Function-Dependent Commitments from Structure-Preserving Homomorphic Authenticators

Lucas Schabhüser, Denis Butin, and Johannes Buchmann

Technische Universität Darmstadt, Germany
{lschabhueser,dbutin,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. Outsourcing costly computations to an untrusted server is common in cloud computing. A basic requirement for a client is the ability to efficiently verify the computation's result. When sensitive data are involved, verification should be possible without the server learning about the computation's inputs, or its result. Homomorphic authenticators, the dominant approach to this problem, support efficient verification but usually leak information about the processed data. A recently introduced primitive, function-dependent commitments (FDCs), enables both types of privacy and fast correctness verification. However, the relation between FDCs and homomorphic authenticators was unclear until now. In this paper, we show that every FDC scheme can be transformed into a homomorphic authenticator scheme, showing that FDCs are at least as powerful as homomorphic authenticators. We then present a generic transformation turning any structure-preserving homomorphic authenticator scheme into an FDC scheme. The resulting schemes enjoy information-theoretic input and output privacy. Finally, we introduce a new structure-preserving, linearly homomorphic authenticator scheme suitable for our transformation. It is the first both context hiding and structure-preserving homomorphic authenticator scheme. Our scheme is also the first structure-preserving homomorphic authenticator scheme to achieve efficient verification.

1 Introduction

Time-consuming computations are commonly outsourced 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. 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 aggregated data. Hence the requirement for efficient

verification procedures for outsourced computing that are privacy-preserving, both for computation inputs and for computation results.

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.

Existing approaches address verifiability and confidentiality to various degrees.

- Homomorphic authenticators [5] sometimes allow for efficient verification, keeping the computational effort of the verifier low. They do, however, not provide information-theoretic confidentiality. Some schemes offer so-called context hiding security, a form of input privacy. However, stronger privacy notions are not achieved.
- Homomorphic commitments [8,23,27] can be used in audit schemes. In particular, Pedersen commitments [23] provide information-theoretic confidentiality. Homomorphic commitments however, lead to costly verification procedures.
- Function-dependent commitments [26] (FDCs) are a generic construction for authentified delegated computing. Their core idea is as follows. First, commit to input values and to a function. Then, authenticate all inputs. Next, compute an authenticated commitment to the computation's result using homomorphic properties. FDCs combine the advantages of homomorphic authenticators and homomorphic commitments. They allow for information-theoretic input and output privacy as well as efficient verification. Compared to homomorphic authenticators they achieve a *hiding* property. The *context hiding* property of homomorphic authenticators guarantees that authenticators to the *output* of a computation do not leak information about the *input* of the computation (beyond what can be inferred from the result). The authenticator to the input, however, leaks information about the authenticated data. The hiding property of FDCs ensures that not even this is possible. In [26] an information-theoretically hiding FDC was combined with secret sharing for efficiently verifiable multi-party computation. Combining homomorphic authenticators with secret sharing can be instantiated by a straightforward composition, or by authenticating individual shares. The former leads to a loss of information-theoretic privacy, the latter requires all storage servers to perform computations on audit data. By contrast, FDC-based verifiable multi-party computation can be instantiated so that auditing only requires a single storage server. FDC-based verifiable multi-party computation thus provides not only privacy gains, but also efficiency improvements with respect to the classical variant using homomorphic authenticators.

For a detailed comparison to related work, see Sec. 6. There are various homomorphic authenticator schemes fine-tailored to specific scenarios. For FDCs, so far only one construction is known [26]. Adding the privacy properties of FDCs to known homomorphic authenticator schemes makes them suitable even when sensitive data are processed. In this paper, we show how to achieve this.

Contribution Overview In this paper, we investigate the relation between homomorphic authenticators and FDCs. Our contribution is threefold.

First, we show how every FDC can be transformed into a homomorphic authenticator scheme, showing that FDCs are at least as powerful schemes as homomorphic authenticators. To this end, we explicitly construct the algorithms making up a homomorphic authenticator scheme, using only the algorithms from the input FDC. We then derive both authentication correctness and evaluation correctness for the homomorphic authenticator scheme output by our transformation. The proof relies on the correctness of the input FDC scheme. For security, we derive the unforgeability of the resulting homomorphic authenticator scheme from two conditions on the underlying FDC: its own unforgeability, and its bindingness. Regarding bandwidth, we prove that the output homomorphic authenticator scheme is succinct if the input FDC scheme is succinct.

Then, we show how an FDC can be generically constructed from a *structure-preserving* homomorphic authenticator scheme, assuming the additional existence of a homomorphic commitment scheme and of a separate classical commitment scheme. We require the commitment space of the homomorphic commitment scheme to be a subset of the structure preserved by the homomorphic authenticator scheme. The message space of the classical commitment scheme allows labeled programs as admissible inputs, unlike the homomorphic commitment scheme. We show that if the two underlying commitment schemes are binding, then the resulting FDC inherits this bindingness. Furthermore, we prove that the output FDC inherits the unconditional hiding from the underlying homomorphic commitment scheme. The correctness of the output FDC is shown to follow from three assumptions on the input homomorphic authenticator scheme: authentication correctness, evaluation correctness and efficient verification. Regarding security, we prove that unforgeability is also inherited. This is done by showing that a simulator can forward adversary queries in the FDC security experiment to queries in the homomorphic authenticator experiment. The resulting forgery can be used to compute a forgery in the other experiment. For performance, we show that if the input scheme is succinct, respectively efficiently verifiable, then the output FDC is also succinct, respectively has amortized efficiency. Our transformation enables the use of certain existing homomorphic authenticator schemes in particularly privacy-sensitive settings. Applying this transformation enables information-theoretic output privacy. This allows third parties to verify the correct computation of a function without even needing to learn the result.

Finally, we introduce a structure-preserving, linearly homomorphic authenticator scheme suitable for our transformation. All known structure-preserving homomorphic authenticator schemes are limited to linear functions. Our scheme is the first such construction to achieve constant-time verification (after a one-time pre-processing). It is also the first structure-preserving homomorphic authenticator scheme to be *context hiding*. This property ensures that a third-party verifier does not learn anything about the inputs to a computation beyond what it knows from the output of the computation. For simplicity, our scheme is limited to a single dataset. However, it can be extended to a multi-dataset scheme following

a result by Fiore et al. [14]. Furthermore, our scheme is succinct. Authenticators consist of two elements of a cyclic group of prime order. The security of our construction relies on the hardness of the Double Pairing Assumption [1], which implies the Decisional Diffie–Hellman assumption with a tight security reduction.

The remainder of this paper is organized as follows. We first provide the necessary background on FDCs and homomorphic authenticators (Sec. 2). We show how to transform FDCs into homomorphic signatures (Sec. 3). We then show how to construct FDCs from homomorphic commitments and structure-preserving homomorphic authenticators (Sec. 4). Next, we present a new instantiation for a context hiding, structure-preserving linearly homomorphic signature scheme (Sec. 5). Finally, we compare our work to the state of the art (Sec. 6) and conclude (Sec. 7).

2 Preliminaries

In this section, we provide the necessary background for our contributions. We formalize the notion of homomorphic commitments. Afterwards we provide the terminology of labeled programs, on which the notions of unforgeability in this paper are based. We then describe FDCs and their properties. These include the classical hiding and binding properties of commitments, as well as further FDC-specific properties such as correctness, unforgeability, succinctness and amortized efficiency. Likewise, we recall definitions for homomorphic authenticators and their properties. Finally, we define structure-preserving signatures.

Commitment Schemes Commitment schemes, particularly homomorphic commitment schemes, are a basic building block in cryptography. We provide the formalizations used in this work.

Definition 1 (Commitment Scheme). *A commitment scheme* Com *is a tuple of the following algorithms (*CSetup*,* Commit*,* $\mathsf{Decommit}$*):*

$\mathsf{CSetup}(1^\lambda)$: *On input a security parameter λ, this algorithm outputs a commitment key* CK*. We implicitly assume that every algorithm uses this commitment key, leaving it out of the notation.*

$\mathsf{Commit}(m, r)$: *On input a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, it outputs the commitment C and the decommitment d.*

$\mathsf{Decommit}(m, d, C)$: *On input a message $m \in \mathcal{M}$, decommitment d, and a commitment C it outputs* $\mathtt{1}$ *or* $\mathtt{0}$*.*

Definition 2. *Let $\mathcal{F}$ be a class of functions. A commitment scheme* $\mathsf{Com} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit})$ *is $\mathcal{F}$-homomorphic if there exists an algorithm* CEval *with the following properties:*

$\mathsf{CEval}(f, C_1, \ldots, C_n)$: *On input a function $f \in \mathcal{F}$ and a tuple of commitments C_i for $i \in [n]$, the algorithm outputs C^*.*

Correctness: *For every $m_i \in \mathcal{M}$, $r_i \in \mathcal{R}$, $i \in [n]$ with $(C_i, d_i) \leftarrow \mathsf{Commit}(m_i, r_i)$ and $C^* \leftarrow \mathsf{CEval}(f, C_1, \ldots, C_n)$, there exists a unique function $\hat{f} \in \mathcal{F}$, such that $\mathsf{Decommit}(f(m_1, \ldots, m_n), \hat{f}(m_1, \ldots, m_n, r_1, \ldots r_n), C^*) = 1$.*

Labeled Programs To accurately describe both correct and legitimate operations for homomorphic authenticators, we use *multi-labeled programs* similarly to Backes, Fiore, and Reischuk [7]. The basic idea is to append a function by several identifiers, in our case *input identifiers* and *dataset identifiers*. Input identifiers label in which order the input values are to be used. Dataset identifiers determine which authenticators can be homomorphically combined. The idea is that only authenticators created under the same dataset identifier can be combined. Intuitively, we need input identifiers to distinguish between messages. This allows restricting homomorphic evaluation to authenticators related to the same set of messages, leading to a stronger unforgeability notion.

Formally, a *labeled program* $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_n)$, where $f : \mathcal{M}^n \to \mathcal{M}$ is a function with n inputs and $\tau_i \in \mathcal{T}$ with $i \in [n]$ is a label for the i^{th} input of f from some set $\mathcal{T}$. Given a set of labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_N$ and a function $g : \mathcal{M}^N \to \mathcal{M}$, they can be composed by evaluating g over the labeled programs, i.e. $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. This is an abuse of notation analogous to function composition. The identity program with label τ is given by $\mathcal{I}_\tau = (f_{id}, \tau)$, where $f_{id} : \mathcal{M} \to \mathcal{M}$ is the identity function. The program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

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

Definition 3 (Well Defined Program). *A labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is* well defined *with respect to a list $L \subset \mathcal{T} \times \mathcal{M}$ if exactly one of the two following cases holds: First, there are messages $m_1, \ldots, m_n$ such that $(\tau_i, m_i) \in L \ \forall i \in [n]$. Second, there is an $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin L$ and $f(\{m_j\}_{(\tau_j, m_j) \in L} \cup \{m'_k\}_{(\tau_k, \cdot) \notin L})$ is constant over all possible choices of $m'_k \in \mathcal{M}$.*

If f is a linear function, $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, with $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$ fulfills the second condition if and only if $f_k = 0$ for all $(\tau_k, \cdot) \notin L$.

FDCs Going beyond the basic functionalities of homomorphic commitments, the idea of FDCs was introduced by Schabhüser et al. [26]. In particular, this framework allows for a notion of unforgeability.

Definition 4 ([26]). *An FDC scheme for a class $\mathcal{F}$ of functions is a tuple of algorithms (*Setup*,* KeyGen*,* PublicCommit*,* PrivateCommit*,* FunctionCommit*,* Eval*,* FunctionVerify*,* PublicDecommit*):*

- Setup(1^λ) *takes as input 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 (except for* KeyGen*).*
- KeyGen(pp) *takes the public parameters* pp *as input and outputs a secret-public key pair* (sk, pk)*.*

$\mathsf{PublicCommit}(m, r)$ *takes as input a message* m *and randomness* r *and outputs commitment* C*.*

$\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$ *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* (m, r, Δ, τ)*.*

$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$ *takes as input the public key* pk *and a labeled program* $\mathcal{P}$ *and outputs a function commitment* F *to* $\mathcal{P}$*.*

$\mathsf{Eval}(f, A_1, \ldots A_n)$ *takes as input a function* $f \in \mathcal{F}$ *and a set of authenticators* $A_1, \ldots, A_n$*, where* A_i *is an authenticator for* $(m_i, r_i, \Delta, \tau_i)$*, for* $i = 1, \ldots, n$*. It computes an authenticator* A^* *using the* A_i *and outputs* A^**.*

$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$ *takes as input a public key* pk*, an authenticator* A*, a commitment* C*, a function commitment* F*, as well as a dataset identifier* Δ*. It outputs either 1 (accept) or 0 (reject).*

$\mathsf{PublicDecommit}(m, r, C)$ *takes as input message* m*, randomness* r*, and commitment* C*. It outputs either 1 (accept) or 0 (reject).*

The intuition behind FDC algorithms is as follows. FDCs allow for two different ways of committing to messages. One is just a standard commitment. This enables output privacy with respect to the verifier. The other way commits to a message under a secret key to produce an authenticator. These authenticators allow for homomorphic evaluation. Given authenticators to the input of a function, one can derive an authenticator to the output of a function. Additionally, one can commit to a function under a public verification key. This results in a function commitment. One can then check if a public commitment C matches an authenticator A (derived from a secret key) and a function commitment F (derived from a public key). As long as a cryptographic hardness assumption holds, such a match is only possible if A was obtained by running the evaluation on the exact function committed to via F.

As for classical commitments, we want our schemes to be *binding*. That is, after committing to a message, it should be infeasible to open the commitment to a different message. We describe the following security experiment between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Definition 5 (Bindingness experiments $\mathbf{EXP}^{Bind}_{\mathcal{A},\mathbf{Com}}(\lambda)$).
Challenger $\mathcal{C}$ *runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and gives* pk *to the adversary* $\mathcal{A}$*.* $\mathcal{A}$ *outputs the pairs* (m, r) *and* (m', r')*, with* $m \neq m'$*. If* $\mathsf{PublicCommit}(m', r') = \mathsf{PublicCommit}(m, r)$ *the experiment outputs* $\mathtt{1}$*, else it returns* $\mathtt{0}$*.*

Definition 6 (Binding).
Using the formalism of Def. 5, an FDC is called binding *if for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*,* $\Pr[\boldsymbol{EXP}^{Bind}_{\mathcal{A},Com}(\lambda) = 1] = \mathsf{negl}(\lambda)$*, where* $\mathsf{negl}(\lambda)$ *denotes any function negligible in the security parameter* λ*.*

The binding property for $\mathsf{FunctionCommit}$ is defined analogously.

Another important notion, targeting privacy, is the hiding property. Commitments are intended not to leak information about the messages they commit to. 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. Context hiding homomorphic authenticators do however leak information about the output.

Definition 7 (Hiding). *An FDC is called* computationally hiding *if the sets of commitments* $\{\mathsf{PublicCommit}(m, r) \mid r \xleftarrow{\$} \mathcal{R}\}$ *and* $\{\mathsf{PublicCommit}(m', r') \mid r' \xleftarrow{\$} \mathcal{R}\}$ *as well as* $\{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\}$ *and* $\{\mathsf{PrivateCommit}(\mathsf{sk}, m', r', \Delta, \tau) \mid r' \xleftarrow{\$} \mathcal{R}\}$ *have distributions that are indistinguishable for any PPT adversary* $\mathcal{A}$ *for all* $m \neq m' \in \mathcal{M}$*. An FDC is called* unconditionally hiding *if these sets have the* same *distribution respectively for all* $m \neq m' \in \mathcal{M}$*.*

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 accepted. This is formalized in the following definition.

Definition 8 (Correctness). *An FDC achieves* correctness *if for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, and any dataset identifier* $\Delta \in \{0,1\}^*$*, the following properties hold:*

For any message $m \in \mathcal{M}$*, randomness* $r \in \mathcal{R}$*, label* $\tau \in \mathcal{T}$*, authenticator* $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$*, commitment* $C \leftarrow \mathsf{PublicCommit}(m, r)$*, and function commitment* $F_{\mathcal{I}} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$*, where* $\mathcal{I}_\tau$ *is the labeled identity program, we have both* $\mathsf{PublicDecommit}(m, r, C) = 1$ *and* $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F_{\mathcal{I}}, \Delta) = 1$*.*

For any tuple $\{(A_i, m_i, r_i, \mathcal{P}_i)\}_{i\in[N]}$ *such that for* $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$*,* $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$*,* $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$*, and any function* $g \in \mathcal{F}$ *the following holds: There exists a function* $\hat{g} \in \mathcal{F}$*, that is efficiently computable from* g*, such that for* $m^* = g(m_1, \ldots, m_N)$*,* $r^* = \hat{g}(m_1, \ldots, m_N, r_1, \ldots r_N)$*,* $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$*,* $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$*,* $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$*,* $A^* \leftarrow \mathsf{Eval}(g, A_1, \ldots, A_N)$*,*

$$\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F^*, \Delta) = 1.$$

The security notion of FDCs is also based on *well defined programs* (see Def. 3). We introduce an experiment the attacker can run in order to generate a successful forgery and present a definition for unforgeability based on it.

Definition 9 (Forgery). *A forgery is a tuple* $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$ *such that*

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

holds and exactly one of the following conditions is met:

Type 1: *No message was ever committed under the data set identifier* Δ^**, i.e. the list* L_{Δ^*} *of tuples* (τ, m, r) *was not initialized during the security experiment.*

Type 2: $\mathcal{P}^*_{\Delta^*}$ *is well defined with respect to list* L_{Δ^*} *and*

$$C^* \neq \mathsf{PublicCommit}(f(\{m_j\}_{(\tau_j,m_j,r_j)\in L_{\Delta^*}}), \hat{f}(\{(m_j,r_j)\}_{(\tau_j,m_j,r_j)\in L_{\Delta^*}})),$$

where f *is taken from* $\mathcal{P}^*$*, that is,* C^* *is not a commitment to the correct output of the computation.*

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

To define unforgeability, we first describe the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Definition 10 ($\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathbf{FDC}}(\lambda)$ [26]).

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

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

Queries $\mathcal{A}$ *adaptively submits queries for* (Δ,τ,m,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* (Δ,τ,m,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* $(\tau,\cdot,\cdot)$*, that is,* $\mathcal{A}$ *never queried* $(\Delta,\tau,\cdot,\cdot)$*,* $\mathcal{C}$ *calls* $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk},m,r,\Delta,\tau)$*, updates the list* $L_\Delta = L_\Delta \cup (\tau,m,r)$*, and gives* A *to* $\mathcal{A}$*.*
- *If* $(\tau,m,r) \in L_\Delta$*, then* $\mathcal{C}$ *returns the same authenticator* A *as before.*
- *If* L_{Δ^*} *already contains a tuple* (τ,m',r') *for* $(m,r) \neq (m',r')$*,* $\mathcal{C}$ *returns* $\perp$*.*

Forgery $\mathcal{A}$ *outputs a tuple* $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$*.*

$\boldsymbol{EXP}^{UF-CMA}_{\mathcal{A},FDC}(\lambda)$ *outputs* $\mathtt{1}$ *if the tuple returned by* $\mathcal{A}$ *is a forgery (Def. 9).*

Definition 11 (Unforgeability). *An FDC is unforgeable if for any PPT adversary* $\mathcal{A}$*,* $\Pr[\boldsymbol{EXP}^{UF-CMA}_{\mathcal{A},FDC}(\lambda) = 1] = \mathsf{negl}(\lambda)$*.*

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 12 (Succinctness). *An FDC is* succinct *if, for fixed* 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 13 (Amortized Efficiency). *Let* $\mathcal{P}_\Delta = (\mathcal{P},\Delta)$ *be a multi-labeled program,* $m_1,\ldots,m_n \in \mathcal{M}$ *a set of messages,* $r_1,\ldots r_n \in \mathcal{R}$ *a set of randomness,* $f \in \mathcal{F}$ *be an arbitrary function, and* $t(n)$ *be the time required to compute* $f(m_1,\ldots,m_n)$*. An FDC achieves* amortized efficiency *if, for any public parameters* pp *and any* $(\mathsf{sk},\mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$*, any authenticator* A*, any commitment* C*, and function commitment* F*, the time required to compute* $\mathsf{FunctionVerify}(\mathsf{pk},A,C,F,\Delta)$ *is* $t' = o(t(n))$*. Note that* A *and* F *may depend on* f *and* n*.*

Homomorphic Authenticators The high-level idea behind homomorphic authenticators is to provide verifiability for outsourced computations. Inputs to a computation are authenticated by the data owner before sending the authenticated data to a server. The server performs a computation and follows the computation on the authenticators due to their homomorphic property. A (possibly third party) verifier can then use the result of the computation and the derived authenticator to check whether the computation was done correctly. We provide the necessary definitions for our work.

Definition 14 (Homomorphic Authenticator ([14]). *A homomorphic authenticator scheme* HAuth *is a tuple of the following PPT algorithms:*

$\mathsf{HSetup}(1^\lambda)$: *On input a security parameter* λ*, the algorithm returns a set of public parameter* pp*, consisting of (at least) the description of an identifier space* $\mathcal{T}$*, a message space* $\mathcal{M}$*, and a set of admissible functions* $\mathcal{F}$*. The public parameters* pp *are inputs to all following algorithms, even if not explicitly specified.*

$\mathsf{HKeyGen}(\mathsf{pp})$: *On input the public parameters* pp*, the algorithm returns a key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*, where* sk *is the secret key authentication key,* ek *is a public evaluation key, and* vk *is a verification key that can be either private or public.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$: *On input a secret key* sk*, a dataset identifier* Δ*, a label* τ*, and a message* m*, the algorithm returns an authenticator* σ*.*

$\mathsf{HEval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$: *On input a function* $f : \mathcal{M}^n \to \mathcal{M}$*, a set* $\{\sigma_i\}_{i\in[n]}$ *of authenticators, and an evaluation key* ek*, it returns an authenticator* σ*.*

$\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$*, a verification key* vk*, a message* $m \in \mathcal{M}$*, and an authenticator* σ *, the algorithm returns either* $\mathsf{1}$ *(accept), or* $\mathsf{0}$ *(reject).*

If vk is private, we call HAuth a *homomorphic MAC*, while for a public vk we call it a *homomorphic signature*.

We now define properties relevant for the analysis of homomorphic authenticator schemes: authentication correctness, evaluation correctness, succinctness, efficient verification, unforgeability and context hiding.

Correctness naturally comes in two forms. We require both authenticators created directly with a secret signing key as well as those derived by the homomorphic property to verify correctly.

Definition 15 (Authentication Correctness [14]). *A homomorphic authenticator scheme* $(\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$ *satisfies* authentication correctness *if, for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$*, any key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$*, any label* $\tau \in \mathcal{T}$*, any dataset identifier* $\Delta \in \{0,1\}^*$*, any message* $m \in \mathcal{M}$*, and any authenticator* $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$ *we have* $\mathsf{Ver}(\mathcal{I}_{(\tau,\Delta)}, \mathsf{vk}, m, \sigma) = \mathsf{1}$*, where* $\mathcal{I}_{\tau,\Delta}$ *is the multi-labeled identity program.*

Definition 16 (Evaluation Correctness [14]). *A homomorphic authenticator scheme* $(\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$ *satisfies* authentication correctness *if, for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$*, any*

key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$*, for any dataset identifier* $\Delta \in \{0,1\}^*$*, and any set of program/message/authenticator triples* $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$*, such that* $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ *the following holds: Let* $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$*, and* $\sigma^* = \mathsf{HEval}(g, \{\sigma_i\}_{i\in[N]}, \mathsf{ek})$*. Then* $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, m^*, \sigma^*) = 1$*.*

We now consider two properties impacting the practicality of homomorphic authenticator schemes. Succinctness on a high level guarantees that bandwidth requirements for deploying such a scheme are low. Efficient verification allows for low computational effort on behalf of the verifier.

Definition 17 (Succinctness [14]). *A homomorphic authenticator scheme* $(\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$ *is said to be succinct if the size of every authenticator depends only logarithmically on the size of a dataset. More formally, let* $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$*,* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*,* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$*, and* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau_i, m_i)$ *for all* $i \in [n]$*. A homomorphic authenticator is said to be succinct if there exists a fixed polynomial* p *such that* $|\sigma| = p(\lambda, \log n)$*, where* $\sigma = \mathsf{HEval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$*.*

Like Libert and Yung [21], we call a key *concise* if its size is independent of the input size n.

Definition 18 (Efficient Verification [10]). *A homomorphic authenticator scheme for multi-labeled programs allows for* efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$: *Given a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*, and verification key* vk*, this algorithm generates a concise verification key* $\mathsf{vk}_\mathcal{P}$*. This does* not *depend on a dataset identifier* Δ*.*

$\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, m, \sigma, \Delta)$**:** *Given a concise verification key* $\mathsf{vk}_\mathcal{P}$*, a message* m*, an authenticator* σ*, and a dataset* Δ*, it outputs* `1` *or* `0`*.*

The above algorithms are required to satisfy the following two properties:

Correctness: *Let* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ *be an honestly generated key triple and* $(\mathcal{P}_\Delta, m, \sigma)$ *be a tuple. Then, for every* $\mathsf{vk}_\mathcal{P} \leftarrow \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$*, we have* $\Pr[\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, m, \sigma, \Delta) \neq \mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)] = \mathsf{negl}(\lambda)$*.*

Amortized Efficiency: *Let* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ *be a program, let* $m_1, \ldots, m_n \in \mathcal{M}$ *and let* $t(n)$ *be the time required to compute* $f(m_1, \ldots, m_n)$ *with output* m*. Then, for any* $\mathsf{vk}_\mathcal{P} \leftarrow \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$*, and any* $\Delta \in \{0,1\}^*$ *the time required to compute* $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, m, \sigma, \Delta)$ *is* $t' = o(t(n))$*, where* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau_i, m_i)$ *for* $i \in [n]$*, and* $\sigma \leftarrow \mathsf{HEval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$*.*

Here, *efficiency* is used in an amortized sense. There is a function-dependent pre-processing phase, so that verification cost amortizes over multiple datasets.

For the notion of unforgeability of a homomorphic authenticator scheme $(\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$, we define the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. During the experiment, the adversary $\mathcal{A}$ can adaptively query the challenger $\mathcal{C}$ for authenticators on messages of his choice

under labels of his choice. He can also make verification queries. Intuitively, the homomorphic property allows anyone (with access to the evaluation keys) to derive new authenticators. This can be checked by the use of the corresponding program in the verification algorithm. An adversary should however not be able to derive authenticators beyond that.

Definition 19 ($\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ **[14]).**

Setup: *$\mathcal{C}$ runs $\mathsf{HSetup}(1^\lambda)$ to obtain the public parameters pp that are sent to $\mathcal{A}$, runs $(\mathsf{sk},\mathsf{ek},\mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$ and gives ek to $\mathcal{A}$.*

Authentication Queries: *$\mathcal{A}$ can adaptively submit queries of the form (Δ, τ, m) where Δ is a dataset identifier, $\tau \in \mathcal{T}$ is a label, and $m \in \mathcal{M}$ is a message of its choice. $\mathcal{C}$ answers as follows:*

- *If (Δ, τ, m) is the first query for the dataset Δ, $\mathcal{C}$ initializes an empty list $L_\Delta = \varnothing$ and proceeds as follows.*
- *If (Δ, τ, m) is such that $(\tau, \cdot) \in L_\Delta$ (which means that the adversary had already made a query (Δ, τ, m'), then $\mathcal{C}$ ignores the query.*
- *If (Δ, τ, m) is such that $(\tau, m) \notin L_\Delta$, $\mathcal{C}$ computes $\sigma_\tau \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$, returns σ_τ to $\mathcal{A}$ and updates the list $L_\Delta \leftarrow L_\Delta \cup (\tau, m)$.*

Verification Queries: *$\mathcal{A}$ is also given access to a verification oracle. Namely the adversary can submit a query $(\mathcal{P}_\Delta, m, \sigma)$ and $\mathcal{C}$ replies with the output of $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$.*

Forgery: *In the end, $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. The experiment outputs $\mathsf{1}$ if the tuple returned by $\mathcal{A}$ is a forgery as defined below (see Def. 20), and $\mathsf{0}$ otherwise.*

This describes the case of privately verifiable homomorphic authenticators. For homomorphic signatures, vk is given to the adversary.

Definition 20 (Forgery [14]). *Consider a run of $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ where $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is the tuple returned by the adversary in the end of the experiment, with $\mathcal{P}^* = (f^*, \tau^*_1, \ldots, \tau^*_n)$. This is a forgery if $\mathsf{Ver}(\mathcal{P}^*_{\Delta^*}, \mathsf{vk}, m^*, \sigma^*) = 1$, and at least one of the following properties is satisfied:*

Type 1: *The list L_{Δ^*} was not initialized during the security experiment, i.e. no message was ever committed under the dataset identifier Δ^*.*

Type 2: *$\mathcal{P}^*_{\Delta^*}$ is well defined with respect to list L_{Δ^*} and m^* is not the correct output of the computation, i.e. $m^* \neq f^*(m_1, \ldots, m_n)$, where the m_i are taken fromL_{Δ^*}.*

Type 3: *$\mathcal{P}^*_{\Delta^*}$ is not well defined with respect to L_{Δ^*} (see Def. 3).*

Definition 21 (Unforgeability [14]). *A homomorphic authenticator scheme* HAuth *is* unforgeable *if for any PPT adversary $\mathcal{A}$,*

$$\Pr[\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

Definition 22 (Context Hiding [10]). *A homomorphic authenticator scheme for multi-labeled programs is context hiding if there exist two additional PPT procedures $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ and $\mathsf{HideVer}(\mathsf{vk}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ such that:*

Correctness: *For any* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and any tuple* $(\mathcal{P}_\Delta, m, \sigma)$, *such that* $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma) = 1$, *and* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$, *it holds that* $\mathsf{HideVer}(\mathsf{vk}, \mathcal{P}_\Delta, m, \tilde{\sigma}) = 1$.

Unforgeability: *The scheme is unforgeable (Def. 21) when replacing the algorithm* Ver *with* HideVer *in the security experiment.*

Context Hiding Security: *There exists a simulator* Sim *such that, for any fixed (worst-case) choice of* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, *any multi-labeled program* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$, *messages* $m_1, \ldots, m_n$, *and distinguisher* $\mathcal{D}$ *there exists a function* $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ *such that* $|\Pr[\mathcal{D}(\mathcal{I}, \mathsf{Hide}(\mathsf{vk}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathcal{I}, \mathsf{Sim}(\mathsf{sk}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda)$, *where* $\mathcal{I} = (\mathsf{sk}, (m_1, \ldots, m_n))$, $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau_i, m_i)$, $m \leftarrow f(m_1, \ldots, m_n)$, $\sigma \leftarrow \mathsf{Eval}(f, \{\sigma_i\}_{i \in [n]})$,*and the probabilities are taken over the randomness of* Auth, Hide *and* Sim.

If $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, *we call the homomorphic authenticator scheme* statistically context hiding. *If* $\epsilon(\lambda) = 0$, *we call it* perfectly context hiding.

Pairings and Structure-Preserving Signatures We formalize the definitions and assumptions related to pairings. These will be the main building block for our construction in Sec. 5. Structure-preserving signatures are also defined.

Definition 23 (Double Pairing Assumption in $\mathbb{G}_2$ (DBP_2, e.g. [1])). *Let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ *and* $R, Z \xleftarrow{\$} \mathbb{G}_2$. *Any PPT adversary* $\mathcal{A}$ *can produce* $(G_R, G_Z) \in \mathbb{G}_1^2 \backslash \{(1,1)\}$ *such that* $1 = e(G_R, R) \cdot e(G_Z, Z)$ *only with a probability negligible in* λ.

DBP_2 implies the DDH assumption in $\mathbb{G}_2$, and the reduction is tight [3].

Definition 24 (Structure-Preserving Signature [18]).
A structure-preserving signature scheme is a triple of PPT algorithms $\mathsf{SPS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$*:*

The probabilistic key generation algorithm $\mathsf{Gen}(1^\lambda)$ *returns the secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$, *where* $\mathsf{pk} \in \mathbb{G}^{n_{\mathsf{pk}}}$ *for some* $n_{\mathsf{pk}} \in \mathsf{poly}(\lambda)$. *We assume that* pk *implicitly defines a message space* $\mathcal{M} = \mathbb{G}^n$ *for some* $n \in \mathsf{poly}(\lambda)$.

The probabilistic signing algorithm $\mathsf{Sign}(\mathsf{sk}, M)$ *returns a signature* $\sigma \in \mathbb{G}^{n_\sigma}$ *for some* $n_\sigma \in \mathsf{poly}(\lambda)$.

The deterministic verification algorithm $\mathsf{Verify}(\mathsf{pk}, M, \sigma)$ *only consists of pairing product equations and returns* `1` *or* `0`.

(Perfect correctness.) for all $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ *and all messages* $M \in \mathcal{M}$ *and all* $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, M)$, *we have* $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$.

Libert et al. [19] adapted this to the scenario of homomorphic signatures. There, a signature scheme is structure-preserving if messages, signature components and public keys are elements of the group bgp.

3 Homomorphic Authenticators from FDCs

In this section, we begin to investigate the relation between FDCs and homomorphic authenticators. In particular we show how any correct FDC can be transformed into a homomorphic signature scheme and show how it inherits properties from the underlying FDC.

We begin by describing a transformation that constructs the algorithms of a homomorphic authenticator scheme from the algorithms of an FDC.

Theorem 1. *Every correct FDC scheme can be transformed into a homomorphic authenticator scheme.*

Proof. We describe a transformation Φ that on input an FDC scheme $\mathsf{FDC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{PublicCommit}, \mathsf{PrivateCommit}, \mathsf{FunctionCommit}, \mathsf{Eval}, \mathsf{FunctionVerify}, \mathsf{PublicDecommit})$ outputs a homomorphic authenticator scheme $\mathsf{HAuth} = (\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$.

$\mathsf{HSetup}(1^\lambda)$: On input a security parameter λ, it runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. It outputs the public parameters pp.

$\mathsf{HKeyGen}(\mathsf{pp})$: On input the public parameters pp it runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. It outputs the secret key sk, the evaluation key $\mathsf{ek} = 0$ as well as the verification key $\mathsf{vk} = \mathsf{pk}$.

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$: On input a secret key sk, a dataset identifier Δ, an input identifier τ, and a message m, it chooses randomness $r \xleftarrow{\$} \mathcal{R}$ uniformly at random. It runs $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$. It sets $M = m$ and outputs the authenticator $\sigma = (M, r, A)$.

$\mathsf{HEval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$: On input an function $f : \mathcal{M}^n \to \mathcal{M}$, a set $\{\sigma_i\}_{i\in[n]}$ of authenticators, and an evaluation key ek (in our construction, no evaluation key is needed), the algorithm parses $\sigma_i = (M_i, r_i, A_i)$. It runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$, and sets $M^* = f(M_1, \ldots, M_n)$ as well as $r^* = \hat{f}(m_1, \ldots, m_n, r_1, \ldots, r_n)$, where $\hat{f} \in \mathcal{F}$ can be derived from f since the FDC scheme is correct in the sense of Def. 8. It sets $\sigma = (M^*, r^*, A^*)$ and outputs σ.

$\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$: On input a multi-labeled program $\mathcal{P}_\Delta$, a verification key vk, a message $m \in \mathcal{M}$, and an authenticator σ , the algorithm parses $\sigma = (M, r, A)$ and $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$. It checks if $m = M$. If not it outputs 0, else it runs $C \leftarrow \mathsf{PublicCommit}(m, r)$ as well as $F \leftarrow \mathsf{FunctionCommit}(\mathsf{vk}, \mathcal{P})$. It runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{vk}, A, C, F, \Delta)$. It outputs b.

For homomorphic authenticators, correctness comes in two flavors. Both authenticators created directly with a secret signing key as well as those derived by the homomorphic property should verify correctly. Both properties are derived from the FDC's correctness.

Lemma 1. *If* FDC *satisfies correctness (see Def. 8), and* $\mathsf{HAuth} = \Phi(\mathsf{FDC})$*, then* HAuth *achieves authentication correctness (see Def. 15) and evaluation correctness (see Def. 16).*

Proof. Let $\tau \in \mathcal{T}$ be an arbitrary label and $\Delta \in \{0,1\}^*$ be an arbitrary dataset identifier. We consider the multi-labeled identity program $\mathcal{P}_\Delta = \mathcal{I}_{\tau,\Delta}$. Let $m \in \mathcal{M}$ be an arbitrary message, and $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. By construction, we know that $\sigma = (M, r, A)$,with $M = m$, where $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$. By correctness of the FDC, we know that $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta) = 1$ for $C \leftarrow \mathsf{PublicCommit}(m, r)$ and $F \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$. Now we have $\mathsf{Ver}(\mathcal{I}_{\tau,\Delta}, \mathsf{vk}, m, \sigma) = 1$, and HAuth achieves authentication correctness.

Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier and $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$ any set of program/message/authenticator triples such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for all $i \in [N]$. Let $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{HEval}(g, \{\sigma_i\}_{i\in[N]}, \mathsf{ek})$. We parse $\sigma_i = (M_i, r_i, A_i)$ and $\sigma^* = (M^*, r^*, A^*)$. We set $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$ and $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$. By construction, we know that $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$, as well as $M^* = f(M_1, \ldots, M_N)$, $r^* = \hat{g}(m_1, \ldots, m_N, r_1, \ldots, r_N)$, where $\hat{g} \in \mathcal{F}$ can be derived from g since the FDC scheme is correct in the sense of Def. 8. We set $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$ and $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$. By assumption, $m_i = M_i$ hence $M^* = m^*$. Since FDC satisfies correctness, $\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F^*, \Delta) = 1$ and thus $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, m^*, \sigma^*) = 1$.

Next, we investigate the relation between the unforgeability notions of FDCs and homomorphic authenticators. We can show that an FDC that is both unforgeable and binding can be transformed into an unforgeable homomorphic authenticator.

Lemma 2. *If* FDC *is unforgeable (see Def. 11) and binding (see Def. 6), and* $\mathsf{HAuth} = \Phi(\mathsf{FDC})$*, then* HAuth *is unforgeable (see Def. 21).*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ (see Def. 19), we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to win the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$ (see Def. 10).

Setup $\mathcal{S}$ receives pp from the challenger of the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$. It gives pp to the adversary $\mathcal{A}$.

Key Generation Simulator $\mathcal{S}$ receives pk from the challenger of the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$. It sets $\mathsf{ek} = 0$, $\mathsf{vk} = \mathsf{pk}$, and outputs $(\mathsf{ek}, \mathsf{vk})$ to $\mathcal{A}$.

Queries When $\mathcal{A}$ asks queries (Δ, τ, m) $\mathcal{S}$ chooses $r \in \mathcal{R}$ uniformly at random and queries (Δ, τ, m, r) to receive an authenticator A. It sends the authenticator $\sigma = (m, r, A)$ to $\mathcal{A}$. Note that σ is perfectly indistinguishable from a response to a query (Δ, τ, m) during $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$.

Forgery $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. $\mathcal{S}$ parses $\sigma^* = (M^*, r^*, A^*)$. It then computes the correct results $\hat{m} = f^*(m_1, \ldots, m_n), \hat{r} = \hat{f}^*(m_1, \ldots, m_n, r_1, \ldots, r_n)$. It computes $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$ and checks whether $C^* = \mathsf{PublicCommit}(\hat{m}, \hat{r})$. If not, it returns $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$

We either find a collision (m^*, r^*) and $(\hat{m}, \hat{r})$ for the bindingness, or produce a forgery since a type 1, 2, 3 forgery in experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ corresponds exactly to a forgery in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$. Thus
$\Pr[\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1] = \Pr[\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda) = 1] +$
$\Pr[\mathbf{EXP}^{Bind}_{\mathcal{A},\mathrm{Com}}(\lambda) = 1]$.

We now analyze this transformation with respect to its efficiency. A trivial construction of a homomorphic authenticator scheme is to (conventionally) sign every input, and during HEval to simply concatenate all authenticators along with the corresponding values. Verification then consists of checking every input value, and then redoing the computation. This naive solution is obviously undesirable in terms of bandwidth, efficiency and fails to provide privacy guarantees.

Succinctness guarantees that a homomorphically derived authenticator is still small, thus keeping bandwidth requirements low.

We show that the homomorphic authenticators derived by our transformation inherits this property from the underlying FDC.

Lemma 3. *If* FDC *is succinct (see Def. 12) and* $\mathsf{HAuth} = \Phi(\mathsf{FDC})$*, then* HAuth *is succinct (see Def. 17).*

Proof. By assumption, the size of the output of $\mathsf{PrivateCommit}$ and Eval depends at most logarithmically on n. By construction, the size of authenticators thus depends at most logarithmically on n.

4 FDCs from Homomorphic Authenticators

In this section, we discuss how to construct an FDC from (homomorphic) commitment schemes and structure-preserving homomorphic signatures schemes over the commitment space. We show how the properties of the resulting FDC depend on the underlying homomorphic signature scheme and commitment scheme.

Assume the homomorphic authenticator scheme $\mathsf{HAuth} = (\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$ is structure-preserving over some structure $\mathcal{X}$. Let Com be a homomorphic commitment scheme $\mathsf{Com} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit}, \mathsf{CEval})$ with message space $\mathcal{M}$ and commitment space $\mathcal{C} \subset \mathcal{X}$. We also assume the existence of an ordinary commitment scheme $\mathsf{Com}' = (\mathsf{CSetup}', \mathsf{Commit}', \mathsf{Decommit}')$ with message space $\mathcal{F} \times \mathcal{T}^n$, so labeled programs are admissible inputs. One can always split up Ver into $(\mathsf{VerPrep}, \mathsf{EffVer})$ as follows.

$\mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$: On input a labeled program $\mathcal{P}$ and a verification key vk, the algorithm sets $\mathsf{vk}_{\mathcal{P}} = (\mathcal{P}, \mathsf{vk})$. It returns $\mathsf{vk}_{\mathcal{P}}$.

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, C, \sigma, \Delta)$**:** On input a concise verification key $\mathsf{vk}_{\mathcal{P}}$, a message C, an authenticator σ, and a dataset identifier $\Delta \in \{0,1\}^*$, the algorithm parses $\mathsf{vk}_{\mathcal{P}} = (\mathcal{P}, \mathsf{vk})$. It runs $b \leftarrow \mathsf{Ver}(\mathcal{P}, \mathsf{vk}, C, \sigma, \Delta)$ and returns b.

We now show how to construct an FDC.

$\mathsf{Setup}(1^\lambda)$ takes the security parameter λ as input. It runs $\mathsf{CK} \leftarrow \mathsf{CSetup}(1^\lambda)$, $\mathsf{CK}' \leftarrow \mathsf{CSetup}'(1^\lambda)$ as well as $\mathsf{pp}' \leftarrow \mathsf{HSetup}(1^\lambda)$. It sets $\mathsf{pp} = (\mathsf{CK}, \mathsf{CK}', \mathsf{pp}')$ and outputs pp. We implicitly assume that every algorithm uses these public parameters pp, leaving them out of the notation.

$\mathsf{KeyGen}(\mathsf{pp})$ takes the public parameters pp and runs $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$. It sets $\mathsf{sk} = (\mathsf{sk}', \mathsf{ek})$, $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ and outputs the key pair $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{PublicCommit}(m, r)$ takes as input a message m and randomness r and runs $(C, d) \leftarrow \mathsf{Commit}(m, r)$. It outputs the commitment C.

$\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$ takes as input the secret key sk, a message m, randomness r, an identifier τ and a dataset identifier Δ. It runs $(C, d) \leftarrow \mathsf{Commit}(m, r)$, $A' \leftarrow \mathsf{Auth}(\mathsf{sk}, \tau, \Delta, C)$ and outputs $A = (A', \mathsf{ek})$.

$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$ takes as input the public key pk and a labeled program $\mathcal{P}$. It parses $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ and runs $\mathsf{vk}_\mathcal{P} \leftarrow \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$. It chooses randomness $r_\mathcal{P} \xleftarrow{\$} \mathcal{R}$ uniformly at random and runs $(C_\mathcal{P}, d_\mathcal{P}) \leftarrow \mathsf{Commit}'(\mathcal{P}, r_\mathcal{P})$. It outputs the function commitment $F = (\mathsf{vk}_\mathcal{P}, C_\mathcal{P})$.

$\mathsf{Eval}(f, A_1, \ldots A_n)$ takes as input a function f and a set of authenticators $A_1, \ldots, A_n$. It parses $A_i = (A_i', \mathsf{ek}_i)$ for all $i \in [n]$, and runs $\hat{A} \leftarrow \mathsf{HEval}(f, \{A_i'\}_{i\in[n]}, \mathsf{ek}_1)$. It outputs $A^* = (\hat{A}, \mathsf{ek}_1)$.

$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$ takes as input a public key pk, an FDC containing an authenticator A and a commitment C, a function commitment F as well as a dataset identifier Δ. It parses $F = (\mathsf{vk}_\mathcal{P}, C_\mathcal{P})$, and $A = (A', \mathsf{ek})$. It runs $b \leftarrow \mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, C, A', \Delta)$ and outputs b.

$\mathsf{PublicDecommit}(m, r, C)$ takes as input message m, randomness r, and commitment C. It runs $(C, d) \leftarrow \mathsf{Commit}(m, r)$ as well as $b \leftarrow \mathsf{Decommit}(m, d, C)$ and outputs b.

We first look at the commitment properties — hiding and binding. In our transformation, these are inherited from the underlying commitment schemes.

Lemma 4. *The construction* FDC *is binding in the sense of Def. 6 if* Com *and* Com′ *used in the construction are binding commitment schemes.*

Proof. Obviously, if Com is binding then PublicCommit is binding. We parse a function commitment as $F = (\mathsf{vk}_\mathcal{P}, C_\mathcal{P})$. Note that $C_\mathcal{P}$ is by assumption a binding commitment, thus FunctionCommit is also binding.

The hiding property of FDCs (Def. 7) is different from the context hiding property of homomorphic authenticators [10]. Context hiding guarantees that authenticators to the *output* of a computation do not leak information about the *inputs* to the computation. By contrast, the hiding property of FDCs guarantees that even authenticators to the inputs do not leak information about inputs to the computation. In [26], this property was used to combine an FDC with secret sharing to construct an efficient verifiable multi-party computation scheme. This privacy gain is one of the major benefits of FDCs over homomorphic authenticators when sensitive data are used as computation inputs.

Lemma 5. *If* Com *is (unconditionally) hiding, then* FDC *is unconditionally hiding in the sense of Def. 7.*

Proof. If Com is (unconditionally) hiding, then the probabilistic distributions over the sets $\{\mathsf{Commit}(m,r) \mid r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{Commit}(m',r') \mid r' \xleftarrow{\$} \mathcal{R}\}$ are perfectly indistinguishable for all $m, m' \in \mathcal{M}$. This is independent of any $\tau \in \mathcal{T}$ and any $\Delta \in \{0,1\}^*$. Hence the probabilistic distributions over sets $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C) \mid C \leftarrow \mathsf{Commit}(m,r), r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C') \mid C' \leftarrow \mathsf{Commit}(m',r'), r' \xleftarrow{\$} \mathcal{R}\}$ are perfectly indistinguishable for all $m, m' \in \mathcal{M}, \tau \in \mathcal{T}, \Delta \in \{0,1\}^*$. Since $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C) \mid C \leftarrow \mathsf{Commit}(m,r), r \xleftarrow{\$} \mathcal{R}\} = \{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\}$ for all $m \in \mathcal{M}, \tau \in \mathcal{T}, \Delta \in \{0,1\}^*$ the probabilistic distributions over $\{\mathsf{PrivateCommit}(\mathsf{sk}, m,\ r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{PrivateCommit}(\mathsf{sk}, m', r', \Delta, \tau) \mid r' \xleftarrow{\$} \mathcal{R}\}$ are also (perfectly) indistinguishable. The $\mathsf{PublicCommit}$ case is trivial.

Next, we investigate the homomorphic property of such an FDC. We can show that if the homomorphic authenticator scheme HAuth satisfies both correctness properties — authentication and evaluation, and furthermore supports efficient verification, then the transformed FDC is also correct.

Lemma 6. *If* HAuth *achieves authentication (see [10]), evaluation correctness (see [10]), and efficient verification (see [10]), then* FDC *is correct in the sense of Def. 8 with overwhelming probability.*

Proof. Let λ be any security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and let $\Delta \in \{0,1\}^*$ be an arbitrary dataset identifier. Let $m \in \mathcal{M}$ be an arbitrary message and $r \in \mathcal{R}$ arbitrary randomness. We set $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$, $C \leftarrow \mathsf{PublicCommit}(m,r)$, $F_{\mathcal{I}} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$, where $\mathcal{I}_\tau$ is the labeled identity program. Then we have $A = \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C)$. By the authentication correctness of HAuth, we know that $\mathsf{Ver}(\mathcal{I}_{\tau,\Delta}, \mathsf{vk}, C, \sigma) = 1$. Since HAuth achieves efficient verification, $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{I}_\tau}, C, \sigma, \Delta) = 1$ with overwhelming probability. By construction, $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F_{\mathcal{I}}, \Delta) = 1$.

Let $\{m_i, \sigma_i, \mathcal{P}_i)\}_{i \in [N]}$ be any set of tuples (parsed as $\sigma_i = (r_i, A_i)$) such that for $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$, $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$, $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$. This implies $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}_i}, C_i, \sigma_i, \Delta) = 1$, thus $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, C_i, \sigma_i) = 1$ with overwhelming probability. Then let $m^* = g(m_1, \ldots, m_N)$, $r^* = \hat{g}(m_1, \ldots, m_N, r_1, \ldots r_N)$, $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$, $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_N)$, and $\sigma^* = (r^*, A^*)$. From the homomorphic property of Com, we have $C^* = \mathsf{CEval}(g, C_1, \ldots, C_N)$. By the evaluation correctness of HAuth we have $\mathsf{Ver}(\mathcal{P}^*, \mathsf{vk}, C^*, \sigma^*) = 1$. Thus $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}^*}, C^*, \sigma^*, \Delta) = 1$ with overwhelming probability, due to the correctness of efficient verification. By construction, $\mathsf{FunctionVerify}(\mathsf{pk},\ A^*, C^*, F^*,\ \Delta) = 1$.

We now look at the essential security property of an FDC — unforgeability. We show how an adversary that can break the security experiment for FDCs can be used to break the security experiment for homomorphic authenticators. A simulator can forward the queries used by the adversary in the FDC experiment as queries in the homomorphic authenticator experiment, and use the resulting forgery in the one experiment to compute a forgery in the other.

Lemma 7. *If* HAuth *is secure (see [10]), then* FDC *is unforgeable (Def. 11).*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$, we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to win the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ (see [10]).

Setup $\mathcal{S}$ gets pp' from the challenger of the experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. It runs $\mathsf{CK} \leftarrow \mathsf{CSetup}(1^\lambda)$, $\mathsf{CK}' \leftarrow \mathsf{CSetup}'(1^\lambda)$. It sets $\mathsf{pp} = (\mathsf{CK}, \mathsf{CK}', \mathsf{pp}')$ and outputs pp to the adversary $\mathcal{A}$.

Key Generation $\mathcal{S}$ receives $(\mathsf{ek}, \mathsf{vk})$ from the challenger of the experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. It sets $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ and outputs pk to $\mathcal{A}$.

Queries When $\mathcal{A}$ ask queries (Δ, τ, m, r), $\mathcal{S}$ computes $(C, d) \leftarrow \mathsf{Commit}(m, r)$ and queries (Δ, τ, C) to receive an authenticator σ. It sets $A = \sigma$ and replies to the query with the private commitment A. This is the exact same reply to a query in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$.

Forgery The adversary $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, r^*, A^*)$. $\mathcal{S}$ computes $(C^*, d^*) \leftarrow \mathsf{Commit}(m^*, r^*)$ and outputs $(\mathcal{P}^*_{\Delta^*}, C^*, A^*)$.

A type 1, 2, 3 forgery in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$ corresponds to a forgery in experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. Thus $\mathcal{S}$ produces a forgery with the same probability as $\mathcal{A}$.

We now analyze an FDC obtained by our transformation with respect to its efficiency properties. On the one hand we have succinctness, which guarantees that authenticators are short, so bandwidth requirements are low. On the other hand, we show how the FDC inherits amortized efficiency, i.e. efficient verification after a one-time preprocessing from the efficient verification of the underlying homomorphic authenticator scheme.

Lemma 8. *If* HAuth *is succinct (see [10]), then* FDC *is succinct (Def. 12).*

Proof. By assumption, HAuth produces authenticators whose size depends at most logarithmically on the data set size n. By construction, the output size of PrivateCommit and Eval thus depends at most logarithmically on n.

Lemma 9. *If* HAuth *is efficiently verifiable (see [10]), then* FDC *has amortized efficiency in the sense of Def. 13.*

Proof. Let $t(n)$ be the runtime of $f(m_1, \ldots, m_n)$. FunctionVerify parses a function commitment $F = (\mathsf{vk}_\mathcal{P}, C_\mathcal{P})$ and runs $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, C, A, \Delta)$. By assumption, the runtime of EffVer is $o(t(n))$. Thus the runtime of FunctionVerify is also $o(t(n))$.

4.1 A New Structure-Preserving Homomorphic Signature Scheme

We now consider the special case of a single-dataset, structure-preserving homomorphic signature scheme. Obviously, our transformation also works for such a scheme. This can easily be seen by interpreting the underlying authenticator scheme as one where all algorithms are constant over all inputs $\Delta \in \{0,1\}^*$. This

leads to a single dataset FDC. It is an immediate corollary of [14, Theorem 2] that a single dataset FDC can be transformed into a multi dataset FDC. On a high level, this transformation uses a keyed pseudorandom function that on input a dataset $\Delta \in \{0,1\}^*$ produces the keys $(\mathsf{sk}_\Delta, \mathsf{ek}_\Delta, \mathsf{vk}_\Delta)$ and then uses a conventional UF-CMA secure signature scheme to bind the dataset to the public keys by signing $\Delta \mid \mathsf{vk}_\Delta$. For details, see Fiore et al. [14]. In Sec. 5, we present such a single dataset structure-preserving homomorphic signature scheme.

5 A New Single-Dataset, Structure-Preserving Linearly Homomorphic Signature Scheme

We now describe a novel structure-preserving linearly homomorphic signature scheme $\mathsf{SPHAuth}$ for a single dataset. As we discussed in Sec. 4, this can be extended to a scheme for multiple datasets by standard methods. Our structure-preserving linearly homomorphic signature scheme is the first structure-preserving homomorphic signature scheme to achieve efficient verification, and the first context hiding. It achieves the latter even in an information-theoretic sense.

$\mathsf{HSetup}(1^\lambda)$: On input a security parameter λ, this algorithm chooses the parameter $n \in \mathbb{Z}$, a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ and the tag space $\mathcal{T} = [n]$. Additionally, it fixes a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (n, F, \mathsf{bgp})$.

$\mathsf{HKeyGen}(\mathsf{pp})$: On input the public parameters pp, the algorithm chooses chooses $x_1, \ldots, x_n, y, z \in \mathbb{Z}_p$ uniformly at random. It sets $h_i = g_t^{x_i}$ for all $i \in [n]$, as well as $Y = g_2^y$, $Z = g_2^z$. Additionally the algorithm chooses a random seed $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function F. It sets $\mathsf{sk} = (K, x_1, \ldots, x_n, y, z)$, $\mathsf{ek} = 0$, $\mathsf{vk} = (h_1, \ldots, h_n, Y, Z)$ and outputs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$.

$\mathsf{Auth}(\mathsf{sk}, \tau, M)$: On input a secret key sk, an input identifier τ, and a message $M \in \mathbb{G}_1$, the algorithm takes x, y from sk. It computes $s = F_K(\tau)$ and sets $S = g_1^s$, $\Lambda = \left(g_1^{x_\tau + s} \cdot M^y\right)^{\frac{1}{z}}$. It outputs $\sigma = (\Lambda, S)$.

$\mathsf{HEval}(f, \{\sigma_i,\}_{i\in[n]}, 0)$: On input an function $f : \mathcal{M}^n \to \mathcal{M}$ and a set $\{\sigma_i\}_{i\in[n]}$ of authenticators, and an empty evaluation key, the algorithm parses $f = (f_1, \ldots, f_n)$ as a coefficient vector. It parses each σ_i as (Λ_i, S_i) and sets $\Lambda = \prod_{i=1}^n \Lambda_i^{f_i}$ and $S = \prod_{i=1}^n S_i^{f_i}$. It returns $\sigma = (\Lambda, S)$.

$\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, M, \sigma)$: On input a labeled program $\mathcal{P}$, a verification key vk, a message $M \in \mathbb{G}_1$, and an authenticator σ, the algorithm parses $\sigma = (\Lambda, S)$. It checks whether $e(\Lambda, Z) = e(M, Y) \cdot \prod_{i=1}^n h_{\tau_i}^{f_i} \cdot e(S, g_2)$. If the equation holds, it outputs $\mathsf{1}$, otherwise it outputs $\mathsf{0}$.

This scheme $\mathsf{SPHAuth}$ is structure-preserving, as messages are taken from $\mathbb{G}_1$, public keys lie in $\mathbb{G}_2$ or $\mathbb{G}_T$ and authenticators lie in $\mathbb{G}_1$. An obvious requirement for this structure-preserving homomorphic signature scheme is to be correct. For homomorphic authenticators, two different notions of correctness are considered. One ensures that freshly generated authenticators obtained by running Auth

verify correctly. The other correctness property ensures that any derived signature, obtained by running HEval verifies correctly w.r.t the correct labeled program $\mathcal{P}$.

Lemma 10. $\mathsf{SPHAuth}$ *satisfies authentication correctness (see [10]).*

Proof. For all public parameters $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$, and key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$, we have $\mathsf{sk} = (x_1, \ldots, x_n, y, z)$, $\mathsf{ek} = 0$, $\mathsf{vk} = (h_1, \ldots, h_n, Y, Z)$. For any input identifier $\tau \in \mathcal{T}$, and any message $M \in \mathbb{G}_1$ we have $\sigma = (\Lambda, S)$ with $\Lambda = \left(g_1^{x_\tau + s} \cdot M^y\right)^{\frac{1}{z}}$ and $S = g_1^s$. We consider $\mathcal{P} = \mathcal{I}_\tau$ the identity program for label τ. During the computation of $\mathsf{Ver}(\mathcal{I}_\tau, \mathsf{vk}, M, \sigma)$, $e(\Lambda, Z) = e\left(\left(g_1^{x_\tau + s} \cdot M^y\right)^{\frac{1}{z}}, g_2^z\right) = e\left(g_1^{x_\tau + s} \cdot M^y, g_2\right) = e(g_1^{x_\tau}, g_2) \cdot e(g_1^s, g_2) \cdot e(M, g_2^y) = e(M, Y) \cdot h_\tau \cdot e(S, g_2)$. Thus $\mathsf{SPHAuth}$ satisfies authentication correctness with probability 1.

Lemma 11. $\mathsf{SPHAuth}$ *satisfies evaluation correctness (see [10]).*

Proof. We fix the public parameters $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$, key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$, a function $g : \mathbb{G}_1^N \to \mathbb{G}_1$, given by its coefficient vector $(g_1, \ldots, g_N)$ and any set of program/message/authenticator triples $\{(\mathcal{P}_i, M_i, \sigma_i)\}_{i \in [N]}$ such that $\mathsf{Ver}(\mathcal{P}_i, \mathsf{vk}, M_i, \sigma_i) = 1$ for all $i \in [N]$. So in particular, for $\sigma_i = (\Lambda_i, S_i)$, $e(\Lambda_i, Z) = e(M_i, Y) \cdot h_{\mathcal{P}_i} \cdot e(S_i, g_2)$ For readability, we write $h_{\mathcal{P}_i} = \prod_{k=1}^n h_{\tau_{i,k}}^{f_{i,k}}$ with $\mathcal{P}_i = (f_{i,1}, \ldots, f_{i,n}, \tau_{i,1}, \ldots, \tau_{i,n})$. Let $M^* = \prod_{i=1}^N M_i^{g_i}$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{HEval}(g, \{\sigma_i\}_{i \in [N]}, 0)$ (we have an empty evaluation key). We parse $\sigma^* = (\Lambda^*, S^*)$. Then $e(\Lambda^*, Z) = e\left(\prod_{i=1}^N \Lambda_i^{g_i}, Z\right) = \prod_{i=1}^N e(\Lambda_i, Z)^{g_i} = \prod_{i=1}^N (e(M_i, Y) \cdot h_{\mathcal{P}_i} \cdot e(S_i, g_2))^{g_i} = \prod_{i=1}^N e(M_i, Y)^{g_i} \cdot \prod_{i=1}^N h_{\mathcal{P}_i}^{g_i} \cdot \prod_{i=1}^N e(S_i, g_2)^{g_i} = e(\prod_{i=1}^N M_i^{g_i}, Y) \cdot h_{\mathcal{P}^*} \cdot e\left(\prod_{i=1}^N S_i^{g_i}, g_2\right) = e(M^*, Y) \cdot h_{\mathcal{P}^*} \cdot e(S^*, g_2)$. Thus $\mathsf{SPHAuth}$ satisfies evaluation correctness with probability 1.

Next, we show that $\mathsf{SPHAuth}$ is efficient with respect to both bandwidth (succinctness) and verification time (efficient verification).

Lemma 12. $\mathsf{SPHAuth}$ *is succinct (see [10]).*

Proof. An authenticator consist of 2 $\mathbb{G}_1$ elements and is thus independent of n.

Lemma 13. $\mathsf{SPHAuth}$ *allows for efficient verification (see [10]).*

Proof. We describe the algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$:

$\mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$: On input the labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, with f given by its coefficient vector $(f_1, \ldots, f_n)$, the algorithm takes Y, Z from vk. For label τ_i it takes h_{τ_i} from vk. It computes $h_{\mathcal{P}} \leftarrow \prod_{i=1}^n h_{\tau_i}^{f_i}$ and outputs $\mathsf{vk}_{\mathcal{P}} \leftarrow (h_{\mathcal{P}}, Y, Z)$. This is independent of the input size n.

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, M, \sigma)$**:** On input a concise verification key $\mathsf{vk}_{\mathcal{P}}$, a message M, and an authenticator σ, the algorithm parses $\sigma = (\Lambda, S)$. It checks whether the following equation holds: $e(\Lambda, Z) = h_{\mathcal{P}} \cdot e(M, Y) \cdot e(S, g_2)$. If it does, it outputs $\mathtt{1}$, otherwise it outputs $\mathtt{0}$.

This obviously satisfies correctness. We can see that the runtime of EffVer is $\mathcal{O}(1)$, and is independent of the input size n. Thus, for large n, this scheme allows for efficient verification.

We now prove the unforgeability of our scheme. To this end, we first describe a sequence of games, allowing us to argue about different variants of forgeries. We show how any noticeable difference between the first two games leads to a distinguisher against the pseudorandomness of F. We then show how both a noticeable difference between the latter two games, as well as a forgery in the final game lead to a solver of the double pairing assumption.

Theorem 2. *If F is a pseudorandom function and the double pairing assumption holds in $\mathbb{G}_2$ (see Def. 23), then* $\mathsf{SPHAuth}$ *is unforgeable.*

Proof. We now provide the security reduction for the unforgeability of our scheme in the standard model. We define a series of games with the adversary $\mathcal{A}$ and we show that $\mathcal{A}$ wins, i.e. any game outputs $\mathtt{1}$, only with negligible probability. Following the notation of [10], we write $G_i(\mathcal{A})$ to denote that a run of game i with $\mathcal{A}$ returns $\mathtt{1}$. We use flag values bad_i, initially set to false. If, at the end of each game, any of these previously defined flags is set to true, the game simply outputs $\mathtt{0}$. Let $\mathsf{Bad_i}$ denote the event that $\mathsf{bad_i}$ is set to true during game i.

Game 1 is defined as the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda)$ between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

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

Game 3 is defined as Game 2, except for the following change. The challenger runs an additional check. It computes $\hat{\sigma} \leftarrow \mathsf{HEval}(f, \{\sigma_i,\}_{i\in[n]}, 0)$ over the σ_i given to the adversary $\mathcal{A}$ in answer to his queries. It parses $\hat{\sigma} = (\hat{\Lambda}, \hat{S})$. It parses the forgery $\sigma^* = (\Lambda^*, S^*)$. If $\hat{S} = S^*$ it sets $\mathsf{bad}_3 = \mathsf{true}$.

First, we show that for every PPT adversary $\mathcal{A}$ running Game 2, there exists a PPT distinguisher $\mathcal{D}$ such that $|\Pr[G_2(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{F,\mathsf{D}}(\lambda)$.

Assume we have a noticeable difference $|\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \geq \epsilon$. Since the only difference between these games is the replacement of the pseudorandom function F by the random function $\mathcal{R}$, this immediately leads to a distinguisher $\mathcal{D}$ that achieves an advantage of ϵ against the pseudorandomness of F.

Now, we show that $\Pr[\mathsf{Bad}_3] = \mathsf{negl}(\lambda)$. The simulator $\mathcal{S}$ gets as input $\mathsf{bgp}, Z \in \mathbb{G}_2$. It simulates Game 3.

Setup Simulator $\mathcal{S}$ chooses the parameter $n \in \mathbb{Z}$ and the tag space $\mathcal{T} = [n]$. It outputs the public parameters $\mathsf{pp} = (n, \mathsf{bgp})$.

KeyGen Simulator $\mathcal{S}$ chooses $a_i, b_i \in \mathbb{Z}_p$ uniformly at random for all $i = 1, \ldots, n$. It sets $h_i = g_t^{a_i} \cdot e\left(g_1, g_2\right)^{b_i}$. It chooses $y \in \mathbb{Z}_p$ uniformly at random and sets $Y = g_2^y$. It gives the verification key $\mathsf{vk} = (h_1, \ldots, h_n, Y, Z)$ to $\mathcal{A}$.

Queries When queried for (M, τ), simulator $\mathcal{S}$ sets $\Lambda = g_1^{b_\tau}$ as well as $S = g_1^{-a_\tau} \cdot M^{-y}$. Since a_τ, b_τ were chosen uniformly at random, the signature is correctly distributed.

Forgery Let $(\mathcal{P}^*, M^*, \sigma^*)$ with $\sigma^* = (\Lambda^*, S^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 3 to compute $\hat{\Lambda}, \hat{S}, \hat{M}$. Since σ^* is a successful forgery, we furthermore know that both $e(\Lambda^*, Z) = e(M^*, Y) \cdot \prod_{i=1}^{n} h_{\tau_i}^{f_i} \cdot e(S^*, g_2)$ and $e\left(\hat{\Lambda}, Z\right) = e\left(\hat{M}, Y\right) \cdot \prod_{i=1}^{n} h_{\tau_i}^{f_i} \cdot e\left(\hat{S}, g_2\right)$. Dividing the equations and considering that $\hat{S} = S^*$ since $\mathsf{bad}_3 = \mathsf{true}$, $e\left(\frac{\Lambda^*}{\hat{\Lambda}}, Z\right) = e\left(\frac{M^*}{\hat{M}}, Y\right)$ or alternatively $e\left(\frac{\Lambda^*}{\hat{\Lambda}}, Z\right) \cdot e\left(\left(\frac{\hat{M}}{M^*}\right)^y, g_2\right) = 1$ and we have found a solution to the double pairing problem. By definition, we have $M^* \neq \hat{M}$.

Now we consider the general case. The simulator $\mathcal{S}$ gets as input $\mathsf{bgp}, Z \in \mathbb{G}_2$. It simulates Game 3.

Setup Simulator $\mathcal{S}$ chooses the parameter $n \in \mathbb{Z}$ and the tag space $\mathcal{T} = [n]$. It outputs the public parameters $\mathsf{pp} = (n, \mathsf{bgp})$.

KeyGen Simulator $\mathcal{S}$ chooses $a_i, b_i \in \mathbb{Z}_p$ uniformly at random for all $i = 1, \ldots, n$. It sets $h_i = g_t^{a_i} \cdot e(g_1, G_2)^{b_i}$. It chooses $y \in \mathbb{Z}_p$ uniformly at random and sets $Y = Z^y$. It gives the verification key $\mathsf{vk} = (h_1, \ldots, h_n, Y, Z)$ to $\mathcal{A}$.

Queries When queried for (M, τ) simulator $\mathcal{S}$ sets $\Lambda = g_1^{b_\tau} \cdot M^y$ as well as $S = g_1^{-a_\tau}$. Note that since a_τ, b_τ were chosen uniformly at random the signature is correctly distributed.

Forgery Let $(\mathcal{P}^*, M^*, \sigma^*)$ with $\sigma^* = (\Lambda^*, S^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 3 to compute $\hat{\Lambda}, \hat{S}, \hat{M}$. Since σ^* is a successful forgery, we furthermore know that both $e(\Lambda^*, Z) = e(M^*, Y) \cdot \prod_{i=1}^{n} h_{\tau_i}^{f_i} \cdot e(S^*, g_2)$ and $e\left(\hat{\Lambda}, Z\right) = e\left(\hat{M}, Y\right) \cdot \prod_{i=1}^{n} h_{\tau_i}^{f_i} \cdot e\left(\hat{S}, g_2\right)$.

Dividing the equations and using the identity $Y = Z^y$ yields $e\left(\frac{\Lambda^*}{\hat{\Lambda}}, Z\right) = e\left(\frac{M^*}{\hat{M}}, Z^y\right) \cdot e\left(\frac{S^*}{\hat{S}}, g_2\right)$ or alternatively $e\left(\frac{\Lambda^*}{\hat{\Lambda}} \cdot \left(\frac{\hat{M}}{M^*}\right)^y, Z\right) \cdot e\left(\frac{\hat{S}}{S^*}, g_2\right) = 1$ and we have found a solution to the double pairing problem. Since we have $\mathsf{bad}_3 = \mathsf{false}$ we know that $\hat{S} \neq S^*$.

Finally, we argue the privacy of SPHAuth. Intuitively, a homomorphic authenticator scheme is context hiding if it is infeasible to derive information about the inputs to a computation from an authenticator to the outcome of a computation (beyond what can be learned from the output itself). We show that for SPHAuth, this holds even against a computationally unbounded adversary.

Theorem 3. SPHAuth *is perfectly context hiding (see [10]).*

Proof. We show that SPHAuth is perfectly context hiding by comparing the distributions of homomorphically derived signatures to that of simulated signatures. First, in our case, the algorithm Hide is just the identity function. More precisely, we have $\mathsf{Hide}(\mathsf{vk}, M, \sigma) = \sigma$, for all possible verification keys vk, messages $M \in \mathbb{G}_1$ and authenticators σ. Thus we have $\mathsf{HideVer} = \mathsf{Ver}$, so correctness and unforgeability hold by Lemmas 10, and 11, and Theorem 2.

We show how to construct a simulator Sim that outputs signatures perfectly indistinguishable from the ones obtained by running Eval. Parse the simulator's

input as $\mathsf{sk} = (K, x_1, \ldots, x_n, y, z)$, $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$. It computes $s_i = F_K(\tau_i)$. It sets $S' = g_1^{\sum_{i=1}^{n} s_i}$ and $\Lambda' = \left(g_1^{\sum_{i=1}^{n} (x_{\tau_i} + s_i)} \cdot M^y \right)^{\frac{1}{z}}$.

Let $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}, \tau_i, M_i)$, $\sigma^* \leftarrow \mathsf{HEval}(f, \{\sigma_i, \}_{i \in [n]}, 0)$. Parsing $\sigma^* = (\Lambda^*, S^*)$, we have by construction $S^* = S'$ and $\Lambda^* = \Lambda'$. Since these elements are identical, they are indistinguishable against a computationally unbounded distinguisher.

6 Related Work

6.1 Transforming homomorphic authenticators

Catalano et al. [12] showed a transformation for linearly homomorphic signatures. They introduced a primitive called *linearly homomorphic authenticated encryption with public verifiability (LAEPuV)*, and how to combine linearly homomorphic signature schemes with Paillier encryption to obtain LAEPuV schemes. Their work is, however, restricted to the computational security of Paillier encryption. Our approach also allows for information-theoretic hiding properties.

We now survey work related to the primitives used in our transformations.

6.2 Commitments

Commitment schemes are a convenient tool to add verifiability to various processes, such as secret sharing [23], multi-party computation [8], or e-voting [22]. The most well-known and widely used commitment schemes used to provide verifiability are Pedersen's commitments [23]. In [26], FDC schemes are introduced. Unlike previous commitment schemes, they allow for succinctness and amortized efficiency. Furthermore, FDCs 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. In this work, we investigate the relations between homomorphic authenticators and FDCs. In particular, we show how to construct FDCs from structure-preserving signatures. In [20], 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 [13], for which finding secure instantiations has been challenging so far.

6.3 Homomorphic authenticators

Homomorphic authenticators have been proposed both in the secret-key setting, as homomorphic MACs (e.g. [5, 7, 9, 28]), and in the public-key setting as homomorphic signatures (e.g. [6, 10, 11, 24, 25]). In contrast, FDCs additionally

consider information-theoretic privacy. Libert et al. [19] presented a structure-preserving, linearly homomorphic signature scheme. For structure-preserving homomorphic signatures, so far, only schemes limited to linear functions are known. Our construction in Sec. 5 is, however, the first such scheme to achieve efficient verification as well as the first to be context hiding.

6.4 Structure-preserving signatures

The notion of signatures to group elements consisting of group elements were introduced by Groth [16]. This property was later called *structure-preserving* [2]. Since then, various constructions have been proposed (e.g. [1, 4, 15, 17]).

7 Conclusion

In this paper, we have investigated the relations between homomorphic authenticators and FDCs. We have shown that every FDC can be transformed into a homomorphic signature. Conversely, we then showed that structure-preserving homomorphic signature schemes can be turned into FDCs. Finally, we introduced a new structure-preserving scheme suitable for our transformation. This construction is indeed the first such scheme to allow for efficient verification. These results give us a more thorough understanding on the relationship between FDCs and homomorphic signatures. Ultimately, they further support the addition of information-theoretic confidentiality to suitable homomorphic signature schemes.

References

1. Abe, M., Chase, M., David, B., Kohlweiss, M., Nishimaki, R., Ohkubo, M.: Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In: ASIACRYPT. LNCS, vol. 7658, pp. 4–24. Springer (2012)
2. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: CRYPTO. LNCS, vol. 6223, pp. 209–236. Springer (2010)
3. Abe, M., Haralambiev, K., Ohkubo, M.: Signing on elements in bilinear groups for modular protocol design. IACR Cryptology ePrint Archive 2010, 133 (2010)
4. Abe, M., Hofheinz, D., Nishimaki, R., Ohkubo, M., Pan, J.: Compact structure-preserving signatures with almost tight security. In: CRYPTO (2). LNCS, vol. 10402, pp. 548–580. Springer (2017)
5. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-based integrity for network coding. In: ACNS. LNCS, vol. 5536, pp. 292–305 (2009)
6. Attrapadung, N., Libert, B.: Homomorphic network coding signatures in the standard model. In: PKC. LNCS, vol. 6571, pp. 17–34. Springer (2011)
7. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: ACM CCS. pp. 863–874. ACM (2013)
8. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: SCN. LNCS, vol. 8642, pp. 175–196. Springer (2014)

9. Catalano, D., Fiore, D., Gennaro, R., Nizzardo, L.: Generalizing homomorphic MACs for arithmetic circuits. In: PKC. LNCS, vol. 8383, pp. 538–555. Springer (2014)
10. Catalano, D., Fiore, D., Nizzardo, L.: Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In: CRYPTO (2). LNCS, vol. 9216, pp. 254–274. Springer (2015)
11. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: CRYPTO (1). LNCS, vol. 8616, pp. 371–389. Springer (2014)
12. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: New homomorphic primitives and applications. In: ASIACRYPT (2). LNCS, vol. 8874, pp. 193–212. Springer (2014)
13. 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)
14. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. In: ASIACRYPT (2). LNCS, vol. 10032, pp. 499–530 (2016)
15. Ghadafi, E.: How low can you go? short structure-preserving signatures for Diffie–Hellman vectors. In: IMACC. LNCS, vol. 10655, pp. 185–204. Springer (2017)
16. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: ASIACRYPT. LNCS, vol. 4284, pp. 444–459. Springer (2006)
17. Jutla, C.S., Roy, A.: Improved structure preserving signatures under standard bilinear assumptions. In: PKC (2). LNCS, vol. 10175, pp. 183–209. Springer (2017)
18. Kiltz, E., Pan, J., Wee, H.: Structure-preserving signatures from standard assumptions, revisited. In: CRYPTO. LNCS, vol. 9216, pp. 275–295. Springer (2015)
19. Libert, B., Peters, T., Joye, M., Yung, M.: Linearly homomorphic structure-preserving signatures and their applications. In: CRYPTO (2). LNCS, vol. 8043, pp. 289–307. Springer (2013)
20. Libert, B., Ramanna, S.C., Yung, M.: Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In: ICALP. LIPIcs, vol. 55, pp. 30:1–30:14. Dagstuhl (2016)
21. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: TCC. LNCS, vol. 5978, pp. 499–517. Springer (2010)
22. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: CRYPTO. LNCS, vol. 4117, pp. 373–392. Springer (2006)
23. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. LNCS, vol. 576, pp. 129–140. Springer (1991)
24. 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)
25. Schabhüser, L., Butin, D., Buchmann, J.: CHQS: publicly verifiable homomorphic signatures beyond the linear case. In: ISPEC. LNCS, vol. 11125, pp. 213–228. Springer (2018)
26. Schabhüser, L., Butin, D., Demirel, D., Buchmann, J.: Function-dependent commitments for verifiable multi-party computation. In: ISC. LNCS, vol. 11060, pp. 289–307. Springer (2018)
27. Schabhüser, L., Demirel, D., Buchmann, J.: An unconditionally hiding auditing procedure for computations over distributed data. In: CNS. pp. 552–560. IEEE (2016)
28. Zhang, L.F., Safavi-Naini, R.: Generalized homomorphic MACs with efficient verification. In: AsiaPKC@AsiaCCS. pp. 3–12. ACM (2014)