Function-Dependent Commitments for Verifiable Multi-Party Computation*

Lucas Schabhüser, Denis Butin, Denise Demirel, and Johannes Buchmann

Technische Universität Darmstadt, Germany
lschabhueser@cdc.informatik.tu-darmstadt.de

Abstract In cloud computing, delegated computing raises the security issue of guaranteeing data authenticity during a remote computation. Existing solutions do not simultaneously provide fast correctness verification, strong security properties, and information-theoretic confidentiality. We introduce a novel approach, in the form of function-dependent commitments, that combines these strengths. We also provide an instantiation of function-dependent commitments for linear functions that is unconditionally, i.e. information-theoretically, hiding and relies on standard hardness assumptions. This powerful construction can for instance be used to build verifiable computing schemes providing information-theoretic confidentiality. As an example, we introduce a verifiable multi-party computation scheme for shared data providing public verifiability and unconditional privacy towards the servers and parties verifying the correctness of the result. Our scheme can be used to perform verifiable computations on secret shares while requiring only a single party to compute the audit data for verification. Furthermore, our verification procedure is asymptotically even more efficient than performing operations locally on the shared data. Thus, our solution improves the state of the art for authenticated computing, verifiable computing and multi-party computation.

Keywords: Commitments · Homomorphic Cryptography · MPC

1 Introduction

Today, it is common practice to outsource time-consuming computations to the cloud. Such infrastructures attractively offer cost savings and dynamic computing resource allocation. In such a situation, it is desirable to be able to verify the outsourced computation. The verification must be *efficient*, by which we mean that the verification procedure is significantly faster than verified computation itself. Otherwise, the verifier could as well carry out the computation by himself, negating the advantage of outsourcing.

Often, not only the data owner is interested in the correctness of a computation; but also third parties, like insurance companies in the case of medical data. For such third party verifiers, another desired property for verification procedures is

* This article is based on an earlier article which will appear in the proceedings of ISC2018.

proof of origin: evidence linking the result of the outsourced computation to their input. This additional guarantee is required because proofs of correct computation usually do not explicitly include input values. It is especially important if the verifier of the correctness proof is a third party who does not trust the cloud provider to provide the correct input to the computation. Together, these two pieces of evidence guarantee that the output received by the provider was indeed correctly computed from the initially provided input.

In addition, there are scenarios in which computations are performed over sensitive data. For instance, a cloud server may collect health data of individuals and compute their averages. So the challenge arises to design efficient verification procedures for outsourced computing that are privacy-preserving. Growing amounts of data are sensitive enough to require *long-term* protection. Electronic health records, voting records, or tax data require protection periods exceeding the lifetime of an individual. Over such a long time, complexity-based confidentiality protection is unsuitable because algorithmic progress is unpredictable. In contrast, *information-theoretic* confidentiality protection is not threatened by algorithmic progress and supports long-term security.

Two categories of solutions simultaneously address verifiability, proof of origin, and confidentiality:

- Homomorphic authenticators [1], which sometimes allow for efficient verification, keeping the computational effort of the verifier low. They do, however, not provide information-theoretic privacy, i.e. they are not long-term secure. Schemes like the one presented in [11] offer context hiding security, i.e. authenticators to the output of a computation do not leak information about the input. In this work, we consider a privacy notion which is even stronger than context hiding. In our case, no information is leaked; in particular, not even about the output.
- Homomorphic commitments [6, 25, 27] can be used for auditing. Authenticity is typically achieved by using a secure bulletin board [16]. In particular, Pedersen commitments [25] provide long-term security: they achieve information-theoretic privacy for the input values to arbitrary linear functions. Homomorphic commitments however, feature computationally costly correctness verification.

For a more detailed comparison to related work, see Sec. 5.

1.1 Contributions

In this paper, we solve the problem of providing efficient verification and proof of origin with information-theoretic privacy for linear functions. To achieve this, we introduce a novel generic construction that combines information-theoretic privacy with strong unforgeability and fast verification. We call this construction *function-dependent commitments* (FDCs). In addition to this main contribution, we provide a concrete, unconditionally hiding instantiation of FDCs for linear functions using pairings, demonstrating that our generic construction can be realized in the

standard model. In terms of hardness assumptions, only a variant of the Diffie–Hellman problem [11] is required. Our instantiation achieves succinctness and efficient verification. Finally, we showcase a verifiable multi-party computation scheme based on the concrete instantiation. This scheme makes it possible to verify whether the reconstructed result has been computed correctly by computing additional audit data on a *single* storage server. Previous proposals require *all* storage servers to perform computations to check correctness. Our scheme provides unconditional input-output privacy towards the servers and parties verifying computational correctness.

1.2 Outline

The remainder of this paper is organized as follows. We first introduce our framework for FDC schemes (Sec. 2). We then present a concrete instantiation of an FDC using pairings, and prove its properties (Sec. 3). A sketch of how this instantiation can be used to build a verifiable computing scheme for shared data is presented next (Sec. 4). Finally, we compare our contribution with related work (Sec. 5) and conclude (Sec. 6).

2 Function-Dependent Commitments

In this section, we present our novel FDC scheme and define its relevant properties. We define the classical properties of commitments, binding and hiding, in the context of FDCs. Furthermore we provide definitions for evaluation correctness and unforgeability. In terms of performance properties, we consider succinctness and amortized efficiency.

Like in the case of homomorphic commitments or authenticators, a function-dependent commitment can be used to derive new commitments by its homomorphic properties. It is necessary that the homomorphic property cannot be abused to create forgeries. In the context of homomorphic authenticators, the notions of labeled and multi-labeled programs (see e.g. [4]) are introduced to provide meaningful security definitions.

Evaluating a function can be modeled as performing a program on a set of labeled inputs that belong to a given dataset. On a high level, a message is uniquely identified by two identifiers: one input identifier τ, and one dataset identifier Δ. One can think of a dataset as an array of message, and of the input identifiers as pointers to specific positions within this array.

This enables a precise description of homomorphic properties. For authenticators, it is usually required that only authenticators created under the same dataset identifier are used for homomorphic evaluation. We now formally describe labeled and multi-labeled programs, in the vein of Backes et al [4].

A *labeled program* $\mathcal{P}$ consists of a tuple $(\mathsf{f}, \tau_1, \ldots, \tau_k)$, where $\mathsf{f} : \mathcal{M}^k \to \mathcal{M}$ is a function with k inputs and $\tau_i \in \chi$ is a label for the i-th input of f from some set χ. Given a set of labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, they can be composed by evaluating g over the labeled programs, i.e. $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$.

The identity program with label τ is given by $\mathcal{I}_\tau = (\mathsf{f}_{id}, \tau)$, where $\mathsf{f}_{id} : \mathcal{M} \to \mathcal{M}$ is the identity function. Program $\mathcal{P} = (\mathsf{f}, \tau_1, \ldots, \tau_k)$ can be expressed as the composition of k identity programs $\mathcal{P} = \mathsf{f}(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_k})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ of the labeled program $\mathcal{P}$ and a dataset identifier Δ. Given a set of t multi-labeled programs with the same data set identifier Δ, i.e. $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$, and a function $g : \mathcal{M}^t \to \mathcal{M}$, a composed multi-label program $\mathcal{P}^*_\Delta$ can be computed, consisting of the pair $(\mathcal{P}^*, \Delta)$, where $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. Analogously to the identity program for labeled programs, we refer to a multi-labeled identity program by $\mathcal{I}_{(\Delta,\tau)} = ((\mathsf{f}_{id}, \tau), \Delta)$.

Using the formalism of multi-labeled programs, we now define FDCs.

Definition 1. *A FDC scheme is a tuple of algorithms (*Setup*,* KeyGen*,* PublicCommit*,* PrivateCommit*,* FunctionCommit*,* Eval*,* FunctionVerify*,* PublicDecommit*):*

- $\mathsf{Setup}(1^\lambda)$ *takes the security parameter* λ *and outputs public parameters* pp*. We implicitly assume that every algorithm uses these public parameters, leaving them out of the notation.*
- $\mathsf{KeyGen}(1^\lambda)$ *takes the security parameter* λ *as input and outputs a secret-public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*
- $\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r})$ *takes as input a message* m *and randomness* r *and outputs commitment* C*.*
- $\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}, \mathsf{r}, \Delta, \tau)$ *takes as input the secret key* sk*, a message* m*, randomness* r*, a dataset* Δ*, and an identifier* τ *and outputs an authenticator* A *for the tuple* $(\mathsf{m}, \mathsf{r}, \Delta, \tau)$*.*
- $\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$ *takes as input the public key* pk *and a labeled program* $\mathcal{P}$ *and outputs a function commitment* F *to* $\mathcal{P}$*.*
- $\mathsf{Eval}(\mathcal{P}_\Delta, \mathsf{A}_1, \ldots, \mathsf{A}_n)$ *takes as input a multi-labeled program* $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ *and a set of authenticators* $\mathsf{A}_1, \ldots, \mathsf{A}_n$*, where* A_i *is an authenticator for* $(\mathsf{m}_i, \mathsf{r}_i, \Delta, \tau_i)$*, for* $i = 1, \ldots, n$*. It computes an authenticator* A^* *to the tuple* $(\mathsf{f}(\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{f}(\mathsf{r}_1, \ldots, \mathsf{r}_n), \Delta, (\tau_1, \ldots, \tau_n))$ *using* $\mathsf{A}_1, \ldots, \mathsf{A}_n$ *and outputs* A^**.*
- $\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}, \mathsf{C}, \mathsf{F})$ *takes as input a public key* pk*, a FDC containing an authenticator* A *and a commitment* C*, as well as a function commitment* F*. It outputs either 1 (accept) or 0 (reject).*
- $\mathsf{PublicDecommit}(\mathsf{m}, \mathsf{r}, \mathsf{C})$ *takes as input message* m*, randomness* r*, and commitment* C*. It outputs either 1 (accept) or 0 (reject).*

FunctionVerify only verifies whether the pair (C, A) is a correct FDC to F while PublicDecommit allows to check that C opens to a specific pair of opening values (m, r).

2.1 Properties of Function-Dependent Commitments

As for classical commitments we want our schemes to be *binding* (see e.g. [30]). That is, after committing to a message, it should be infeasible to open the commitment to a different message. For a formal definition we refer to Appendix B.

Another important notion, targeting privacy, is the hiding property. Commitments are intended to not leak information about the messages they contain. This is not to be confused with the context hiding property, where homomorphic authenticators to the output of a computation do not leak information about the inputs to the computation. They do however leak information about the output.

Definition 2 (Hiding). *A FDC is called* computationally hiding *if the sets of commitments* $\{\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}) \mid \mathsf{r} \xleftarrow{\$} \mathcal{R}\}$ *and* $\{\mathsf{PublicCommit}(\mathsf{m}', \mathsf{r}') \mid \mathsf{r}' \xleftarrow{\$} \mathcal{R}\}$ *as well as* $\{\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}, \mathsf{r}, \Delta, \tau) \mid \mathsf{r} \xleftarrow{\$} \mathcal{R}\}$ *and* $\{\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}', \mathsf{r}', \Delta, \tau) \mid \mathsf{r}' \xleftarrow{\$} \mathcal{R}\}$ *have distributions that are indistinguishable for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *for all* $\mathsf{m} \neq \mathsf{m}' \in \mathcal{M}$.
A FDC is called unconditionally hiding *if these sets have the* same *distribution respectively for all* $\mathsf{m} \neq \mathsf{m}' \in \mathcal{M}$.

An obvious requirement for an FDC is to be *correct*, i.e. if messages are authenticated properly and evaluation is performed honestly, the resulting commitment should be verified. This is formalized in the following definition.

Definition 3 (Evaluation Correctness). *A FDC achieves* evaluation correctness *if for any set of messages,* $\mathsf{m}_1, \ldots, \mathsf{m}_n \subset \mathcal{M}$*, any set of randomness* $\mathsf{r}_1, \ldots, \mathsf{r}_n \subset \mathcal{R}$*, any set of identifiers* $\tau_1, \ldots, \tau_n \subset \chi$*, and any multi-labeled program* $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ *we have* $\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}, \mathsf{C}, \mathsf{F}) = 1$*, where* $\mathsf{A}_i = \mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}_i, \mathsf{r}_i, \Delta, \tau_i)$ *for* $i \in [n]$*,* $\mathsf{A} = \mathsf{Eval}(\mathcal{P}_\Delta, \mathsf{A}_1, \ldots, \mathsf{A}_n)$*,* $\mathsf{C} = \mathsf{PublicCommit}(\mathsf{f}(\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{f}(\mathsf{r}_1, \ldots, \mathsf{r}_n))$*, and* $\mathsf{F} = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$.

For the security notion of FDCs, we first provide a definition for *well defined programs* and *forgeries* on these programs. Then, we introduce an experiment the attacker can run in order to generate a successful forgery and present a definition for unforgeability based on this experiment.

Definition 4 (Well Defined Program). *A labeled program* $\mathcal{P} = (\mathsf{f}, \tau_1, \ldots, \tau_n)$ *is well defined with respect to a list* L_Δ *if one of the two following cases holds:*

1. *There are messages* $\mathsf{m}_1, \ldots, \mathsf{m}_n$ *such that* $(\tau_i, \mathsf{m}_i) \in L_\Delta \ \forall\ i = 1, \ldots, n$.
2. *There is an* $i \in \{1, \ldots, n\}$ *such that* $(\tau_i, \cdot) \notin L_\Delta$ *and* $\mathsf{f}(\{\mathsf{m}_j\}_{(\tau_j, \mathsf{m}_j) \in L_\Delta} \cup \{\tilde{\mathsf{m}}_k\}_{(\tau_k, \cdot) \notin L_\Delta})$ *does not depend on the choice of* $\tilde{\mathsf{m}}_k \in \mathcal{M}$.

Definition 5 (Forgery). *A forgery is a tuple* $(\mathcal{P}_\Delta, \mathsf{C}, \mathsf{A})$ *such that*

$$\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}, \mathsf{C}, \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_\Delta)) = 1$$

holds and one of the following conditions is met:

Type 1: *The list* L_Δ *was not initialized during the game, i.e. no message was ever committed under the data set identifier* Δ.
Type 2: $\mathcal{P}_\Delta$ *is well defined with respect to list* L_Δ *and* $\mathsf{C} \neq \mathsf{PublicCommit}(\mathsf{f}(\{\mathsf{m}_j\}_{(\tau_j, \mathsf{m}_j) \in L_\Delta}), \mathsf{f}(\{\mathsf{r}_j\}_{(\tau_j, \mathsf{r}_j) \in L_\Delta}))$*, i.e.* C *is not the correct commitment to the output of the computation.*

Type 3: *$\mathcal{P}_\Delta$ is not well defined with respect to L_Δ.*

This definition of forgeries is consistent with existing definitions, e.g. [11]. It is an immediate corollary of [18, Theorem 5.1] that if $\mathcal{P}$ contains a linear function, then any adversary who outputs a Type 3 forgery can be converted into one that outputs a Type 2 forgery. To define unforgeability, we first describe the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{Com}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

$\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{Com}}(\lambda)$:

Setup $\mathcal{C}$ calls $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and gives pp to $\mathcal{A}$.

Key Generation $\mathcal{C}$ calls $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ and gives pk to $\mathcal{A}$.

Queries $\mathcal{A}$ adaptively submits queries for $(\Delta, \tau, \mathsf{m}, \mathsf{r})$ where Δ is a dataset, τ is an identifier, m is a message, and r is a random value. $\mathcal{C}$ proceeds as follows:

- If $(\Delta, \tau, \mathsf{m}, \mathsf{r})$ is the first query with dataset identifier Δ, it initializes an empty list $L_\Delta = \emptyset$ for Δ.
- If L_Δ does not contain a tuple $(\tau, \cdot, \cdot)$, i.e. $\mathcal{A}$ never queried $(\Delta, \tau, \cdot, \cdot)$, $\mathcal{C}$ calls $\mathsf{A} \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}, \mathsf{r}, \Delta, \tau)$, updates the list $L_\Delta = L_\Delta \cup (\tau, \mathsf{m}, \mathsf{r})$, and gives A to $\mathcal{A}$.
- If $(\tau, \mathsf{m}, \mathsf{r}) \in L_\Delta$, then $\mathcal{C}$ returns the same authenticator A as before.
- If L_Δ already contains a tuple $(\tau, \mathsf{m}', \mathsf{r}')$ for $(\mathsf{m}, \mathsf{r}) \neq (\mathsf{m}', \mathsf{r}')$, $\mathcal{C}$ returns $\perp$.

Forgery $\mathcal{A}$ outputs a tuple $(\mathcal{P}_\Delta, \mathsf{m}, \mathsf{r}, \mathsf{A})$.

$\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{Com}}(\lambda)$ outputs 1 if the tuple returned by $\mathcal{A}$ is a forgery as defined before in Def. 5.

Definition 6 (Unforgeability). *A FDC is unforgeable if for any PPT adversary $\mathcal{A}$ we have*

$$\Pr[\boldsymbol{EXP}^{UF-CMA}_{\mathcal{A},Com}(\lambda) = 1] = \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ denotes any function negligible in the security parameter λ.

Regarding performance, we consider additional properties. *Succinctness* specifies a limit on the size of the FDCs, thus keeping the required bandwidth low, when using FDCs to verify the correctness of an outsourced computation.

Definition 7 (Succinctness). *A FDC is* succinct *if, for a fixed security parameter λ, the size of the authenticators depends at most logarithmically on the dataset size n.*

Amortized efficiency specifies a bound on the computational effort required to perform verifications.

Definition 8 (Amortized Efficiency). *Let $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ be a multi-labeled program, $\mathsf{m}_1, \ldots, \mathsf{m}_n \subset \mathcal{M}$ a set of messages, $\mathsf{r}_1, \ldots, \mathsf{r}_n \subset \mathcal{R}$ a set of randomness, and $t(n)$ be the time required to compute $\mathsf{f}(\mathsf{m}_1, \ldots, \mathsf{m}_n)$. A FDC achieves* amortized efficiency *if for given authenticator A and function commitment F the time required to compute $\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}, \mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}), \mathsf{F})$ is $t' = o(t(n))$.*

Analogous definitions for amortized efficiency haven been given in [4], [11], [14], and [31]. The usual one-time pre-computation is captured by our algorithm FunctionCommit. In the case of reuse of the same function over multiple datasets, this property enables an improvement in terms of runtime.

3 A Pairing-Based FDC Instantiation

In this section, we present an instantiation of a FDC scheme based on pairings. Our construction uses asymmetric bilinear groups. It can be use to verify the correct evaluation of linear functions. In the following, we analyze our scheme with regard to their hiding and binding property, as well as correctness, unforgeability, succinctness and amortized efficiency.

Definition 9. *Let $\mathcal{G}$ be a generator of cyclic groups of order p and let $\mathbb{G} \xleftarrow{\$} \mathcal{G}(1^\lambda)$. We say the Discrete Logarithm assumption (DL) holds in $\mathbb{G}$ if there exists no PPT adversary $\mathcal{A}$ that, given (g, g^a) for a random generator $g \in \mathbb{G}$ and random $a \in \mathbb{Z}_p$, can output a with more than negligible probability, i.e. if* $\Pr[a \leftarrow \mathcal{A}(g, g^a) \mid g \xleftarrow{\$} \mathbb{G},\ a \xleftarrow{\$} \mathbb{Z}_p] = \mathsf{negl}(\lambda)$.

Definition 10 (Asymmetric bilinear groups [8]). *An* asymmetric bilinear group *is a tuple* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$*, such that:*

- $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *are cyclic groups of prime order* p*,*
- $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *are generators for their respective groups,*
- *the DL assumption holds in* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$*,*
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is bilinear, i.e.* $e\left({g_1}^a, {g_2}^b\right) = e\left(g_1, g_2\right)^{ab}$ *holds for all* $a, b \in \mathbb{Z}$*,*
- e *is non-degenerate, i.e.* $e\left(g_1, g_2\right) \neq 1$*, and*
- e *is efficiently computable.*

We write $g_t = e\left(g_1, g_2\right)$.

The security of our pairing-based instantiation relies on a hardness assumption previously introduced by Catalano et al. [11], the Flexible Diffie-Hellman Inversion (FDHI) problem. It was shown by these authors that the FDHI holds in the generic group model.

Definition 11 ([11]). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the Flexible Diffie–Hellman Inversion (FDHI) assumption holds in* bgp *if for every PPT adversary $\mathcal{A}$*

$$\Pr[W \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\} \wedge W' = W^{\frac{1}{z}} : (W, W') \leftarrow \mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}) \mid z, r, v \xleftarrow{\$} \mathbb{F}_p] = \mathsf{negl}(\lambda).$$

3.1 Construction

We are now ready to describe the algorithms making up our FDC. We use a signature scheme $\Sigma = (\mathsf{SigKeyGen}, \mathsf{Sign}, \mathsf{SigVerify})$ and a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{F}_p$. For a set of possibly different messages $\mathsf{m}_1, \ldots, \mathsf{m}_n$, we denote by m_i the i-th message. Since our messages are vectors, i.e. $\mathsf{m} \in \mathbb{F}_p^T$, we write $\mathsf{m}[j]$ to indicate the j-th entry of message vector m. Therefore $\mathsf{m}_i[j]$ denotes the j-th entry of the i-th message. Given a linear function f, its i-th coefficient is denoted by f_i, i.e. $\mathsf{f}(\mathsf{m}_1, \ldots, \mathsf{m}_n) = \sum_{i=1}^{n} \mathsf{f}_i \mathsf{m}_i$.

- Setup takes as input the security parameter λ. It defines the parameters $n, T \in \mathbb{N}$. Then, it first chooses a bilinear map $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ with $e(g_1, g_2) = g_t$. Afterwards, it chooses $a_0, \ldots, a_T \in \mathbb{F}_p$ uniformly at random. It checks whether $a_j \neq a_i$ for all $j \neq i$. If it fails it chooses a new a_j. Then, for all $j = 0, \ldots, T$ it computes $H_j = g_1^{a_j}$. It outputs the public parameters $\mathsf{pp} = (n, T, \mathsf{bgp}, H_0, \ldots H_T)$.
- KeyGen takes as input the security parameter λ, and the public parameters pp. Then it chooses $y \in \mathbb{F}_p$ uniformly at random and computes $Y = g_2^y$. Additionally it chooses $b_1, \ldots b_n \in \mathbb{F}_p$ uniformly at random and checks whether $b_i \neq b_j$ for all $i \neq j$. If it fails it chooses a new b_i. Then, for all $i = 1, \ldots, n$ it computes $\hat{H}_i = g_1^{b_i}$ and $\hat{h}_i = g_t^{b_i}$. Then, the algorithm chooses random seeds $K, K' \in \mathcal{K}$ for a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{F}_p$. Finally, it generates keys for the signature scheme by calling $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{SigKeyGen}(1^\lambda)$ and outputs public key $\mathsf{pk} = (\mathsf{pk}_{\mathsf{Sig}}, Y, \hat{h}_1, \ldots, \hat{h}_n)$ and secret key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, y, \hat{H}_1, \ldots, \hat{H}_n, K, K')$.
- $\mathsf{PublicCommit}$ takes as input the public parameters pp, a message $\mathsf{m} \in \mathbb{F}_p^T$, and randomness $\mathsf{r} \in \mathbb{F}_p$. It computes $\mathsf{C} = H_0^r \cdot \prod_{j=1}^{T} H_j^{\mathsf{m}[j]}$, where $\mathsf{m}[j]$ is the j-th entry of message vector $\mathsf{m} \in \mathbb{F}_p^T$, and outputs commitment C.
- $\mathsf{PrivateCommit}$ takes as input the secret key sk, a message $\mathsf{m} \in \mathbb{F}_p^T$, randomness $\mathsf{r} \in \mathbb{F}_p$, a dataset $\Delta \in \{0,1\}^*$, and an identifier $\tau \in [n]$. It first computes $z = F_K(\Delta)$ with the pseudorandom function F and calculates $Z = g_2^z$. Then, it binds Z to the dataset identifier Δ by signing their concatenation, i.e. $\sigma_\Delta = \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, \Delta \mid Z)$. Then, it computes $u = F_{K'}(\Delta \mid \tau)$, $U = g_1^u$, and $V = (U \cdot \hat{H}_\tau \cdot H_0^{yr} \cdot \prod_{j=1}^{T} H_j^{ym[j]})^{\frac{1}{z}}$. It returns authenticator $\mathsf{A} = (\sigma_\Delta, Z, U, V)$.
- $\mathsf{FunctionCommit}$ takes as input the public key pk and a labeled program $\mathcal{P} = (\mathsf{f}, \tau_1, \ldots, \tau_n)$. It computes $\mathsf{F} = \prod_{i=1}^{n} \hat{h}_i^{\mathsf{f}_i}$, where f_i denotes the i-th coefficient of f, and outputs function commitment F.
- Eval takes as input a linear function f and authenticators $\mathsf{A}_1, \ldots, \mathsf{A}_n$ where $\mathsf{A}_i = (\sigma_{\Delta,i}, Z_i, U_i, V_i)$. It sets $\sigma_\Delta = \sigma_{\Delta,1}$, $Z = Z_1$ and computes $U = \prod_{i=1}^{n} U_i^{\mathsf{f}_i}$ and $V = \prod_{i=1}^{n} V_i^{\mathsf{f}_i}$ and outputs authenticator $\mathsf{A} = (\sigma_\Delta, Z, U, V)$.
- $\mathsf{FunctionVerify}$ takes as input the public key pk, an authenticator $\mathsf{A} = (\sigma_\Delta, Z, U, V)$, a commitment C, and a function commitment F. It checks whether $\mathsf{SigVerify}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta \mid Z) = 1$ holds. If not it outputs 0, otherwise it checks whether $e(V, Z) = e(U, g_2) \cdot \mathsf{F} \cdot e(\mathsf{C}, Y)$ holds. If it does, it outputs 1; otherwise it outputs 0.

$\mathsf{PublicDecommit}$ takes as input the public parameters pp, a message $\mathsf{m} \in \mathbb{F}_p^T$, randomness $\mathsf{r} \in \mathbb{F}_p$, and a commitment C. It outputs 1 if $\mathsf{C} = \mathsf{PublicCommit}(\mathsf{pp}, \mathsf{m}, \mathsf{r})$ and 0 otherwise.

Our construction can also be used to provide authenticity in the form of unconditionally hiding authenticators, similarly to signatures. First, algorithm KeyGen is called. To authenticate a message m, the owner of the corresponding secret key sk generates a random value r and computes an authenticator A with algorithm $\mathsf{PrivateCommit}$. The authenticator A serves as a signature for m. To verify the authenticity of m, the verifier first requests m and r from the data owner. It next computes commitment C by calling $\mathsf{PublicCommit}$ with m, r, and the public key pk. Then, it uses pk and algorithm $\mathsf{FunctionCommit}$ to generate a function commitment F_{id} to the identity function of m. Finally, it calls algorithm $\mathsf{FunctionVerify}$ to check whether the triple $\mathsf{C}, \mathsf{A}, \mathsf{F}_{id}$ is valid.

3.2 Properties

In the following, we first prove that our concrete scheme is indeed correct in the sense of Def. 3. We then prove that it satisfies the classical commitment properties — binding and hiding. With respect to efficiency, we next show succinctness and amortized efficiency. Finally, we reduce the security of our scheme to the hardness of the FDHI assumption.

Theorem 1. *Our construction is a correct FDC (Def. 3).*

Proof. Let $\mathsf{m}_1, \ldots, \mathsf{m}_n \in \mathbb{F}_p^T$ be a set of messages, $\mathsf{r}_1, \ldots, \mathsf{r}_n \in \mathbb{F}_p$ a set of randomness, $\tau_1, \ldots, \tau_n \in \chi$ set of identifiers, and $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ a linear multi-labeled program. We set $\mathsf{m} = \mathsf{f}(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, $\mathsf{r} = \mathsf{f}(\mathsf{r}_1, \ldots, \mathsf{r}_n)$, $\mathsf{A}_i = (\sigma_{\Delta,i}, Z_i, U_i, V_i) \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}_i, \mathsf{r}_i, \Delta, \tau_i)$, for $i = 1, \ldots, n$, as well as $\mathsf{F} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_\Delta)$ and $\mathsf{C} \leftarrow \mathsf{PublicCommit}(\mathsf{m}, \mathsf{r})$.
Let $\mathsf{A} = (\sigma_\Delta, Z, U, V) \leftarrow \mathsf{Eval}(\mathcal{P}_\Delta, \mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}_1, \mathsf{r}_1, \Delta, \tau_1), \ldots,$
$\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}_n, \mathsf{r}_n, \Delta, \tau_n))$. By construction, we have $\sigma_\Delta = \sigma_{\Delta,1}$ which is correctly verified as long as the underlying signature scheme is correct. Furthermore, consider $(\mathsf{f}_{id}, \tau_i)$ the identity function on the i-th input and let $\mathsf{F}_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, ((\mathsf{f}_{id}, \tau_i), \Delta)) = \mathsf{F}_i = \hat{h}_i$. Since $z_i = F_K(\Delta)$ and $Z_i = g_2^{z_i}$ we have $Z_i = Z,\ \forall\ i \in [n]$ and therefore $e\,(V_i, Z_i) = e\,(V_i, Z)$
$= e\left((U_i \cdot \hat{H}_{\tau_i} \cdot H_0^{y\mathsf{r}_i} \cdot \prod_{j=1}^T H_j^{y\mathsf{m}_i[j]})^{\frac{1}{z}}, Z\right) = e\left(U_i \cdot \hat{H}_{\tau_i} \cdot H_0^{y\mathsf{r}_i} \cdot \prod_{j=1}^T H_j^{y\mathsf{m}_i[j]}, g_2\right)$
$= e\,(U_i, g_2) \cdot e\left(\hat{H}_{\tau_i}, g_2\right) \cdot e\left(H_0^{y r_i} \cdot \prod_{j=1}^T H_j^{y\mathsf{m}_i[j]}, g_2\right)$
$= e\,(U_i, g_2) \cdot \hat{h}_{\tau_i} \cdot e\left(H_0^{\mathsf{r}_i} \cdot \prod_{j=1}^T H_j^{\mathsf{m}_i[j]}, g_2^y\right)$.
Hence $e\,(V, Z) = e\left(\prod_{i=1}^n V_i^{\mathsf{f}_i}, Z\right)$
$= e\left((\prod_{i=1}^n U_i^{\mathsf{f}_i} \cdot \hat{H}_{\tau_i}^{\mathsf{f}_i} \cdot H_0^{y\mathsf{r}_i\mathsf{f}_i} \cdot \prod_{j=1}^T H_j^{\mathsf{f}_i \cdot y\mathsf{m}_i[j]})^{\frac{1}{z}}, Z\right)$
$= e\left(\prod_{i=1}^n U_i^{\mathsf{f}_i}, g_2\right) \cdot (\prod_{i=1}^n \hat{h}_{\tau_i}^{\mathsf{f}_i}) \cdot e\left(H_0^{\sum_{i=1}^n \mathsf{f}_i \cdot \mathsf{r}_i} \cdot (\prod_{j=1}^T H_j^{\sum_{i=1}^n \mathsf{f}_i \cdot \mathsf{m}_i[j]}), g_2^y\right)$

$= e(U, g_2) \cdot \mathsf{F} \cdot e\left(H_0^{\mathsf{r}} \cdot \prod_{j=1}^{T} H_j^{\mathsf{m}[j]}, Y\right) = e(U, g_2) \cdot \mathsf{F} \cdot e(\mathsf{C}, Y)$
This shows the correctness of our scheme.

Theorem 2. *Our construction is a binding FDC scheme as long as the discrete logarithm problem in $\mathbb{G}_1$ is hard.*

Proof. Assume we have access to an oracle $\mathcal{O}(\cdot)$ that on input pk outputs $(\mathsf{m}, \mathsf{r}) \neq (\mathsf{m}', \mathsf{r}')$ such that $\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}) = \mathsf{PublicCommit}(\mathsf{m}', \mathsf{r}')$. Given $g_1 \in \mathbb{G}_1$ from bgp we show how to use $\mathcal{O}(\cdot)$ to solve the discrete logarithm problem in $\mathbb{G}_1$, i.e. computing x for $g_1' = g_1^x$, where $g_1' \in \mathbb{G}_1$. During the generation of public key pk in KeyGen, we choose random $\alpha_0, \ldots, \alpha_T \in \mathbb{F}_p^*$ and compute $H_0 = {g_1'}^{\alpha_0}$ and $H_i = g_1^{\alpha_i}$ for $i = 1, \ldots, T$.

Afterwards, we query $\mathcal{O}(\mathsf{pk})$, and receive $(\mathsf{m}, \mathsf{r}) \neq (\mathsf{m}', \mathsf{r}')$. If $\mathsf{r} = \mathsf{r}'$ we submit a new query. Otherwise we have $H_0{}^{\mathsf{r}} \cdot \prod_{j=1}^{T} H_j{}^{\mathsf{m}_j} = H_0{}^{\mathsf{r}'} \cdot \prod_{j=1}^{T} H_j{}^{\mathsf{m}'_j}$
$\Leftrightarrow {g_1'}^{\alpha_0 \mathsf{r}} \cdot \prod_{j=1}^{T} g_1^{\alpha_j \mathsf{m}_j} = {g_1'}^{\alpha_0 \mathsf{r}'} \cdot \prod_{j=1}^{T} g_1^{\alpha_j \mathsf{m}'_j} \Leftrightarrow g_1^{x \cdot \alpha_0(\mathsf{r}-\mathsf{r}') + \sum_{j=1}^{T} \alpha_j(\mathsf{m}_j - \mathsf{m}'_j)} = g_1^0$
$\Leftrightarrow x \cdot \alpha_0(\mathsf{r} - \mathsf{r}') + \sum_{j=1}^{T} \alpha_j(\mathsf{m}_j - \mathsf{m}'_j) = 0 \Leftrightarrow x = \frac{1}{\alpha_0(\mathsf{r}-\mathsf{r}')} \sum_{j=1}^{T} \alpha_j(\mathsf{m}'_j - \mathsf{m}_j)$

and found the discrete logarithm $g_1^x = g_1'$. The binding property of algorithm $\mathsf{FunctionCommit}$ can be proven completely analogously.

Theorem 3. *Our construction is an unconditionally hiding FDC (Def. 2).*

Proof. If $\mathsf{r} \xleftarrow{\$} \mathbb{F}_p$ is chosen uniformly at random then $\{H_0^{\mathsf{r}} \mid \mathsf{r} \xleftarrow{\$} \mathbb{F}_p\}$ is uniformly distributed over $\mathbb{G}_1$. Therefore the set $\{H_0^{\mathsf{r}} \cdot \prod_{j=1}^{T} H_j^{\mathsf{m}[j]} \mid \mathsf{r} \xleftarrow{\$} \mathbb{F}_p\}$ is uniformly distributed over $\mathbb{G}_1$. So $\{\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}) \mid \mathsf{r} \in \mathbb{F}_p\}$ and $\{\mathsf{PublicCommit}(\mathsf{m}', \mathsf{r}') \mid \mathsf{r}' \in \mathbb{F}_p\}$ have the same distribution $\forall \mathsf{m}, \mathsf{m}' \in \mathbb{F}_p^T$. The output of $\mathsf{PrivateCommit}$ is an authenticator $\mathsf{A} = (\sigma_\Delta, Z, U, V)$. By construction σ_Δ, Z, U are all independent of m. Considering the V component, we have $V = \left(U \cdot \hat{H}_\tau \cdot (H_0^{\mathsf{r}} \cdot \prod_{j=1}^{T} H_j^{\mathsf{m}[j]})^y\right)^{\frac{1}{z}}$.
This is uniformly distributed over $\mathbb{G}_1$ if and only if the set $\{H_0^{\mathsf{r}} \cdot \prod_{j=1}^{T} H_j^{\mathsf{m}[j]} \mid \mathsf{r} \xleftarrow{\$} \mathbb{F}_p\}$ is. As we have shown this to be true $\{\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}, \mathsf{r}, \Delta, \tau) \mid \mathsf{r} \in \mathbb{F}_p\}$ and $\{\mathsf{PrivateCommit}(\mathsf{sk}, \mathsf{m}', \mathsf{r}', \Delta, \tau) \mid \mathsf{r}' \in \mathbb{F}_p\}$ have the same distribution for all $\mathsf{m}, \mathsf{m}' \in \mathbb{F}_p^T$.

Theorem 4. *Our construction is succinct (Def. 7).*

Proof. The number of elements contained in an authenticator $\mathsf{PrivateCommit}$ is constant and does therefore not depend on n, the size of the dataset.

Theorem 5. *Our construction achieves amortized efficiency (Def. 8).*

Proof. $\mathsf{PublicCommit}$ is independent of n. $\mathsf{FunctionVerify}$ consists of a signature verification, two pairing evaluations, and two group operations. Thus their combined running time is independent of n whereas an evaluation of f is $\geq \mathrm{O}(n)$. Therefore, our construction achieves amortized efficiency for suitably large n.

Theorem 6. *Our construction is an unforgeable FDC scheme (Def. 6) if Σ is an unforgeable signature scheme, F is a pseudorandom function and the FDHI assumption (see Def. 11) holds in* bgp.

Proof. This proof follows the structure of [11, Theorem 8]. A major difference is that, in our security reduction, the actual outcome of the computation function f is never required. In particular [12, Lemmata 5 and 7], knowledge of the forged outcome of the computation is a crucial part of the security reductions that prove indistinguishability between games. We present a new indistinguishability reduction that only uses group elements.

To prove Theorem 6, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. the game outputs 1, only with negligible probability. Following the notation of [11], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns 1. We use flag values bad_i, initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs 0. Let Bad_i denote the event that bad_i is set to true during game i.

Due to Theorem 5.1 in [18], any adversary who outputs a Type 3 forgery (see Def. 5) can be converted into one that outputs a Type 2 forgery. Therefore we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{Com}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where $\mathcal{A}$ only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 1, except for the following change. Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, \mathsf{m}^*, \mathsf{r}^*, \mathsf{A}^*)$ with $\mathsf{A}^* = (\sigma^*_\Delta, Z^*, U^*, V^*)$ and Z^* has not been generated by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery.

Game 3 is defined as Game 2, except that the pseudorandom function F is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{F}_p$.

Game 4 is defined as Game 3, except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_4 = \mathsf{true}$.

Game 5 is defined as Game 4, except for the following change. At the beginning, $\mathcal{C}$ chooses $z_\mu \in \mathbb{F}_p$ at random and computes $Z_\mu = g_2^{z_\mu}$. It uses Z_μ whenever queried for dataset Δ_μ. It chooses $b_i, s_i \in \mathbb{F}_p$ uniformly at random for $i = 1, \ldots n$ and sets $\hat{H}_i = g_1^{b_i + z_\mu s_i}$ as well as $\hat{h}_i = g_t^{b_i + z_\mu s_i}$. If $k = \mu$, simulator $\mathcal{S}$ sets the component $U_\tau = g_1^{-b_\tau - a_0 y\mathsf{r} - \sum_{j=1}^{T} a_j y\mathsf{m}[j]}$.

Game 6 is defined as Game 5, except for the following change. The challenger runs an additional check. It computes $\hat{m} = \mathcal{P}(m_1, \ldots, m_n), \hat{r} = \mathcal{P}(r_1, \ldots, r_n)$, as well as $\hat{\mathsf{A}} = \mathsf{Eval}(\mathcal{P}^*_{\Delta^*}, \mathsf{A}_1, \ldots, \mathsf{A}_n)$, i.e. it runs an honest computation over the messages, randomness and authenticators in dataset Δ_μ. If

$$\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}^*, C^*, \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*})) = 1$$

and $U^* = \hat{U}$, then $\mathcal{C}$ sets $\mathsf{bad}_6 = \mathsf{true}$.

Game 7 is defined as Game 6, except for the following change. During a query for $(\Delta_\mu, \tau, \mathsf{m}, \mathsf{r})$, the challenger sets $U_\tau = g_1^{-b_\tau}$.

- Any noticeable difference between Games 1 and 2 can be reduced to producing a forgery for the signature scheme. If Bad_2 occurs, then $\mathcal{A}$ produced a valid signature $\sigma^*_{\Delta^*}$ for $(\Delta^* \mid Z^*)$ despite never having queried a signature on any $(\Delta^* \mid \cdot)$. This is a forgery on the signature scheme.
- Under the assumption that F is pseudorandom, Games 2 and 3 are computationally indistinguishable.
- By definition, $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.
- $\Pr[G_4(\mathcal{A})] = \Pr[G_5(\mathcal{A})]$, since the public keys are perfectly indistinguishable.
- Clearly, $|\Pr[G_5(\mathcal{A})] - \Pr[G_6(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_6]$. This occurs only with negligible probability if the FDHI assumption holds. For a proof of this statement, we refer to Lemma 1 in the Appendix.
- Since the b_i were chosen uniformly at random, Game 7 is perfectly indistinguishable from Game 6. After these modifications, Game 7 can only output 1 if $\mathcal{A}$ produces a forgery $(\mathcal{P}^*_{\Delta^*}, \mathsf{m}^*, \mathsf{r}^*, \mathsf{A}^*)$ with $\mathsf{A}^* = (\sigma^*_\Delta, Z^*, U^*, V^*)$ s.t.

$$\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{A}^*, \mathsf{PublicCommit}(\hat{m}, \hat{r}), \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*})) = 1$$

and $(\hat{\mathsf{m}}, \hat{\mathsf{r}}) \neq (\mathsf{m}^*, \mathsf{r}^*)$, $\hat{U} \neq U^*$, and $\hat{V} \neq V^*$. This only occurs with negligible probability if the FDHI assumption holds. For a corresponding proof, we refer to Lemma 2 in the Appendix.

4 Verifiable Computing on Shared Data from our FDC

We now show how to build a verifiable multi-party computation scheme for linear functions using our pairing-based FDC construction from Sec. 3. A trivial solution would be to use a linearly homomorphic authenticator on each set of shares, running the homomorphic evaluation multiple times in parallel. Our construction only requires a single evaluation over the authenticators. We first recall the algorithms making up a verifiable computing scheme. We then briefly list relevant properties, and then present our construction. Finally, we sketch proofs for the properties of our verifiable computing scheme, notably input and output privacy.

Definition 12 (Verifiable Computing Scheme). *A* Verifiable Computing Scheme $\mathcal{VC}$ *is a tuple of the following PPT algorithms ([19]):*

$\mathsf{VKeyGen}(1^\lambda, f)$: *The probabilistic key generation algorithm takes a security parameter λ and the description of a function f. It generates a secret key* sk*, a corresponding verification key* vk*, and a public evaluation key* ek *(that encodes the target function f) and returns all these keys.*

$\mathsf{ProbGen}(\mathsf{sk}, x)$: *The problem generation algorithm takes a secret key* sk *and data x. It outputs a decoding value ρ_x and a public value σ_x which encodes x.*

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *The computation algorithm takes the evaluation key* ek *and the encoded input σ_x. It outputs an encoded version σ_y of the function's output $y = f(x)$.*

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: *The verification algorithm obtains a verification key* vk *and the decoding value* ρ_x*. It converts the encoded output* σ_y *into the output of the function* y*. If* $y = f(x)$ *holds, it returns* y *or outputs* $\perp$ *indicating that* σ_y *does not represent a valid output of* f *on* x*.*

Relevant properties for *verifiable computing schemes* are *correctness*, *publicly verifiability*, and *security*. For formal definitions, see [17]. Further privacy properties are *input privacy w.r.t. the servers*, *output privacy w.r.t. the servers*, *input privacy w.r.t. the verifier*, and *output privacy w.r.t. the verifier* (see [17]). These computationally secure versions can naturally be extended to information-theoretically secure versions; for more details we refer to Appendix C.

4.1 Construction

Our instantiation of a FDC can be used to build a verifiable computing scheme for shared data supporting linear functions. Secure multi-party computation performed on shared data is realized using a secret sharing scheme, e.g. Shamir secret sharing [30], which we briefly describe. To share a secret $\mathsf{m} \in \mathbb{F}_p$, the client chooses random $a_1, \ldots, a_{t-1} \in \mathbb{F}_p$ and computes the polynomial $P(x) = \mathsf{m} + a_1 x + \ldots + a_{t-1} x^{t-1}$. By evaluating $P(j)$ for $j = 1, \ldots, k$ it creates k shares which are given to k shareholders. Since a polynomial of degree $t-1$ is uniquely determined by t points $(j, P(j))$ one can recover the secret by requesting t shares. At the same time, even a computationally unbounded adversary cannot learn anything about m from $t-1$ shares or less (see [30]). Shamir secret sharing is linearly homomorphic, i.e. $\alpha P(j) + \beta P'(j) = (\alpha P + \beta P')(j)$ for any two polynomials $P, P' \in \mathbb{F}_p[x]$ and constants $\alpha, \beta \in \mathbb{F}_p$. Linear functions can thus be evaluated locally on the shares.

Verifiable computing for shared data can be performed as follows. For $\mathsf{VKeyGen}$, the client runs Setup, Gen, and $\mathsf{FunctionCommit}$ of our construction (see Sec. 3). In our construction, the verification key consists of the public key and the function commitment, i.e. $\mathsf{vk} = (\mathsf{pk}, \mathsf{F})$ and the evaluation key ek is just the multi-labeled program $\mathcal{P}_\Delta$. Assume the client outsourced its secret data to a distributed storage system, i.e. it computed for each secret m_i a polynomial $P_i(x)$ and sent $\phi_j(\mathsf{m}_i) = P_i(j)$ to shareholder j. To allow the shareholders to perform operations on this data, for each secret m_i for $i = 1, \ldots, n$ it first chooses a random value r_i and sends a corresponding share $\phi'_j(\mathsf{r}_i) = P'_i(j)$ to shareholder j.

For $\mathsf{ProbGen}$, the secret key sk is used by the client to run $\mathsf{PrivateCommit}$ of our construction computing a public value σ_{m_i} in form of an authenticator A_i for the pair $(\mathsf{m}_i, \mathsf{r}_i)$. Then, the client sends this value to a dedicated shareholder. The authenticators are unconditionally hiding, i.e. they reveal no information about secrets m_i nor randomness r_i even to a computationally unbounded attacker. The share $P_i(j)$ and the authenticator A_i is in our construction the encoding required by $\mathsf{ProbGen}$ with no decoding value needed.

For $\mathsf{Compute}$, a distinct principal (or the client) gives program $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ to the shareholders. Since a majority of storage servers is assumed to be honest (this is a common assumption), privacy-violating functions can

be denied. Each shareholder j performs program $\mathcal{P}_\Delta$ by evaluating f on its shares, i.e. it computes shares $\phi_j(\mathsf{m}) = \mathsf{f}(\phi_j(\mathsf{m}_1), \ldots, \phi_j(\mathsf{m}_n))$ and $\phi'_j(\mathsf{r}) = \mathsf{f}(\phi'_j(\mathsf{r}_1), \ldots, \phi'_j(\mathsf{r}_n))$. Furthermore, the dedicated shareholder computes an authenticator for the result by performing Eval of our construction on $\mathsf{A}_1, \ldots, \mathsf{A}_n$. The shareholders then use their shares to reconstruct [30] the claimed outcome of the function evaluation m and r, and call $\mathsf{PublicCommit}$ with m and r to obtain a corresponding commitment C. These commitments are linearly homomorphic, and the shareholders can construct C by reconstructing m and r in the exponent.

For Verify, anyone can run $\mathsf{FunctionVerify}$ with C, A, and F as input, in order to prove that C is a commitment to the correct solution. If output privacy is not desired, m and r can be made public; this allows checking that these are opening values to C by calling $\mathsf{PublicDecommit}$ of our construction.

4.2 Properties

Theorem 7. *The verifiable computing scheme for shared data presented above provides correctness, public verifiability, security, input privacy w.r.t. the servers, output privacy w.r.t the servers, input privacy w.r.t. the verifier, and output privacy w.r.t. the verifier.*

Proof. These properties mainly follow from the properties of our FDC:

Correctness follows directly from the evaluation correctness of FDCs.

Public verifiability follows from the construction of algorithm Gen of FDCs. The output of this algorithm are two keys, a secret key to generate the authenticators and a public key to generate commitments to messages and functions.

Security follows from the unforgeability of our FDC.

Input privacy w.r.t. the servers follows from using multi-party computation, where each shareholder independently computes the function on its shares. Our scheme offers input privacy against an adversary actively corrupting at most $t-1$ shareholders, while unforgeability holds even if the adversary can actively corrupt *all* shareholders. We prove privacy w.r.t. the servers by showing that a simulator $\mathcal{S}$ can simulate the protocol without needing to know any input values. Assume that $s_j(\mathsf{m}) = (s_j(\mathsf{m}[1]), \ldots s_j(\mathsf{m}[T]))$ and the simulator $\mathcal{S}$ stores each x_j, where $H_j = g_1^{x_j}$. Then, the simulator chooses $\mathsf{r}_i \in \mathbb{F}_p$ uniformly at random for $i = 1, \ldots, n$ and gives $\mathsf{PrivateCommit}(\mathsf{sk}, 0, \mathsf{r}_i, \Delta, \tau)$, $s_j(0), s_j(\mathsf{r}_i)$ to the adversary $\mathcal{A}$ for $i = 1, \ldots, n, j = 1, \ldots, t-1$. Afterwards, $\mathcal{A}$ outputs A^* and $t-1$ shares of the result $(\hat{\mathsf{m}}, \hat{\mathsf{r}})$. $\mathcal{S}$ can produce shares that reconstruct to $\hat{\mathsf{m}}$ and use its knowledge of each x_j to find an opening $(\hat{\mathsf{m}}, \hat{\mathsf{r}})$ such that $\mathsf{PublicCommit}(\hat{\mathsf{m}}, \hat{\mathsf{r}})$ satisfies
$\mathsf{FunctionVerify}(\mathsf{pk}, \mathsf{Eval}(\mathcal{P}_\Delta, \mathsf{PrivatCommit}(\mathsf{sk}, \mathsf{m}_1, \mathsf{r}_1, \Delta, \tau_1), \ldots,$
$\mathsf{PrivatCommit}(\mathsf{sk}, \mathsf{m}_n, \mathsf{r}_n, \Delta, \tau_n)), \mathsf{PublicCommit}(\hat{\mathsf{m}}, \hat{\mathsf{r}}),$
$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_\Delta)) = 1$.

Output privacy w.r.t. the servers follows directly from the input privacy w.r.t. the servers and the unconditional hiding property of the public commitments.

Input privacy w.r.t. the verifier is derived as follows. We show that a simulator $\mathcal{S}$ given access to the secret key sk, a message $\hat{\mathsf{m}}$, and randomness $\hat{\mathsf{r}}$ can compute the authenticator $\hat{\mathsf{A}} = (\sigma_\Delta, Z, \hat{U}, \hat{V})$ to the outcome $(\hat{\mathsf{m}}, \hat{\mathsf{r}})$ of a computation $\mathcal{P}_\Delta = ((\mathsf{f}, \tau_1, \ldots, \tau_n), \Delta)$ without needing to know any valid input values. It first computes $z = F_K(\Delta)$ with the pseudorandom function F and calculates $Z = g_2^z$. Then, it binds Z to the dataset identifier Δ by concatenating both and signing it, i.e. $\sigma_\Delta = \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, \Delta \mid Z)$. Then, it computes $u_i = F_{K'}(\Delta \mid i)$, and $U_i = g_1^{u_i}$ for $i = 1, \ldots, n$. Afterwards it computes $\hat{U} = \prod_{i=1}^{n} U_i^{\mathsf{f}_i}$. It sets $\hat{V} = (\hat{U} \cdot \prod_{i=1}^{n} \hat{H}_i^{\mathsf{f}_i} \cdot H_0^y \hat{r} \cdot \prod_{j=1}^{T} H_j^{y\hat{m}[j]})^{\frac{1}{z}}$. It returns authenticator $\hat{\mathsf{A}} = (\sigma_\Delta, Z, \hat{U}, \hat{V})$. By construction, this is the authenticator created by Eval for any authenticators to valid input values (m_j, r_j). Therefore an authenticator created by Eval hides the input values perfectly, i.e. even against a computationally unbounded adversary.

Output privacy w.r.t. the verifier follows directly from the input privacy w.r.t. the verifier and the unconditional hiding property of the public commitments.

5 Related Work

Commitments: Commitment schemes are a convenient tool to add verifiability to various processes, such as secret sharing [25], multi-party computation [6], or e-voting [22]. The most well-known and widely used commitment schemes used to provide verifiability are Pedersen's commitments [25]. However, this work presents the first *function-dependent* commitment scheme. Unlike previous commitment schemes, our solution provides succinctness and amortized efficiency. Furthermore, function-dependent commitments support messages stored in datasets and thus enables a much more expressive notion of public verifiability and more rigorous definition of forgery. Besides, a secure bulletin board is not required for our solution. In [21], the notion of *functional commitments* is introduced. Their notion of function bindingness, however, is strictly weaker than our notion of adaptive unforgeability. The instantiation proposed supports linear functions on field elements, i.e. vectors of length 1, while we support vectors of arbitrary polynomial length. Furthermore, notions such as amortized efficiency and succinctness are not considered. In commitment based audit schemes authenticity is typically achieved by using a secure bulletin board [16], for which finding secure instantiations has been challenging so far.

Homomorphic authenticators: Homomorphic authenticators have been proposed both in the secret key setting, as homomorphic MACs (e.g. [1,4,10,31]), and in the public key setting as homomorphic signatures (e.g. [2, 11, 13, 14, 26]). In contrast, our approach additionally considers information-theoretic privacy. Most existing constructions do not consider output privacy. In [26] a solution is proposed in the random oracle model for computational output privacy. Our construction is the first to achieve this in an information-theoretic sense.

Catalano–Fiore–Nizzardo homomorphic signature scheme [11]: Both our FDC and the homomorphic signature scheme presented in [11] are based on the FDHI assumption, and indeed our FDC builds on this homomorphic signature scheme. The Catalano–Fiore–Nizzardo construction is context hiding, i.e. signatures to the output of a function do not leak information about the inputs to the function beyond knowledge of the output to a third party verifier. By contrast, our FDC achieves an even stronger privacy property: information-theoretic input–output privacy with respect to both verifiers and servers. A freshly signed signature in the case of [11] still reveals information about the message to an adversary corrupting a server. Our unconditionally hiding FDC, however, does not. Furthermore, our verification algorithm FunctionVerify only requires a commitment to the output of a computation, enabling output privacy, while verification in [11] requires the output itself. This called for a novel strategy in our security reduction.

Functional cryptography: Functional dependencies in general were, until now, only available for primitives required to be either hiding, i.e. functional encryption, or binding, i.e. functional signatures. Our notion of FDCs is both binding and (depending on the instantiation, even unconditionally) hiding. Functional encryption and functional signatures have been used to build verifiable computing schemes. However, the functional-encryption-based solution proposed by Parno et al. [24] does not provide privacy nor public verifiability. The solution by Barbosa and Farshim [5] makes use of additional primitives, such as predicate encryption schemes for general predicates for which no efficient construction is available. Furthermore, this solution only provides computational input privacy with respect to the verifier. For functional signatures, only one verifiable computing scheme has been proposed by Boyle et al. [9]; it does not provide any privacy.

Verifiable computation: There are many more verifiable computing schemes using proof- and argument-based systems, or based on fully homomorphic encryption. However, there are only few approaches that address public verifiability and input privacy. There are also argument-based verifiable computing schemes available that provide public verifiability and statistical input privacy towards the verifier [3, 7, 15, 23, 29]. All of these are based on strong, so called non-falsifiable assumptions [20]. However, these verifiable computing schemes are not tailored to perform computations on secret shares. Schoenmakers and Veeningen show [28] how to achieve this, but they demand that every shareholder performs computations in order to allow for verification and thereby produces a significant overhead. Our verifiable computing scheme for shared data allows to process secret shares while only one storage server has to compute the audit data. Furthermore, since our solution only makes use of FDC and signatures we are able to provide a concrete instantiation of this approach using our FDC construction. The resulting scheme provides public verifiability, unconditional input privacy towards the servers and the verifier, and relies on standard assumptions.

Multi-party computation: Regarding multi-party computation, only two schemes enable a publicly verifiable audit trail. They have been proposed by Baum et

al. [6] and Schabhüser et al. [27]. Unlike our approach, they achieve public verifiability for arbitrary arithmetic circuits. However, our approach is the first with amortized efficiency. The client incurs setup costs, but setup is only performed once. Afterwards, multiple functions can be evaluated and verified. The verification process itself is more efficient than performing the operations locally for a suitably large number of inputs n.

6 Conclusion

In this paper, we introduced a novel approach to guarantee data authenticity in delegated computing settings. Our function-depend commitments enable fast correctness verification, proof of origin, and information-theoretic input-output privacy. We also provided a concrete instantiation of this generic construction for linear functions. Using this instantiation, we introduced a verifiable computing scheme for shared data with unconditional privacy both towards the server and the verifier. Furthermore, this scheme only requires a single party to perform the computationally more costly computation of the authenticators. Our instantiation does not require revealing a computation's outcome, but merely a commitment to it, thus also satisfying information-theoretic output privacy.

Future Work Our FDC instantiation adds verifiability and authenticity to applications while maintaining privacy — even unconditionally. We intend to further analyse this impact of FDCs on applications processing sensitive data. To this end, we plan to examine the composability of FDCs and investigate black-box constructions of FDC-based verifiable multi-party computation schemes.

Acknowledgments

This work has been co-funded by the DFG as part of project "Long-Term Secure Archiving" within CRC 1119 CROSSING. It has also received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement 644962.

References

1. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-Based Integrity for Network Coding. In: ACNS 2009. pp. 292–305 (2009)
2. Attrapadung, N., Libert, B.: Homomorphic Network Coding Signatures in the Standard Model. In: PKC 2011. pp. 17–34 (2011)
3. Backes, M., Barbosa, M., Fiore, D., Reischuk, R.M.: ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In: SP 2015. pp. 271–286. IEEE Computer Society (2015)
4. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable Delegation of Computation on Outsourced Data. In: CCS'13. pp. 863–874. ACM (2013)
5. Barbosa, M., Farshim, P.: Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation. In: CT-RSA 2012. LNCS, vol. 7178, pp. 296–312. Springer (2012)

6. Baum, C., Damgård, I., Orlandi, C.: Publicly Auditable Secure Multi-Party Computation. In: SCN 2014. LNCS, vol. 8642, pp. 175–196. Springer (2014)
7. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In: CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer (2013)
8. Boneh, D., Franklin, M.K.: Identity-Based Encryption from the Weil Pairing. SIAM J. Comput. 32(3), 586–615 (2003)
9. Boyle, E., Goldwasser, S., Ivan, I.: Functional Signatures and Pseudorandom Functions. In: PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer (2014)
10. Catalano, D., Fiore, D., Gennaro, R., Nizzardo, L.: Generalizing Homomorphic MACs for Arithmetic Circuits. In: PKC 2014. pp. 538–555 (2014)
11. Catalano, D., Fiore, D., Nizzardo, L.: Programmable Hash Functions Go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. In: CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 254–274. Springer (2015)
12. Catalano, D., Fiore, D., Nizzardo, L.: Programmable Hash Functions go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. IACR Cryptology ePrint Archive 2015, 826 (2015)
13. Catalano, D., Fiore, D., Warinschi, B.: Efficient Network Coding Signatures in the Standard Model. In: PKC 2012. pp. 680–696 (2012)
14. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic Signatures with Efficient Verification for Polynomial Functions. In: CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 371–389. Springer (2014)
15. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile Verifiable Computation. In: SP 2015. pp. 253–270. IEEE Computer Society (2015)
16. Culnane, C., Schneider, S.A.: A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In: CSF. pp. 169–183. IEEE Computer Society (2014)
17. Demirel, D., Schabhüser, L., Buchmann, J.A.: Privately and Publicly Verifiable Computing Techniques: A Survey. Springer Briefs in Comp. Science, Springer (2017)
18. Freeman, D.M.: Improved Security for Linearly Homomorphic Signatures: A Generic Framework. In: PKC 2012. LNCS, vol. 7293, pp. 697–714. Springer (2012)
19. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer (2010)
20. Gentry, C., Wichs, D.: Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions. In: STOC. pp. 99–108. ACM (2011)
21. Libert, B., Ramanna, S.C., Yung, M.: Functional Commitment Schemes: From Polynomial Commitments to Pairing-Based Accumulators from Simple Assumptions. In: ICALP 2016. LIPIcs, vol. 55, pp. 30:1–30:14. Schloss Dagstuhl (2016)
22. Moran, T., Naor, M.: Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In: CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer (2006)
23. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: SP 2013. pp. 238–252. IEEE Computer Society (2013)
24. Parno, B., Raykova, M., Vaikuntanathan, V.: How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In: TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer (2012)
25. Pedersen, T.P.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: CRYPTO '91. LNCS, vol. 576, pp. 129–140. Springer (1991)
26. Schabhüser, L., Buchmann, J.A., Struck, P.: A Linearly Homomorphic Signature Scheme from Weaker Assumptions. In: IMACC. LNCS, vol. 10655, pp. 261–279. Springer (2017)

27. Schabhüser, L., Demirel, D., Buchmann, J.A.: An Unconditionally Hiding Auditing Procedure for Computations over Distributed Data. In: CNS 2016. pp. 552–560. IEEE (2016)
28. Schoenmakers, B., Veeningen, M.: Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In: ACNS 2015. LNCS, vol. 9092, pp. 3–22. Springer (2015)
29. Schoenmakers, B., Veeningen, M., de Vreede, N.: Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation. In: ACNS 2016. LNCS, vol. 9696, pp. 346–366. Springer (2016)
30. Shamir, A.: How to Share a Secret. Commun. ACM 22(11), 612–613 (1979)
31. Zhang, L.F., Safavi-Naini, R.: Generalized Homomorphic MACs with Efficient Verification. In: ASIAPKC'14. pp. 3–12. ACM (2014)

A Proving Pairing-Based FDC Unforgeability

Lemma 1. *Assume there exists a PPT adversary $\mathcal{A}$ for whom Bad_6 occurs with non-negligible probability during Game 6 as described in Theorem 6. Then there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Def. 11) with non-negligible probability.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce the result Bad_6 during Game 6. We show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^u, g_1^{\frac{u}{v}})$, simulator $\mathcal{S}$ simulates Game 6.

Setup Simulator $\mathcal{S}$ chooses $a_j \in \mathbb{F}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{a_j}$. It gives the public parameters $\mathsf{pp} = (T, n, \mathsf{bgp}, H_0, \ldots H_T)$ to $\mathcal{A}$.

KeyGen Simulator $\mathcal{S}$ chooses an index $\mu \in \{1, \ldots, Q\}$ uniformly at random. During key generation, it chooses $b_i, s_i \in \mathbb{F}_p$ uniformly at random for all $i = 1, \ldots, n$. It sets $\hat{h}_i = g_t^{b_i} \cdot e\,(g_1, g_2^z)^{s_i}$ corresponding to $\hat{H}_i = g_1^{b_i + z s_i}$ (which $\mathcal{S}$ can not compute). It chooses $y \in \mathbb{F}_p$ uniformly at random and sets $Y = g_2^y$. It gives the public key $\mathsf{pk} = (\mathsf{pk}_{\mathsf{Sig}}, Y, \hat{h}_1, \ldots, \hat{h}_n)$ to $\mathcal{A}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list T_Δ of tuples $(\tau, \mathsf{m}, \mathsf{r})$, which collects all the label/message/randomness tuples queried by the adversary on Δ and the respectively generated authenticators. Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_\mu \in \mathbb{F}_p$, sets $Z_\mu = (g_2^z)^{\xi_\mu}$ and stores ξ_μ. If $k \neq \mu$, it samples a random $\xi_k \in \mathbb{F}_p$ and sets $Z_k = (g_2^v)^{\xi_k}$ and stores ξ_k. Since all Z_k are randomly distributed in $\mathbb{G}_2$ they have the same distribution as in Game 6. Given a query $(\Delta, \tau, \mathsf{m}, \mathsf{r})$ with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k} = \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_k \mid Z_k)$.

If $k \neq \mu$, it samples $\rho_\tau \in \mathbb{F}_p$ uniformly at random and computes $U_\tau = g_1^{-b_\tau} \cdot (g_1^u)^{\rho_\tau} \cdot g_1^{-a_0 y \mathsf{r} - \sum_{j=1}^{T} a_j y \mathsf{m}[j]}$, $V_\tau = \left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau}\right)^{\frac{1}{\xi_k}}$, and gives $\mathsf{A} =$

$(\sigma_{\Delta_k}, \mathsf{Z_k}, \mathsf{U}_\tau, \mathsf{V}_\tau)$ to $\mathcal{A}$. We have

$$e\left(V_\tau, Z_k\right) = e\left(\left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau}\right)^{\frac{1}{\xi_k}}, Z_k\right) = e\left(((g_1^z)^{s_\tau} \cdot (g_1^u)^{\rho_\tau})^{\frac{1}{v\xi_k}}, Z_k\right) =$$

$$e\left(((g_1^z)^{s_\tau} \cdot g_1^{b_\tau} \cdot g_1^{-b_\tau} \cdot g_1^{-a_0 y\mathsf{r} - \sum_{j=1}^{T} a_j y\mathsf{m}[j]} \cdot g_1^{a_0 y\mathsf{r} + \sum_{j=1}^{T} a_j y\mathsf{m}[j]} \cdot (g_1^u)^{\rho_\tau})^{\frac{1}{z_k}}, Z_k\right)$$

$$= \hat{h}_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(g_1^{a_0 \mathsf{r}} \prod_{j=1}^{T} g_1^{a_j \mathsf{m}[j]}, g_2^y\right)$$

$$= \hat{h}_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}), Y\right).$$

Thus this output is indistinguishable from the challenger's output during Game 6.

If $k = \mu$, simulator $\mathcal{S}$ sets $U_\tau = g_1^{-b_\tau - a_0 y\mathsf{r} - \sum_{j=1}^{T} a_j y\mathsf{m}[j]}$, computes $V_\tau = (g_1^{s_\tau})^{\frac{1}{\xi_\mu}}$ and gives $\mathsf{A} = (\sigma_{\Delta_\mu}, \mathsf{Z}_\mu, \mathsf{U}_\tau, \mathsf{V}_\tau)$ to $\mathcal{A}$. We have

$$e\left(V_\tau, Z_\mu\right) = e\left((g_1^{s_\tau})^{\frac{1}{\xi_\mu}}, Z_\mu\right) = e\left((g_1^{zs_\tau})^{\frac{1}{z\xi_\mu}}, Z_\mu\right) = e\left((g_1^{zs_\tau})^{\frac{1}{z_\mu}}, Z_\mu\right)$$

$$= e\left((g_1^{zs_\tau} \cdot g_1^{b_\tau - b_\tau + a_0 y\mathsf{r} + \sum_{j=1}^{T} a_j y\mathsf{m}[j] - a_0 y\mathsf{r} - \sum_{j=1}^{T} a_j y\mathsf{m}[j]})^{\frac{1}{z_\mu}}, Z_\mu\right)$$

$$= e\left(g_1^{zs_\tau} \cdot g_1^{b_\tau - b_\tau + a_0 y\mathsf{r} + \sum_{j=1}^{T} a_j y\mathsf{m}[j] - a_0 y\mathsf{r} - \sum_{j=1}^{T} a_j y\mathsf{m}[j]}, g_2\right)$$

$$= g_t^{zs_\tau} \cdot g_t^{-b_\tau} \cdot g_t^{b_\tau} \cdot g_t^{ya_0\mathsf{r} + \sum_{j=1}^{T} ya_j\mathsf{m}[j]} = \hat{h}_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}), Y\right).$$

Thus this output is indistinguishable from the challenger's output during Game 6.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, C^*, \mathsf{A}^*)$ with $\mathsf{A}^* = (\sigma^*_{\Delta}, Z^*, U^*, V^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 6 to compute $\hat{U}, \hat{V}, \hat{C} = \mathsf{PublicCommit}(\hat{\mathsf{m}}, \hat{\mathsf{r}})$. If Bad_6 occurs, we have $e\left(V^*, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(U^*, g_2\right) \cdot e\left(C^*, Y\right)$ and $e\left(\hat{V}, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(\hat{U}, g_2\right) \cdot e\left(\hat{C}, Y\right)$ Dividing those equations and using the fact that $\hat{U} = U^*$ we obtain $\frac{V^*}{\hat{V}} = (\frac{C^*}{\hat{C}})^{\frac{y}{z\xi_\mu}}$ or equivalently $(\frac{V^*}{\hat{V}})^{\xi_\mu} = (\frac{C^*}{\hat{C}})^{\frac{y}{z}}$ and therefore $W = (\frac{C^*}{\hat{C}})^y$ and $W' = (\frac{V^*}{\hat{V}})^{\xi_\mu}$ are a solution to the FDHI problem. By the definition of unforgeability, we have $W \neq 1$.

Lemma 2. *Assume there exists a PPT adversary $\mathcal{A}$ who wins Game 7 with non-negligible probability. Then there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Def. 11) with non-negligible probability.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that wins Game 7. We show how a simulator $\mathcal{S}$ can use this to solve the FDHI problem.

Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^u, g_1^{\frac{r}{u}})$, simulator $\mathcal{S}$ simulates Game 7.

Setup Simulator $\mathcal{S}$ chooses $a_j \in \mathbb{F}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{a_j}$. It gives the public parameters $\mathsf{pp} = (T, n, \mathsf{bgp}, H_0, \ldots H_T)$ to $\mathcal{A}$.

KeyGen Simulator $\mathcal{S}$ chooses an index $\mu \in \{1,\dots,Q\}$ uniformly at random. During key generation, it chooses $b_i, s_i \in \mathbb{F}_p$ uniformly at random for all $i = 1,\dots,n$. It sets $\hat{h}_i = g_t^{b_i} \cdot e(g_1, g_2^z)^{s_i}$ corresponding to $\hat{H}_i = g_1^{b_i + z s_i}$ (which $\mathcal{S}$ can not compute). It sets $Y = g_2^z$. It gives the public key $\mathsf{pk} = (\mathsf{pk}_{\mathsf{Sig}}, Y, \hat{h}_1, \dots, \hat{h}_n)$ to $\mathcal{A}$.

Setup $\mathcal{S}$ chooses an index $\mu \in \{1,\dots,Q\}$ uniformly at random. During key generation, it chooses $b_i, s_i \in \mathbb{F}_p$ uniformly at random for all $i = 1,\dots,n$. It sets $\hat{h}_i = g_t^{b_i} \cdot e(g_1, g_2^z)^{s_i}$ corresponding to $\hat{H}_i = g_1^{b_i + z s_i}$ (which $\mathcal{S}$ can not compute). It chooses $a_j \in \mathbb{F}_p$ uniformly at random for $j = 0,\dots,T$ and sets $h_j = e(g_1, g_2^z)^{a_j}$ corresponding to $H_j = g_1^{a_j z}$ (which $\mathcal{S}$ can not compute). It gives the public key $\mathsf{pk} = (\mathsf{bgp}, \mathsf{pk}_{\mathsf{Sig}}, H_0 \dots H_T, \hat{H}_1, \dots, \hat{H}_n)$ to $\mathcal{A}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list T_Δ of tuples $(\tau, \mathsf{m}, \mathsf{r})$, which collects all label/message/randomness tuples queried by the adversary on Δ and the respectively generated authenticators. Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{S}$ does the following. If $k = \mu$, it samples a random $\xi_\mu \in \mathbb{F}_p$, sets $Z_\mu = (g_2^z)^{\xi_\mu}$ and stores ξ_μ. If $k \neq \mu$, it samples a random $\xi_k \in \mathbb{F}_p$ and sets $Z_k = (g_2^v)^{\xi_k}$ and stores ξ_k. Since all Z_k are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query $(\Delta, \tau, \mathsf{m}, \mathsf{r})$ with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k} = \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_k \mid Z_k)$.

If $k \neq \mu$ it samples $\rho_\tau \in \mathbb{F}_p$ uniformly at random and computes $U_\tau = g_1^{-b_\tau} \cdot (g_1^u)^{\rho_\tau}$, $V_\tau = \left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j \mathsf{m}[j]} \right)^{\frac{1}{\xi_k}}$ and gives $\mathsf{A} = (\sigma_{\Delta_k}, \mathsf{Z_k}, \mathsf{U}_\tau, \mathsf{V}_\tau)$ to $\mathcal{A}$. We have

$$e(V_\tau, Z_k) = e\left(\left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j \mathsf{m}[j]} \right)^{\frac{1}{\xi_k}}, Z_k \right)$$

$$= e\left(\left((g_1^{z})^{s_\tau} \cdot (g_1^{u})^{\rho_\tau} \cdot (g_1^{z})^{a_0 r + \sum_{j=1}^T a_j \mathsf{m}[j]} \right)^{\frac{1}{v \xi_k}}, Z_k \right)$$

$$= e\left(\left((g_1^{\frac{z}{v}})^{s_\tau} \cdot g_1^{b_\tau} \cdot g_1^{-b_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j \mathsf{m}[j]} \right)^{\frac{1}{z_k}}, Z_k \right)$$

$$= \hat{h}_\tau \cdot e(U_\tau, g_2) \cdot g_t^{a_0 z \mathsf{r}} \prod_{j=1}^T g_t^{a_j z \mathsf{m}[j]} = \hat{h}_\tau \cdot e(U_\tau, g_2) \cdot e(\mathsf{PublicCommit}(\mathsf{m}, \mathsf{r}), Y).$$

Thus this output is indistinguishable from the challenger's output during Game 7.

If $k = \mu$, simulator $\mathcal{S}$ sets $U_\tau = g_1^{-b_\tau}$, $V_\tau = (g_1^{s_\tau} \cdot g_1^{a_0 \mathsf{r} + \sum_{j=1}^T a_j \mathsf{m}[j]})^{\frac{1}{\xi_\mu}}$ and gives $\mathsf{A} = (\sigma_{\Delta_\mu}, \mathsf{Z}_\mu, \mathsf{U}_\tau, \mathsf{V}_\tau)$ to $\mathcal{A}$. We have

$$e(V_\tau, Z_\mu) = e\left(\left(g_1^{s_\tau} \cdot g_1^{a_0 \mathsf{r} + \sum_{j=1}^T a_j \mathsf{m}[j]} \right)^{\frac{1}{\xi_\mu}}, Z_\mu \right)$$

$$= e\left(\left(g_1^{z s_\tau + z a_0 \mathsf{r} + \sum_{j=1}^T z a_j \mathsf{m}[j]} \right)^{\frac{1}{z \xi_\mu}}, Z_\mu \right)$$

$$= e\left(\left(g_1^{zs_\tau} \cdot g_1^{-b_\tau} \cdot g_1^{b_\tau} \cdot g_1^{za_0\mathsf{r}+\sum_{j=1}^{T} za_j\mathsf{m}[j]}\right)^{\frac{1}{z\mu}}, Z_\mu\right)$$
$$= e\left(g_1^{zs_\tau} \cdot g_1^{-b_\tau} \cdot g_1^{b_\tau} \cdot g_1^{za_0\mathsf{r}+\sum_{j=1}^{T} za_j\mathsf{m}[j]}, g_2\right)$$
$$= g_t^{zs_\tau} \cdot g_t^{-b_\tau} \cdot g_t^{b_\tau} \cdot g_t^{za_0\mathsf{r}+\sum_{j=1}^{T} za_j\mathsf{m}[j]} = \hat{h}_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(g_1^{a_0\mathsf{r}+\sum_{j=1}^{T} a_j\mathsf{m}[j]}, g_2^z\right)$$

$= \hat{h}_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(\mathsf{PublicCommit}(\mathsf{m},\mathsf{r}), Y\right)$ and thus this output is indistinguishable from the challenger's output during Game 7.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, \mathsf{m}^*, \mathsf{r}^*, \mathsf{A}^*)$ with $\mathsf{A}^* = (\sigma^*_\Delta, Z^*, U^*, V^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{U}, \hat{V}, \hat{\mathsf{m}}, \hat{\mathsf{r}}$. If Game 7 outputs 1, we have $e\left(V^*, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(U^*, g_2\right) \cdot e\left(C^*, Y\right)$, as well as $e\left(\hat{V}, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(\hat{U}, g_2\right) \cdot e\left(\hat{C}, Y\right)$. Dividing those equations yields $\frac{V^*}{\hat{V}} = \left(\frac{U^*}{\hat{U}} \cdot (\frac{C^*}{\hat{C}})^z\right)^{\frac{1}{z\xi_\mu}} = (\frac{U^*}{\hat{U}})^{\frac{1}{z\xi_\mu}} \cdot (\frac{C^*}{\hat{C}})^{\frac{1}{\xi_\mu}}$. Thus $\mathcal{S}$ can compute $W = \frac{U^*}{\hat{U}}$, $W' = (\frac{V^*}{\hat{V}})^{\xi_\mu} \cdot \frac{C^*}{\hat{C}}$. We have $(W')^z = (\frac{V^*}{\hat{V}})^{z\xi_\mu} \cdot (\frac{C^*}{\hat{C}})^z = \frac{U^*}{\hat{U}} = W$ and thus (W, W') is a solution to the FDHI problem. Our simulation has the same distribution as a real execution of Game 7.

B Formally Defining Bindingness

For a formal definition, we first define two experiments $\mathbf{EXP}^{Bind_M}_{\mathcal{A},\mathrm{Com}}(\lambda)$ and $\mathbf{EXP}^{Bind_F}_{\mathcal{A},\mathrm{Com}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Definition 13 (Bindingness experiments).

$\boldsymbol{EXP}^{Bind_M}_{\mathcal{A},Com}(\lambda)$:
$\mathcal{C}$ *runs* $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and gives* pk *to* $\mathcal{A}$.
$\mathcal{A}$ *outputs the pairs* (m,r) *and* $(\mathsf{m}',\mathsf{r}')$, *with* $(\mathsf{m},\mathsf{r}) \neq (\mathsf{m}',\mathsf{r}')$.
If $\mathsf{PublicCommit}(\mathsf{m}',\mathsf{r}') = \mathsf{PublicCommit}(\mathsf{m},\mathsf{r})$ *output 1, else return 0.*

$\boldsymbol{EXP}^{Bind_F}_{\mathcal{A},Com}(\lambda)$:
$\mathcal{C}$ *runs* $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and gives* pk *to* $\mathcal{A}$.
$\mathcal{A}$ *outputs* $\mathcal{P}_\Delta = ((\mathsf{f},\tau_1,\ldots,\tau_n),\Delta)$ *and* $\mathcal{P}'_\Delta = ((\mathsf{f}',\tau_1,\ldots,\tau_n),\Delta)$, *with* $\mathsf{f} \neq \mathsf{f}'$.
If $\mathsf{FunctionCommit}(\mathsf{pk},\mathcal{P}'_\Delta) = \mathsf{FunctionCommit}(\mathsf{pk},\mathcal{P}_\Delta)$ *output 1, else return 0.*

Using these experiments, we can now define bindingness.

Definition 14 (Bindingness).
Using the formalism of Def. 13, a FDC is called binding *if for any PPT adversary* $\mathcal{A}$,

$$\Pr[\boldsymbol{EXP}^{Bind_M}_{\mathcal{A},Com}(\lambda) = 1] = \mathsf{negl}(\lambda) \wedge \Pr[\boldsymbol{EXP}^{Bind_F}_{\mathcal{A},Com}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Server}}}[\mathcal{VC}, f, \lambda, t]$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^{\lambda})$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek})$
$(\sigma_0, \rho_0) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_1, \rho_1) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{B} \leftarrow \mathcal{A}$ with $\mathcal{B} \subset \{1, \ldots, n\}$ and $|\mathcal{B}| = t-1$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek}, x_0, x_1, \{(\sigma_{y_b,j}, \rho_{x_b,j}\}_{j\in\mathcal{B}})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

C Formalizing Unconditional Privacy for Verifiable MPC

Verifiable computing can guarantee the integrity of a computation. Beyond that, a desirable property is to protect the secrecy of the client's inputs towards the server and when using a publicly verifiable scheme also towards the verifiers. To formally define *input privacy w.r.t the server* we define experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Server}}}$, during which the adversary controls $t-1$ shareholders. We use the oracle $\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},x)}$ which calls $\mathsf{ProbGen}(\mathsf{sk}, x)$ to obtain (σ_x, ρ_x) and only returns the public part σ_x.

In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1 and is given the encoding of one of the two inputs chosen at random. The adversary then must determine which input has been encoded. During this process, the adversary may request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda] = 1\right] - \tfrac{1}{2} \right|.$$

Definition 15 (Input privacy w.r.t. the server). *A verifiable computing scheme $\mathcal{VC}$ provides unconditional input privacy if any computationally unbounded adversary $\mathcal{A}$ has $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Server}}}(\mathcal{VC}, f, \lambda, t) = 0$.*

We give an analogous definition for output privacy. To formally define *output privacy w.r.t the server*, we define experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Server}}}$, during which the adversary controls $t-1$ shareholders. We use the oracle $\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},x)}$ which calls $\mathsf{ProbGen}(\mathsf{sk}, x)$ to obtain (σ_x, ρ_x) and only returns the public part σ_x.

In this experiment, the adversary first receives the public evaluation key for the scheme. Then, it selects two inputs x_0, x_1. It is given the results of the computation $y_0 = f(x_0), y_1 = f(x_1)$ and is given the encoding of one of the two inputs chosen at random. The adversary corrupts $t-1$ shareholders and receives their share of the encoding and then must determine which encoded output has been computed. During this process, the adversary may request the encoding of

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t]$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{ek})$
$(\sigma_{x_0}, \rho_{x_0}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_{x_1}, \rho_{x_1}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$\sigma_{y_0} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_0})$
$\sigma_{y_1} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_1})$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{B} \leftarrow \mathcal{A}$ with $\mathcal{B} \subset \{1, \ldots, n\}$ and $|\mathcal{B}| = t - 1$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{ek}, x_0, x_1, y_0, y_1, \{(\sigma_{y_b,j}, \rho_{x_b,j}\}_{j\in\mathcal{B}})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Verifier}}}[\mathcal{VC}, f, \lambda]$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek})$
$(\sigma_0, \rho_0) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_1, \rho_1) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$b \xleftarrow{\$} \{0,1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek}, x_0, x_1, \sigma_b, \rho_b)$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t] = 1\right] - \frac{1}{2} \right|$$

Definition 16 (Output privacy w.r.t. the server). *A verifiable computing scheme $\mathcal{VC}$ provides unconditional output privacy if any computationally unbounded adversary $\mathcal{A}$ has* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = 0$.

If we have a publicly verifiable computing scheme a third party verifier might try to learn about the input data from the publicly available verification data. To formally define *input privacy w.r.t. a third party verifier* we use experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Verifier}}}$.

In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1 and is given the encoding of one of the two inputs chosen at random. The adversary then must determine which

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{out-Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda]$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^{\lambda})$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk}, \cdot)}}(\mathsf{ek})$
$(\sigma_{x_0}, \rho_{x_0}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_{x_1}, \rho_{x_1}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$\sigma_{y_0} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_0})$
$\sigma_{y_1} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_1})$
$b \xleftarrow{\$} \{0, 1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk}, \cdot)}}(\mathsf{ek}, x_0, x_1, y_0, y_1 \sigma_{y_b}, \rho_{y_b})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

input has been encoded. During this process, the adversary may request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda] = 1 \right] - \frac{1}{2} \right|.$$

Definition 17 (Input privacy w.r.t. the verifier). *A verifiable computing scheme $\mathcal{VC}$ provides unconditional input privacy if for any computationally unbounded adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = 0$.*

For a publicly verifiable computing scheme, it is interesting to show correctness of a computation without revealing its outcome. A third party verifier might try to learn about the output data from the publicly available verification data. To formally define *output privacy w.r.t a third party verifier*, we define experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{out-Privacy}_{\mathsf{Verifier}}}$.

In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1. It is given the results $y_0 = f(x_0), y_1 = f(x_1)$ and is given the encoding of one of the two outputs chosen at random. The adversary then must determine which input has been encoded. During this process, the adversary may request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda] = 1 \right] - \frac{1}{2} \right|.$$

Definition 18 (Output privacy w.r.t. the verifier). *A verifiable computing scheme $\mathcal{VC}$ provides unconditional output privacy if for any computationally unbounded adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = 0$.*