Fully Homomorphic Message Authenticators

Rosario Gennaro[*] Daniel Wichs[†]

Abstract

We define and construct a new primitive called a *fully homomorphic message authenticator*. With such scheme, anybody can perform arbitrary computations over authenticated data and produce a *short* tag that authenticates the result of the computation (without knowing the secret key). This tag can be verified using the secret key to ensure that the claimed result is indeed the correct output of the specified computation over previously authenticated data (without knowing the underlying data). For example, Alice can upload authenticated data to "the cloud", which then performs some specified computations over this data and sends the output to Bob, along with a short tag that convinces Bob of correctness. Alice and Bob only share a secret key, and Bob never needs to know Alice's underlying data. Our construction relies on fully homomorphic encryption to build fully homomorphic message authenticators.

1 Introduction

The rise of the *cloud computing* paradigm requires that users can securely outsource their data to a remote service provider while allowing it to reliably perform computations over the data. The recent groundbreaking development of *fully homomorphic encryption* [24] allows us to maintain *confidentiality/privacy* of outsourced data in this setting. In this work, we look at the analogous but orthogonal question of providing *integrity/authenticity* for computations over outsourced data. In particular, if a remote server claims that the execution of some program $\mathcal{P}$ over the user's outsourced data results in an output y, how can the user be sure that this is indeed the case?

More generally, we can consider a group of mutually-trusting users that share a secret key – each user can authenticate various data items at various times (without keeping state) and upload the authenticated data to an untrusted cloud. The cloud should be able to perform a joint computation over various data of several users and convince any user in the group of the validity of the result.

Toward this goal, we define and instantiate a new primitive, called a *fully homomorphic message authenticator*. This primitive can be seen as a *symmetric-key* version of fully homomorphic *signatures*, which were defined by Boneh and Freeman [10], but whose construction remains an open problem. We will return to survey the related work on partially homomorphic signatures and authenticators, as well as related work on *delegating memory and computation*, in Section 1.3. First, we describe our notion of fully homomorphic message authenticators, which will be the focus of this work.

1.1 What are Homomorphic Message Authenticators?

Simplified Description. In a homomorphic message-authenticator scheme, Alice can authenticate some large data D using her secret key sk. Later, anybody can homomorphically execute an arbitrary program $\mathcal{P}$ over the authenticated data to produce a short tag ψ (without knowing sk), which certifies the value $y = \mathcal{P}(D)$ as the output of $\mathcal{P}$. It is important that ψ does *not* simply authenticate y out of

[*] City College, CUNY. `rosario@cs.ccny.cuny.edu`. Work done while at the IBM T.J.Watson Research Center.

[†] Northeastern University. `wichs@ccs.neu.edu`. Work done while at IBM T.J.Watson Research Center.

context; it only certifies y as the output of a specific program $\mathcal{P}$. Another user Bob, who shares the secret key sk with Alice, can verify the triple $(y, \mathcal{P}, \psi)$ to ensure that y is indeed the output of the program $\mathcal{P}$ evaluated on Alice's previously authenticated data D (without knowing D). The tag ψ should be *succinct*, meaning that its size is independent of the size of the data D or the complexity of the program $\mathcal{P}$. In other words, homomorphic message authenticators allow anyone to certify the output of a complex computation over a large authenticated data with only a short tag.

Labeled Data and Programs. The above high-level description considers a restricted scenario where a single user Alice authenticates a single large data D in one shot. We actually consider a more general setting where many users, who share a secret key, can authenticate various data-items (say, many different files) at different times without keeping any local or joint state. In this setting, we need to establish some syntax for specifying *which* data is being authenticated and which data a program $\mathcal{P}$ should be evaluated on. For this purpose, we rely on the notion of *labeled* data and programs.

Whenever the user wants to authenticate some data-item D, she chooses a *label* τ for it, and the authentication algorithm authenticates the data D with respect to the label τ. For example, the label τ could be a file name. For the greatest level of granularity, we will assume that the user authenticates individual *bits* of data separately. Each bit b is authenticated with respect to its own label τ via a secretly-keyed authentication algorithm $\sigma \leftarrow \mathsf{Auth}_{sk}(b, \tau)$. For example, to authenticate a long file named "salaries" containing the data-bits $D = (b_1, \ldots, b_t)$, the user can create separate labels $\tau_i =$("salaries", i) to denote the i^{th} bit of the file, and then authenticate each bit b_i of the file under the label τ_i. Our scheme is oblivious to how the labels for each bit are chosen and whether they have any meaningful semantics.

Correspondingly, we consider *labeled programs* $\mathcal{P}$, where each input bit of the program has an associated label τ_i indicating which data it should be evaluated on. For example, a labeled-program $\mathcal{P}$ meant to compute the *median* of the "salaries" data would have its input bits labeled by $\tau_i =$("salaries", i). In general, the description of the labeled program $\mathcal{P}$ could be as long as, or even longer than, the input data itself. However, as in the above example, we envision the typical use-case to be one where $\mathcal{P}$ has some succinct description, such as computing the "median of the salaries data". We note that a labeled program can compute over data authenticated by different users at different times, as long as it was authenticated with the same shared secret key,

Homomorphic authenticators allow us to certify the output of a labeled program, given authentication tags for correspondingly labeled input data. In particular, there is a public homomorphic evaluation algorithm $\psi = \mathsf{Eval}(\mathcal{P}, \sigma_1, \ldots, \sigma_t)$ that takes as input tags σ_i authenticating some data-bits b_i with respect to some labels τ_i, and a labeled program $\mathcal{P}$ with matching input labels $\tau_1, \ldots, \tau_t$. It outputs a tag ψ that certifies the value $y = \mathcal{P}(b_1, \ldots, b_t)$ as the correct output of the program $\mathcal{P}$. The verification algorithm $\mathsf{Ver}_{sk}(y, \mathcal{P}, \psi)$ uses the secret key sk to verify that y is indeed the output of the labeled program $\mathcal{P}$ on previously authenticated labeled input-data, without needing to know the original data itself.

Composition. Our homomorphic authenticators are also *composable* so that we can incrementally combine authenticated outputs of partial computations to derive an authenticated output of a larger computation. In particular, if the tags $\psi_1, \ldots, \psi_t$ authenticate some bits $b_1, \ldots, b_t$ as the outputs of some labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ respectively, then $\psi^* = \mathsf{Eval}(\mathcal{P}^*, \psi_1, \ldots, \psi_t)$ should authenticate the bit $b^* = \mathcal{P}^*(b_1, \ldots, b_t)$ as the output of the *composed program*, which first evaluates $\mathcal{P}_1, \ldots, \mathcal{P}_t$ on the appropriately labeled authenticated data, and then runs $\mathcal{P}^*$ on the outputs.

Succinct Tags vs. Efficient Verification. The main requirement that makes our definition of homomorphic authenticators interesting is that the tags should be *succinct*. Otherwise, there is a trivial solution where we can authenticate the output of a computation $\mathcal{P}$ by simply providing all of its input bits and their authentication tags. The succinctness requirement ensures that we can certify the output of a computation $\mathcal{P}$ over authenticated data with much smaller *communication* than that of simply transmitting the input data.[1] Therefore, this primitive is especially useful when verifying computations that read a lot of input data but have a short output (e.g., computing the median in a large database).

However, we note that the verification algorithm in a homomorphic authenticator schemes is allowed to have a large *computational complexity*, proportional to the complexity of the computation $\mathcal{P}$ being verified. Therefore, although homomorphic authenticators allow us to save on *communication*, they do not necessarily save on the *computational complexity* of verifying computations over outsourced data. We believe that communication-efficient solutions are already interesting, and may be useful, even without the additional constraint of computational efficiency. In Section 4, we explore how to combine our communication-efficient homomorphic authenticators with techniques from *delegating computation* to also achieve computationally efficient verification.

1.2 Overview of Our Construction

Our construction of fully homomorphic authenticators relies on the use of fully homomorphic encryption (FHE). We let n denote the *security parameter* and define $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. The secret key of the authenticator scheme consists of: a random *verification set* $S \subseteq [n]$ of size $|S| = n/2$, a key-pair (pk, sk) for a fully homomorphic encryption (FHE) scheme and a pseudo-random function (PRF) $f_K(\cdot)$.

To authenticate a bit b under a label τ, Alice creates n ciphertexts $c_1, \ldots, c_n$ as follows. For $i \in [n] \setminus S$, she chooses the ciphertexts $c_i \leftarrow \mathsf{Enc}_{pk}(b)$ as random encryptions of the actual bit b being authenticated. For $i \in S$, she computes $c_i = \mathsf{Enc}_{pk}(0; f_K((i, \tau)))$ as *pseudorandom encryptions of 0*, where the random coins are derived using the PRF. Notice that for the indices $i \in S$ in the verification set, the pseudorandom ciphertexts c_i can be easily re-computed from Alice's secret key and the label τ alone, *without knowing the data* bit b. Alice outputs the *authentication tag* $\sigma = (c_1, \ldots, c_n)$, consisting of the n ciphertexts.

Given some program $\mathcal{P}$ with t input-labels and t authentication tags $\{\sigma_j = (c_{1,j}, \ldots, c_{n,j})\}_{j \in [t]}$ for the correspondingly labeled data, we can homomorphically derive an authentication tag $\psi = (c_1^*, \ldots, c_n^*)$ for the output by setting $c_i^* = \mathsf{Eval}(\mathcal{P}, c_{i,1}, \ldots, c_{i,t})$, where Eval is the homomorphic evaluation of the FHE scheme. In other words, for each position $i \in [n]$, we perform a homomorphic evaluations of the program $\mathcal{P}$ over the t FHE ciphertexts that lie in position i. We assume (without loss of generality) that the evaluation procedure for the FHE scheme is deterministic so that the results are reproducible.

Alice can verify the triple $(y, \mathcal{P}, \psi)$, where the tag $\psi = (\hat{c}_1, \ldots, \hat{c}_n)$ is supposed to certify that y is the output of the labeled program $\mathcal{P}$. Let $\tau_1, \ldots, \tau_t$ be the input labels of $\mathcal{P}$. For the indices $i \in S$ in the verification set, Alice can re-compute the pseudo-random input ciphertexts $\{c_{i,j} = \mathsf{Enc}_{pk}(0; f_K((i, \tau_j)))\}_{j \in [t]}$ using the PRF, without knowing the actual input bits. She then computes $c_i^* = \mathsf{Eval}(\mathcal{P}, c_{i,1}, \ldots, c_{i,t})$ and checks that the ciphertexts in the tag ψ were computed correctly with $\hat{c}_i \stackrel{?}{=} c_i^*$ for indices $i \in S$. [2] If this is the case, and all of the other ciphertexts $\hat{c}_i$ for $i \in [n] \setminus S$ decrypt to the claimed bit y, then Alice accepts.

Intuitively, the only way that an attacker can lie about the output of some program $\mathcal{P}$ is by producing a tag $\psi = (\hat{c}_1, \ldots, \hat{c}_t)$ where the ciphertexts $\hat{c}_i$ for indices in the verification set $i \in S$ are computed correctly but for $i \in [n] \setminus S$ they are all modified so as to encrypt the wrong bit. But this is impossible since the security of the FHE should ensure that the attacker cannot distinguish encryptions of 0 from those of the authenticated bits, and hence cannot learn anything about the set S. In particular, the FHE hides the difference between data-independent pseudorandom ciphertexts $c_i \ : \ i \in S$, which allow Alice to check that the computation was performed correctly, and data-containing ciphertexts $c_i \ : \ i \in [n] \setminus S$, which allow Alice to check that the output bit y is the correct one. Note that the authentication tags ψ in our scheme always consist of n (= security parameter) ciphertexts, no matter how many inputs the program $\mathcal{P}$ takes and what its complexity is. Therefore, we satisfy the succinctness requirement.

[1] Note that we do not count the cost of transmitting the labeled-program $\mathcal{P}$ itself. As a previous note explains, we envision that in the typical use-case such programs should have a succinct description.

[2] Note that, in this step, Alice has to perform work comparable to that of computing $\mathcal{P}$.

We remark that several recent schemes for *delegating computation* [21, 16, 2] (see Section 1.3 on related work) also use FHE in a similar manner to check that a computation is performed correctly by a remote server. In particular, the work of Chung, Kalai and Vadhan [16] relies on a similar idea, where the output of the homomorphic evaluation for some "data-independent ciphertexts" is known in advance and used to check that the computation was done correctly for the relevant "data-containing" ciphertexts. However, in all these previous works, the technique is used to verify computations over a short known input, whereas in our case we use it to verify computations over unknown authenticated data. The main novelty in our use of the technique is to notice that the "data-independent ciphertexts" can be made *pseudorandom*, so that they can be re-derived in the future given only a short secret PRF key without needing to sacrifice any significant storage.

Security & Verification Queries. We show that our construction is secure in the setting where the attacker can adaptively make arbitrarily many *authentication queries* for various labels, but cannot make *verification queries* to test if a maliciously constructed tag verifies correctly. In practice, this means that the user needs to abort and completely stop using the scheme whenever she gets the first tag that doesn't verify correctly. It is easy to allow for some fixed a-priori bounded number of verification queries q, just by increasing the number of ciphertexts contained in an authentication tag from n to $n + q$.

The difficulty of allowing arbitrarily many verification queries also comes up in most prior schemes for delegating computation in the "pre-processing" model [21, 16, 2] (see Section 1.3 on related work), and remains an important open problem in both areas.

Fast Verification. One of the limitations of our solution above is that the verification algorithm is *no more efficient* than running the computation $\mathcal{P}$. Therefore, although it saves tremendously on the *communication complexity* of verifying computations over outsourced data, it does not save on user's computational complexity. In Section 4, we explore the option of using schemes for *delegating computation* to also offload the computational cost of the verification procedure to the remote server. As one of our contributions, we show how to achieve fully homomorphic MACs with fast verification using our initial construction and *succinct non-interactive arguments* for polynomial-time computation: **P**-SNARGs. In contrast, a simple solution that bypasses fully-homomorphic MACs would require succinct non-interactive arguments *of knowledge* for all *non-deterministic polynomial-time* computation: **NP**-SNARKs.

1.3 Related work

Homomorphic Signatures and MACs. Many prior works consider the question of homomorphic message authentication (private verification) and signatures (public verification) for restricted homomorphisms, and almost exclusively for *linear functions.* Perhaps the first work to propose this problem is that of Johnson et al. [29]. Since then, many works have considered this notion in the context of *network coding*, yielding a long line of positive results [1, 9, 23, 10, 5, 11, 15, 20]. Another line of works considered this notion in the context of *proofs of data possession and retrievability* [3, 33, 18, 4].

The only work that considers a larger class of homomorphisms *beyond linear functions* is that of Boneh and Freeman [10], who show how to get homomorphic signatures for *bounded (constant) degree polynomials.* In that work, they also present a general definition along the same lines as the definition we use in this work, and pose the question of constructing *fully* homomorphic signatures for *arbitrary* functions.[3] Although the question of fully homomorphic *publicly verifiable* signatures under standard assumptions still remains open, our work provides the first positive result for the case of private verification.

Succinct Arguments of Knowledge. One method that would allow us to construct fully homomorphic (publicly verifiable) signatures is to rely on CS-Proofs [30] or, more generally, any *succinct*

[3] See [10, 20] for an explanation of how this definition generalizes that of prior works on network coding.

non-interactive argument of knowledge for all of **NP** (**NP**-SNARK) [7]. This primitive allows us to create a short "argument" π for any **NP** statement, to prove "knowledge" of the corresponding witness. The length of π is independent of the statement/witness size, and the complexity of verifying π only depends on the size size of the statement.

Using SNARKs, we can authenticate the output y of a labeled program $\mathcal{P}$, by creating a short argument π that proves the knowledge of some "labeled input data D along with valid signatures authenticating D under the appropriate labels, such that $\mathcal{P}(D) = y$". Since this is an argument of *knowledge*, a forged signature for the output of some program $\mathcal{P}$ would allow us to extract out a forged signature for the underlying input data, breaking the security of signatures.

Unfortunately, constructing succinct non-interactive arguments for **NP** is known to require the use of *non-standard* assumptions [25]. Current constructions either rely on the *random-oracle model* [30] or on various "knowledge" assumptions (see, e.g., [28, 7, 8, 22]).

Delegating Computation. Several prior works consider the problem of *delegating computation* to a remote server while maintaining the ability to efficiently verify the result [27, 21, 16, 2, 6, 31]. In this scenario, the server needs to convince the user that $\mathcal{P}(x) = y$, where the user knows the program $\mathcal{P}$, the input x and the output y, but does not want to do the *work* of computing $\mathcal{P}(x)$. In contrast, in our scenario the verifier only knows $\mathcal{P}, y$, but does *not* know the previously authenticated inputs that $\mathcal{P}$ should have been executed on. On the other hand, we are not trying to minimize work, just communication.

Despite these differences, some of the results on delegating computation in the "pre-processing" model [21, 16, 2], can also be (re-)interpreted for our setting. In this model, the user "pre-processes" a circuit C and stores some value σ on the server. Later, the user can ask the server to compute $C(x)$ for various inputs x, and the server uses σ to derive a short and efficiently verifiable proof ψ that certifies correctness of the computation. One caveat is that, in all these schemes, the user first needs to send some challenge $c = \mathit{chall}(x)$ to the server, and the proof ψ is computed as a response to c.

We can apply these results to our setting of outsourced data as follows. Consider outsourcing the "universal circuit" $C_D(\cdot)$ that has the data D "hard-coded", gets as input a program $\mathcal{P}$, and outputs $C_D(\mathcal{P}) = \mathcal{P}(D)$. Then, we can think of the pre-processing of C_D as creating an authentication tag σ for the data D. Later, the user can take a program $\mathcal{P}$, create a challenge $c = \mathit{chall}(\mathcal{P})$, and get back a short tag ψ that authenticates $y = C_D(\mathcal{P}) = \mathcal{P}(D)$.[4] One advantage of this approach is that the tag ψ can be verified efficiently, with less work than that of computing $\mathcal{P}(D)$. However, there are several important disadvantages of this approach as compared to our notion of homomorphic authenticators:

1. *Interaction:* Homomorphic authenticators allow anybody to evaluate a chosen program $\mathcal{P}$ over authenticated data and *non-interactively* authenticate the output. The above delegation-based schemes require a *round of interaction*; the user first creates a challenge $\mathit{chall}(\mathcal{P})$ for the program $\mathcal{P}$, and only then can the server authenticate the output $\mathcal{P}(D)$ with respect to this challenge.

2. *Single Use:* Homomorphic authenticators allow several users to authenticate various labeled data *"on the fly"* (without any state) and verify arbitrary computations over all of the data in the future using a fixed secret key. The above delegation-based schemes require that a single user outsources all of the data D in *one shot*, and stores some small secret state associated with the data to verify computations over only this data in the future.

3. *Bounded Size:* The above delegation-based schemes require that the circuit-size of the computations $\mathcal{P}$ is *a-priori bounded* by some fixed polynomial chosen during authentication. Furthermore, the complexity of authentication is proportional to this polynomial. Our fully homomorphic authenticators have no such restriction.

[4] Here, we assume that $\mathcal{P}$ has a short uniform description so that reading /transmitting $\mathcal{P}$ is much more efficient than evaluating $\mathcal{P}$.

4. *No composition:* The above delegation-based schemes does not support the *composition* of several partial authenticated computations.

Memory Delegation. The work of Chung et al. [17] on *memory delegation* explicitly considers the problem of outsourcing a large amount of data while maintaining the ability to efficiently verify later computations over it. The main *advantages* of memory delegation over our work are that: (I) the verification is more efficient than the computation, (II) the data can be efficiently updated by the user in the future, (III) it does not suffer from the verification problem. However, memory delegation suffers from many of the same disadvantaged outlined above for delegation-based schemes. In particular, the general memory delegation scheme of [17] is interactive, requiring 4 rounds of interaction during verification. The paper also provides a non-interactive solution where the size of the tag grows with the *depth* of the computation-circuit (and therefore does not satisfy our succinctness property). Furthermore, memory delegation considers the setting where a single user outsources a single data item D in *one shot*, and store some small secret state associated with the data to verify computations over it in the future. In particular, it does not provide a method where various users can authenticate various (small) pieces of data independently, and verify joint computations over all of the data in the future. Lastly, the memory-delegation solutions do not support composition.

Follow-up work. Following our work, Catalano and Fiore [14] (Eurocrypt '13) give a very efficient and simple construction of homomorphic-message authenticators for low-depth arithmetic circuits – in particular, the computation/verification time and the tag size depend polynomially on the *degree* of the circuit, which can be exponential in the depth of the circuit. Their construction only relies on one-way functions. In contrast, our construction here is significantly more general (works for any polynomial-size boolean circuit) but relies on the "heavier machinery" of fully homomorphic encryption.

2 Definitions

2.1 Homomorphic Authenticators

Labeled Programs. We begin by defining the concept of a labeled program, where the labels denote *which* data the program should be evaluated on. Formally, a *labeled-program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_k)$ consists of a circuit $f : \{0,1\}^k \to \{0,1\}$ along with a distinct *input label* $\tau_i \in \{0,1\}^*$ for each input wire $i \in [k]$. [5]

Given some labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a circuit $g : \{0,1\}^t \to \{0,1\}$, we can define the *composed program*, denoted by $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, which corresponds to evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$. The labeled inputs of the composed program $\mathcal{P}^*$ are just all the *distinct* labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, meaning that we collect all the input wires with the same label and convert them into a single input wire.

We define the *identity program with label* τ as $\mathcal{I}_\tau := (g_{id}, \tau)$ where g_{id} is the *canonical identity circuit* and $\tau \in \{0,1\}^*$ is some label. Notice that any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_k)$ can be written as a composition of identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_k})$.

Syntax. A *homomorphic authenticator scheme* consists of the probabilistic-polynomial time algorithms $(\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Ver}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{KeyGen}(1^n) \to (evk, sk)$: Outputs the secret key sk and an evaluation key evk.
- $\mathsf{Auth}_{sk}(b, \tau) \to \sigma$: Creates a *tag* σ that authenticates the bit $b \in \{0,1\}$ under the label $\tau \in \{0,1\}^*$. (Equivalently, we say that σ authenticates b as the output of the identity program $\mathcal{I}_\tau$.)

[5] Although the above description of $\mathcal{P}$ is long (proportional to its input size and complexity), in many scenarios it is possible that $\mathcal{P}$ may also have an alternative succinct description. For example, $\mathcal{P}$ may compute the *median* value in a large file called "salaries" and its input labels are simply $\tau_i =$ ("salaries", i) for each bit i. Therefore, although we are concerned with succinctness, we will ignore the cost of communicating the program $\mathcal{P}$ from future consideration.

- $\mathsf{Eval}_{evk}(f, \boldsymbol{\sigma}) \to \psi$: The deterministic evaluation procedure takes a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$ and a circuit $f : \{0,1\}^k \to \{0,1\}$. It outputs a tag ψ. If each σ_i authenticates a bit b_i as the output of some labeled-program $\mathcal{P}_i$ (possibly the identity program), then ψ should authenticate $b^* = f(b_1, \ldots, b_k)$ as the output of the composed program $\mathcal{P}^* = f(\mathcal{P}_1, \ldots, \mathcal{P}_k)$.
- $\mathsf{Ver}_{sk}(e, \mathcal{P}, \psi) \to \{\mathtt{accept}, \mathtt{reject}\}$: The deterministic verification procedure uses the tag ψ to check that $e \in \{0,1\}$ is the output of the program $\mathcal{P}$ on previously authenticated labeled data.

We require that the scheme satisfies the following properties, defined below: *authentication correctness*, *evaluation correctness*, *succinctness* and *authenticator security.*

Authentication Correctness. We require that for any $b \in \{0,1\}$ and any label $\tau \in \{0,1\}^*$, we have:

$$\Pr\left[\mathsf{Ver}_{sk}(b, \mathcal{I}_\tau, \sigma) = \mathtt{accept} \;\middle|\; (evk, sk) \leftarrow \mathsf{KeyGen}(1^n), \sigma \leftarrow \mathsf{Auth}_{sk}(b, \tau) \;\right] = 1$$

where $\mathcal{I}_\tau$ is the identity program with label τ. In other words, the tag $\sigma = \mathsf{Auth}_{sk}(b, \tau)$ correctly authenticates b under the label τ, which is equivalent to saying that it authenticates b as the output of the identity program $\mathcal{I}_\tau$.

Evaluation Correctness. Fix any (evk, sk) in the support of $\mathsf{KeyGen}(1^n)$. Fix any circuit $g : \{0,1\}^t \to \{0,1\}$ and any set of program/bit/tag triples $\{\ (\mathcal{P}_i, b_i, \psi_i)\ \}_{i=1}^t$ such that $\mathsf{Ver}_{sk}(b_i, \mathcal{P}_i, \psi_i) = \mathtt{accept}$. Set:

$$b^* := g(b_1, \ldots, b_t),\ \mathcal{P}^* := g(\mathcal{P}_1, \ldots, \mathcal{P}_t),\ \psi^* := \mathsf{Eval}_{evk}(g, (\psi_1, \ldots, \psi_t)).$$

Then we require $\mathsf{Ver}_{sk}(b^*, \mathcal{P}^*, \psi^*) = \mathtt{accept}$.

In words, assume that each tag ψ_i certifies that the output of the labeled program $\mathcal{P}_i$ is b_i. Then the tag ψ^* certifies that b^* is the output of the composed program $\mathcal{P}^*$. If all of the programs $\mathcal{P}_i = \mathcal{I}_{\tau_i}$ are identity programs, then the above says that as long as the tags ψ_i authenticate bits b_i under the labels τ_i, the tag ψ^* authenticates b^* as the output of $\mathcal{P}^* = (g, \tau_1, \ldots, \tau_t)$. Therefore, the above definition captures the basic correctness of computing over freshly authenticated data, as well as the *composability* of computing over the authenticated outputs of prior computations.

Succinctness. We require that the tag-size is always bounded by some fixed polynomial in the security parameter n, and is independent of the size of the evaluated circuit or the number of inputs it takes. That is, there exists some polynomial $p(\cdot)$ such that, for any (evk, sk) in the support of $\mathsf{KeyGen}(1^n)$, the output-size of $\mathsf{Auth}_{sk}(\cdot,\cdot)$ and of $\mathsf{Eval}_{evk}(\cdot,\cdot)$ is bounded by $p(n)$ for any choice of their input.

Authenticator Security. Consider the following game $\mathsf{ForgeGame}^{\mathcal{A}}(1^n)$ between an attacker $\mathcal{A}(1^n)$ and a challenger:

1. The challenger chooses $(evk, sk) \leftarrow \mathsf{KeyGen}(1^n)$ and gives evk to $\mathcal{A}$. It initializes $T := \emptyset$.
2. The attacker $\mathcal{A}$ can adaptively submit arbitrarily many *authentication queries* of the form (b, τ) to the challenger. On each such query, if there is some $(\tau, \cdot) \in T$ (i.e. the label τ is not fresh) then the challenger ignores it. Else it updates $T := T \cup \{(\tau, b)\}$, associating the label τ with the authenticated bit b, and replies with $\sigma \leftarrow \mathsf{Auth}_{sk}(b, \tau)$.
3. Finally, the attacker outputs some *forgery* $(e^*, \mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_k^*), \psi^*)$. The output of the game is 1 iff $\mathsf{Ver}_{sk}(e^*, \mathcal{P}^*, \psi^*) = \mathtt{accept}$ *and* one of the following two conditions holds:
    - *Type I Forgery:* There is some $i \in [k]$ such that the label $(\tau_i^*, \cdot)$ does not appear in T. (*i.e., No bit was ever authenticated under the label τ_i^* involved in the forgery.*)

- *Type II Forgery:* The set T contains tuples $(\tau_1^*, b_1), \ldots, (\tau_k^*, b_k)$, for some bits $b_1, \ldots, b_k \in \{0,1\}$ such that $f^*(b_1, \ldots, b_k) \neq e^*$. (*i.e., The labeled program $\mathcal{P}^*$ does not output e^* when executed on previously authenticated labeled data $b_1, \ldots, b_k$.)*

We say that a homomorphic authenticator scheme is *secure (without verification queries)* if, for any probabilistic polynomial-time $\mathcal{A}$, we have $\Pr[\mathsf{ForgeGame}^{\mathcal{A}}(1^n) = 1] \leq \mathsf{negl}(n)$. We can also define a stronger variant, called *security with verification queries*, where we insist that the above probability holds for a modified version of the game, in which the attacker can also adaptively make arbitrarily many verification queries of the form $(e, \mathcal{P}, \psi)$, and the challenger replies with $\mathsf{Ver}_{sk}(e, \mathcal{P}, \psi)$.

2.2 Homomorphic Encryption

A *fully homomorphic (public-key) encryption* (FHE) scheme is a quadruple of PPT algorithms $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ defined as follows.

- $\mathsf{HE.KeyGen}(1^n) \to (pk, evk, sk)$: Outputs a public encryption key pk, a public evaluation key evk and a secret decryption key sk.

- $\mathsf{HE.Enc}_{pk}(b) \to c$: Encrypts a bit $b \in \{0,1\}$ under public key pk. Outputs ciphertext c.

- $\mathsf{HE.Dec}_{sk}(c) \to b$: Decrypts ciphertext c using sk to a plaintext bit $b \in \{0,1\}$.

- $\mathsf{HE.Eval}_{evk}(g, c_1, \ldots, c_t) \to c^*$: The *deterministic evaluation algorithm* takes the evaluation key evk, a boolean circuit $g : \{0,1\}^t \to \{0,1\}$, and a set of t ciphertexts $c_1, \ldots, c_t$. It outputs the result ciphertext c^*.

An FHE should also satisfy the following properties.

Encryption Correctness. For all $b \in \{0,1\}$ we have:

$$\Pr\left[\mathsf{HE.Dec}_{sk}(\mathsf{HE.Enc}_{pk}(b)) = b \mid (pk, evk, sk) \leftarrow \mathsf{HE.KeyGen}(1^n)\right] = 1$$

Evaluation Correctness. For any (pk, evk, sk) in the support of $\mathsf{HE.KeyGen}(1^n)$, any ciphertexts $c_1, \ldots, c_t$ such that $\mathsf{HE.Dec}_{sk}(c_i) = b_i \in \{0,1\}$, and any circuit $g : \{0,1\}^t \to \{0,1\}$ we have

$$\mathsf{HE.Dec}_{sk}(\mathsf{HE.Eval}_{evk}(g, c_1, \ldots, c_t)) = g(b_1, \ldots, b_t).$$

Succinctness. We require that the ciphertext-size is always bounded by some fixed polynomial in the security parameter, and is independent of the size of the evaluated circuit or the number of inputs it takes. That is, there exists some polynomial $p(\cdot)$ such that, for any (pk, evk, sk) in the support of $\mathsf{HE.KeyGen}(1^n)$, the output-size of $\mathsf{HE.Enc}_{pk}(\cdot)$ and of $\mathsf{Eval}_{evk}(\cdot)$ is bounded by $p(n)$, for any choice of their inputs.

Semantic Security. Lastly, an FHE should satisfy the standard notion of *semantic security* for public-key encryption, where we consider the evaluation key evk as a part of the public key. That is, for any PPT attacker $\mathcal{A}$ we have:

$$\left|\ \Pr\left[\mathcal{A}(1^n, pk, evk, c_0) = 1\right] - \Pr\left[\mathcal{A}(1^n, pk, evk, c_1) = 1\right]\ \right| \leq \mathsf{negl}(n)$$

where the probability is over $(pk, evk, sk) \leftarrow \mathsf{KeyGen}(1^n)$, $\{c_b \leftarrow \mathsf{HE.Enc}_{pk}(b)\}_{b \in \{0,1\}}$, and the coins of $\mathcal{A}$.

Canonical FHE. We can take any FHE scheme and make it *canonical*, meaning that the $\mathsf{HE.Eval}$ procedure just evaluates the circuit recursively, level-by-level and gate-by-gate. In particular, for any circuit $g : \{0,1\}^k \to \{0,1\}$ taking input bits $b_1, \ldots, b_k$, if the top gate of g is $h : \{0,1\}^t \to \{0,1\}$ and the inputs to h are computed by sub-circuits $f_1(b_{i_{1,1}}, \ldots b_{i_{1,k_1}}) \ldots, f_t(b_{i_{t,1}}, \ldots b_{i_{t,k_t}})$ then

$$\mathsf{HE.Eval}_{evk}(g, c_1, \ldots, c_k) = \mathsf{HE.Eval}_{evk}(h, c_1^*, \ldots, c_t^*)$$

where $c_j^* = \mathsf{HE.Eval}_{evk}(f_j, c_{i_{j,1}}, \ldots, c_{i_{j,k_j}})$ for $j \in [t]$. We also assume that, if g_{id} is the canonical identity circuit on one input, then $\mathsf{HE.Eval}_{evk}(g_{id}, c) = c$. Making the FHE scheme canonical will be important when we want to reason about *composition*, since it will ensure that the evaluation procedure outputs the exact same ciphertext when we homomorphically evaluate the entire circuit in one shot as when we first homomorphically evaluate some sub-circuits and then combine the results via additional independent homomorphic evaluations.

See, e.g., the works of [24, 34, 13, 12] for constructions of fully homomorphic encryption.

3 Constructing Homomorphic Authenticators

We now describe our construction of homomorphic authenticators. Although it closely follows our high-level description in the introduction, there are some differences. Most notably, the simple scheme in the introduction does not appropriately protect against *type I* forgeries. For example, it is possible that for the function g_0 which always outputs 0, the FHE scheme always outputs $\mathsf{HE.Eval}_{evk}(g_0, c) = C_0$ where C_0 is some fixed and known ciphertext encrypting 0. In that case, the attacker can always authenticate the output of the labeled-program $\mathcal{P}_0 = (g_0, \tau)$ for any label τ, even if he never saw any previously authenticated data under the label τ. This would qualify as a type I forgery, breaking our definition. To fix this, we add an extra component to our tags that ensures that the attacker must have seen authentication tags for all of the underlying input labels. We describe this component below.

Hash Tree of a Circuit. If $g : \{0,1\}^k \to \{0,1\}$ is a circuit and $H : \{0,1\}^* \to \{0,1\}^m$ is some hash function, we define the *hash-tree of* g, denoted g^H, as a *Merkle-Tree* that has the same structure as the circuit g, but replaces all internal gates with the hash function H. More precisely, the hash-tree $g^H : (\{0,1\}^*)^k \to \{0,1\}^m$ is a function which takes as input strings $\nu_i \in \{0,1\}^*$ for each input wire of g. For every wire w in the circuit g, we define the *value of* $g^H(\nu_1, \ldots, \nu_k)$ *at* w inductively as:

- $val(w) = H(\nu_i)$ if w is the ith input wire of g.
- $val(w) = H(val(w_1), \ldots, val(w_t))$ if w is the output wire of some gate with input wires $w_1, \ldots, w_t$.

We define the output of the function $g^H(\nu_1, \ldots, \nu_k)$ to be its its value at the output wire of g.

Construction. Let $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ be a canonical fully homomorphic encryption scheme, where the encryption algorithm uses $r = r(n) = \omega(\log(n))$ random bits. Let $\left\{ f_K : \{0,1\}^* \to \{0,1\}^{r(n)} \right\}_{K \in \{0,1\}^n}$ be a (variable-input-length) *pseudo-random function* PRF family. Let $\mathcal{H}$ be a family of (variable-length) *collision-resistant hash functions (CRHF)* $H : \{0,1\}^* \to \{0,1\}^{m(n)}$. We define the authenticator scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ as follows:

$\mathsf{KeyGen}(1^n)$**:** Choose a PRF key $K \leftarrow \{0,1\}^n$ and a CRHF $H \leftarrow \mathcal{H}$. Choose an encryption key $(pk, evk', sk') \leftarrow \mathsf{HE.KeyGen}(1^n)$. Select a subset $S \subseteq [n]$ by choosing whether to add each index $i \in [n]$ to the set S independently with probability $\frac{1}{2}$. Output $evk = (evk', H), sk = (pk, evk', H, sk', K, S)$.

$\mathsf{Auth}_{sk}(b, \tau)$**:** Given $b \in \{0,1\}$ and $\tau \in \{0,1\}^*$ do the following:

1. Choose random coins $rand_1, \ldots, rand_n$ by setting $rand_i = f_K((\tau, i))$. Set $\nu := f_K(\tau)$.
2. Create n ciphertexts $c_1, \ldots, c_n$ as follows. For $i \in [n] \setminus S$, choose $c_i = \mathsf{HE.Enc}_{pk}(b; rand_i)$ as encryptions of the bit b. For $i \in S$, choose $c_i = \mathsf{HE.Enc}_{pk}(0; rand_i)$ as encryption of 0.
3. Output $\sigma = (c_1, \ldots, c_n, \nu)$.

$\mathsf{Eval}_{evk}(g, \boldsymbol{\sigma})$**:** Given $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_t)$, parse each $\sigma_j = (c_{1,j}, \ldots, c_{n,j}, \nu_j)$.

- For each $i \in [n]$, compute $c_i^* = \mathsf{HE.Eval}_{evk'}(g, c_{i,1}, \ldots, c_{i,t})$.
- Compute $\nu^* = g^H(\nu_1, \ldots, \nu_t)$ to be the output of the hash-tree of g evaluated at $\nu_1, \ldots \nu_t$.

Output $\psi = (c_1^*, \ldots, c_n^*, \nu^*)$.

$\mathsf{Ver}_{sk}(e, \mathcal{P}, \psi)$**:** Parse $\mathcal{P} = (g, \tau_1, \ldots, \tau_t)$ and $\psi = (c_1^*, \ldots, c_n^*, \nu^*)$.

1. Compute $\nu_1 := f_K(\tau_1), \ldots, \nu_t = f_K(\tau_t)$ and $\nu' := g^H(\nu_1, \ldots, \nu_t)$. If $\nu' \neq \nu^*$, output `reject`.
2. For $i \in S$, $j \in [t]$, compute $rand_{i,j} := f_K((\tau_j, i))$ and set $c_{i,j} := \mathsf{HE.Enc}_{pk}(0; rand_{i,j})$.
   For each $i \in S$, evaluate $c_i' := \mathsf{HE.Eval}_{evk'}(g, c_{i,1}, \ldots, c_{i,t})$ and if $c_i' \neq c_i^*$ output `reject`.
3. For each $i \in [n] \setminus S$, decrypt $e_i := \mathsf{HE.Dec}_{sk'}(c_i^*)$ and if $e \neq e_i$ output `reject`.

If the above doesn't reject, output `accept`.

Theorem 3.1. *If $\{f_K\}$ is a PRF family, $\mathcal{H}$ is a CRHF family and* HE *is a semantically secure canonical FHE, then the homomorphic authenticator scheme Π is secure without verification queries.*

Proof. It is easy to verify that the *authentication correctness* of Π just follows from the encryption correctness of HE, and the *evaluation correctness* of Π follows from that of HE, along with the fact that HE is *canonical.*

We now prove the security of Π (without verification queries). Let $\mathcal{A}$ be some PPT attacker and let $\mu(n) = \Pr[\mathsf{ForgeGame}^{\mathcal{A}}(1^n) = 1]$. We use a series of hybrid games modifying ForgeGame to prove that $\mu(n)$ must be negligible.

Game_1**:** We modify ForgeGame so as to replace the PRF outputs with truly random consistent values. That is, the challenger replaces all calls to f_K needed to answer authentication queries and to check the winning condition $\mathsf{Ver}_{sk}(e^*, \mathcal{P}^*, \psi^*) \overset{?}{=} \texttt{accept}$, with calls to a completely random function $F : \{0,1\}^* \to \{0,1\}^{r(n)}$, whose outputs it chooses efficiently "on the fly".

By the pseudo-randomness of $\{f_K\}$, we must have $\Pr[\mathsf{Game}_1^{\mathcal{A}}(n) = 1] \geq \mu(n) - \mathsf{negl}(n)$.

Game_2**:** We now define Game_2 by modifying the winning condition, so that the attacker only wins (the game outputs 1) if the attacker outputs a valid *type (II) forgery* (and the game outputs 0 on a type I forgery). Let E be the event that attacker wins with a type I forgery in Game_1. Then we claim that, under the collision-resistance of $\mathcal{H}$, we have $\Pr[E] = \mathsf{negl}(n)$ and therefore:

$$\Pr[\mathsf{Game}_2^{\mathcal{A}}(n) = 1] \geq \Pr[\mathsf{Game}_1^{\mathcal{A}}(n) = 1] - \Pr[E] \geq \mu(n) - \mathsf{negl}(n)$$

Assume otherwise that $\Pr[E]$ is non-negligible. Recall that the event E only occurs when the attacker submits a forgery

$$e^*, \mathcal{P}^* = (g, \tau_1^*, \ldots, \tau_t^*), \psi^* = (c_1^*, \ldots, c_n^*, \nu^*)$$

such that the attacker never asked any authentication query containing one of the labels τ_j^* for some $j \in [t]$, and verification accepts. During the computation of $\mathsf{Ver}_{sk}(e^*, \mathcal{P}^*, \psi^*)$, when checking that verification accepts in Game_1, the challenger chooses the value $\nu_j = F(\tau_j^*) \leftarrow \{0,1\}^{r(n)}$ freshly at random, since the label τ_j^* was never queried before. If we rewind and re-sample $\nu_j' \leftarrow \{0,1\}^{r(n)}$ freshly an independently at random again, then the probability that verification accepts *both* times, which we denote by the event E^2, is at least $\Pr[E^2] \geq \Pr[E]^2$. Let C be the event that E^2 occurs *and* the values $\nu_j \neq \nu_j'$ are distinct, so that $\Pr[C] \geq \Pr[E]^2 - 2^{-r(n)} = \Pr[E]^2 - \mathsf{negl}(n)$ is *non-negligible.* When the event C occurs then we must have $\nu^* = g^H(\nu_1, \ldots, \nu_j, \ldots, \nu_t) = g^H(\nu_1, \ldots, \nu_j', \ldots, \nu_t)$ which immediately gives us some collision on H at some level of the hash tree g^H. Therefore, $\mathcal{A}$ can be used to efficiently find collisions on $H \xleftarrow{\$} \mathcal{H}$ with non-negligible probability, which gives us a contradiction.

Game_3: In Game_3 we modify the winning condition yet again. When answering authentication queries, the challenger now also remembers the tag σ that it uses, storing (τ, b, σ) in T. If the attacker outputs a type II forgery

$$e^*, \mathcal{P}^* = (g, \tau_1^*, \ldots, \tau_t^*), \psi^* = (c_1^*, \ldots, c_n^*, \nu^*),$$

we modify how the challenger checks $\mathsf{Ver}_{sk}(e^*, \mathcal{P}^*, \psi^*) \stackrel{?}{=} \texttt{accept}$. Recall that for a type II forgery, the tags τ_i^* were previously used in authentication queries, so that T must contain some tuples

$$((\tau_1^*, b_1, \sigma_1 = (c_{1,1}, \ldots, c_{n,1}, \nu_1)), \ldots, (\tau_t^*, b_t, \sigma_t = (c_{1,t}, \ldots, c_{n,t}, \nu_t)).$$

Let $\hat{c}_i := \mathsf{HE.Eval}_{evk}(g, c_{i,1}, \ldots, c_{i,t})$ for $i \in [n]$ be the *"honest ciphertexts"* that would be included in an honestly generated tag ψ for the program $\mathcal{P}^*$. In Game_3, we replace steps (2), (3) of the verification procedure as follows:

2'. For each $i \in S$: if $\hat{c}_i \neq c_i^*$ then output `reject`.

3'. For each $i \in [n] \setminus S$: if $\hat{c}_i = c_i^*$ then output `reject`.

Notice that step (2') is actually the same as the original step (2) used in Game_2, since in both cases we just check the forgery ciphertexts c_i^* against the honest ciphertexts $c_i' = \hat{c}_i$. The only difference is that previously we re-computed c_i' from scratch by re-encrypting $c_{i,j}$, and now we compute $\hat{c}_i$ using the stored ciphertexts $c_{i,j}$ in T (but the values are equivalent).

Step (3'), in Game_3 is different from the original step (3) in Game_2. In the original step (3), we decrypted the forgery ciphertexts for $i \in [n] \setminus S$ and checked that they decrypt to the claimed output $e^* \stackrel{?}{=} \mathsf{Dec}_{sk'}(c_i^*)$. Let $e = g(b_1, \ldots, b_t)$ be the correct output of g on previously authenticated data. In an accepting type II forgery, we must have $e^* \neq e$ but the decryption of the "honest ciphertexts" will satisfy $\mathsf{HE.Dec}_{sk'}(\hat{c}_i) = e$. So it must be the case that $c_i^* \neq \hat{c}_i$ for all $i \in [n] \setminus S$ for any accepting type II forgery in Game_2. Therefore, any type II forgery that's accepting in Game_2 is also accepting in Game_3 and hence: $\Pr[\mathsf{Game}_3^{\mathcal{A}}(n) = 1] \geq \Pr[\mathsf{Game}_2^{\mathcal{A}}(n) = 1] \geq \mu(n) - \mathsf{negl}(n)$.

Game_4: We modify Game_3 so that, when answering *authentication queries*, the challenger computes all of the ciphertexts c_i (even for $i \in S$) as encryptions of the correct bit b in step (2) of the authentication procedure. In particular, the choice of S is ignored when answering authentication queries.

We claim that:

$$\Pr[\mathsf{Game}_4^{\mathcal{A}}(n) = 1] \geq \Pr[\mathsf{Game}_3^{\mathcal{A}}(n) = 1] - \mathsf{negl}(n) \geq \mu(n) - \mathsf{negl}(n). \quad (1)$$

This simply follows by the semantic security of the encryption scheme HE. Given challenge ciphertexts which either encrypt the attacker's bits b or 0, we can embed these into the authentication procedures for positions $i \in S$ and either simulate Game_3 or Game_4. We can efficiently determine if the output of the game is 1, since the decryption secret key sk' is never used in these games. Therefore, if the above didn't hold, the attacker $\mathcal{A}$ would break semantic security.

Negligible Advantage. We now claim that, information theoretically, $\Pr[\mathsf{Game}_4^{\mathcal{A}}(n) = 1] \leq 2^{-n}$. Together with equation (1), this shows that $\mu(n) \leq 2^{-n} + \mathsf{negl}(n) = \mathsf{negl}(n)$, as we wanted to show.

In Game_4, the choice of the set $S \subseteq [n]$ is not used at all when answering authentication queries and so we can think of the challenger as only picking the set S during verification. For any type II forgery $e^*, \mathcal{P}^*, \psi^* = (c_1^*, \ldots, c_n^*, \nu^*)$, let $c_1', \ldots, c_n'$ be the *"honest ciphertexts"* that would be included in an honestly generated tag ψ for the output of $\mathcal{P}^*$ (see description of Game_3). Let $S' := \{i \in [n] \ : \ c_i^* = c_i'\}$ be the indices on which the forged and honest ciphertexts match. The attacker only wins if steps (2'), (3') of verification pass, which only occurs if $S = S'$. But this only occurs with probability 2^{-n} over the random choice of S.

□

3.1 Fully Homomorphic Authenticator-Encryption

We can also extend homomorphic message authenticators to homomorphic *authenticator-encryption*. Given the secret key sk, it should be possible to *decrypt* the correct bit b from the tag authenticating it, but given only the evaluation key evk, the tags should not reveal any information about the authenticated bits.

We can allow decryption generically. Take any homomorphic authenticator scheme (KeyGen, Auth, Eval, Ver) and define $\mathsf{VerDec}(\mathcal{P}, \psi) \to \{0, 1, \mathtt{reject}\}$ as follows: run $\mathsf{Ver}(e, \mathcal{P}, \psi)$ for both choices of $e \in \{0, 1\}$ and, if exactly one of the runs is accepting, return the corresponding e, else return `reject`.

We notice that our specific construction of homomorphic authenticators already provides *chosen-plaintext-attack (CPA)* security for the above authenticator-encryption scheme. Even if the attacker gets evk and access to the authentication oracle $\mathsf{Auth}_{sk}(\cdot, \cdot)$, if he later sees a tag $\sigma \leftarrow \mathsf{Auth}_{sk}(b, \tau)$ for a fresh label τ, he cannot distinguish between the cases $b = 0$ and $b = 1$.

3.2 Security with Verification Queries?

The scheme presented in Section 3 only provides security *without* verification queries, and it remains an interesting open problem to construct fully homomorphic authenticators that allow for the stronger notion of security *with* verification queries. We make several observations here.

An Efficient Attack. Firstly, we note that there is an efficient attack against our basic scheme from Section 3, in the setting of security *with* verification queries.

The attacker gets evk and makes a single authentication query to get a tag $\sigma \leftarrow \mathsf{Auth}_{sk}(1, \tau)$, authenticating the bit $b = 1$ under some arbitrary label τ. We parse $\sigma = (c_1, \ldots, c_n, \nu)$.

The attacker then makes several verification queries whose aim is to learn the secret set $S \subseteq [n]$. It does so as follows: for each $i \in [n]$ he computes c_i' by adding an encryption of 0 to c_i (or performing any homomorphic operation that changes the ciphertext while preserving the plaintext) and sets the modified tag σ_i to be the same as σ, but with c_i replaced by c_i'. Then, for each $i \in [n]$, the attacker makes a verification query $\mathsf{Ver}_{sk}(1, \mathcal{I}_\tau, \sigma_i)$ to test if the modified tag σ_i is valid. If the query rejects then the attacker guesses that $i \in S$ and otherwise guesses $i \notin S$. With overwhelming probability the attacker correctly recovers the entire set S.

Now the attacker can construct a type II forgery for the identity program $\mathcal{I}_\tau$, claiming that its value is 0 (recall, that we previously authenticated $b = 1$ under the label τ). To do so, the attacker takes the tag σ and, for $i \notin S$, replaces the ciphertexts c_i with fresh encryptions of 0. Let's call the resulting modified tag σ^*. Then it's easy to see that $(0, \mathcal{I}_\tau, \sigma^*)$ is a valid type II forgery, breaking the scheme.

Bounded Verification Queries. The above attack requires n verification queries to break the scheme. It is relatively easy to show that the scheme is secure against $O(\log(n))$ verification queries. In particular, the attacker only gets $O(\log(n))$ bits of information about the set S, which is not enough to break the scheme. Similarly, for any a-priori bound q, we can modify our scheme so that the tags contain $n + q$ ciphertexts, and then we get security against q verification queries.

Computation with Long Output. Our basic scheme considers homomorphic authentication for a program $\mathcal{P}$ with a 1-bit output. Of course, we can extend the scheme to authenticate a program with longer output, by simply authenticating each bit of the output separately. However, this means that the tag is proportional to the output size of the computation. A simple trick allows us to authenticate a long output of some program $\mathcal{P}$ with only a short tag which is independent of the output size. Instead of homomorphically authenticating the output of the program $\mathcal{P}$, we just authenticate the output of a program $H(\mathcal{P})$ which first computes the output y of $\mathcal{P}$ and then outputs $H(y)$ where H is a collision-resistant hash function. Since $H(\mathcal{P})$ has a short output even when $\mathcal{P}$ has a long output, the resulting tag is short. Moreover, by getting the short tag ψ computed as above and the long output y, the verifier can check $\mathsf{Ver}_{sk}(H(y), H(\mathcal{P}), \psi) = \texttt{accept}$ to ensure that y is the output of the computation $\mathcal{P}$ over previously authenticated data. It is easy to check that this preserves security.

4 Improving Verification Complexity

One of the main limitations of our homomorphic authenticator scheme from the previous section is that the complexity of the verification algorithm is no better than that of executing the program $\mathcal{P}$. Therefore, although the scheme saves on the *communication complexity* of transmitting the input data, it does not save on the *computation complexity* of executing $\mathcal{P}$. As we discussed in the introduction, works in the area of *delegating computation* obtain efficient verification whose complexity is independent (or at least much smaller than) the computation of $\mathcal{P}$, but require that the user knows the entire input x. We explore the idea of "marrying" these two techniques by *delegating* the computation required to *verify* the authentication tag in our homomorphic authenticator scheme.

Firstly, we notice that the verification procedure $\mathsf{Ver}_{sk}(e, \mathcal{P}, \psi)$ of our scheme as described in Section 3 has special structure. The only "expensive" computation (proportional to the complexity of the the program $\mathcal{P}$) is *independent* of the tag ψ. In particular, this computation uses the secret key sk and the program $\mathcal{P}$ to compute the output of the "hash-tree" ν' and the ciphertexts c'_i for $i \in S$ derived by evaluating $\mathcal{P}$ over the pseudorandom encryptions of 0. We call this computation $\texttt{Expensive}(\mathcal{P}, sk)$. Given the outputs of $\texttt{Expensive}(\mathcal{P}, sk)$, the rest of the verification procedure $\mathsf{Ver}_{sk}(e, \mathcal{P}, \psi)$ consists of simple comparisons and is incredibly efficient (independent of the complexity of $\mathcal{P}$). Therefore, we can delegate the computation of $\texttt{Expensive}(\mathcal{P}, sk)$ prior to knowing the tag ψ that needs to be verified. One issue is that the computation does depend on the secret key sk, which needs to be kept private. We note that we can always (generically) keep the input of a delegated computation private by encrypting it under an FHE scheme. In our context, we can encrypt the value sk under an independently chosen FHE key, and publish this ciphertext C_{sk} in the evaluation key. We can then delegate the computation $\texttt{Expensive}'(\mathcal{P}, C_{sk})$ which takes $\mathcal{P}$ as an input and homomorphically executed $\texttt{Expensive}(\mathcal{P}, sk)$.

We now explore the advantages of using the above approach with some concrete delegation schemes.

Using SNARGs. We can use succinct non-interactive arguments for polynomial-time computation (**P**-SNARGs). This primitive allows anyone to provide a short proof π certifying the correctness of an arbitrary polynomial-time computation $y = f(x)$, where f is a Turing Machine. The tuple (f, y, x, π) can be verified in some fixed polynomial-time $p(|x|, |y|, |f|)$, that only depends on the *description-length* of the machine f, but is independent of the *running time* of f. Given **P**-SNARGs, we get a completely non-interactive delegation of computation (assuming the computation has a short uniform description). Therefore, using the above approach, we get homomorphic authenticators satisfying our original non-interactive syntax and security, but also allowing efficient verification for programs $\mathcal{P}$ having a short uniform description. During *evaluation*, we simply also have the server compute $\texttt{Expensive}'(\mathcal{P}, C_{sk})$ and provide a SNARG proof π that it was done correctly. Recall that, in the introduction, we described a significantly simpler solution to the problem of efficiently verifiable homomorphic authenticators (and signatures) using succinct non-interactive argument of knowledge for all of **NP**, or **NP**-SNARKs. Therefore, the main advantage of the above technique is that now we only require **P**-SNARGs, which is a much weaker primitive. For example, if we instantiate the random-oracle CS-Proofs of Micali [30] with some cryptographic hash function, it may be more reasonable to assume that we get a **P**-SNARG, than it is to assume that we get an **NP**-SNARK. In particular, the former *is* a falsifiable assumption whereas the latter cannot be proved under any falsifiable assumption (see [25]).

Using Delegation with Pre-Processing. Alternatively, we can use the delegation techniques in the "pre-processing" model [21, 16, 2] to outsource the computation of $\texttt{Expensive}'(\mathcal{P}, C_{sk})$ where $\mathcal{P}$ is given as an input. This scheme will have many of the same advantages and disadvantages as the approach of using delegation with "pre-processing" directly to outsource the data (see a description of this latter approach in Section 1.3). In particular, in both approaches, the verification will be efficient but the scheme will now require one round of interaction, where the user needs to create a challenge $chall(\mathcal{P})$ for the computation $\mathcal{P}$ that she wants to verify. The main advantages of our approach, combining delegation with homomorphic authenticators, over a direct delegation-based scheme is the following. When using delegation directly, the user needs to outsource all of the data in one shot and remember some short partial information about it; now the user can arbitrarily authenticate fresh labeled data "on the fly" and verify computations over all of it using a single independent secret key.

5 Conclusions

In this work we give the first solution to fully homomorphic message authenticators, allowing a user to verify computations over previously authenticated data. The authentication tag is short, independent of the size of the authenticated input to the computation. Our work leaves many interesting open questions. Perhaps the most ambitious one is to construct fully homomorphic signatures with *public verification.* Less ambitiously, construct fully homomorphic authenticators that allow an unbounded number of verification queries. Lastly, it would be interesting to improve the verification efficiency of our construction. One pressing question is to make the verification complexity independent of the complexity of the program $\mathcal{P}$ while maintaining all of the advantages of our scheme (standard assumptions, no interaction). But a less ambitious, still interesting question is to just reduce the tag size from $O(n)$ ciphertexts to something smaller, say a single ciphertext.

6 Acknowledgement

We'd like to thank Craig Gentry for his valuable comments. In particular, a prior version of this work included a speculative suggestion for achieving security with verification queries via "randomness-homomorphic encryption"; Craig pointed out that this latter primitive cannot exist.

References

[1] S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 292–305. Springer, June 2009.

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

[3] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 598–609. ACM Press, Oct. 2007.

[4] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 319–333. Springer, Dec. 2009.

[5] N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 17–34. Springer, Mar. 2011.

[6] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In Rogaway [32], pages 111–131.

[7] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Goldwasser [26], pages 326–349.

[8] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. Cryptology ePrint Archive, Report 2012/095, 2012. `http://eprint.iacr.org/`.

[9] D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Mar. 2009.

[10] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, May 2011.

[11] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16. Springer, Mar. 2011.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Goldwasser [26], pages 309–325.

[13] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In R. Ostrovsky, editor, *FOCS*, pages 97–106. IEEE, 2011.

[14] D. Catalano and D. Fiore. Practical homomorphic macs for arithmetic circuits. In *Eurocrypt*, 2013.

[15] D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In Fischlin et al. [19], pages 680–696.

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

[17] K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In Rogaway [32], pages 151–168.

[18] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 109–127. Springer, Mar. 2009.

[19] M. Fischlin, J. Buchmann, and M. Manulis, editors. *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*. Springer, 2012.

[20] D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Fischlin et al. [19], pages 697–714.

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

[22] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. Cryptology ePrint Archive, Report 2012/215, 2012. `http://eprint.iacr.org/`.

[23] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 142–160. Springer, May 2010.

[24] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[25] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[26] S. Goldwasser, editor. *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. ACM, 2012.

[27] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

[28] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Dec. 2010.

[29] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Feb. 2002.

[30] S. Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[31] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[32] P. Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[33] H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, Dec. 2008.

[34] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, May 2010.