $\hat{x}_1 = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x_1, R)$ (bit-by-bit) and sends them to the receiver Serv.
2. For $i \in \{2,\ldots,n\}, j \in [\ell]$, P_1 computes $\hat{c}_{i,j,0} = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, 0; R)$, and $\hat{c}_{i,j,1} = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, 1; R)$, and then sends $(\hat{c}_{i,j,0}, \hat{c}_{i,j,1})$ to the (i,j)-th instance of $\mathcal{F}_{\mathsf{POT}}$.

[3]Recall that our protocol for realizing $\mathcal{F}_{\mathsf{POT}}$ relies on a setup phase for establishing a PKI, so we could rely on this PKI for encrypting messages. If we instead were to use a protocol for $\mathcal{F}_{\mathsf{POT}}$ that did not rely on a PKI, we could simply add the establishment of a PKI to the setup phase of this protocol. Finally, we note that the assumption of private channels is not necessary: we could instead choose to leak P_{n+1}'s output to an eavesdropper. This would suffice for our purposes, but makes the resulting ideal functionality and the security proof a bit more involved.

– For $i \in \{2, \ldots, n\}$, upon receiving (sid, x_i), the party P_i sends, in parallel, $x_i[j]$ to the (i, j)-th instance of $\mathcal{F}_{\mathsf{POT}}$ for all $j \in [\ell]$. Here $x_i[j]$ denotes the j-th bit of x_i.
– Party Serv receives $\mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m_0, m_1)$, $\mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x_1)$ (bit-by-bit) from the sender P_1, and $\mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x_2)$, …, $\mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x_n)$ (bit-by-bit) via the instances of the functionality $\mathcal{F}_{\mathsf{POT}}$. He outputs m' by running the ABE decryption algorithm on the received ciphertexts using decryption key sk_f.

In Appendix D, we prove the following:

Theorem 3.2 *Assuming the existence of two-outcome ABE for a function $f : (\{0,1\}^\ell)^n \to \{0,1\}$ with the additional encoding property as above, then the protocol above securely realizes the ideal functionality $\mathcal{F}^f_{\mathsf{mABE}}$ in the $(\mathcal{F}_{\mathsf{POT}}, \mathcal{G}^{\mathsf{ABE}})$- hybrid model, against either (1) malicious server corruption, or (2) any semi-honest (static) corruption among any fixed set of clients.*

Using mABE as a building block, we can easily achieve verifiable computation without privacy. In Section C, we present the formal definition of MVC without privacy, the protocol that achieves this notion using mABE, and its security proof. We note that the construction is very similar to the one in the next section (see Theorem 3.4).

3.2 Achieving Attribute Hiding

In Figure 3, we define an attribute-hiding version of mABE, where the sender attributes are not leaked to the receiver. The attribute-hiding mABE functionality, denoted $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, is defined in almost the same way as $\mathcal{F}_{\mathsf{mABE}}$, except that when the functionality notifies the server, it only notifies $(\mathsf{ssid}, \mathsf{P}_i)$, without leaking the attributes x_i's.

ah-mABE Functionality

Functionality $\mathcal{F}^f_{\mathsf{ah\text{-}mABE}}$ interacts with multiple senders $\mathsf{P}_1, \ldots, \mathsf{P}_n$, a server Serv, as well as a simulator $\mathcal{S}im$. The functionality is parameterized by a function $f : (\{0,1\}^\ell)^n \to \{0,1\}$.

- Upon receiving $(\mathsf{ssid}, m_0, m_1, x_1)$ from the sender P_1, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_1)$. Later, if $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_1)$, store $(\mathsf{ssid}, m_0, m_1, x_1)$, and notify Serv with $(\mathsf{ssid}, \mathsf{P}_1)$.
- Similarly, upon receiving (ssid, x_i) from other senders P_i for $i \in \{2, .., n\}$, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_i)$. Later when $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_i)$, if no (ssid, x_i) recorded yet, store it, and notify Serv with $(\mathsf{ssid}, \mathsf{P}_i)$.
- Upon receiving $(\mathsf{ssid}, 1)$ from Serv, if all $(\mathsf{ssid}, m_0, m_1, x_1)$, and (ssid, x_i) for $i \in \{2, .., n\}$ are recorded, return $(\mathsf{ssid}, m_{f(x_1,\ldots,x_n)})$ to Serv. Otherwise, if some tuple for ssid has not been recorded, return fail to Serv.

Figure 3: Functionality $\mathcal{F}^f_{\mathsf{ah\text{-}mABE}}$

We present our protocol that realizes $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$ in the $\mathcal{G}^{\mathsf{FHE}}$ setup plus $\mathcal{F}_{\mathsf{mABE}}$ hybrid model, where $\mathcal{G}^{\mathsf{FHE}}$ serves as a self-registered PKI which allows the sender to generate $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^k)$, and register $\mathsf{pk}_{\mathsf{FHE}}$. When queried by parties other than the sender, it returns $\mathsf{pk}_{\mathsf{FHE}}$. Our construction can be viewed as a distributed version of that of Goldwasser et al. [26], who constructed attribute-hiding ABE (or functional encryption) from a non-hiding one. Briefly speaking, the first party P_1 generates a garbled circuit of the FHE decryption circuit, and then all

parties input ciphertexts of their attributes to $\mathcal{F}_{\mathsf{mABE}}$, to allow the server to learn *only* a set of labels to the garbled circuit. Then the server can learn only the outcome by evaluating the garbled circuit. Intuitively, since the attributes are encrypted, and the server can learn *only* a set of labels of the garbled circuit, the server can only learn the outcome but not the attributes of the parties.

Construction of $\mathsf{ah\text{-}mABE}$. Let $f : (\{0,1\}^\ell)^n \to \{0,1\}$ be a policy function, let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ be the senders, and let Serv be the receiver. Denote $g := \mathsf{Eval}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, f', (c, c', c_1), \ldots, c_n)$ where $\mathsf{pk}_{\mathsf{FHE}}$ is an FHE public key, $c, c', c_1 \ldots, c_n$ are ciphertexts and f' is an n-nary function that on input $((m_0, m_1, x_1), \ldots, x_n)$ outputs $m_{f(x_1,\ldots,x_n)}$. Assume the function g has an λ-bit output, and denote g_i as the function that outputs the i-bit of g. Then the parties do as follows:

- Upon receiving input $(\mathsf{ssid}, m_0, m_1, x_1)$, P_1 does the following:
  - Obtain $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$, and compute $(\Gamma, \{L_i^0, L_i^1\}_{i\in[\lambda]}) \leftarrow \mathsf{Gb.Garble}(1^k, \mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_{\mathsf{FHE}}, \cdot))$ where $\mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_{\mathsf{FHE}}, \cdot)$ is a circuit that takes a λ-bit ciphertext as input and outputs a single bit message.
  - Send (ssid, Γ) to the receiver Serv, and in parallel,
  - Compute $\hat{m}_0 \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, m_0)$, $\hat{m}_1 \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, m_1)$, $\hat{x}_1 \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, x_1)$, and send $(\mathsf{ssid}, L_j^0, L_j^1, (\hat{m}_0, \hat{m}_1, \hat{x}_1))$ to the functionality $\mathcal{F}_{\mathsf{mABE}}^{g_j}$ for all $j \in [\lambda]$.
- For $i \in [n] \setminus \{1\}$, upon receiving input (ssid, x_i), P_i first calls $\mathcal{G}^{\mathsf{FHE}}$ to obtain $\mathsf{pk}_{\mathsf{FHE}}$. Then he computes $\hat{x}_i \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, x_i)$ and sends $(\mathsf{ssid}, \hat{x}_i)$ to the functionality $\mathcal{F}_{\mathsf{mABE}}^{g_j}$ for all $j \in [\lambda]$.
- Upon receiving input $(\mathsf{ssid}, \hat{x}_1, \ldots, \hat{x}_n, \{L^{d_i}\}_{i\in[\lambda]}, \Gamma)$ from the ideal functionalities and P_1, the receiver Serv computes $\mathsf{Gb.Eval}(\Gamma, \{L^{d_i}\}_{i\in[\lambda]})$, and outputs the result of the evaluation.

We prove the following in Appendix D:

Theorem 3.3 *Assuming the existence of a fully homomorphic encryption scheme and a garbling scheme, the protocol above securely realizes the ideal functionality $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^f$ for any efficiently computable f in the $(\mathcal{F}_{\mathsf{mABE}}, \mathcal{G}^{\mathsf{FHE}})$-hybrid model, against either (1) malicious server corruption, or (2) semi-honest (static) corruption among any fixed set of senders.*

Using the functionality $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, we are able to build an MVC scheme that also achieves input and output privacy, in a similar fashion that (single-sender) private-index functional encryption implies private verifiable computation [26]. As before, we assume f outputs only one bit and only the first party receives the output. The construction is in the $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^f$ hybrid model. More formally, let $f : (\{0,1\}^\ell)^n \to \{0,1\}$ be a function to be delegated, let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ be the clients and Serv be the server. The the parties do as the following:

- Upon receiving input (ssid, x_1), P_1 samples two random inputs $m_0, m_1 \leftarrow \{0,1\}^\ell$ and sends $(\mathsf{ssid}, m_0, m_1, x_1)$ to the functionality $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^f$. Locally, he stores m_0, m_1.
- For $i \in [n]\setminus\{1\}$, upon receiving message (ssid, x_i), P_i sends (ssid, x_i) to the functionality $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^f$.
- Upon receiving (ssid, m) from $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^f$, the server sends P_1 the message (ssid, m).
- Upon receiving (ssid, m) from the server, P_1 checks whether $m = m_b$ for some $b \in \{0,1\}$. If so, he outputs b, and otherwise he outputs $\perp$.

In Appendix D, we prove:

Theorem 3.4 *The protocol above securely realizes $\mathcal{F}_{\mathsf{pVC}}$ in the $\mathcal{F}^{f}_{\mathsf{ah\text{-}mABE}}$-hybrid world, against either (1) malicious server corruption, or (2) semi-honest corruption of set of clients.*

Remark 3.5 *In fact, we can show that the protocol is secure against* any *set of corruptions in the $\mathcal{F}^{f}_{\mathsf{ah\text{-}mABE}}$-hybrid world. However, in the previous Theorems 3.2 and 3.3, we only know how to realize $\mathcal{F}^{f}_{\mathsf{ah\text{-}mABE}}$ against either (1) malicious server corruption, or (2) semi-honest (static) corruption of any fixed set of clients. Therefore, by putting things together we can obtain an input-private verifiable computation (*pVC*) protocol against such patterns of corruption. In Section 5, we will show that the corruption pattern cannot be extended; i.e., it is impossible to construct general* pVC *protocols against arbitrary server-client collusions. This in particular implies that it is impossible to construct a protocol for $\mathcal{F}^{f}_{\mathsf{ah\text{-}mABE}}$ against arbitrary server-client collusions.*

Efficiency of our construction. We outline the efficiency of a scheme where every client receives 1 bit of output — this can be achieved by a parallel repetition of our basic construction where only P_1 receives output. For such a private MVC scheme, the server runs in $\mathrm{poly}(\kappa) \cdot O(|f| \cdot n)$. If we instantiate using the ABE construction of Gorbunov et al. [30], the run-time and the communication cost for each client is $O(d \cdot n\ell\kappa)$, where d is the depth of the function f being delegated, ℓ is the input length, and κ is the security parameter. In Appendix 6 we also offer more detailed discussion. We note that if some non-falsifiable assumption is used, it is possible to remove the dependence on the circuit depth. As mentioned, the focus of this paper is on using falsifiable assumptions.

Also we note that efficiency of Choi et al.'s construction [15] does not depend on circuit depth — however they security is weaker in many respects. An interesting direction for future research is to construct a scheme (or prove impossibility) where the client online computation and communication does not depend on the number of parties n and the circuit depth d, by only using standard assumptions.

4 From Semi-Honest to Malicious Clients

In the previous section, we considered the case where the clients can be corrupted in the semi-honest way. In this section, we present a simple compiler that upgrades the previous protocol to one that is secure against any maliciously corrupted clients, and remains non-interactive. That is, the resulting protocol is secure against either malicious server, or against a set of malicious clients. Our construction only needs an additional setup $\mathcal{F}_{\mathsf{CRS}}$.

We note that if we allow more rounds of communication, it is already known how to achieve security against arbitrary malicious corruptions (i.e. of clients and/or the server) [2]. However in the non-interactive multi-client verifiable computation MVC, there are no known constructions. We have already demonstrated security against a malicious server, and we will consider arbitrary corruptions of both the server and the clients in Section 5. Here we address the case where multiple clients are corrupted, and demonstrate that if an MVC protocol offers security against the semi-honest corruption of an arbitrary subset of the clients, and, additionally it offers the clients *perfect privacy* from one another (as defined in Definition 4.1), then there exists a simple compiler for guaranteeing security against the malicious corruption of clients. Of course, if we are allowed for a trusted PKI during the setup phase, we could include honestly generated, committed randomness for each party, and then use a NIZK to prove that all messages were honestly generated. However,

we are interested in avoiding the use of trusted PKI, instead allowing each party to register the key of their choice; see Remark 2.2 for a discussion about trusted PKI and self-registered PKI.

Definition 4.1 *An* MVC *protocol* Π *has* perfect client privacy *if for all inputs* $x_1, \ldots, x_n$*, for an adversary* $\mathcal{A}$ *that semi-honestly corrupts some subset of the parties* $\{\mathsf{P}_i\}_{i\in\mathcal{I}}$ *where* $\mathcal{I} \subset [n]$*, and for every random tape* $r_{\mathcal{A}}$ *belonging to* $\mathcal{A}$*, there exists a simulator* $\mathcal{S}im$ *such that the following distributions are identical*

$$\left\{\mathsf{View}_{\Pi(x_1,\ldots,x_n),\mathcal{A}}\right\} \equiv \{\mathcal{S}im(\{x_i, y_i\}_{i\in\mathcal{I}}, r_{\mathcal{A}})\}$$

where $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$*, and* $\mathsf{View}_{\Pi(x_1,\ldots,x_n),\mathcal{A}}$ *is the view of the adversary when the inputs to the clients are* $(x_1, \ldots, x_n)$*. In particular, the view contains random string* $r_{\mathcal{A}}$*, inputs* $\{x_i\}_{i\in\mathcal{I}}$*, and the message received from the server, and the messages generated by honest clients.*

Note that what distinguishes this from a standard requirement for semi-honest corruption is that we require indistinguishability to hold for every random tape of the adversary, rather than only on average. Intuitively, if a protocol meets this requirement, we can simplify the standard compilation techniques, since the adversary is free to use the random tape of his choice. To achieve security in the presence of malicious adversaries, it suffices to have the clients prove (using a NIZK) that their messages are consistent with *some* random string. Formally, we are able to achieve the following theorem. We present the proof in Section D.

Theorem 4.2 *Suppose there exists an* MVC *protocol* Π *in self-registered PKI setup hybrid model that is secure against semi-honest client corruptions, and that* Π *has perfect client privacy. Then there exists an* MVC *protocol* Π' *in the ZK and the self-registered PKI setup hybrid model, which is secure against malicious client corruptions.*

In order to apply the compiler results to our protocol, we need to show that our constructions have the desired property. We show this by the following claim:

Claim 4.3 *If the underlying* ABE *and* FHE *and the garbling schemes is perfectly correct, then the private* MVC *protocol from Section 3 has perfect client privacy.*

Proof: Since $\mathsf{P}_2, \ldots, \mathsf{P}_n$ do not receive messages or output, their views can be simulated easily. Now, we give a simulation of P_1's view. In the honest protocol, P_1 samples random strings m_0, m_1 and some r for generating ciphertexts of the ABE, FHE and garbling schemes. Suppose these schemes have perfect correctness. Then for every r the honest P_1 will receive either m_0 or m_1 from the server, depending on $b := f(x_1, \ldots, x_n)$. Therefore, given the result of the computation, b, and the random tape of P_1, $R = (m_0, m_1, r)$, the simulator can simply output m_b as the message from the server, producing an identical view. This completes the simulation of his view. ■

As a consequence of this theorem and Theorems 3.2, 3.3, 3.4, and the fact that non-interactive ZK can be implemented in the CRS model, we are able to construct a private MVC protocol using CRS and self-registered PKI. We summarize this by the following theorem:

Theorem 4.4 *Assume the existence of a fully homomorphic encryption scheme, a garbling scheme, and an ABE that has local encoding property. Assume the primitives have perfect correctness. Then for any efficiently computable* f*, there exists an* MVC *protocol that securely realizes the ideal functionality* $\mathcal{F}^f_{\mathsf{pVC}}$ *in the CRS and self-registered PKI hybrid model, against (1) any malicious server corruption, or (2) any malicious (static) corruption among any fixed set of clients.*

5 Client-Server Collusion

In this section, we consider the remaining, more complicated case where the server and clients can be corrupted at the same time. We show that even for a seemingly simple case where only one client and the server are corrupted together, it is impossible to construct private MVC for general functions, under a large class of instantiations of $\mathcal{G}$ setup including trusted PKI (which is stronger than self-registered PKI, see Remark 2.2), CRS, shared secret randomness, etc. The lower bound holds even in the standalone setting, and for semi-honest corruptions.

In particular, we consider the case with two clients and one server, where the function being delegated is a universal circuit $U(\cdot,\cdot)$, the first client's input is a circuit C, the second client's is a string x. The server returns $U(C,x) = C(x)$ to both parties. If there exists a private MVC protocol with respect to such U, i.e. if there exists a protocol that realizes $\mathcal{F}_{\mathsf{pVC}}$, then, even if it is only secure against semi-honest corruption and only in the standalone setting, we can construct an obfuscator for any circuit. (We refer the reader to the remark following Definition 2.3 for a definition of the standalone setting.) By previous lower bounds for obfuscation [4], this leads to an impossibility result. We present the formal statement below.

We note that there is a similar lower bound argument in the server-aided MPC setting in the work [2]. Our lower bound further shows that even a natural relaxation of security (where the ideal functionality can be called multiple times if there is server-client corruption) is not achievable for all functionalities.

Theorem 5.1 *Suppose there exist an instantiation of $\mathcal{G}$ setup and a private* MVC *protocol* Π *(i.e., one that realizes $\mathcal{F}_{\mathsf{pVC}}$) for all efficiently computable functions in the $\mathcal{G}$ setup hybrid world, against semi-honest corruptions for arbitrary parties in the standalone model, then there exists an obfuscator for any circuit C secure under the virtual black-box simulation.*

Proof: Consider the case where two clients want to delegate the computation of the universal circuit $U(\cdot,\cdot)$ to the server; the first client provides a circuit C, and the second provides an input x. Then the honest server returns $C(x)$ to both parties. Suppose there exists an instantiation of setup $\mathcal{G}$ and a secure protocol Π that achieve this goal, then we construct an obfuscator O that on input C does the following:

- O simulates the setup $\mathcal{G}$ and the role of each client in the offline stage to obtain $\mathsf{pub}, \mathsf{sk}_1, \mathsf{sk}_2, \hat{f}$.
- O simulates the first client's procedure on input C in the online stage. Let $\hat{C}$ be the message that P_1 sends to the server.
- O outputs $(\hat{C}, \mathsf{pub}, \mathsf{sk}_2, \hat{f})$ as an obfuscation of C, i.e. $O(C)$.

To evaluate $O(C)$ on input x, the evaluator simulates P_2's online phase to create an encoding of x using $\mathsf{pub}, \mathsf{sk}_2$, and then simulates the (corrupted) server to evaluate $\hat{C}, \hat{x}$ with the encoded version of the function $\hat{f}$.

The correctness follows immediately from the correctness of the protocol Π, and the efficiency of the obfuscator follows directly from the efficiency of the parties in the protocol Π. In the rest of the proof, we are going to show the virtual black-box (VBB) simulation property. In particular we will turn the protocol simulator into a VBB obfuscation simulator.

Now we analyze the construction. In particular, we want to show given an adversary $\mathcal{A}$ attacking the security of the obfuscation, we are going to construct a simulator $\mathcal{S}im$ such that the probability $\mathcal{A}(O(C)) = 1$ is close to that of $\mathcal{S}im^C(1^k) = 1$ up to a negligible factor for all polynomial-sized

circuits C. We do this by defining a particular adversary $\mathcal{A}^*$ that attacks $\Pi^{\mathcal{G}}$, and using the protocol simulator that is guaranteed to exist for this adversary by the security of the MVC protocol.

Given $\mathcal{A}$ and any poly-sized circuit C, we define the following experiment in the $\mathcal{G}$-hybrid world. Let $\mathcal{Z}^*$ be an environment and $\mathcal{A}^*$ be an adversary attacking protocol $\Pi^{\mathcal{G}}$. $\mathcal{A}^*$ corrupts the server and the second party at the beginning. He queries the ideal functionality $\mathcal{G}$ and stores the reply $(\mathsf{pub}, \mathsf{sk}_2)$. Upon receiving a message from P_1 during the offline stage (on behalf of the server), he uses this message, along with the offline message that an honest P_2 would send, to construct $\hat{f}$ as the server would do. When P_1 sends a message $\hat{C}$ to the server during the online phase, $\mathcal{A}^*$ interprets $(\hat{C}, \mathsf{pub}, \mathsf{sk}_2, \hat{f})$ as an obfuscation of $O(C)$. Then $\mathcal{A}^*$ runs $\mathcal{A}$ on the interpreted $O(C)$ and passes $\mathcal{A}$'s output to $\mathcal{Z}^*$. $\mathcal{Z}^*$ outputs this as the output of the experiment $\mathsf{EXEC}_{\mathcal{A}^*,\Pi,\mathcal{Z}^*}$.

Now we are ready to construct the simulator $\mathcal{S}im$. By the premise that $\Pi^{\mathcal{G}}$ realizes $\mathcal{F}_{\mathsf{pVC}}$, for this $\mathcal{A}^*$, there exists $\mathcal{S}im^*$ such that for this particular $\mathcal{Z}^*$, we have $\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{A}^*,\Pi,\mathcal{Z}^*} \approx \mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{pVC}},\mathcal{Z}^*}$. Given such $\mathcal{S}im^*$, we define a simulator $\mathcal{S}im$ for the VBB obfuscation as follows:

- $\mathcal{S}im$ basically simulates the execution of $\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{pVC}},\mathcal{Z}^*}$.
- Whenever the protocol simulator $\mathcal{S}im^*$ queries the oracle $O_{U,\{2\}}(\cdot)$ in the ideal functionality with some modified P_2's input x_2', $\mathcal{S}im$ simulates it using a black-box query to C with input x_2' and returns $C(x_2')$ to $\mathcal{S}im^*$.
- Then $\mathcal{S}im$ outputs whatever the output of the experiment $\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{pVC}},\mathcal{Z}^*}$.

Lemma 5.2 *For the simulator $\mathcal{S}im$ described above, there exists a negligible function $\nu(\cdot)$ such that $|\Pr[\mathcal{A}(O(C)) = 1] - \Pr[\mathcal{S}im^C(1^k) = 1]| < \nu(k)$.*

Proof: Assume there is a non-negligible function ε with $\Pr[\mathcal{A}(O(C)) = 1] - \Pr[\mathcal{S}im^C(1^k) = 1] > \varepsilon$. We show that the real and simulation worlds in the protocol are distinguishable.

According to the description of $\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{A}^*,\Pi,\mathcal{Z}^*}$, the output of such experiment is identical to that of $\mathcal{A}(O(C))$. On the other hand, the output of $\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{pVC}},\mathcal{Z}^*}$ is exactly the same as that of $\mathcal{S}im^C(1^k)$. So this means the executions of the protocol are distinguishable by ε, which reaches a contradiction. Thus we complete the proof. ∎

The above shows that $\mathcal{S}im$ is a good VBB simulator. ∎

6 Instantiations and Efficiency

In this section, we discuss the instantiations of our building blocks. We need a two-outcome attribute encryption scheme with the local encoding property, a fully homomorphic encryption scheme, and a garbling scheme. In particular we can use any instantiation of FHE schemes, e.g. one by Brakerski [11], and any instantiation of Yao's garbling scheme.

The attribute based encryption (ABE) constructed by Gorbunov, Vaikuntanathan, Wee [30] actually achieves the requirements of regular ABE with the local encoding property. Goldwasser et al. [26] showed a generic way to achieve two-outcome ABE from a regular one. So by plugging the GSW ABE scheme and using the generic technique, we achieve the two-outcome ABE as required by Definition 3.1.

For our private MVC scheme, the server clearly runs in $\mathrm{poly}(\kappa, f)$. For the clients, $\mathsf{P}_2, \ldots, \mathsf{P}_n$ runs in time $O(\ell\kappa)$, where ℓ is the input length; P_1 generates $O(n\ell\kappa)$ ABE ciphertexts plus a

garble circuit of size $O(\kappa)$, where n is the number of parties, ℓ is the input length, and κ is the security parameter. However, the ciphertexts' length for the currently best known ABE construction of Gorbunov, Vaikuntanathan, Wee [30] depends on the circuit depth (independent of the size). Therefore, P_1's running time (the communication complexity as well) depends on $O(d \cdot n\ell\kappa)$, where d is the depth of the function being delegated. The construction of Choi et al. [15] has better online efficiency for clients that is independent of the function complexity, but has the issues of adaptive soundness and is vulnerable to selective failure attacks. The construction using multi-input functional encryption [24] can achieve better efficiency but their solution inherently requires the existence of indistinguishable obfuscation, which is a stronger assumption and has large overhead.

Acknowledgments

This research was sponsored in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence, or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163. Springer, July 2010.

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Apr. 2012.

[3] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 863–874. ACM Press, Nov. 2013.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Aug. 2001.

[5] M. Bellare, V. T. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Dec. 2012.

[6] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, Oct. 2012.

[7] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Aug. 2013.

[8] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Usenix Security Symposium*, 2014.

[9] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, Aug. 2011.

[10] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[11] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Aug. 2012.

[12] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[13] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

[14] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[15] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 499–518. Springer, Mar. 2013.

[16] K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Aug. 2010.

[17] K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 151–168. Springer, Aug. 2011.

[18] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 501–512. ACM Press, Oct. 2012.

[19] S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[20] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Aug. 2010.

[21] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, May 2013.

[22] C. Gentry, A. B. Lewko, A. Sahai, and B. Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[23] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[24] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology—Eurocrypt 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, 2014.

[25] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Aug. 2013.

[26] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[27] S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[28] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[29] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[30] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[31] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. `http://eprint.iacr.org/`.

[32] S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 797–808. ACM Press, Oct. 2012.

[33] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. Trueset: Nearly practical veriable set computations. In *Usenix Security Symposium*, 2014.

[34] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242. Springer, Mar. 2013.

[35] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 91–110. Springer, Aug. 2011.

[36] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Mar. 2012.

[37] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*. The Internet Society, Feb. 2011.

A Definitions

Here we provide the various definitions we use in this paper.

A.1 Two-outcome ABE

Definition A.1 (Correctness of two-outcome ABE [26]) *For any polynomial $n(\cdot)$, for every sufficiently large security parameter κ, if $n = n(\kappa)$, for all boolean functions $f \in \mathcal{F}_n$, attributes $x \in \{0,1\}^n$, messages $M_0, M_1 \in \mathcal{M}$, there exists some negligible $\nu(\cdot)$ such that*

$$\Pr\left[\begin{array}{l}(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa);\\ \mathsf{sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, f);\\ c \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_{\mathsf{ABE}}, x, m_0, m_1);\\ m = \mathsf{ABE.Dec}(\mathsf{sk}_f, c) :\\ m = m_{f(x)}\end{array}\right] = 1 - \nu(\kappa).$$

If $\nu = 0$, then the scheme has perfect correctness.

Then we define the security for single-key two-outcome ABE.

Definition A.2 (Security of two-outcome ABE [26]) *Let* ABE *be a two-outcome ABE scheme for the class of boolean functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ and associated message space $\mathcal{M}$ and let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be a triple of PPT adversaries. Consider the following experiment.*

- $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$
- $(f, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{mpk}_{\mathsf{ABE}})$
- $\mathsf{sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, f)$
- $(m, m_0, m_1, x, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{st}_1, \mathsf{sk}_f)$
- *choose a bit b at random. Then let*

$$c = \begin{cases} \mathsf{ABE.Enc}(\mathsf{mpk}_{\mathsf{ABE}}, x, m, m_b), & \text{if } f(x) = 0,\\ \mathsf{ABE.Enc}(\mathsf{mpk}_{\mathsf{ABE}}, x, m_b, m), & \text{otherwise.}\end{cases}$$

- $b' \leftarrow \mathcal{A}_3(\mathsf{st}_2, c)$. *If $b = b'$, and there exists n such that, for all $f \in \mathcal{F}_n$, messages $m.m_0, m_1 \in \mathcal{M}$, $|m_0| = |m_1|$, $x \in \{0,1\}^n$, output 1. Else output 0.*

We say the scheme is a full-secure single-key two-outcome ABE if for all PPT adversaries $\mathcal{A}$, and for all sufficiently large κ, the probability that the experiment outputs 1 is bounded by $1/2+\nu(k)$ for some negligible function ν.

A.2 Garbling Schemes

Definition A.3 (Garbling schemes [6]) *A garbling scheme for a family of circuits $C = \{C_n\}_{n\in\mathbb{N}}$ with C_n a set of boolean circuits taking as input n bits, is a tuple of PPT algorithms $\mathsf{Gb} = \mathsf{Gb}.\{\mathsf{Garble}, \mathsf{Enc}, \mathsf{Eval}\}$ such that*

- $\mathsf{Gb}.\mathsf{Garble}(1^\kappa, C)$ *takes as input the security parameter κ and a circuit $C \in C_n$ for some n and outputs the garbled circuit Γ and a secret key sk.*
- $\mathsf{Gb}.\mathsf{Enc}(\mathsf{sk}, x)$ *takes as input x and outputs an encoding c,*
- $\mathsf{Gb}.\mathsf{Eval}(\Gamma, c)$ *takes as input a garbled circuit Γ and an encoding c, and outputs a value y which should be $C(x)$.*

The correctness and efficiency properties are straight-forward. Next we consider a special property of the encoding of the Yao's garbled scheme, which we will use in this paper. The secret key has the form $\mathsf{sk} = \{L_i^0, L_i^1\}_{i\in[n]}$, and the encoding of an input x of n bits is of the form $c = (L^{x_1}, L^{x_2}, \ldots, L^{x_n})$, where x_i is the i-th bit of x.

Then we are going to define the security of garbling schemes.

Definition A.4 (Input and circuit privacy) *A garbling scheme Gb for a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ is input and circuit private if there exists a PPT simulator $\mathcal{S}im$ such that for every adversaries $\mathcal{A}$ and D, for all sufficiently large κ,*

$$\left|\Pr\left[\begin{array}{c}(x,C,\alpha)\leftarrow A(1^k);\\ (\Gamma,\mathsf{sk})\leftarrow \mathsf{Gb}.\mathsf{Garble}(1^\kappa,C);\\ c\leftarrow \mathsf{Gb}.\mathsf{Enc}(\mathsf{sk},x):\\ D(\alpha,x,C,\Gamma,c)=1\end{array}\right] - \Pr\left[\begin{array}{c}(x,C,\alpha)\leftarrow A(1^k);\\ (\tilde{\Gamma},\tilde{c})\leftarrow \mathcal{S}im(1^\kappa,C(x),1^{|C|},1^{|x|}):\\ D(\alpha,x,C,\tilde{\Gamma},\tilde{c})=1\end{array}\right]\right| = \nu(k)$$

for some negligible $\nu(\cdot)$, where we consider only $\mathcal{A}$ such that for some $n, x \in \{0,1\}^n$ and $C \in C_n$.

A.3 Extractable Witness Encryption

Definition A.5 (Witness encryption [19]) *A witness encryption scheme for language $L \in NP$ with witness relation R_L consists of polynomial-time algorithms $\mathsf{WE}.\{\mathsf{Enc}, \mathsf{Dec}\}$ such that:*

- *Encryption $\mathsf{WE}.\mathsf{Enc}(1^\kappa, x, b)$: takes as input a security parameter κ, a statement $x \in \{0,1\}^*$, a bit b and outputs a ciphertext c.*
- *Decryption $\mathsf{WE}.\mathsf{Dec}(w, c)$: takes as input a witness $w \in \{0,1\}^*$ and a ciphertext c and outputs a bit b or $\perp$.*

Correctness: *For all $(x, w) \in R_L$, for all bits b for every sufficiently large security parameter κ, we have*

$$\Pr[c \leftarrow \mathsf{WE}.\mathsf{Enc}(1^\kappa, x, b) : \mathsf{WE}.\mathsf{Dec}(w, c) = b] = 1 - \nu(\kappa),$$

for some negligible ν.

Definition A.6 (Extractable security [25]) *A witness encryption scheme for a language $L \in NP$ is secure if for all PPT adversaries $\mathcal{A}$, and all poly q, there exists a PPT extractor E and a poly p such that for all auxiliary input z and all $x \in \{0,1\}^*$, the following holds:*

$$\Pr[b \leftarrow \{0,1\}; c \leftarrow \mathsf{WE.Enc}(1^\kappa, x, b) : A(x, c, b) = b] \geq 1/2 + 1/q(|x|)$$
$$\Rightarrow \Pr[E(x, z) = w : (x, w) \in R_L] \geq 1/p(|x|).$$

A.4 Obfuscations

Definition A.7 (Circuit obfuscation [4]) *A probabilistic algorithm O is a (circuit) obfuscator for the collection $\mathcal{F}$ of circuits if the following holds:*

- *(functionality) For every circuit $C \in \mathcal{F}$, the string $O(C)$ describes a circuit that computes the same function as C.*
- *(polynomial slowdown) There is a polynomial p such that for every circuit $C \in \mathcal{F}$, we have $|O(C)| \leq p(|C|)$.*
- *("virtual black box" (VBB) property) For any PPT $\mathcal{A}$, there is a PPT $\mathcal{S}im$ and a negligible function ν such that for all circuits $C \in \mathcal{F}$, it holds that*

$$\left|\Pr\left[A(O(C)) = 1\right] - \Pr\left[\mathcal{S}im^C(1^{|C|}) = 1\right]\right| \leq \nu(|C|).$$

We say that O is efficient if it runs in polynomial time. If we omit specifying the collection $\mathcal{F}$, then it is assumed to be the collection of all circuits.

A.5 Proxy Oblivious Transfer

Choi et al. [15] recently defined and constructed proxy oblivious transfer. Instead of taking the game based security definitions from the paper by Choi et al., here we define the security of POT in the real/ideal paradigm, which provides a stronger security guarantee. In the ideal functionality below, we omit the session id for notational simplicity. We remark that in each session, the functionality could accept multiple new inputs; we assign a sub-session id, i.e., ssid, for each new input.

Theorem A.8 ([15]) *There is a* non-interactive *protocol which realizes $\mathcal{F}_{\mathsf{POT}}$ in the self-registered PKI setup $\mathcal{G}^{\mathsf{Diffie-Hellman}}$-hybrid model, against (1) any malicious server corruption, or (2) any semi-honest (static) corruption among any fixed set of clients.*

Choi et al. [15] constructed a *non-interactive* protocol in the offline/online model that realizes the ideal functionality $\mathcal{F}_{\mathsf{POT}}$. In this model, two clients run some protocol in an offline stage, prior to learning their inputs, and then complete the protocol in the online stage, after receiving their inputs. In their construction, the clients do not need to interact in the offline stage, and in the online stage both the sender and chooser send a single message to the server. The construction relies on the existence of non-interactive key agreement.

Proxy Oblivious Transfer

Functionality $\mathcal{F}_{\mathsf{POT}}$ interacts with a sender P_S, a chooser P_C, a receiver P_R, and the simulator $\mathcal{S}im$.

- Upon receiving $(\mathsf{ssid}, m_0, m_1)$ from the sender P_S, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_S)$. Later, when $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_S)$, if no value $(\mathsf{ssid}, m_0', m_1')$ has been recorded yet, store it and notify P_R with $(\mathsf{ssid}, \mathsf{P}_S)$.
- Similarly, upon receiving (ssid, b) from the chooser P_C, notify $\mathcal{S}im$ with $(\mathsf{ssid}, \mathsf{P}_C)$. Later, when $\mathcal{S}im$ replies with $(\mathsf{ssid}, \mathsf{P}_C)$, if no value (ssid, b') has been recorded yet, store it and notify P_R with $(\mathsf{ssid}, \mathsf{P}_C)$.
- Upon receiving $(\mathsf{ssid}, 1)$ from P_R, if both $(\mathsf{ssid}, m_0, m_1)$ and (ssid, b) are recorded, send (ssid, m_b) to the receiver P_R; else send fail.

Figure 4: Functionality $\mathcal{F}_{\mathsf{POT}}$

B Multi-Sender ABE and Extractable Witness Encryption

Our impossibility result for private MVC against client-server corruptions also rules out the possibility of attribute hiding multi-sender attribute based encryption schemes for general functions under the same corruption (server-sender corruptions) model. In this section, we consider whether we can achieve the weaker functionality mABE (without attribute hiding) for such model. In particular, we show a similar lower bound result – any secure mABE protocol against server-sender (or server-client) corruptions, (even in the standalone model, against semi-honest corruptions) implies extractable witness encryptions. So via the route of mABE, it is not possible to construct a non-private MVC protocol against arbitrary corruption only based on standard assumptions.

On the other hand, we show a construction of mABE from extractable witness encryption (plus SNARK if we want the message to be succinct) with better asymptotic efficiency, i.e. the complexity in the online phase does not depend on the circuit depth. The construction is UC secure against either a malicious server or malicious senders corruptions. Here we leave as an open question how to construct a UC secure mABE against arbitrary corruptions (which requires non-falsifiable assumptions).

B.1 mABE Implies Extractable Witness Encryption

In this section we show that mABE for general functions implies an extractable witness encryption for all NP relations in the following theorem.

Theorem B.1 *Suppose there exist an instantiation of $\mathcal{G}$ setup and a non-interactive protocol Π in the offline/online model that realizes the $\mathcal{F}_{\mathsf{mABE}}^{f}$ for all efficiently computable functions, in the $\mathcal{G}$-setup hybrid world, against semi-honest corruptions for arbitrary parties in the standalone model, then there exists a extractable witness encryption (WE) scheme for all NP relations.*

Proof: To prove the theorem, we need to construct a extractable WE for all NP relations. Consider any NP relation $R(x, w)$ that on inputs a string x and witness w outputs whether x is in the language. Then we define a WE (for bits) scheme as follows:

- To encrypt a message $b \in \{0, 1\}$ with respect to a statement x, first we define a functionality $\mathcal{F}_{\mathsf{mABE}}^{R}$ that interacts with three parties: $\mathsf{P}_1, \mathsf{P}_2, \mathsf{Serv}$, where $\mathsf{P}_1, \mathsf{P}_2$ are the senders holding

messages $(m_0, m_1, x), (w)$ respectively, and Serv is the server. Serv will learn the input message m_1 of P_1 if $R(x, w) = 1$, or otherwise the other m_0. By the premise of the theorem, there exists a non-interactive protocol Π in the offline/online model that realizes $\mathcal{F}^R_{\mathsf{mABE}}$ with some setup $\mathcal{G}$.

Now the encryptor first simulates the parties of the protocol Π in the offline stage. Then he simulates P_1 with input $(m_0 := \perp, m_1 = b, x)$. Then he outputs all the messages $\mathsf{P}_2, \mathsf{Serv}$ receives in the offline stage, and the message of P_1 sent to the server.

- To decrypt a ciphertext c with some statement x and its witness w, i.e. $R(x, w) = 1$, the decryptor first interprets the ciphertext as the encoded message of P_1 plus the information of P_2 and Serv. Now he simulates P_2 with input a witness w and then applies Serv's strategy. At the end, he outputs whatever Serv outputs.

The correctness property follows directly from the correctness of the protocol. Now we want to prove extractability: for any statement x, if there exists an adversary $\mathcal{A}$ who can predict messages from the corresponding ciphertexts, there exists an extractor who can output a witness w such that $R(x, w) = 1$. We show this in the following lemma:

Lemma B.2 *For any statement x, suppose there exists an adversary $\mathcal{A}$ such that $\Pr[\mathcal{A}(x, C) = b] > 1/2 + \varepsilon$, where C is an encryption of a random bit $b \in \{0, 1\}$, and ε is some non-negligible quantity. Then there exists an extractor E such that $\Pr[E(x) = w] > \varepsilon'$ such that $R(x, w) = 1$ for some non-negligible ε'.*

Proof: Given any adversary $\mathcal{A}$ who can decrypt the WE ciphertexts as the premise, we are going to construct an extractor from the simulator in the protocol Π.

First consider the experiment (in the $\mathcal{G}$ setup hybrid world) with the following protocol adversary $\mathcal{A}^*$ and environment $\mathcal{Z}^*$. $\mathcal{A}^*$ corrupts the parties P_2 and Serv. In the offline stage, P_1 receives some $(\mathsf{pub}, \mathsf{sk}_1)$ and sends some $\hat{f}$ to $\mathcal{A}^*$. At the end of the offline stage, $\mathcal{A}^*$ receives $(\mathsf{pub}, \mathsf{sk}_2, \hat{f})$.

Then in the online stage, $\mathcal{Z}^*$ sends x and a random bit b to P_1. Then P_1 sends some message to $\mathcal{A}^*$. Upon receiving the message from P_1, $\mathcal{A}^*$ interprets it and the information $(\mathsf{pub}, \mathsf{sk}_2, \hat{f})$ he got in the offline stage as a WE ciphertext C. Then $\mathcal{A}^*$ runs $\mathcal{A}(C)$ and outputs whatever $\mathcal{A}$'s output to the environment. Then $\mathcal{Z}^*$ outputs this bit as the view of the execution $\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{A}^*, \Pi, \mathcal{Z}^*}$.

By the assumption that the protocol Π realizes $\mathcal{F}^R_{\mathsf{mABE}}$, there exists a simulator $\mathcal{S}im^*$ such that $\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{A}^*, \Pi, \mathcal{Z}^*} \approx \mathsf{EXEC}_{\mathcal{S}im^*, \mathcal{F}^R_{\mathsf{mABE}}, \mathcal{Z}^*}$. Given such $\mathcal{S}im^*$, we define an extractor E as follows: E simulates the experiment of $\mathsf{EXEC}_{\mathcal{S}im^*, \mathcal{F}^R_{\mathsf{mABE}}, \mathcal{Z}^*}$. Also E records all queries the protocol simulator $\mathcal{S}im^*$ had made to the functionality $\mathcal{F}^R_{\mathsf{mABE}}$. Let W be the list, and E checks every element $w \in W$. If there exists any w such that $R(x, w) = 1$, E outputs w as a witness of x. We know the extractor succeeds if W contains a witness of x.

Now we will argue the probability that W contains a witness of x is non-negligible. Denote this event as P and the associate probability as ε'. We want to show that ε' is also non-negligible.

By the definition of $\mathcal{A}^*$ and $\mathcal{Z}^*$, we know that $\Pr[\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{A}^*, \Pi, \mathcal{Z}^*} = b] = \Pr[A(x, C) = b] > 1/2 + \varepsilon$, where ε is non-negligible. Since the simulator $\mathcal{S}im^*$ is a good protocol simulator, we must have

$\Pr[\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{ah\text{-}mABE}},\mathcal{Z}^*} = b] > 1/2 + \varepsilon/2$. We observe that:

$$\begin{aligned}
&1/2 + \varepsilon/2 \\
&< \Pr\left[\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{ah\text{-}mABE}},\mathcal{Z}^*} = b\right] \\
&< \Pr\left[\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{ah\text{-}mABE}},\mathcal{Z}^*} = b \middle| \neg P\right] \cdot \Pr[\neg P] + \Pr[P] \\
&= \Pr\left[\mathsf{EXEC}_{\mathcal{S}im^*,\mathcal{F}_{\mathsf{ah\text{-}mABE}},\mathcal{Z}^*} = b \middle| \neg P\right] \cdot (1 - \varepsilon') + \varepsilon' \\
&= 1/2 \cdot (1 - \varepsilon') + \varepsilon' \\
&= 1/2 + \varepsilon'/2.
\end{aligned}$$

The first two inequalities are explained as above. The first equality comes from the definition of $\Pr[P] = \varepsilon'$. The second one comes from the fact that conditioning on the event $\neg P$, the ideal functionality will always return $\perp$ whenever $\mathcal{S}im^*$ queries. Thus the message bit b is independent of other randomness in the experiment. In such case, the probability that the experiment outputs the bit is clearly $1/2$. The last two equalities are trivial.

The above calculations show that $\varepsilon' > \varepsilon$ and therefore, ε' must be non-negligible. ■

This completes the proof of the theorem. ■

B.2 More Efficient mABE Using Non-falsifiable Assumptions

We show a construction of mABE from extractable witness encryption and SNARKs. Here the complexity does not depend on the circuit depth. In particular we achieve the following:

Theorem B.3 *Suppose there exists an extractable witness encryption scheme for general NP relations, and a digital signature scheme existentially unforgeable against adaptive chosen-message attacks, and a secure SNARK scheme. Then, in the PKI-setup hybrid world, there exists a* succinct *non-interactive* mABE *protocol UC secure against (1) maliciously corrupted server or (2) semi-honest senders.*

Our construction uses extractable witness encryption schemes with a PKI setup. The construction is similar in the spirit of the ABE scheme by Goldwasser et al. [25].

Let $\Sigma := (\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$ be a secure signature scheme, and $\mathsf{WE} := (\mathsf{Enc}, \mathsf{Dec})$ denote a secure witness encryption scheme. Let $\mathsf{SNARK} := (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ be a secure SNARK scheme.

Setup. In the PKI setup, everyone obtains a key-pair $(\mathsf{vk}_i, \mathsf{sk}_i)$ of a secure signature scheme.

Offline. In the offline stage, P_1 sends a signature $\sigma_f := \Sigma.\mathsf{Sign}(\mathsf{sk}_1, f)$ to the server.

Online. The protocol for the online phase is as follows.

- Each party P_i for $i \in \{2, \ldots, n\}$, upon receiving input (ssid, x_i), sends to the server $(\mathsf{ssid}, x_i, \sigma_i)$, where $\sigma_i := \Sigma.\mathsf{Sign}(\mathsf{sk}_i, \mathsf{ssid}, x_i)$.
- Upon receiving input $(\mathsf{ssid}, m_0, m_1, x_1)$, the sender P_1 does the following:
  first he computes a signature $\sigma_1 := \Sigma.\mathsf{Sign}(\mathsf{sk}_1, \mathsf{ssid}, x_1)$ *SNARK language and setup.* P_1 runs $\mathsf{CRS} := \mathsf{SNARK.Gen}(1^\lambda)$, where SNARK is for the following language L_{SNARK}:

SNARK language L_{SNARK}:

A statement x is of the form $x := (\mathsf{ssid}, f, b)$.
A witness w is of the form $w := (x_1, x_2, \ldots, x_n, \sigma_f, \sigma_1, \ldots, \sigma_n)$.
Further, let R_{SNARK} denote the relation for the language L_{SNARK}. $R_{\text{SNARK}}(x, w) = 1$ if and only if: $\Sigma.\mathsf{Verify}(\mathsf{vk}_1, f, \sigma_f) = 1$, and for all $i \in [n]$, $\Sigma.\mathsf{Verify}(\mathsf{vk}_i, \mathsf{ssid}, x_i, \sigma_i)$, and $f(x_1, x_2, \ldots, x_n) = b$.

It then sends to the server witness encryption ciphertexts $(\mathsf{WE.Enc}_{s_0}(m_0), \mathsf{WE.Enc}_{s_1}(m_1))$, where m_0, m_1 are the inputs he received, and for $b \in \{0, 1\}$, $x_b := (\mathsf{ssid}, f, b, \mathsf{CRS})$ is a statement of the following WE language:

WE language L_{WE}:

A statement s is of the form $s := (\mathsf{ssid}, f, b, \mathsf{CRS})$.
A witness is of the form $w := \pi$.
$x \in L_{\text{WE}}$ iff there is a proof π such that $\mathsf{SNARK.Verify}(\mathsf{CRS}, (\mathsf{ssid}, f, b), \pi) = 1$.

- **Server action.** Based on CRS, f, ssid, $x_1, \ldots, x_n, \sigma_f, \sigma_1, \ldots, \sigma_n$, compute $b := f(x_1, \ldots, x_n)$. Computer $\pi := \mathsf{SNARK.Prove}(\mathsf{CRS}, (\mathsf{ssid}, f, b), (x_1, \ldots, x_n, \sigma_f, \sigma_1, \ldots, \sigma_n))$.

  Let c_0, c_1 be the two WE ciphertexts the server received from P_1. Compute and output $m_b := \mathsf{WE.Dec}_{s_b}(c_b, \pi)$.

Remarks. If succinctness is not required, the SNARK is not needed. The protocol can be used to build an $\mathsf{ah\text{-}mABE}$, which implies private MVC (in the malicious server or semi-honest client corruptions. Again by our compiler, we can reach security against malicious client corruptions). The protocol here has the advantage that the client's online efficiency is better in that their computational times depend on the f in a poly-logarithmic way. For standard assumptions, the best known construction of ABE's ciphertexts depend on the depth of the circuits being computed. Here if we are using stronger assumptions, we can achieve better efficiency.

We now sketch a proof of security. The simulator's strategy isas follows:

Case 1: A subset of senders are corrupted. The simulator construction is simple in this case.

Case 2: The server is corrupted. The simulator first learn from the ideal functionality the tuple $(x_1, \ldots, x_n, m_{f(x_1,\ldots,x_n)})$. For simplicity, assume $f(x_1, \ldots, x_n) = 0$. Then he simulate the view of by computing $(\mathsf{WE.Enc}_{s_0}(m_0), \mathsf{WE.Enc}_{s_1}(r))$, where r is a random message. Now, suppose there exists an environment who can distinguish, then we can use the extraction property of WE and SNARK to extract a valid witness, $(x'_1, \ldots, x'_n, \sigma'_f, \sigma'_1, \ldots, \sigma'_n))$. This must be different from $(x_1, \ldots, x_n, \sigma_f, \sigma_1, \ldots, \sigma_n))$, and thus gives a forgery of the signature scheme.

C Multi-Client Verifiable Computation without Input Privacy

In this section, we show how to construct a protocol for MVC without input privacy. We define the ideal functionality in Figure 5. For simplicity, we assume the function f being delegated outputs only one bit, and that only the first party receives the output. We remark that for general functions, we can do this bit-by-bit at the cost of a blowup up to the length of output. We present a construction in the $\mathcal{F}^f_{\mathsf{mABE}}$ hybrid model. More formally, let $f : (\{0,1\}^\ell)^n \to \{0,1\}$ be a function

Multi-Client Verifiable Computation (non-private)

The functionality is parameterized with an n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$. The functionality interacts with n clients P_i for $i \in [n]$, a distinguished server Serv, and the simulator $\mathcal{S}im$.

Initialization: Same as $\mathcal{F}_{\mathsf{pVC}}$.

Computation:
Upon receiving $(\mathsf{Input}, \mathsf{ssid}, x_i)$ from client P_i, send $(\mathsf{ssid}, \mathsf{P}_i)$ to notify $\mathcal{S}im$. Later, when $\mathcal{S}im$ returns $(\mathsf{ssid}, \mathsf{P}_i)$, store (ssid, x_i), and send a notification $(\mathsf{Input}, \mathsf{ssid}, \mathsf{P}_i, x_i)$ to the server Serv.
Upon receiving $(\mathsf{Input}, \mathsf{ssid}, 1)$ from the server Serv, retrieve (ssid, x_i) for $i \in [n]$. If some (ssid, x_i) has not been stored, send $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to the server and all clients.

- **Server is not corrupted.** Compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$, and send $(\mathsf{ssid}, (y_1, \ldots, y_n))$ to the server Serv and a notification to $\mathcal{S}im$. Later when the simulator $\mathcal{S}im$ returns $(\mathsf{ssid}, \mathsf{P}_i, \phi)$, if $\phi = \mathsf{ok}$, send $(\mathsf{Output}, \mathsf{ssid}, y_i)$ to client P_i; if $\phi = \mathsf{fail}$, send $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to client P_i.
- **Server is corrupted.** Let $\mathcal{I} \subseteq [n]$ denote the set of indices corresponding to corrupted clients. Let $\overline{\mathcal{I}} := [n] \setminus \mathcal{I}$. Without loss of generality, we can renumber the clients such that $\mathcal{I} := \{1, 2, \ldots, |\mathcal{I}|\}$.
  For each $i \in \overline{\mathcal{I}}$: the simulator sends $(\mathsf{ssid}, \mathsf{P}_i, \phi)$ to the ideal functionality where $\phi = \mathbf{x}^*_{\mathcal{I}}$ or $\phi = \mathsf{fail}$. If $\phi = \mathbf{x}^*_{\mathcal{I}}$, the functionality sends $(\mathsf{Output}, \mathsf{ssid}, f_i(\mathbf{x}^*_{\mathcal{I}}, \mathbf{x}_{\overline{\mathcal{I}}}))$ to P_i. Else, the functionality sends $(\mathsf{Output}, \mathsf{ssid}, \mathsf{fail})$ to P_i.

Figure 5: Functionality $\mathcal{F}_{\mathsf{VC}}$

to be delegated, let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ be the clients and Serv be the server. We consider an instance of the functionality $\mathcal{F}^f_{\mathsf{mABE}}$ where P_1 plays the sender, the server plays Serv, and all the clients are the senders. The parties act as follows:

- Upon receiving input (ssid, x_1), P_1 samples two random inputs $m_0, m_1 \leftarrow \{0,1\}^\ell$ and sends $(\mathsf{ssid}, m_0, m_1, x_1)$ to the functionality $\mathcal{F}^f_{\mathsf{mABE}}$. Locally, he stores m_0, m_1.
- For $i \in [n] \setminus \{1\}$, upon receiving message (ssid, x_i), P_i sends (ssid, x_i) to the functionality $\mathcal{F}^f_{\mathsf{mABE}}$.
- Upon receiving $(\mathsf{ssid}, x_1, \ldots, x_n, m)$ from $\mathcal{F}^f_{\mathsf{mABE}}$, the server sends (ssid, m) to P_1.
- Upon receiving (ssid, m) from the server, P_1 checks whether $m = m_b$ for some $b \in \{0,1\}$. If so, he outputs b, and otherwise he outputs $\perp$.

Theorem C.1 *The above protocol securely realizes $\mathcal{F}_{\mathsf{VC}}$ in the $\mathcal{F}^f_{\mathsf{mABE}}$ hybrid model against (1) malicious server corruption, or (2) semi-honest (static) corruption of any fixed subset of clients.*

Remark C.2 *Similar to Remark 3.5 the above theorem can be more general: we can show that the protocol is secure against any malicious corruption pattern in the $\mathcal{F}^f_{\mathsf{mABE}}$-hybrid world. We leave it as an interesting open question to construct VC protocols based on falsifiable assumptions. As we point out in Section B, any protocol that realizes $\mathcal{F}^f_{\mathsf{mABE}}$ against server-client collusions requires extractable witness encryption, so any construction of VC based on standard assumptions must avoid this route.*

Proof: We prove the stronger statement that the protocol securely realizes $\mathcal{F}_{\mathsf{VC}}$ in the $\mathcal{F}^f_{\mathsf{mABE}}$-hybrid model against any corruption pattern.

Let Π denote the $\mathcal{F}_{\mathsf{mABE}}$-hybrid protocol described above. To show the security of the protocol, we need to construct a simulator $\mathcal{S}im$ for any non-uniform PPT environment $\mathcal{Z}$ such that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{mABE}}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{VC}},\mathcal{S}im,\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary. We first construct a simulator and then argue the indistinguishability between the two ensembles.

Simulating the communication with $\mathcal{Z}$: Upon receiving an input value from $\mathcal{Z}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{A}$'s input tape (as if coming from $\mathcal{Z}$); upon obtaining an output value from $\mathcal{A}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{Z}$'s output tape (as if coming from $\mathcal{A}$). The simulator $\mathcal{S}im$ interacts with the (external) ideal functionality $\mathcal{F}_{\mathsf{VC}}$. In addition, $\mathcal{S}im$ internally emulates a copy of $\mathcal{F}_{\mathsf{mABE}}$ as well as honest players to interact with corrupted players (who are under control by $\mathcal{Z}$ thru the dummy $\mathcal{A}$).

Case 1: Simulating honest senders P_i, $i \in \{1,..,n\}$ with honest server Serv: $\mathcal{S}im$ must simulate the view of adversary $\mathcal{A}$ in the real world. Since we assume private and authenticated channels, $\mathcal{S}im$ can simply emulate all honest players by using dummy inputs (e.g., 0's for all honest parties). We omit the analysis.

Case 2: Simulating a corrupted subset of $\mathcal{I} \subset \{\mathsf{P}_1,\ldots,\mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1,\ldots,\mathsf{P}_n\}\backslash\mathcal{I}$ and honest server Serv: $\mathcal{S}im$ through the internally emulated copy of $\mathcal{F}_{\mathsf{mABE}}$, learns inputs of the parties in $\mathcal{I}$ from $\mathcal{Z}$, and stores them; then $\mathcal{S}im$ submit $\{x_i\}_{i\in\mathcal{I}}$ to the ideal functionality, $\mathcal{F}_{\mathsf{VC}}$ on the behalf of the parties in $\mathcal{I}$.

If P_1 is not corrupt, this concludes the simulation because none of the members of $\mathcal{I} \setminus \mathsf{P}_1$ ever receives any messages or output, so simulating their view is unnecessary.

If P_1 is corrupt, if $\mathcal{S}im$ receives $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ from the external ideal functionality $\mathcal{F}_{\mathsf{VC}}$, he simulates the internal copy of honest server to send fail to the corrupted P_1 (who is under control by$\mathcal{Z}$). Otherwise, $\mathcal{S}im$ receives a bit y from the ideal functionality $\mathcal{F}_{\mathsf{VC}}$ on the behalf of corrupted P_1. Now the simulator $\mathcal{S}im$ retrieves previously stored $(\mathsf{ssid}, m_0, m_1, x_1)$ which was submitted by $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{mABE}}$; then $\mathcal{S}im$ simulates the internal copy of honest server to send (ssid, m_y) to the corrupted P_1 (who is under control by$\mathcal{Z}$).

It is straightforward to verify that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{mABE}}}_{\Pi,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{\mathcal{F}_{\mathsf{VC}},\mathcal{S}im,\mathcal{Z}}$.

The simulation here is for malicious corrupted clients. We can easily obtain a simulator for semi-honestly corrupted clients: instead of extracting the inputs from corrupted clients through $\mathcal{F}_{\mathsf{mABE}}$, the semi-honest simulator is provided with such inputs.

Case 3: Simulating honest senders P_i for $i \in \{1,..,n\}$ with corrupted (malicious) server Serv: The simulator $\mathcal{S}im$ begins by querying the external ideal functionality $\mathcal{F}_{\mathsf{VC}}$ and receives $(\mathsf{ssid}, x_1, \ldots, x_n)$. He generates a random $m \leftarrow \{0,1\}^\ell$ and simulates the internal copy of $\mathcal{F}_{\mathsf{mABE}}$ to send $(\mathsf{ssid}, x_1, \ldots, x_n, m)$ to the corrupted server (who is under the control of $\mathcal{Z}$). Later, $\mathcal{S}im$ receives $(\mathsf{ssid}, \widetilde{m})$ from the corrupted server in response. If $\widetilde{m} \neq m$, the simulator sends $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ to the external ideal functionality $\mathcal{F}_{\mathsf{VC}}$ (indicating that P_1 should receive output fail). Otherwise, the simulator sends $(\mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{VC}}$. It is straightforward to verify that this is a perfect simulation, and we omit the analysis.

Case 4: Simulating a corrupted subset of $\mathcal{I} \subset \{\mathsf{P}_1,\ldots,\mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1,\ldots,\mathsf{P}_n\}\backslash\mathcal{I}$ with corrupted server Serv: The simulation here is a combination of Cases 2 and 3. Thru the internally emulated $\mathcal{F}_{\mathsf{mABE}}$, the simulator learns inputs of the parties in $\mathcal{I}$ from $\mathcal{Z}$, and

stores them; then $\mathcal{S}im$ submit $\{x_i\}_{i\in\mathcal{I}}$ to the ideal functionality, $\mathcal{F}_{\mathsf{VC}}$ on the behalf of the parties in $\mathcal{I}$. The simulator $\mathcal{S}im$ then by querying the external ideal functionality $\mathcal{F}_{\mathsf{VC}}$ receives $(\mathsf{ssid}, x_1, \ldots, x_n)$. (Note that if the environment $\mathcal{Z}$ sends different values for inputs of the parties in $\mathcal{I}$ to $\mathcal{F}_{\mathsf{mABE}}$, then the simulator will repeat the above again.)

If P_1 is not corrupted, the simulator generates a random $m \leftarrow \{0,1\}^\ell$ and simulates the internal copy of $\mathcal{F}_{\mathsf{mABE}}$ to send $(\mathsf{ssid}, x_1, \ldots, x_n, m)$ to the corrupted server (who is under the control of $\mathcal{Z}$); Later, $\mathcal{S}im$ receives $(\mathsf{ssid}, \widetilde{m})$ from the corrupted server in response; If $\widetilde{m} \neq m$, the simulator sends $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ to the external ideal functionality $\mathcal{F}_{\mathsf{VC}}$ (indicating that P_1 should receive output fail). Otherwise, the simulator sends $(\mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{VC}}$.

If P_1 is corrupted, the simulator $\mathcal{S}im$ retrieves previously stored $(\mathsf{ssid}, m_0, m_1, x_1)$ which was submitted by $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{mABE}}$; then $\mathcal{S}im$ simulates the internal copy of honest server to send $(\mathsf{ssid}, m_{f(x_1,x_2,\ldots,x_n)})$ to the corrupted P_1 (who is under control by $\mathcal{Z}$).

It is straightforward to verify that this is a perfect simulation. ■

D Deferred Proofs

D.1 Proof of Theorem 3.2 (mABE)

Proof: Let Π denote the mABE protocol described above. To show the security of the protocol, we need to construct a simulator $\mathcal{S}im$ for any non-uniform PPT environment $\mathcal{Z}$ such that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{POT}},\mathcal{G}^{\mathsf{ABE}}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{mABE}},\mathcal{S}im,\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary who is allowed either to corrupt Serv maliciously, or to corrupt some subset of the clients semi-honestly. We first construct a simulator and then argue the indistinguishability between the two ensembles.

Simulating the communication with $\mathcal{Z}$: Upon receiving an input value from $\mathcal{Z}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{A}$'s input tape (as if coming from $\mathcal{Z}$); upon obtaining an output value from $\mathcal{A}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{Z}$'s output tape (as if coming from $\mathcal{A}$).

Case 1: Simulating honest senders P_i, $i \in \{1,..,n\}$ with honest server Serv: $\mathcal{S}im$ must simulate the view of an eavesdropping adversary $\mathcal{A}$ in the hybrid world. Since we assume private channels, $\mathcal{S}im$ can simply encrypt strings of 0's to simulate the messages over-heard by the adversary. (See the discussion above.) We omit the analysis.

Case 2: Simulating a corrupted (semi-honest) subset of $\mathcal{I} \subset \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\} \setminus \mathcal{I}$ and honest server Serv: Each party in the corrupted set receives exactly one message during setup: P_1 receives $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}})$, and for $i \in \{2, \ldots, n\}$, P_i receives $\mathsf{mpk}_{\mathsf{ABE}}$. Note that none of these parties receive any messages during the online phase of the protocol. The description of $\mathcal{S}im$ is as follows:

- $\mathcal{S}im$ runs $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE}}$. If P_1 is corrupt, $\mathcal{S}im$ simulates his view using this pair. For $i \in \{2, \ldots, n\}$, if P_i is corrupt $\mathcal{S}im$ simulates his view using $\mathsf{mpk}_{\mathsf{ABE}}$. He outputs the simulated views to $\mathcal{Z}$.
- During the online phase, $\mathcal{S}im$ receives input for each corrupted party and forwards these inputs to the ideal functionality.

The view of each corrupted party in the offline phase is drawn from a distribution that is identical to the corresponding distribution in the hybrid world. Since the parties are semi-honest, they will always use the inputs they were given from $\mathcal{Z}$. It is easy to verify that security holds.

Case 3: Simulating honest senders P_i for $i \in \{1, .., n\}$ and corrupted (malicious) server Serv: Note that Serv has no input and never sends any messages in this protocol, so there is no difference between a semi-honest adversary and a fully malicious adversary. The simulator acts as follows:

- $\mathcal{S}im$ runs $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE}}$. He adds $\mathsf{mpk}_{\mathsf{ABE}}$ to the view of Serv.
- To simulate the message received from P_1 during the offline phase, $\mathcal{S}im$ computes $\mathsf{sk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, f)$ and adds this to the view of Serv.
- $\mathcal{S}im$ then requests output from $\mathcal{F}_{\mathsf{mABE}}$ and receives $(x_1, \ldots, x_n, m)$. He samples a random string R, and, for $i \in [n]$, $j \in [\ell]$, he computes $c_{i,j} = \mathsf{enc}(x_i[j]; R)$. Finally, he computes

$$c = \begin{cases} \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m, 0^\ell; R) & \text{if } f(x_1, \ldots, x_n) = 0. \\ \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, 0^\ell, m; R) & \text{if } f(x_1, \ldots, x_n) = 1. \end{cases}$$

  To complete the simulation,

  - He uses $c_{1,1}, \ldots, c_{1,\ell}$ and c to simulate the message received from P_1 during the online phase.
  - For $i \in \{2, \ldots, n\}$ and $j \in [\ell]$, he uses $c_{i,1}, \ldots, c_{i,\ell}$ to simulate the output from the (i, j)th instance of $\mathcal{F}_{\mathsf{POT}}$.

Note that all simulated messages are drawn from distributions that are identical to the corresponding real world distributions, with the exception of c. We claim that if there exists some environment $\mathcal{Z}^*$ such that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{POT}}, \mathcal{G}^{\mathsf{ABE}}}_{\Pi, \mathcal{A}, \mathcal{Z}^*} \not\approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{mABE}}, \mathcal{S}im, \mathcal{Z}^*}$, then there exists an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ that can break the security of the ABE scheme. Specifically, $\mathcal{B}$ simulates $\mathcal{Z}^*$ internally, by simulating all messages sent to $\mathcal{Z}^*$ by any of $\mathcal{S}im, \mathsf{P}_1, \ldots, \mathsf{P}_n$. $\mathcal{B}$ receives the inputs $(m_0, m_1, x_1), x_2, \ldots, x_n$ that $\mathcal{Z}^*$ puts on the input tapes of each of these players. Then,

- After receiving $\mathsf{mpk}_{\mathsf{ABE}}$ from his challenger, $\mathcal{B}$ forwards the value to $\mathcal{Z}^*$ on behalf of Serv, simulating the message sent during setup. He sends f to his own challenger.
- After receiving sk_f from his challenger, he sends this to $\mathcal{Z}^*$ on behalf of Serv, simulating the offline message received by Serv from P_1.
- If $f(x_1, \ldots, x_n) = 0$, $\mathcal{B}_2$ sends $(m_0, m_1, 0^\ell, x_1 || \cdots || x_n)$ to his challenger. Otherwise, he sends $(m_1, m_0, 0^\ell, x_1 || \cdots || x_n)$. Going forward, we will assume the former case: the proof proceeds in a parallel way when $f(x_1, \ldots, x_n) = 1$. He receives back a challenge $(c_0, c_1, \ldots, c_n)$, where

$$c_0 = \begin{cases} \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m_0, m_1; R) & \text{if challenge bit } \delta = 0. \\ \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, m_0; 0^\ell; R) & \text{if challenge bit } \delta = 1. \end{cases}$$

  for some randomly chosen R, and $c_i = \mathsf{enc}(\mathsf{mpk}_{\mathsf{ABE}}, x_i; R)$ (bit by bit) using the same random R. $\mathcal{B}$ sends these values to $\mathcal{Z}^*$ as a simulation of Serv's view in the online phase of the protocol. When $\mathcal{Z}^*$ outputs some bit b, $\mathcal{B}_3$ outputs the same value.

It is easy to verify that when $\delta = 0$ in Exp.ABE, $\mathcal{Z}^*$'s view in $\mathcal{B}$'s simulation and his view in $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{POT}},\mathcal{G}^{\mathsf{ABE}}}_{\Pi,\mathcal{A},\mathcal{Z}^*}$ are identically distributed, while when $\delta = 1$, $\mathcal{Z}^*$'s view in $\mathcal{B}$'s simulation and his view in $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{mABE}},\mathcal{S}im,\mathcal{Z}^*}$ are identically distributed. We conclude that $\mathcal{B}$'s advantage in Exp.ABE is the same as the advantage of $\mathcal{Z}^*$ in distinguishing $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{POT}},\mathcal{G}^{\mathsf{ABE}}}_{\Pi,\mathcal{A},\mathcal{Z}^*}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{mABE}},\mathcal{S}im,\mathcal{Z}^*}$. ■

D.2 Proof of Theorem 3.3 (ah-mABE)

Proof: Let Π denote the ah-mABE protocol described above. To show the security of the protocol, we need to construct a simulator $\mathcal{S}im$ for any non-uniform PPT environment $\mathcal{Z}$ such that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{mABE}},\mathcal{G}^{\mathsf{FHE}}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{ah\text{-}mABE}},\mathcal{S}im,\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary who is allowed either to maliciously corrupt the server, or to semi-honestly corrupt any subset of the senders. We first construct a simulator and then argue the indistinguishability between the two ensembles.

Simulating the communication with $\mathcal{Z}$: Upon receiving an input value from $\mathcal{Z}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{A}$'s input tape (as if coming from $\mathcal{Z}$); upon obtaining an output value from $\mathcal{A}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{Z}$'s output tape (as if coming from $\mathcal{A}$).

Case 1: Simulating honest senders $\mathsf{P}_1,\ldots,\mathsf{P}_n$ with honest server Serv: $\mathcal{S}im$ must simulate the view of an eavesdropping adversary $\mathcal{A}$ in the hybrid world. Since we assume private channels, $\mathcal{S}im$ can simply encrypt strings of 0's to simulate the messages over-heard by the adversary. (See the discussion above.) We omit the analysis.

Case 2: Simulating a corrupted (semi-honest) subset of $\mathcal{I} \subset \{\mathsf{P}_1,\ldots,\mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1,\ldots,\mathsf{P}_n\} \setminus \mathcal{I}$ and honest server Serv: Each party in the corrupted set receives exactly one message during setup: P_1 receives $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$, and for $i \in \{2,\ldots,n\}$, P_i receives $\mathsf{pk}_{\mathsf{FHE}}$. Note that none of these parties receive any messages during the online phase of the protocol. The description of $\mathcal{S}im$ is as follows:

- If P_1 is corrupt, $\mathcal{S}im$ simulates his view using this key pair he generated $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$. For $i \in \{2,\ldots,n\}$, if P_i is corrupt $\mathcal{S}im$ simulates his view using $\mathsf{pk}_{\mathsf{FHE}}$. He outputs the simulated views to $\mathcal{Z}$.
- During the online phase, $\mathcal{S}im$ receives input for each corrupted party and forwards these inputs to the ideal functionality.

The view of each corrupted party in the offline phase is drawn from a distribution that is identical to the corresponding distribution in the hybrid world. Since the parties are semi-honest, they will always use the inputs they were given from $\mathcal{Z}$. It is easy to verify that security holds.

Case 3: Simulating honest senders $\mathsf{P}_1,\ldots,\mathsf{P}_n$ with corrupted (malicious) server Serv: Note that Serv has no input and never sends any messages in this protocol, so there is no difference between a semi-honest adversary and a fully malicious adversary. The simulator acts as follows:

- He adds $\mathsf{pk}_{\mathsf{FHE}}$ to the view of Serv.
- He learns the output m from the ideal functionality $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$. We remind the reader that the honest senders give input $((m_0, m_1, x_1), x_2, \ldots, x_n)$ to the functionality, who will output $m_{f(x_1,\ldots,x_n)}$.

- He then samples $\hat{x}'_1, \ldots, \hat{x}'_n \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, 0)$.
- Then he generates a simulated garble circuit $\tilde{\Gamma}$ with simulated labels $\{\tilde{L}_i\}_{i\in[\lambda]}$ such that $\mathsf{Gb.Eval}(\tilde{\Gamma}, \{\tilde{L}_i\}_{i\in[\lambda]}) = m$. $\mathcal{S}im$ adds $\tilde{\Gamma}$ to Serv's view.
- $\mathcal{S}im$ computes $(d_1, \ldots, d_\lambda) = \mathsf{Eval}_{\mathsf{FHE}}(\mathsf{pk}_{\mathsf{FHE}}, f, \hat{x}'_1, \ldots, \hat{x}'_n)$, where d_i is the i-th bit of the evaluation outcome.
- For $j \in [\lambda]$, $\mathcal{S}im$ sets $L_j^{d_j} = \tilde{L}_j$ and $L_j^{1-d_j} = 0$. The he simulates the ideal functionalities $\mathcal{F}_{\mathsf{mABE}}^{g_j}$ with P_1's input $(L_0^j, L_1^j, \hat{x}'_1)$, and $\hat{x}'_i$ for the other P_i, for $i \in [n] \setminus \{1\}$. At the end, the server learns $\{\tilde{L}_j\}_{j\in[\lambda]}$, and $\hat{x}'_1 \ldots, \hat{x}'_n$. This completes the simulation.

Next we are going to show that the simulation is correct. We denote the view in the real (hybrid) world as $\mathcal{Z}$ by $H := (\mathsf{pk}_{\mathsf{FHE}}, \Gamma, \{L^{d_j}\}_{j\in[\lambda]}, \hat{x}_1, \ldots, \hat{x}_n)$, and the view in the ideal world as $H' := (\mathsf{pk}'_{\mathsf{FHE}}, \tilde{\Gamma}, \{\tilde{L}_j\}_{j\in[\lambda]}, \hat{x}'_1, \ldots, \hat{x}'_n)$. Note that $\mathsf{pk}_{\mathsf{FHE}}$ and $\mathsf{pk}'_{\mathsf{FHE}}$ are identically distributed and will not influence the joint distribution. To see that for all PPT $\mathcal{Z}$, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{mABE}}, \mathcal{G}^{\mathsf{FHE}}}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{ah\text{-}mABE}}, \mathcal{S}im, \mathcal{Z}}$, it suffices to consider just a single hybrid game, where $\mathcal{Z}$ is given a simulated garbled circuit, with encryptions of real input values, and thus his view becomes $H_1 := (\mathsf{pk}_{\mathsf{FHE}}, \tilde{\Gamma}, \{\tilde{L}^{d_j}\}_{j\in[\lambda]}, \hat{x}_1, \ldots, \hat{x}_n)$. Specifically, suppose there exists an environment $\mathcal{Z}^*$ that is capable of distinguishing H from H_1. Then there exists a reduction that can break the security of the garbled circuit scheme. The reduction simulates the environment $\mathcal{Z}^*$ internally. When $\mathcal{Z}^*$ specifies inputs $x_1, \ldots, x_n$ and the parties compute $\hat{x}_1, \ldots, \hat{x}_n$ using $\mathsf{pk}'_{\mathsf{FHE}}$. The reduction submits these, along with the circuit for $\mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_{\mathsf{FHE}}, \cdot)$, to his challenger, and receives a challenge $(\tilde{\Gamma}, \{L_j\}_{j\in[\lambda]})$. He uses these to construct the message for $\mathcal{Z}^*$, and outputs whatever $\mathcal{Z}^*$ outputs. The advantage is clear in this case.

To complete the proof, we argue that if there exists an environment $\mathcal{Z}^*$ that can distinguish H_1 from the ideal world H', then there exists a reduction that can break the FHE. We observe that the only difference between H_1 and H' is the ciphertexts. Thus, the reduction can embed the challenge ciphertexts to the experiment and simulate the rest. Then by calling the Enc^*, the reduction can tell whether the ciphertexts were encryptions of 0's or $(x_1, \ldots, x_n)$. ■

D.3 Proof of Theorem 3.4 (Private MVC)

The proof of Theorem 3.4 is very similar to that of Theorem C.1. For completeness, we present the details below.

Proof: We here prove the stronger statement that the described protocol securely realizes $\mathcal{F}_{\mathsf{pVC}}$ in the $\mathcal{F}_{\mathsf{ah\text{-}mABE}}^{f}$-hybrid model against any corruption pattern.

Let Π denote the $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$-hybrid protocol described above. To show the security of the protocol, we need to construct a simulator $\mathcal{S}im$ for any non-uniform PPT environment $\mathcal{Z}$ such that $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{ah\text{-}mABE}}}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{pVC}}, \mathcal{S}im, \mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary. We first construct a simulator and then argue the indistinguishability between the two ensembles.

Simulating the communication with $\mathcal{Z}$: Upon receiving an input value from $\mathcal{Z}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{A}$'s input tape (as if coming from $\mathcal{Z}$); upon obtaining an output value from $\mathcal{A}$, the simulator $\mathcal{S}im$ writes it on $\mathcal{Z}$'s output tape (as if coming from $\mathcal{A}$). The simulator $\mathcal{S}im$ interacts with the (external) ideal functionality $\mathcal{F}_{\mathsf{pVC}}$. In addition, $\mathcal{S}im$ internally emulates a

copy of $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$ as well as honest players to interact with corrupted players (who are under control by $\mathcal{Z}$ thru the dummy $\mathcal{A}$).

Case 1: Simulating honest senders P_i, $i \in \{1,..,n\}$ with honest server Serv: $\mathcal{S}im$ must simulate the view of adversary $\mathcal{A}$ in the real world. Since we assume private and authenticated channels, $\mathcal{S}im$ can simply emulate all honest players by using dummy inputs (e.g., 0's for all honest parties). We omit the analysis.

Case 2: Simulating a corrupted subset of $\mathcal{I} \subset \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\} \setminus \mathcal{I}$ and honest server Serv: $\mathcal{S}im$ through the internally emulated copy of $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, learns inputs of the parties in $\mathcal{I}$ from $\mathcal{Z}$, and stores them; then $\mathcal{S}im$ submit $\{x_i\}_{i\in\mathcal{I}}$ to the ideal functionality, $\mathcal{F}_{\mathsf{pVC}}$ on the behalf of the parties in $\mathcal{I}$.

If P_1 is not corrupt, this concludes the simulation because none of the members of $\mathcal{I} \setminus \mathsf{P}_1$ ever receives any messages or output, so simulating their view is unnecessary.

If P_1 is corrupt, if $\mathcal{S}im$ receives $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ from the external ideal functionality $\mathcal{F}_{\mathsf{pVC}}$, he simulates the internal copy of honest server to send fail to the corrupted P_1 (who is under control by$\mathcal{Z}$). Otherwise, $\mathcal{S}im$ receives a bit y from the ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ on the behalf of corrupted P_1. Now the simulator $\mathcal{S}im$ retrieves previously stored $(\mathsf{ssid}, m_0, m_1, x_1)$ which was submitted by $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$; then $\mathcal{S}im$ simulates the internal copy of honest server to send (ssid, m_y) to the corrupted P_1 (who is under control by$\mathcal{Z}$).

It is straightforward to verify that the joint distributions are the same: $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{ah\text{-}mABE}}}_{\Pi,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{\mathcal{F}_{\mathsf{pVC}},\mathcal{S}im,\mathcal{Z}}$.

The simulation here is for maliciously corrupted clients. We can easily obtain a simulator for semi-honestly corrupted clients: instead of extracting the inputs from corrupted clients through $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, the semi-honest simulator is provided with such inputs.

Case 3: Simulating honest senders P_i for $i \in \{1,..,n\}$ with corrupted (malicious) server Serv: The simulator $\mathcal{S}im$ begins by querying the external ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ and receives $(\mathsf{ssid}, x_1, \ldots, x_n)$. He generates a random $m \leftarrow \{0,1\}^\ell$ and simulates the internal copy of $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$ to send $(\mathsf{ssid}, x_1, \ldots, x_n, m)$ to the corrupted server (who is under the control of $\mathcal{Z}$). Later, $\mathcal{S}im$ receives $(\mathsf{ssid}, \widetilde{m})$ from the corrupted server in response. If $\widetilde{m} \neq m$, the simulator sends $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ to the external ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ (indicating that P_1 should receive output fail). Otherwise, the simulator sends $(\mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{pVC}}$. It is straightforward to verify that this is a perfect simulation, and we omit the analysis.

Case 4: Simulating a corrupted subset of $\mathcal{I} \subset \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, honest senders $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\} \setminus \mathcal{I}$ with corrupted server Serv: The simulation here is a combination of Cases 2 and 3. Thru the internally emulated $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, the simulator learns inputs of the parties in $\mathcal{I}$ from $\mathcal{Z}$, and stores them; then $\mathcal{S}im$ submit $\{x_i\}_{i\in\mathcal{I}}$ to the ideal functionality, $\mathcal{F}_{\mathsf{pVC}}$ on the behalf of the parties in $\mathcal{I}$. The simulator $\mathcal{S}im$ then by querying the external ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ receives $(\mathsf{ssid}, x_1, \ldots, x_n)$. (Note that if the environment $\mathcal{Z}$ sends different values for inputs of the parties in $\mathcal{I}$ to $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$, then the simulator will repeat the above again.)

If P_1 is not corrupted, the simulator generates a random $m \leftarrow \{0,1\}^\ell$ and simulates the internal copy of $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$ to send $(\mathsf{ssid}, x_1, \ldots, x_n, m)$ to the corrupted server (who is under the control of $\mathcal{Z}$); Later, $\mathcal{S}im$ receives $(\mathsf{ssid}, \widetilde{m})$ from the corrupted server in response; If

$\widetilde{m} \neq m$, the simulator sends $(\mathsf{ssid}, \mathsf{P}_1, \mathsf{fail})$ to the external ideal functionality $\mathcal{F}_{\mathsf{pVC}}$ (indicating that P_1 should receive output fail). Otherwise, the simulator sends $(\mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{pVC}}$.

If P_1 is corrupted, the simulator $\mathcal{S}im$ retrieves previously stored $(\mathsf{ssid}, m_0, m_1, x_1)$ which was submitted by $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{ah\text{-}mABE}}$; then $\mathcal{S}im$ simulates the internal copy of honest server to send $(\mathsf{ssid}, m_{f(x_1,x_2,\ldots,x_n)})$ to the corrupted P_1 (who is under control by $\mathcal{Z}$).

It is straightforward to verify that this is a perfect simulation. ■

D.4 Proof of Theorem 4.2

Proof: The construction of Π' is simple. Assume that in the offline stage, client P_i obtains the secret and public information $(\mathsf{pub}, \mathsf{sk}_i)$ and sends some message $\alpha_i = \Pi_i^{\mathsf{offline}}(\mathsf{pub}, \mathsf{sk}_i, f; R_i)$ to the serve, where Π_i is the prescribed strategy. At any time in the online stage, P_i sends some message $M_i = \Pi^{\mathsf{online}}(\mathsf{pub}, \mathsf{sk}_i, x_i; R_i')$ to the server where $M_i()$ is the prescribed strategy.

In the protocol Π', the clients now instead of sending their messages to the server directly, they send the message via the ideal functionality $\mathcal{F}_{\mathsf{ZK}}$ to show consistency. That is, in the offline stage, P_i provides both α_i as the statement, and (sk_i, f, R_i) as the witness to $\mathcal{F}_{\mathsf{ZK}}$; then $\mathcal{F}_{\mathsf{ZK}}$ returns $(\alpha_i, 1)$ to the server if the witness is valid, and $(\alpha_i, 0)$ otherwise. Similarly in the online stage, P_i provides both M_i as the statement, and $(\mathsf{sk}_i, x_i, R_i')$ as the witness to $\mathcal{F}_{\mathsf{ZK}}$; then $\mathcal{F}_{\mathsf{ZK}}$ returns $(M_i, 1)$ to the server if the witness is valid, and $(M_i, 0)$ otherwise.

Security follows in a straightforward way. The simulator, by simulating $\mathcal{F}_{\mathsf{ZK}}$, first extracts the inputs and randomness of the corrupted clients. Then he just runs the simulator as defined in Definition 4.1, which by definition generates an identical view of the corrupted parties. ■