Practical Homomorphic MACs for Arithmetic Circuits[⋆]

Dario Catalano[1] and Dario Fiore[2⋆⋆]

[1] Dipartimento di Matematica e Informatica, Università di Catania, Italy
catalano@dmi.unict.it
[2] IMDEA Software Institute, Madrid, Spain
dario.fiore@imdea.org

Abstract. Homomorphic message authenticators allow the holder of a (public) evaluation key to perform computations over previously authenticated data, in such a way that the produced tag σ can be used to certify the authenticity of the computation. More precisely, a user knowing the secret key sk used to authenticate the original data, can verify that σ authenticates the correct output of the computation. This primitive has been recently formalized by Gennaro and Wichs, who also showed how to realize it from fully homomorphic encryption. In this paper, we show new constructions of this primitive that, while supporting a smaller set of functionalities (i.e., polynomially-bounded arithmetic circuits as opposite to boolean ones), are much more efficient and easy to implement. Moreover, our schemes can tolerate any number of (malicious) verification queries. Our first construction relies on the sole assumption that one way functions exist, allows for arbitrary composition (i.e., outputs of previously authenticated computations can be used as inputs for new ones) but has the drawback that the size of the produced tags grows with the degree of the circuit. Our second solution, relying on the D-Diffie-Hellman Inversion assumption, offers somewhat orthogonal features as it allows for very short tags (one single group element!) but poses some restrictions on the composition side.

1 Introduction

Cloud Computing allows a user to outsource his data to remote service providers in such a way that he can later access the data from multiple platforms (e.g., his desktop at work, his laptop, his smartphone, etc.), and virtually from everywhere. Moreover, using this paradigm, even clients with very limited storage capacity (e.g., smart phones) can have access "on demand" to very large amounts of data. Having access to the outsourced data does not necessarily mean only to retrieve such data. Indeed, a user may wish to perform a computation on (a subset of) the outsourced data, and this too can be delegated to the service provider. These and other benefits are the key success of Cloud Computing. The paradigm, however, raises security concerns essentially because cloud providers cannot always be trusted. One problem is related to preserving the privacy of the outsourced data. This question has been successfully addressed by the recent work on fully homomorphic encryption [24]. The second question deals with enforcing the *authenticity* of the computations performed on the outsourced data, and is the focus of this work. In a nutshell, this problem can be described as follows. Assume that a client outsources a collection of data $m_1, \dots, m_n$ to a server, and later asks the server to run a program $\mathcal{P}$ over $(m_1, \dots, m_n)$. The server computes $m \leftarrow \mathcal{P}(m_1, \dots, m_n)$ and sends m to the client. The problem here is that the client wants to be sure that m is the value obtained by running $\mathcal{P}$ on its own data. A trivial solution would be to have the server send $m_1, \dots, m_n$ to the client, who can then compute/check $m = \mathcal{P}(m_1, \dots, m_n)$ by itself. This however vanishes the advantages of the outsourcing and is too costly in terms of

[⋆] This paper is based on an earlier article which appears in the proceedings of EUROCRYPT 2013 [12], © IACR 2013.

[⋆⋆] Work done while postdoc at NYU.

bandwidth. Therefore, the main goal here is to find solutions in which the server can authenticate the output of the computation by sending some value whose size is much shorter than $m_1, \ldots, m_n$. Such property is also motivated by the fact that, in spite of the continuous progress in increasing the computational power of small devices, bandwidth (especially in mobile data connections) seems to remain the most serious and expensive bottleneck.

The research community has recently put a notable effort in developing new cryptographic tools that can help in solving this and related problems. It is the case, for instance, for works on verifiable computation [28,29,26,21,16,3] and memory delegation [17].

Another line of research has explored the idea of enabling computation on authenticated data [2] by means of homomorphic authentication primitives.

In the public key setting Boneh and Freeman introduced the notion of *(fully) homomorphic signatures* [9]. Roughly speaking, a homomorphic signature allows a user to generate signatures $\sigma_1, \ldots, \sigma_n$ on messages $m_1, \ldots, m_n$ so that later anyone (without knowledge of the signing key) can compute a signature σ that is valid for the value $m = f(m_1, \ldots, m_n)$. Boneh and Freeman also showed a realization of homomorphic signatures for bounded (constant) degree polynomials, from ideal lattices.

Very recently, Gennaro and Wichs proposed, formally defined and constructed the secret-key analogue of homomorphic signatures, that is *homomorphic message authenticators* (homomorphic MACs, for short) [23]. Their construction makes use of fully homomorphic encryption and allows to evaluate every circuit.

In this work, we continue the study of homomorphic MACs and propose new constructions which, while less general than that given in [23], are much more efficient.

Homomorphic Message Authenticators. Informally, a homomorphic MAC scheme enables a user to use his secret key for generating a tag σ which authenticates a message m so that later, given a set of tags $\sigma_1, \ldots, \sigma_n$ authenticating messages $m_1, \ldots, m_n$ respectively, anyone can homomorphically execute a program $\mathcal{P}$ over $(\sigma_1, \ldots, \sigma_n)$ to generate a short tag σ that authenticates m as the output of $\mathcal{P}(m_1, \ldots, m_n)$.

Given such a primitive, it is not hard to imagine how it can be employed to solve the problem of verifying computations on outsourced data. However, the above description needs some refinements, in particular to explain what means to authenticate a message as the output of a program. To do this Gennaro and Wichs introduce the notion of *labeled data and programs*. The *label* τ of a data m is some binary string τ chosen by the user to authenticate m, i.e., $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \tau, m)$. One can think of labels as some indexing of the data. For example, assume that a company outsources a database with informations on its customers, in which each column contains a different attribute (e.g., age, expended amount, etc.). Then, to authenticate the "age" column of the database the user can define a label "(age, i)" for the age value in record i. On the other hand, a *labeled program* $\mathcal{P}$ is defined by a circuit f and a set of labels $\tau_1, \ldots, \tau_n$, one for each input wire of f. This can be seen as a way to specify on which inputs the circuit should be evaluated upon, without knowing the input values themselves. So, given a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and a set of tags $\sigma_1, \ldots, \sigma_n$ that authenticate messages m_i under label τ_i, anyone can run the homomorphic evaluation algorithm $\sigma \leftarrow \mathsf{Eval}(\mathcal{P}, \sigma_1, \ldots, \sigma_n)$ whose output σ will authenticate $m = \mathcal{P}(m_1, \ldots, m_n)$. Precisely, the secret-key verification algorithm takes as input a triple $(m, \mathcal{P}, \sigma)$ and verifies that m is the output of the program $\mathcal{P}$ run on some previously authenticated and labeled messages, without knowing such messages themselves.

Informally, homomorphic MACs are secure if any adversary who can adaptively query tags for messages of its choice cannot produce a valid tag σ that authenticates m as the output of $\mathcal{P}$ unless σ can be honestly computed by applying Eval on the queried tags.

Homomorphic MACs are also required to be *succinct*. Informally, succinctness requires that the output of $\mathcal{P}$ run over (previously) authenticated data can be certified with significantly less communication than that of sending the original inputs.

Another property one might want from homomorphic MACs is *composability*, which allows to combine tags authenticating previous computations to create a tag that authenticates a composition of such computations. More precisely, given tags $\sigma_1, \ldots, \sigma_t$ that authenticate $m_1, \ldots, m_t$ as the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$ respectively, composability allows to further compute $\sigma \leftarrow \mathsf{Eval}(\mathcal{P}, \sigma_1, \ldots, \sigma_t)$ which authenticates $m = \mathcal{P}(m_1, \ldots, m_t)$ as the output of $\mathcal{P}^*$, the composed program obtained by running $\mathcal{P}$ on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$.

1.1 Our Contribution

In this paper we propose the first practically efficient constructions of homomorphic MACs. The most attractive feature of our schemes is that they are efficient, simple to implement and rely on well studied assumptions. Moreover, they are secure against PPT adversaries that can make an unbounded number of verification queries, as opposite to the construction in [23] that supports only an a-priori bounded number of verification queries (see next section for more details about this). On the negative side our solution works only for functionalities that can be expressed as arithmetic circuits with certain additional restrictions that we describe below.

Our first construction is surprisingly simple and relies only on the existence of pseudorandom functions. While it offers arbitrary composition, it does not achieve full succinctness. More precisely, the size of the authentication tags grows with the degree d of the circuit[3], and thus we are able to guarantee succinct authenticators only when d is smaller than the input size n.

Our second construction enjoys succinct, constant-size tags (just one group element!) but only supports a limited form of composition . More precisely, for a fixed bound D (polynomial in the security parameter) the scheme allows to evaluate any arithmetic circuit of degree $d \leq D$. In general, the evaluation has to be done in a "single shot", that is the authentication tags obtained from the Eval algorithm cannot be used again to be composed with other tags. However, we interestingly show that the scheme achieves what we call *local composition.* The idea is that one can keep locally a non-succinct version of the tag that allows for arbitrary composition. Next, when it comes to send an authentication tag to the verifier, one can securely compress such large tag in a very compact one of constant-size. We prove the security of our second construction under the D-Diffie Hellman Inversion assumption [11,30] (where D is the bound on the maximal circuit's degree supported by the scheme).

Succinct Tags and Composition. Even though our solutions do not achieve succinctness and composition at the same time, we argue that these limitations might not be too relevant in many real life scenarios. First, we notice that several interesting functions and statistics (e.g., the standard deviation function) can be represented by constant-degree polynomials. In such a case, our first construction perfectly fits the bill as it is efficient, simple to implement and produces constant-size tags (and, of course, it only requires the existence of a PRF to be proved secure).

[3] Informally, the degree of an arithmetic circuit is related to the degree of the polynomial computed by the circuit (see next section for more details).

For the case of polynomials of large degree d (i.e., d polynomial in the security parameter), our scheme fits well in those applications where composition is not needed. Think for example of the application described at the beginning of this section. There, if the server just runs $m \leftarrow \mathcal{P}(m_1, \ldots, m_n)$ on the client's data, using our second construction it can produce a succinct tag that authenticates m as $\mathcal{P}$'s output, and this tag is only one group element.

Finally, in applications where composition is needed but does not involve different parties, the notion of local composition achieved by our second scheme still allows to save in bandwidth and to (locally) compose tags of partial computations.

Overview of Our Techniques. The main idea behind our construction is a "re-interpretation" of some classical techniques for information-theoretic MACs. The authentication tag of a message $m \in \mathbb{Z}_p$ with label τ is a degree-1 polynomial $y(z) \in \mathbb{Z}_p[z]$ that evaluates to m on the point 0, and to r_τ on a random point x (i.e., $y(0) = m$ and $y(x) = r_\tau$). Here $r_\tau = F_K(\tau)$ is a pseudorandom value, unique per each label, defined by the PRF, while x is the secret key. If we do not care about the homomorphic property and we assume that each r_τ is truly random, then this is a secure information-theoretic MAC. Now, the basic observation that allows to show the homomorphic property is the following. Let f be an arithmetic circuit and assume to evaluate the circuit over the tags (i.e., over these polynomials $y(z)$) as follows: for every additive gate we compute the addition of the two input polynomials, and for every multiplicative gate we compute the multiplication of them (i.e., the convolution of their coefficients). Now, we observe that these operations are naturally *homomorphic* with respect to the evaluation of the polynomial in every point. In particular, if we have two tags $y^{(1)}$ and $y^{(2)}$ (i.e., we are given only the coefficients of these polynomials) such that $y^{(1)}(0) = m_1$ and $y^{(2)}(0) = m_2$, then for $y = y^{(1)} + y^{(2)}$ (resp. $y = y^{(1)} * y^{(2)}$) we clearly obtain $y(0) = m_1 + m_2$ (resp. $y(0) = m_1 \cdot m_2$). The same holds for its evaluation at the random point x, i.e., $y(x) = r_{\tau_1} + r_{\tau_2}$ (resp. $y(x) = r_{\tau_1} \cdot r_{\tau_2}$). By extending this argument to the evaluation of the entire circuit f, this allows to verify a tag y for a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and a message m, by simply checking that $m = y(0)$ and $f(r_{\tau_1}, \ldots, r_{\tau_n}) = y(x)$, where $r_{\tau_i} = F_K(\tau_i)$.

A drawback of this construction is that the tag's size grows linearly with the degree of the evaluated circuit f. The reason is that the above homomorphic evaluation increases the degree of the "tag polynomial" y at every multiplication gate. This is why this MAC fails in achieving the succinctness property when the degree d becomes greater than the input size n of the circuit.

Our second construction overcomes this drawback as follows. First, the evaluation algorithm computes a tag $y = (y_0, \ldots, y_d)$ as before, and then it "accumulates" these coefficients in a single group element $\Lambda = \prod_{i=1}^{d} (g^{x^i})^{y_i}$. Verification will check that $g^{f(r_{\tau_1}, \ldots, r_{\tau_n})} = g^m \cdot \Lambda$. If Λ is computed correctly, then $\Lambda = g^{y(x) - y(0)}$, and thus one can easily see why correctness holds. The need to resort to the $(D-1)$-Diffie Hellman Inversion assumption[4], comes from the fact that, in order to perform the evaluation procedure correctly, the values $g^x, g^{x^2}, \ldots, g^{x^D}$ need to be published as part of the evaluation key ek. Once a tag of the Λ form is created, it can be composed with other tags of the same form only for additions but not for multiplications. To satisfy partial composition, the idea is that one can keep locally the large version of the tag consisting of the coefficients $y_0, \ldots, y_d$, and always send to the verifier its compact version $\Lambda = \prod_{i=1}^{d} (g^{x^i})^{y_i}$. In Appendix A we also show an extension of this scheme that, by using bilinear pairings, allows to further compute an additional level of multiplications and unbounded additions on tags of the Λ form.

[4] Very briefly, this assumption states that it is computationally infeasible to compute $g^{1/x}$, given $g, g^x, g^{x^2}, \ldots, g^{x^{D-1}}$

1.2 Related Work

Homomorphic Message Authenticators and Signatures. Recently, many papers considered the problem of realizing homomorphic (mostly linear) authenticators either in the symmetric setting (MAC) or in the asymmetric one (signatures). This line of research has been initiated by the work of Johnson et al. [27] and became very popular in recent years because of the important application to linear network coding. Efficient solutions for this latter application have been proposed both in the random oracle [8,22,10,13] and in the standard model [1,4,14,15,20]. Linearly-homomorphic message authenticators have been considered also for proofs of retrievability for outsourced storage [33]. Only two works, however, consider the problem of realizing solutions for more complex functionalities (i.e., beyond linear).

Boneh and Freeman defined the notion of (fully) homomorphic signatures and showed a realization for bounded (constant) degree polynomials, from ideal lattices [9]. With respect to our work this solution has the obvious advantage of allowing for public verifiability. On the negative side it is not truly practical and the bound on the degree of the supported polynomials is more stringent than in our case (as they can support only polynomials of constant degree).

Closer to our setting is the recent work of Gennaro and Wichs [23] where fully homomorphic MACs are introduced, formally defined and constructed. The solution given there supports a wider class of functionalities with respect to ours, and it allows to achieve succinct tags and composability at the same time. Their tags have size $\mu(\lambda) = \mathsf{poly}(\lambda)$ where λ is the security parameter, and thus they are asymptotically succinct as long as the circuit's input size n is greater than $\mu(\lambda)$. Despite its nice properties, the proposed construction seems unfortunately far from being truly practical as it relies on fully homomorphic encryption. Moreover, it is proven secure only for a bounded and a-priori fixed number of verification queries[5], meaning with this that the scheme becomes insecure if the verifier leaks information on whether it accepts/rejects tags.

Succinct Non-interactive Arguments of Knowledge. The problem of realizing homomorphic signatures can be solved in theory using *Succinct Non-interactive Arguments of Knowledge* (SNARKs) [6]. In a nutshell, given any NP statement a SNARK allows to construct a succinct argument that can be used to prove knowledge of the corresponding witness. The nice feature of SNARKs is that the size of the argument is independent of the size of both the statement and the witness. The bad thing about SNARKs is that they are not very efficient (or at least not nearly as practical as we require) and require either the random oracle model [29] or non standard, non-falsifiable assumptions [25]. Moreover, SNARK-based solutions seem to allow for only very limited composability [35,7].

Other Related Work. The notion of homomorphic authenticators is also (somewhat) related to the notion of verifiable computation [28,29,26,21,16,3,5,31,19]. There, one wants to delegate a computationally heavy task to a remote server while keeping the ability to verify the result in a very efficient way. While the two primitives might seem quite different at first, one can reinterpret some of the results on verifiable computation in our setting. The resulting solutions however present several limitations that make them of limited practical interest compared to homomorphic authenticators. We refer the reader to [23] for a nice discussion about this.

Homomorphic authenticators are also related to memory delegation [17]. This primitive allows a client to outsource large amounts of data to a server so that he can later verify computations

[5] More precisely, their basic construction cannot support verification queries at all. This can be extended to allow for some fixed a-priori number of queries q at the cost of increasing by $O(q)$ the size of the tag.

on the data. The advantage of this approach over ours is that it offers an efficient verification procedure, and it supports a dynamic memory in which the client can update the outsourced data. However, current (non-interactive) realizations of memory delegation, in the standard model, are rather inefficient and require the user to keep a state. Moreover, in known constructions, efficient verification comes at the price of an offline phase where the runtime of both the delegator and the server depends polynomially on the size of the memory.

Organization. The paper is organized as follows. In Section 2 we provide a background and relevant definitions of arithmetic circuits and homomorphic authenticators. Section 3 describes our first construction from PRFs while our second compact construction is given in Section 4. Finally, we defer to Appendix A an extension of the second construction that allows to further compute one level of multiplication and unbounded additions on tags previously obtained from the homomorphic evaluation.

2 Background and Definitions

Arithmetic Circuits. Here we provide a very brief overview of arithmetic circuits. The interested reader is referred to [34] for a more detailed treatment of the subject.

An arithmetic circuit over a field $\mathbb{F}$ and a set of variables $X = \{\tau_1 \dots \tau_n\}$, is a directed acyclic graph with the following properties. Each node in the graph is called *gate*. Gates with in-degree 0 are called *input* gates (or input nodes) while gates with out-degree 0 are called *output* gates. Each input gate is labeled by either a *variable* or a *constant*. Variable input nodes are labeled with binary strings $\tau_1, \dots, \tau_n$, and can take arbitrary values in $\mathbb{F}$. A constant input node instead is labeled with some constant c and it can take only some fixed value $c \in \mathbb{F}$.

Gates with in-degree and out-degree greater than 0 are called *internal* gates. Each internal gate is labeled with an arithmetic operation symbol. Gates labeled with × are called product gates, while gates labeled with + are called sum gates. In this paper, we consider circuits with a single output node and where the in-degree of each internal gate is 2. The *size* of the circuit is the number of its gates. The *depth* of the circuit is the length of the longest path from input to output.

Arithmetic circuits evaluate polynomials in the following way. Input gates compute the polynomial defined by their labels. Sum gates compute the polynomial obtained by the sum of the (two) polynomials on their incoming wires. Product gates compute the product of the two polynomials on their incoming wires. The output of the circuit is the value contained on the outgoing wire of the output gate. The *degree of a gate* is defined as the total degree of the polynomial computed by that gate. The *degree of a circuit* is defined as the maximal degree of the gates in the circuit.

We stress that arithmetic circuits should be seen as computing *specific* polynomials in $\mathbb{F}[X]$ rather than functions from $\mathbb{F}^{|X|}$ to $\mathbb{F}$. In other words, when studying arithmetic circuits one is interested in the formal computation of polynomials rather than the functions that these polynomials define[6].

In this paper we restrict our interest to families of polynomials $\{f_n\}$ over $\mathbb{F}$ which have *polynomially bounded degree*, meaning with this that both the number of variables and the degree of f_n are bounded by some polynomial $p(n)$. The class **VP** (also known as $\mathbf{AlgP}_{/\mathbf{poly}}$) contains all such polynomials. More precisely it contains all polynomially bounded degree families of polynomials that are computable by arithmetic circuits of polynomial size and degree.

[6] While, in general, every polynomial defines a unique function the converse is not true as a function may be expressed as a polynomial in several ways.

2.1 Homomorphic Message Authenticators

Labeled Programs. First, we recall the notion of labeled programs introduced by Gennaro and Wichs in [23]. A labeled program $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_n)$ where $f : \mathbb{F}^n \to \mathbb{F}$ is a circuit, and the binary strings $\tau_1, \ldots, \tau_n \in \{0,1\}^*$ are the *labels* of the input nodes of f. Given some labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathbb{F}^t \to \mathbb{F}$ it is possible to define the *composed program* $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$ which consists in evaluating a circuit g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$ respectively. The labeled inputs of $\mathcal{P}^*$ are all distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, i.e., all inputs with the same label are put together in a single input of the new program. We denote with $\mathcal{I}_\tau = (g_{id}, \tau)$ the *identity program* with label τ where g_{id} is the canonical identity function and $\tau \in \{0,1\}^*$ is some input label. Finally, we notice that any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

While Gennaro and Wichs [23] defined labeled programs for Boolean circuits (i.e., $f : \{0,1\}^n \to \{0,1\}$), here we consider its extension to the case of arithmetic circuits $f : \mathbb{F}^n \to \mathbb{F}$ where $\mathbb{F}$ is some finite field, e.g., $\mathbb{Z}_p$ for a prime p.

Homomorphic Authenticator Scheme. A homomorphic message authenticator scheme HomMAC is a 4-tuple of algorithms working as follows:

$\mathsf{KeyGen}(1^\lambda)$**:** on input the security parameter λ, the key generation algorithm outputs a secret key sk and a public evaluation key ek.

$\mathsf{Auth}(\mathsf{sk}, \tau, m)$**:** given the secret key sk, an input-label τ and a message $m \in \mathcal{M}$, it outputs a tag σ.

$\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$**:** given the secret key sk, a message $m \in \mathcal{M}$, a program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and a tag σ, the verification algorithm outputs 0 (reject) or 1 (accept).

$\mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma})$**:** on input the evaluation key ek, a circuit $f : \mathcal{M}^n \to \mathcal{M}$ and a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$, the evaluation algorithm outputs a new tag σ.

Authentication Correctness. Intuitively, a homomorphic MAC satisfies this property if any tag σ generated by the algorithm $\mathsf{Auth}(\mathsf{sk}, \tau, m)$ authenticates with respect to the identity program $\mathcal{I}_\tau$. Formally, we require that for any message $m \in \mathcal{M}$, all keys $(\mathsf{sk}, \mathsf{ek}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, any label $\tau \in \{0,1\}^*$, and any tag $\sigma \xleftarrow{\$} \mathsf{Auth}(\mathsf{sk}, \tau, m)$, it holds: $\Pr[\mathsf{Ver}(\mathsf{sk}, m, \mathcal{I}_\tau, \sigma) = 1] = 1$.

Evaluation Correctness. Informally, this property states that if the evaluation algorithm is given a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ such that each σ_i authenticates some message m_i as the output of a labeled program $\mathcal{P}_i$, then the tag σ produced by Eval must authenticate $f(m_1, \ldots, m_n)$ as the output of the composed program $f(\mathcal{P}_1, \ldots, \mathcal{P}_n)$.

More formally, let us fix a pair of keys $(\mathsf{sk}, \mathsf{ek}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, a function $g : \mathcal{M}^t \to \mathcal{M}$ and any set of message/program/tag triples $\{(m_i, \mathcal{P}_i, \sigma_i)\}_{i=1}^t$ such that $\mathsf{Ver}(\mathsf{sk}, m_i, \mathcal{P}_i, \sigma_i) = 1$. If $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, (\sigma_1, \ldots, \sigma_t))$, then it must hold: $\mathsf{Ver}(\mathsf{sk}, m^*, \mathcal{P}^*, \sigma^*) = 1$.

Succinctness. The size of a tag is bounded by some fixed polynomial in the security parameter, that is independent of the number of inputs taken by the evaluated circuit.

Security. Let HomMAC be a homomorphic MAC scheme as defined above. Consider the following experiment $\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC}}(\lambda)$ between a challenger and an adversary $\mathcal{A}$ against HomMAC:

Setup The challenger generates $(\mathsf{sk}, \mathsf{ek}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ and gives ek to $\mathcal{A}$. It also initializes a list $T = \emptyset$.

Authentication queries The adversary can adaptively ask for tags on label-message pairs of its choice. Given a query (τ, m), if there is some $(\tau, \cdot) \in T$ (i.e., the label was already queried), then the challenger ignores the query. Otherwise, it computes $\sigma \xleftarrow{\$} \mathsf{Auth}(\mathsf{sk}, \tau, m)$, returns σ to $\mathcal{A}$ and updates the list $T = T \cup (\tau, m)$. If $(\tau, m) \in T$ (i.e., the query was previously made), then the challenger replies with the same tag generated before.

Verification queries The adversary is also given access to a verification oracle. Namely, $\mathcal{A}$ can submit a query $(m, \mathcal{P}, \sigma)$ and the challenger replies with the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$.

Forgery At some point the adversary is supposed to output a forgery $(m^*, \mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*), \sigma^*)$. Notice that such tuple can be returned by $\mathcal{A}$ also as a verification query $(m^*, \mathcal{P}^*, \sigma^*)$.

Before describing the outcome of this experiment, we define the notion of well defined program with respect to a list T. Informally, there are two ways for a program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ to be well defined. Either all the τ_i^*s are in T or, if there are labels τ_i^* not in T, then the inputs associated with such labels are somewhat "ignored" by f^* when computing the output. In other words input corresponding to labels not in T do not affect the behavior of f^* in any way.

More formally, we say that a labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is *well defined on* T if either one of the following two cases occurs:

1. there exists $i \in \{1, \ldots, n\}$ such that $(\tau_i^*, \cdot) \notin T$ (i.e., $\mathcal{A}$ never asked an authentication query with label τ_i^*), and $f^*(\{m_j\}_{(\tau_j, m_j) \in T} \cup \{\tilde{m}_j\}_{(\tau_j, \cdot) \notin T})$ outputs the same value for all possible choices of $\tilde{m}_j \in \mathcal{M}$;
2. T contains tuples $(\tau_1^*, m_1), \ldots, (\tau_n^*, m_n)$, for some messages $m_1, \ldots, m_n$.

The experiment $\mathsf{HomUF{-}CMA}$ outputs 1 if and only if $\mathsf{Ver}(\mathsf{sk}, m^*, \mathcal{P}^*, \sigma^*) = 1$ and one of the following conditions holds:

- *Type 1 Forgery:* $\mathcal{P}^*$ is not well-defined on T.
- *Type 2 Forgery:* $\mathcal{P}^*$ is well defined on T and $m^* \neq f^*(\{m_j\}_{(\tau_j, m_j) \in T})$, i.e., m^* is not the correct output of the labeled program $\mathcal{P}^*$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$.

We say that a homomorphic MAC scheme HomMAC is *secure* if for every PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A}, \mathsf{HomMAC}}(\lambda) = 1]$ is negligible.

Remark 1 (Comments on our definition). First, we observe that our definition explicitly disallow the possibility of re-using a label to authenticate more than one value. Essentially, this is a way to uniquely keep track of the authenticated inputs. We notice that such restriction is implicitly present in the Gennaro-Wichs construction as well as in all previous works on homomorphic signatures.

Second, the notion of well defined programs aims at capturing, in a formal way, which tuples generated by the adversary should be considered as forgeries. The catch here is that, since we are dealing with a homomorphic primitive, we should be able to differentiate MACs produced by Eval from MACs generated in some other, possibly malicious, way. Notice, however, that even maliciously generated MACs should not necessarily be considered as forgeries. This is because, in our setting, the adversary can trivially modify a circuit C she is allowed to evaluate by adding dummy gates and inputs that are simply ignored in the evaluation of the modified circuit (i.e., the new circuit is semantically equivalent to C). This last case does not constitute an infringement of our security requirements. Our notion of well defined program $\mathcal{P}$ captures exactly this: either $\mathcal{P}$ is run on legal (i.e. in T) inputs only, or, if this is not the case, those inputs not in T do not affect the computation in any way.

Finally, we observe that for arbitrary computations checking whether a program is well defined may not be efficiently computable. In particular, the difficult task is to check the first condition, i.e., whether a program always outputs the same value for all possible choices of the inputs that are not in T. However, for the case of arithmetic circuits in (exponentially) large fields and of polynomial degree this check can be efficiently performed by using probabilistic polynomial identity testing [18,36,32]. Below we state a simple proposition that shows how to do the testing for arithmetic circuits of degree d, over a finite field of order p such that $d/p < 1/2$.[7]

Proposition 1. *Let $\lambda, n \in \mathbb{N}$ and let $\mathcal{F}$ be the class of arithmetic circuits $f : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$ of order p and such that the degree of f is at most d, with $\frac{d}{p} < \frac{1}{2}$. Then, there exists a probabilistic algorithm that for any given $f \in \mathcal{F}$, decides if there exists $y \in \mathbb{F}$ such that $f(\boldsymbol{u}) = y, \forall \boldsymbol{u} \in \mathbb{F}^n$ (i.e., if f is constant) and is correct with probability at least $1 - 2^{-\lambda}$.*

Proof. The algorithm first samples $\lambda+1$ tuples $\{\boldsymbol{u}_i\}_{i=0}^{\lambda}$ uniformly at random in $\mathbb{F}^n$. Next, if $f(\boldsymbol{u}_0) = \cdots = f(\boldsymbol{u}_\lambda)$ the algorithm says "constant", otherwise it says "non-constant". If f is really constant, then it is easy to see that this algorithm is correct with probability 1. In the case when f is not constant, then the probability that the algorithm is wrong is essentially $\Pr[f(\boldsymbol{u}_0) = \cdots = f(\boldsymbol{u}_\lambda)]$ over the random choices of all the $\boldsymbol{u}_i$'s. Since the samples are independent, this probability can be upper bounded by $(\Pr_{\boldsymbol{u}_i \overset{\$}{\leftarrow} \mathbb{F}^n}[f(\boldsymbol{u}_i) = y_0 | y_0 = f(\boldsymbol{u}_0)])^\lambda \le (\frac{d}{p})^\lambda \le 2^{-\lambda}$, where the upper bound by d/p follows from the Schwartz-Zippel Lemma [18,36,32]. □

Remark 2 (Relations with previous definitions). Our definition is very similar to that proposed by Gennaro and Wichs in [23] except for two modifications. First, we explicitly allow the adversary to query the verification oracle. Second, we adopt a definition of forgery slightly weaker than that in [23]. More precisely, Gennaro and Wichs define Type 1 forgeries as ones where at least one new label is present. Type 2 forgeries, on the other hand, contain only labels that have been already queried, but m^* is not the correct output of the program when executed on the previously queried inputs.

Notice that our notion becomes equivalent to that given in [23] by simply changing the definition of "well defined program" so that $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is said well defined on T if $(\tau_i, m_i) \in T$ $\forall i = 1, \ldots n$. The difference between the two definitions is that, as we explained above, we do not consider forgeries all those tuples where "fresh" labels (i.e. labels not in T) do not contribute to the output of the program.

Even though our security definition is weaker than the one in [23], we stress that it is perfectly meaningful for the notion of homomorphic MAC. Indeed, we are still excluding from forgeries all those MACs that can be trivially computed by the adversary from what it queried during the game.

On a technical level, our definition of forgery is inspired by the security definition recently proposed by Freeman for homomorphic signatures [20], except that in our case we do not consider the notion of data set.

3 Our Homomorphic MAC from OWFs

In this section we propose our first construction of homomorphic MACs whose security relies only on a pseudo-random function (and thus on one-way functions). The scheme is simple and efficient

[7] The same argument can be actually extended to $d/p < 1/c$ for some constant c.

and allows to homomorphically evaluate arithmetic circuits $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ for a prime p of roughly λ bits, where λ is the security parameter.

Our Scheme. In our construction we restrict to circuits whose additive gates do not get inputs labeled by constants. This can be done without loss of generality as, when needed, one can use an equivalent circuit in which there is a special variable/label for the value 1, and can publish the MAC of 1.

The description of our scheme follows.

$\mathsf{KeyGen}(1^\lambda)$. Let p be a prime of roughly λ bits. Choose a seed K of a pseudorandom function $F_K : \{0,1\}^* \to \mathbb{Z}_p$ and a random value $x \overset{\$}{\leftarrow} \mathbb{Z}_p$. Output $\mathsf{sk} = (K, x), \mathsf{ek} = p$ and let the message space $\mathcal{M}$ be $\mathbb{Z}_p$.

$\mathsf{Auth}(\mathsf{sk}, \tau, m)$. To authenticate a message $m \in \mathbb{Z}_p$ with label $\tau \in \{0,1\}^\lambda$, compute $r_\tau = F_K(\tau)$, set $y_0 = m$, $y_1 = (r_\tau - m)/x \bmod p$ and output $\sigma = (y_0, y_1)$. Basically, y_0, y_1 are the coefficients of a degree-1 polynomial $y(z)$ with the special property that it evaluates to m on the point 0 ($y(0) = m$), and it evaluates to r_τ on a hidden random point x ($y(x) = r_\tau$).

In our construction we will interpret tags σ as polynomials $y \in \mathbb{Z}_p[z]$ of degree $d \geq 1$ in some (unknown) variable z, i.e., $y(z) = \sum_i y_i z^i$.

$\mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma})$. The homomorphic evaluation algorithm takes as input the evaluation key $\mathsf{ek} = p$, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$.

Intuitively, Eval consists in evaluating the circuit f on the tags $\sigma_1, \ldots, \sigma_n$ instead of evaluating it on messages. However, since the values σ_i's are not messages in $\mathbb{Z}_p$, but rather are polynomials $y^{(i)} \in \mathbb{Z}_p[z]$, we need to specify how this evaluation is carried through.

Eval proceeds gate-by-gate as follows. At each gate g, given two tags σ_1, σ_2 (or a tag σ_1 and a constant $c \in \mathbb{Z}_p$), it runs the algorithm $\sigma \leftarrow \mathsf{GateEval}(\mathsf{ek}, g, \sigma_1, \sigma_2)$ described below that returns a new tag σ, which is in turn passed on as input to the next gate in the circuit.

When the computation reaches the last gate of the circuit f, Eval outputs the tag vector σ obtained by running $\mathsf{GateEval}$ on such last gate.

To complete the description of Eval we describe the subroutine $\mathsf{GateEval}$.

- $\mathsf{GateEval}(\mathsf{ek}, g, \sigma_1, \sigma_2)$. Let $\sigma_i = \boldsymbol{y}^{(i)} = (y_0^{(i)}, \ldots, y_{d_i}^{(i)})$ for $i = 1, 2$ and $d_i \geq 1$ (see below for the special case when one of the two inputs is a constant $c \in \mathbb{Z}_p$).

  If $g = +$, then:
  - let $d = \max(d_1, d_2)$. Here we assume without loss of generality that $d_1 \geq d_2$ (i.e., $d = d_1$).
  - Compute the coefficients $(y_0, \ldots, y_d)$ of the polynomial $y(z) = y^{(1)}(z) + y^{(2)}(z)$. This can be efficiently done by adding the two vectors of coefficients, $\boldsymbol{y} = \boldsymbol{y}^{(1)} + \boldsymbol{y}^{(2)}$ ($\boldsymbol{y}^{(2)}$ is eventually padded with zeroes in positions $d_1 ... d_2$).

  If $g = \times$, then:
  - let $d = d_1 + d_2$.
  - Compute the coefficients $(y_0, \ldots, y_d)$ of the polynomial $y(z) = y^{(1)}(z) * y^{(2)}(z)$ using the convolution operator $*$, i.e., $\forall k = 0, \ldots, d$, define $y_k = \sum_{i=0}^{k} y_i^{(1)} \cdot y_{k-i}^{(2)}$.

  If $g = \times$ and one of the two inputs, say σ_2, is a constant $c \in \mathbb{Z}_p$, then:
  - let $d = d_1$.
  - Compute the coefficients $(y_0, \ldots, y_d)$ of the polynomial $y(z) = c \cdot y^{(1)}(z)$.

  Return $\sigma = (y_0, \ldots, y_d)$.

As one can notice, the size of a tag grows only after the evaluation of a multiplication gate (where both inputs are not constants). It is not hard to see that after the homomorphic evaluation of a circuit f, it holds $|\sigma| = d + 1$, where d is the degree of f.

$\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$**.** Let $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ be a labeled program, $m \in \mathbb{Z}_p$ and $\sigma = (y_0, \ldots, y_d)$ be a tag for some $d \geq 1$. Verification proceeds as follows:

– If $y_0 \neq m$, then output 0 (reject). Otherwise continue as follows.

– For every input wire of f with label τ compute $r_\tau = F_K(\tau)$.

– Next, evaluate the circuit on $r_{\tau_1}, \ldots, r_{\tau_n}$, i.e., compute $\rho \leftarrow f(r_{\tau_1}, \ldots, r_{\tau_n})$, and use x to check whether the following equation holds:

$$\rho = \sum_{k=0}^{d} y_k x^k \tag{1}$$

If this is true, then output 1. Otherwise output 0.

Observe that the above applies also to identity programs $\mathcal{I}_\tau$, in which case the algorithm just checks that $r_\tau = y_0 + y_1 \cdot x$ and $y_0 = m$.

Efficiency. Our scheme is extremely efficient in generating a tag using the Auth algorithm: just one PRF evaluation (e.g., one AES evaluation, in practice).

If we analyze the Eval algorithm, its complexity is dominated by the cost of evaluating the circuit f with an additional overhead due to the modified gate evaluation and to that the tag's size grows with the degree of the circuit. If the circuit has degree d, in the worst case, this overhead is going to be $O(d)$ for addition gates, and $O(d \log d)$ for multiplication gates[8].

The cost of verification is basically the cost of computing $\rho = f(r_{\tau_1}, \ldots, r_{\tau_n})$, that is $O(|f|)$, plus the cost of computing $\sum_{i=0}^{d} y_i x^i$, that is $O(d)$.

3.1 Correctness

In this section we prove that the scheme described above satisfies authentication and evaluation correctness.

Theorem 1. *The homomorphic MAC scheme described above satisfies authentication correctness.*

Proof. Let $\mathsf{sk} = (K, x)$ and $\mathsf{ek} = p$ be honestly generated keys, and let $\sigma \xleftarrow{\$} \mathsf{Auth}(\mathsf{sk}, \tau, m)$. By construction of Auth, the tag $\sigma = (y_0, y_1)$ is a degree-1 polynomial y such that $y(0) = m$ and $y(x) = y_0 + y_1 x = r_\tau$. Therefore it is trivial to verify that the checks made by the verification algorithm are satisfied and it outputs 1. □

Theorem 2. *The homomorphic MAC scheme described above satisfies evaluation correctness.*

Proof. Let $\mathsf{sk} = (K, x)$ and $\mathsf{ek} = p$ be honestly generated keys, and let $\{m_i, \mathcal{P}_i, \sigma_i\}_{i=1}^n$ be n triples of message/program/tag such that $\mathsf{Ver}(\mathsf{sk}, m_i, \mathcal{P}_i, \sigma_i) = 1$, $\forall i = 1, \ldots, n$. Then we want to show that for the message $m^* = f^*(m_1, \ldots, m_n)$, the composed program $\mathcal{P}^* = f^*(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the tag $\sigma^* = \mathsf{Eval}(\mathsf{ek}, f^*, (\sigma_1, \ldots, \sigma_n))$, it holds $\mathsf{Ver}(\mathsf{sk}, m^*, \mathcal{P}^*, \sigma^*) = 1$.

Let us first recall that for any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, any message $m \in \mathcal{M}$ and any tag $\sigma = (y_0, \ldots, y_d)$ the algorithm $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ outputs 1 only if the following two equations are satisfied:

$$m = y(0)$$
$$\rho = y(x) = \sum_{k=0}^{d} y_k x^k$$

[8] This bound follows from that one can use optimized algorithms based on FFT to compute the convolution.

where ρ is the value obtained by computing f on $r_{\tau_1}, \ldots, r_{\tau_n}$.

Let $\rho_1, \ldots, \rho_n$ be the values computed in the run of $\mathsf{Ver}(\mathsf{sk}, m_i, \mathcal{P}_i, \sigma_i) = 1$, for all $i = 1$ to n. Since the verification algorithm outputs 1, it must hold $m_i = y^{(i)}(0)$ and $\rho_i = y^{(i)}(x)$ (where $y^{(i)} = \sigma_i$). Our goal is to show that the above conditions imply $m^* = y^*(0)$ and $\rho^* = y^*(x)$ where y^* is the polynomial obtained by evaluating f^* over the polynomials $y^{(1)}, \ldots, y^{(n)}$ as described in the algorithm Eval, $m^* = f^*(m_1, \ldots, m_n)$ and $\rho^* = f^*(\rho_1, \ldots, \rho_n)$.

To do this we simply show that this property holds for every gate g, and then we observe that it easily extends by induction to the computation of f^*. Namely, if the above conditions hold *before and after* the evaluation of every single gate, then they must hold even for the values generated at the end of the computation of f^*, i.e., the output of the last gate of f^*.

The core of the proof is the following Lemma, which basically shows that our homomorphic evaluation of a gate g using two polynomials $y^{(1)}, y^{(2)}$ preserves their nice structure.

Lemma 1. *Let $b_1, b_2 \in \mathbb{Z}_p, y^{(1)}, y^{(2)} \in \mathbb{Z}_p[z]$ and $z \in \mathbb{Z}_p$ be such that*

$$b_i = y^{(i)}(z), \quad \forall i = 1, 2 \tag{2}$$

and let $b = g(b_1, b_2)$, $y = \mathsf{GateEval}(\mathsf{ek}, g, y^{(1)}, y^{(2)})$ for some gate g that is either $\times$ or $+$. Then, it holds

$$b = y(z) \tag{3}$$

Proof. If g is a multiplicative gate, then $b = b_1 \cdot b_2$ and $y = y^{(1)} * y^{(2)}$. By definition of the $*$ operator we then have $y(z) = y^{(1)}(z) \cdot y^{(2)}(z) = b_1 \cdot b_2$.

If g is an additive gate, then $b = b_1 + b_2$, and $y = y^{(1)} + y^{(2)}$. Hence, it is easy to see that $y(z) = y^{(1)}(z) + y^{(2)}(z) = b_1 + b_2$. □

Our Lemma does not consider the case of multiplicative gates in which one of the two inputs is a constant $c \in \mathbb{Z}_p$. However, it is trivial to observe that it extends to this case as well.

To complete the proof of the Theorem we have to observe that by definition of composed program the value $\rho^* = f^*(\rho_1, \ldots, \rho_n)$ is the same $\rho^* = f(r_{\tau_1^*}, \ldots, r_{\tau_n^*})$ computed in $\mathsf{Ver}(\mathsf{sk}, m^*, \mathcal{P}^*, \sigma^*)$.

3.2 Proof of Security

The security of our scheme is established by the following theorem.

Theorem 3. *If F is a PRF, then the homomorphic MAC scheme described in Section 3 is secure.*

Towards proving the theorem we define the following hybrid games. We denote with G_i the event that the experiment Game i, run with the adversary $\mathcal{A}$, outputs 1.

Game 0: this is the same as the real $\mathsf{HomUF{-}CMA}$ experiment, except that in every verification query $(m, \mathcal{P}, \sigma)$, in order to check whether $\mathcal{P}$ is well-defined or not, the challenger uses the probabilistic test of Proposition 1.

Thus we have that for all adversaries $\mathcal{A}$ making at most Q verification queries we have

$$|\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC}}(\lambda)] - \Pr[G_0(\mathcal{A})]| \le Q \cdot 2^{-\lambda} \tag{4}$$

Game 1: this is the same as Game 0, except that the PRF is replaced with a truly random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$: basically, each value r_τ is generated uniformly at random in $\mathbb{Z}_p$.

Game 2: This is the same as Game 1 except for the following change in answering verification queries. First, for every verification query $(m, \mathcal{P}, \sigma = (y_0, \ldots, y_d))$ such that $y_0 \neq m$ output 0 (reject). Next, for all verification queries $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P}$ is *not* well defined on T, the challenger proceeds as follows.

First, for every τ_i such that $(\tau_i, \cdot) \notin T$ it takes $r_{\tau_i} \leftarrow \mathcal{R}(\tau_i)$. Next, it computes ρ using the internal procedure of Ver, and finally it computes $Z = \rho - \sum_{k=0}^{d} y_k x^k$. If $Z = 0 \bmod p$, then it returns 1, otherwise it outputs 0.

Game 3: this is the same as Game 2 except for an additional change in answering verification queries. For every verification query $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T, the challenger proceeds as follows.

For every index $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin T$ (i.e., $\mathcal{P}$ contains a new label) it chooses a dummy tag σ_i (e.g., for a random message). Let $\sigma_1, \ldots, \sigma_n$ be the tags associated to labels $\tau_1, \ldots, \tau_n$ respectively. The challenger homomorphically computes $\hat{\sigma} = (\hat{y}_0, \ldots, \hat{y}_d) \leftarrow \mathsf{Eval}(\mathsf{ek}, f, (\sigma_1, \ldots, \sigma_n))$, and proceeds as follows:

1. If $(y_0, \ldots, y_d) = (\hat{y}_0, \ldots, \hat{y}_d)$, then output 1.
2. If $(y_0, \ldots, y_d) \neq (\hat{y}_0, \ldots, \hat{y}_d)$, compute $Z = \sum_{k=0}^{d} (y_k - \hat{y}_k) x^k$. If $Z = 0 \bmod p$, then output 1. Otherwise, output 0.

Game 4: This is the same as Game 3 except for the following additional code run by the challenger. Let bad be a flag value which is initially set to false. Once the challenger receives a verification query $(m, \mathcal{P}, \sigma)$ such that

1. $(m, \mathcal{P}, \sigma)$ requires the computation of Z (see the games above),
2. $Z = 0 \pmod{p}$,

then it answers with 0 (reject), instead of 1 (accept), and sets $\mathsf{bad} \leftarrow \mathsf{true}$. Notice that the flag bad is updated to true when the *first* bad (i.e. meeting the two requirements above) verification query is received.

Let Bad_4 be the event that $\mathsf{bad} \leftarrow \mathsf{true}$ is set in Game 4. Notice that any adversary has zero probability of winning in Game 4, as all verification queries that may be Type 1 or Type 2 forgeries are answered with 0. Therefore, it clearly holds $\Pr[G_4] = 0$.

To prove Theorem 3 we first prove the following claims.

Claim 1 $|\Pr[G_0] - \Pr[G_1]| \leq \mathbf{Adv}^{PRF}_{\mathcal{B},F}(\lambda)$.

The proof can be obtained via a straightforward reduction to the security of the PRF.

Claim 2 $\Pr[G_1] \equiv \Pr[G_2]$.

This is only a syntactic change, and it is easy to see that the two games are basically the same.

Claim 3 $\Pr[G_2] \equiv \Pr[G_3]$.

Proof. The only difference between Game 2 and Game 3 is in the way the challenger answers verification queries $(m, \mathcal{P}, \sigma)$ for a program $\mathcal{P}$ that is well defined on T. We claim that the change introduced in Game 3 is only syntactic, and thus the adversary has exactly the same view in both games.

Let $(m, \mathcal{P}, \sigma)$ be a verification query with $\sigma = (y_0, \ldots, y_d)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T, and let us first consider the case in which $\forall i = 1, \ldots, n$ it holds that $(\tau_i, m_i) \in T$ and a tag σ_i had already been computed. Recall that the challenger computes $\hat{\sigma} = (\hat{y}_0, \ldots, \hat{y}_d)$ using the Eval algorithm. If we consider the answer provided by the challenger in each case, then we have:

1. $(y_0, \ldots, y_d) = (\hat{y}_0, \ldots, \hat{y}_d)$*:* the answer is correct by correctness of the scheme.
2. $(y_0, \ldots, y_d) \neq (\hat{y}_0, \ldots, \hat{y}_d)$. Let ρ be the value computed by the verification algorithm to check equation (1), and observe that ρ is the same when running both $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ and $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \hat{\sigma})$ as the same values r_τ's are generated.
   By correctness of $\hat{\sigma}$ we have that $\rho = \sum_{k=0}^{d} \hat{y}_k \cdot x^k$. In order for $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ to output 1, it must hold $y_0 = m$ and $\rho = \sum_{k=0}^{d} y_k \cdot x^k$. Notice also that, by definition of our verification algorithm, the check $y_0 = m$ is performed first (i.e. the equality $\sum_{k=0}^{d} y_k \cdot x^k = \rho$ is verified only if $y_0 = m$ holds). So, returning 1 only if $Z = \sum_{k=0}^{d}(y_k - \hat{y}_k)x^k = 0$ is the same as returning the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$.

To complete the proof, let us consider the case in which $\mathcal{P}$ is well defined on T but there exists some $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin T$. By definition of well defined program, this means that if we fix the input values of all wires labeled with τ where $(\tau, \cdot) \in T$, then the circuit f always returns the same output whatever are the values taken by the input wires labeled with $\tilde{\tau}$ for $(\tilde{\tau}, \cdot) \notin T$. In other words, the value corresponding to input wires $\tilde{\tau} \notin T$ are irrelevant when it comes to evaluating f. Of course, this remains true even in the homomorphic evaluations of f: the one using the polynomials $y^{(i)}$ in Eval and the other one using the values r_{τ_i}'s in Ver. This means that for all wires labeled with $\tilde{\tau}$ (for $(\tilde{\tau}, \cdot) \notin T$) the dummy tags chosen for such indices do not contribute to the computation of $(\hat{y}_0, \ldots, \hat{y}_d)$, and the same holds for the random values $r_{\tilde{\tau}}$ with respect to R. Therefore the above argument for the case when $(\tau_i, m_i) \in T, \forall i = 1, \ldots, n$ applies here as well.

Claim 4 $|\Pr[G_3] - \Pr[G_4]| \leq \Pr[\mathsf{Bad}_4]$.

Game 3 and Game 4 are identical unless the event Bad_4 occurs in Game 4. Indeed, in this case the challenger is providing a different answer to some verification queries. Hence, $\Pr[G_4 \wedge \neg\mathsf{Bad}_4] = \Pr[G_3]$, that is $|\Pr[G_3] - \Pr[G_4]| \leq \Pr[\mathsf{Bad}_4]$.

To finalize the proof of our theorem we are left with bounding the probability of the event Bad_4, which is done below.

Claim 5 $\Pr[\mathsf{Bad}_4] \leq \frac{2dQ}{p-d(Q-1)}$ *where p is the prime used in the construction and Q is an upper bound on the number of verification queries made by $\mathcal{A}$ during the Game.*

Proof. For $j = 1$ to Q, let B_j be the event that $\mathsf{bad} \leftarrow \mathsf{true}$ is set *after* the j-th verification query, but not before. Hence, we have:

$$\Pr[\mathsf{Bad}_4] = \Pr\left[\bigvee_{j=1}^{Q} B_j\right] \leq \sum_{j=1}^{Q} \Pr\left[B_j\right] \tag{5}$$

The main part of this proof will consist in estimating the probability $\Pr[B_j]$ taken over the random choices of x and all values r_τ sampled by the challenger, and for *any* possible values chosen by the adversary.

For ease of exposition, we call "easy queries" those verification queries that do not "force" the challenger to compute Z to be answered. In our analysis we only consider *non*-"easy" queries as, by game definition, "easy" queries cannot cause the setting of $\mathsf{bad} \leftarrow \mathsf{true}$, i.e., B_j never occurs when the j-th query is "easy".

Let $(m, \mathcal{P}, \sigma)$ be the j-th verification query, that is not easy. According to whether $\mathcal{P}$ is well defined or not, we have only two possible cases for B_j to occur:

1. $Z = \sum_{k=0}^{d}(y_k - \hat{y}_k)x^k = 0 \pmod p$, where there exists at least an index $\tilde{k} \in \{0, \ldots, d\}$ such that $y_{\tilde{k}} \neq \hat{y}_{\tilde{k}}$.
2. $Z = \rho - \sum_{k=0}^{d} y_k x^k = 0 \pmod p$, where $\mathcal{P}$ is not well defined and ρ is computed by using at least one value $r_{\tau^*} \in \mathbb{Z}_p$ such that $(\tau^*, \cdot) \notin T$.

For $j = 1, \ldots, Q$ we denote with Z_j the Z value computed in the j-th such query. By definition of the event B_j, bad was not set true in the previous $j-1$ queries, i.e., $Z_1, \ldots, Z_{j-1} \neq 0 \pmod p$. Therefore we have:

$$\Pr[B_j] = \Pr[Z_j = 0 | Z_1 \neq 0 \wedge \cdots \wedge Z_{j-1} \neq 0] \tag{6}$$

Let us fix the value of x at the beginning of the game. Now, the crucial observation here is that, before the adversary starts making verification queries there are exactly p tuples $(x, \{r_\tau\}_{\tau \in T})$ that are consistent with her view. More precisely, conditioning on the tags σ_i seen by the adversary there is exactly one valid r_{τ_i} for each potential value of $x \in \mathbb{Z}_p$. Moreover, each "easy" query does not reveal any additional information about x and the relevant r_τ's. This is because our modified verification algorithm run by the challenger (introduced in Games 2 and 3) does not even need such values to answer "easy" verification queries. So, from now on we assume that all the Q queries in our analysis are non-easy.

After the first query, if $Z_1 \neq 0 \pmod p$, the number of possible values $x, \{r_\tau\}_{\tau \in T}$ becomes at least $p - d$, as the zeroes of a non-zero polynomial $c(x) = \sum_{k=0}^{d} c_k x^k$ of degree d are at most d. In general, after the i-th query, if $Z_1 \neq 0 \wedge \cdots \wedge Z_i \neq 0$, then the number of remaining possible values $x, \{r_\tau\}_{\tau \in T}$ is at least $p - di$.

Let $(m, \mathcal{P}, \sigma)$ be the j-th verification query. We define E_j^1 as the event that "$\mathcal{P}$ is well defined and there exists an index $\tilde{k} \in \{0, \ldots, d\}$ such that $y_{\tilde{k}} \neq \hat{y}_{\tilde{k}}$". Similarly, we define the event E_j^2 as "$\mathcal{P}$ is not well defined and ρ is computed by using some $r_{\tau^*} \in \mathbb{Z}_p$ such that $(\tau^*, \cdot) \notin T$". Finally, let $\mathtt{NotZero_j}$ be the event "$Z_1 \neq 0 \wedge \cdots \wedge Z_{j-1} \neq 0$".

Clearly, $\Pr[B_j] = \Pr[B_j \wedge E_j^1] + \Pr[B_j \wedge E_j^2]$ and thus

$$\begin{aligned}\Pr[B_j] &= \Pr[Z_j = 0 \mid \mathtt{NotZero_j}] = \\ &= \Pr[Z_j = 0 \wedge E_j^1 \mid \mathtt{NotZero_j}] + \Pr[Z_j = 0 \wedge E_j^2 \mid \mathtt{NotZero_j}] \\ &\leq \Pr[Z_j = 0 \mid E_j^1 \ \wedge \ \mathtt{NotZero_j}] + \Pr[Z_j = 0 \mid E_j^2 \ \wedge \ \mathtt{NotZero_j}]\end{aligned} \tag{7}$$

The first probability is

$$\Pr[Z_j = 0 \mid E_j^1 \ \wedge \ \mathtt{NotZero_j}] \leq \frac{d}{p - d(j-1)} \tag{8}$$

as the zeroes of the polynomial $\sum_{k=0}^{d}(y_k - \hat{y}_k)x^k$ are at most d, and there are $p - d(j-1)$ possible values of x.

To evaluate the second probability $\Pr[Z_j = 0 \mid E_j^2 \ \wedge \ \mathtt{NotZero_j}]$ we observe that one can think of ρ as a univariate polynomial $\rho = \eta(r_{\tau^*})$ in the variable r_{τ^*} whose degree is at most d. Moreover, being $\mathcal{P}$ not well defined, η must be a non-constant polynomial. The value r_{τ^*} was sampled uniformly at random and it was *never* used before to produce a tag (since $(\tau^*, \cdot) \notin T$)[9].

[9] However, this r_{τ^*} *might* have been implicitly used before. In particular the adversary might have already asked some non easy verification query containing the input label τ^*.

Initially, i.e. when no verification queries involving τ^* have been made, r_{τ^*} is clearly perfectly hidden to the adversary, and she cannot guess it with probability better than $1/p$. The worst case to consider is then the one where all queries, before the j-th we are considering, involve τ^*. In such a case, we can use an argument very similar to that given before, so that, conditioned on $Z = \eta(r_{\tau^*}) \neq 0$, one can exclude at most d possible values of r_{τ^*}. Therefore, if we condition on the event $\texttt{NotZero}_\texttt{j}$, the probability of the adversary to guess r_{τ^*} at the j-th query cannot be better than $1/(p - d(j-1))$.

Hence,

$$\Pr[Z_j = 0 \mid E_j^2 \ \wedge\ \texttt{NotZero}_\texttt{j}] \leq \frac{d}{p - d(j-1)} \tag{9}$$

Finally, if we apply the results of equations (8) and (9) to equation (7), then we obtain:

$$\Pr[B_j] \leq \frac{2d}{p - d(j-1)} \tag{10}$$

and thus we can upper bound:

$$\Pr[\mathsf{Bad}_4] \leq \frac{2dQ}{p - d(Q-1)} \tag{11}$$

which proves the Claim. □

The proof of the Theorem follows by putting together the results of the above claims:

$$\mathbf{Adv}^{\mathsf{HomUF-CMA}}_{\mathcal{A},\mathsf{HomMAC}}(\lambda) \leq \mathbf{Adv}^{PRF}_{\mathcal{B},F}(\lambda) + \frac{2dQ}{p - d(Q-1)} + \frac{Q}{2^\lambda}.$$

Since $p \approx 2^\lambda$ and both d and Q are $\mathsf{poly}(\lambda)$, $\frac{2dQ}{p-d(Q-1)} = \mathsf{negl}(\lambda)$. Therefore, if the PRF is secure (i.e., $\mathbf{Adv}^{PRF}_{\mathcal{B},F}(\lambda) = \mathsf{negl}(\lambda)$) then any PPT adversary $\mathcal{A}$ has at most negligible advantage of breaking the unforgeability of our homomorphic MAC.

4 A Compact Homomorphic MAC for Circuits of Bounded Polynomial Degree

As we mentioned earlier, the homomorphic MAC of Section 3 has the drawback that the tags' size grows linearly with the degree of the evaluated circuit. While this may be acceptable in some cases, e.g., circuits evaluating constant-degree polynomials, it may become impractical in other situations, e.g., when the degree is greater than the input size of the circuit. In this section, we propose a second scheme that solves this issue and enjoys tags of constant size. The scheme keeps almost the same efficiency of the previous one, even though constant-size tags come at the price of a couple of restrictions. First, we have to fix an a-priori bound D on the degree of the circuits that can be evaluated. Second, the homomorphic evaluation has to be done in a "single shot", that is the authentication tags obtained from the Eval algorithm cannot be used again to be composed with other tags.

Nevertheless, we show that the scheme achieves an interesting property that we call *local composition.* The idea is that one can keep locally a non-succinct version of the tag that allows for arbitrary composition. Later, when it comes to send an authentication tag to the verifier, one can securely compress such large tag in a very compact one of constant-size.

Before describing the scheme in details, we give a high level description of our technique. The main idea is to use the same scheme of Section 3 with the following modifications. Tags on messages are the same, i.e., a degree-1 polynomial with the special property that $y(0) = m$ and $y(x) = r_\tau$. We publish the values $g^x, \ldots, g^{x^D}$ in the public evaluation key. Then, for the homomorphic evaluation one first computes the coefficients $(y_0, \ldots, y_d)$ as before, and then "accumulates" such values into a single group element $\Lambda = \prod_{i=1}^{d} (g^{x^i})^{y_i}$. The verification is carried in the exponent as follows: the verifier first computes ρ as before and next checks that $g^\rho = g^m \Lambda$. If Λ is computed correctly, then $\Lambda = g^{y(x)-y(0)}$ and thus correctness follows.

For security, in addition to a PRF we need to rely on a computational assumption that says that one cannot compute g given values $g^x, \ldots, g^{x^D}$. This problem is basically a re-writing of a problem already considered in the past: the ℓ-Diffie-Hellman Inversion. We recall its definition below.

Definition 1 (ℓ-DHI [11,30]). *Let $\lambda \in \mathbb{N}$ be the security parameter, and $\mathbb{G}$ be a group of order $p > 2^\lambda$. For a generator $g \in \mathbb{G}$ and a randomly chosen $x \xleftarrow{\$} \mathbb{Z}_p$ we define the advantage of an adversary $\mathcal{A}$ in solving the ℓ-DHI problem as*

$$\mathbf{Adv}_{\mathcal{A}}^{DHI}(\lambda) = \Pr[\mathcal{A}(g, g^x, \ldots, g^{x^\ell}) = g^{1/x}]$$

and we say that the ℓ-DHI assumption holds in $\mathbb{G}$ if for every PPT $\mathcal{A}$ and for $\ell = \mathsf{poly}(\lambda)$, the advantage $\mathbf{Adv}_{\mathcal{A}}^{DHI}(\lambda)$ is at most negligible in λ.

Our Construction. The description of our scheme follows.

$\mathsf{KeyGen}(1^\lambda, D)$**.** Let λ be the security parameter and $D = \mathsf{poly}(\lambda)$ be an upper bound so that the scheme can support the homomorphic evaluation of circuits of degree at most D. The key generation works as follows.

Generate a group $\mathbb{G}$ of order p where p is a prime of roughly λ bits, and choose a random generator $g \xleftarrow{\$} \mathbb{G}$. Choose a seed K of a pseudorandom function $F_K : \{0,1\}^* \to \mathbb{Z}_p$ and a random value $x \xleftarrow{\$} \mathbb{Z}_p$. For $i = 1$ to D compute $h_i = g^{x^i}$. Output $\mathsf{sk} = (K, g, x), \mathsf{ek} = (h_1, \ldots, h_D)$ and let the message space $\mathcal{M}$ be $\mathbb{Z}_p$.

$\mathsf{Auth}(\mathsf{sk}, \tau, m)$**.** The tagging algorithm is the same as the one of the construction in Section 3. To authenticate a message $m \in \mathbb{Z}_p$ with label $\tau \in \{0,1\}^\lambda$, compute $r_\tau = F_K(\tau)$, set $y_0 = m$, $y_1 = (r_\tau - m)/x \bmod p$, and output $\sigma = (y_0, y_1)$.

$\mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma})$**.** The homomorphic evaluation algorithm takes as input the evaluation key ek, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$ so that $\sigma_i \in \mathbb{Z}_p^2$ (i.e., it is a tag for a degree-1 polynomial).

First, proceed exactly as in the construction of Section 3 to compute the coefficients $(y_0, \ldots, y_d)$. If $d = 1$ (i.e., the circuit f computes a degree-1 polynomial), then return $\sigma = (y_0, y_1)$. Otherwise, compute

$$\Lambda = \prod_{i=1}^{d} h_i^{y_i}$$

and return $\sigma = \Lambda$.

$\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$**.** Let $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ be a labeled program, $m \in \mathbb{Z}_p$ and σ be a tag of either the form $(y_0, y_1) \in \mathbb{Z}_p^2$ or $\Lambda \in \mathbb{G}$.

First, proceed as in the construction of Section 3 to compute $\rho = f(r_{\tau_1}, \ldots, r_{\tau_n})$.

If the program $\mathcal{P}$ computes a polynomial of degree 1, then proceed exactly as in the construction of Section 3 and check that

$$\rho = y_0 + y_1 \cdot x \ \wedge \ y_0 = m.$$

Otherwise, use g to check whether the following equation holds:

$$g^{\rho} = g^m \cdot \Lambda \tag{12}$$

If the checks are satisfied, then output 1. Otherwise output 0.

Correctness. The correctness easily follows from the correctness of the scheme described in Section 3 and by observing that equation (12) is essentially equivalent to checking that

$$\rho = \sum_{i=0}^{d} y_i x^i$$

which is the verification equation (1) in the scheme of Section 3.

Local Composition. The above scheme satisfies an interesting property that we call *local composition*. The idea is that one can keep locally the large version of the tag, i.e., the polynomial y with its $d+1$ coefficients $y_0, \ldots, y_d$, but still send its compact version $\Lambda = \prod_{i=1}^{d} (g^{x^i})^{y_i}$ to the verifier. Keeping y allows for arbitrary composition as in the scheme of Section 3. In applications where composition does not involve many parties, this property allows to achieve succinct tags and local composition of partial computations at the same time.

Extension. In Appendix A we show an extension of this scheme that, by using pairings, allows to further compute an additional level of multiplications and unbounded additions on tags of the Λ form.

4.1 Proof of Security

Theorem 4. *If F is a PRF and the $(D-1)$-Diffie Hellman Inversion Assumption holds in $\mathbb{G}$, then the homomorphic MAC scheme described in Section 4 is secure.*

To prove the security of our scheme we define the following hybrid games. We denote with G_i the event that the experiment Game i, run with the adversary $\mathcal{A}$, outputs 1.

Game 0: this is the same as the real $\mathsf{HomUF\!-\!CMA}$ experiment, except that in every verification query $(m, \mathcal{P}, \sigma)$, in order to check whether $\mathcal{P}$ is well-defined or not, the challenger uses the probabilistic test of Proposition 1.

Thus we have that for all adversaries $\mathcal{A}$ making at most Q verification queries we have

$$|\Pr[\mathsf{HomUF\!-\!CMA}_{\mathcal{A},\mathsf{HomMAC}}(\lambda)] - \Pr[G_0(\mathcal{A})]| \leq Q \cdot 2^{-\lambda} \tag{13}$$

Game 1: this is the same as Game 0, except that the PRF is replaced with a truly random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$: basically, each value r_τ is generated uniformly at random in $\mathbb{Z}_p$.

Game 2: This is the same as Game 1 except for a change in answering verification queries. Let bad be a flag value which is initially set to false. For all verification queries $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is *not* well defined on T, the challenger answers 0 (reject) and proceeds as follows.

First, for every τ_i such that $(\tau_i, \cdot) \notin T$ it takes $r_{\tau_i} \leftarrow \mathcal{R}(\tau_i)$. Next, it computes ρ using the internal procedure of Ver.

If $\sigma = (y_0, y_1)$, compute $z = \rho - y_0 - x \cdot y_1$. If $z = 0 \pmod p$, then set $\mathsf{bad} \leftarrow \mathsf{true}$.

If $\sigma = \Lambda$ compute $Z = g^{\rho - m}\Lambda^{-1}$. If $Z = 1$, then set $\mathsf{bad} \leftarrow \mathsf{true}$.

Game 3: this is the same as Game 2 except for the following change in answering authentication queries. Given a query (τ, m) such that $(\tau, m) \notin T$, if $r_\tau = \mathcal{R}(\tau)$ was previously used to answer a verification query, then resample a fresh $r'_\tau \xleftarrow{\$} \mathbb{Z}_p$ to create the tag, and from now on use r'_τ in every verification query involving label τ.

Game 4: this is the same as Game 3 except for the following change in answering verification queries. Let bad' be a flag value which is initially set to false. For every verification query $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T, the challenger proceeds as follows.

For every index $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin T$ (i.e., $\mathcal{P}$ contains a new label) it chooses a dummy tag σ_i (e.g., for a random message).

Let $\sigma_1, \ldots, \sigma_n$ be the tags associated to labels $\tau_1, \ldots, \tau_n$ respectively. The challenger homomorphically computes $\hat{\sigma}$ (which is either $(\hat{y}_0, \hat{y}_1)$ or $\hat{\Lambda}$) and proceeds as follows.

If $\sigma = \hat{\sigma}$, then output 1 (accept). Otherwise, output 0 (reject).

Moreover,

1. If $(y_0, y_1) \neq (\hat{y}_0, \hat{y}_1)$, compute $z = (y_0 - \hat{y}_0) + x(y_1 - \hat{y}_1)$. If $z = 0 \pmod p$, then set $\mathsf{bad}' \leftarrow \mathsf{true}$.
2. If $\Lambda \neq \hat{\Lambda}$, compute $Z = (\Lambda/\hat{\Lambda})g^{(m-\hat{m})}$. If $Z = 1$, then set $\mathsf{bad}' \leftarrow \mathsf{true}$.

To prove the theorem we prove the following claims.

Claim 6 $|\Pr[G_0] - \Pr[G_1]| \leq \mathbf{Adv}^{PRF}_{\mathcal{B},F}(\lambda)$.

The proof can be obtained via a straightforward reduction to the security of the PRF.

Claim 7 $|\Pr[G_1] - \Pr[G_2]| \leq \frac{Q(d+1)}{p - d(Q-1)}$ *where p is the order of the group used in the scheme and Q is the number of verification queries made by the adversary $\mathcal{A}$ during the experiment.*

Proof. Let Bad_2 be the event that $\mathsf{bad} \leftarrow \mathsf{true}$ is set in Game 2. Game 1 and Game 2 are identical unless the event Bad_2 occurs. Indeed, in this case the challenger is providing a different answer to some verification queries. It holds $\Pr[G_2 \wedge \neg\mathsf{Bad}_2] = \Pr[G_1]$, that is $|\Pr[G_1] - \Pr[G_2]| \leq \Pr[\mathsf{Bad}_2]$. Hence, to prove the Claim, we estimate the probability $\Pr[\mathsf{Bad}_2]$. The proof is very similar to that of Claim 5. We provide it below for completeness.

For $j = 1$ to Q, let B_j be the event that $\mathsf{bad} \leftarrow \mathsf{true}$ is set *after* the j-th verification query, but not before. Clearly, we have:

$$\Pr[\mathsf{Bad}_2] = \Pr\left[\bigvee_{j=1}^{Q} B_j\right] \leq \sum_{j=1}^{Q} \Pr[B_j] \tag{14}$$

Moreover, by definition of B_j, $\mathsf{bad} \leftarrow \mathsf{true}$ did not occur in the previous $j-1$ queries, thus

$$\Pr[B_j] = \Pr[B_j | \neg B_1 \wedge \cdots \wedge \neg B_{j-1}]$$

The main part of this proof will consist in estimating the probability $\Pr[B_j]$ taken over the random choices of the values r_τ sampled by the challenger, and for any possible values chosen by the adversary. In our analysis we will consider only verification queries $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P}$ is *not* well defined, as they are the only queries that may cause setting $\mathsf{bad}\leftarrow\mathsf{true}$.

Let $(m, \mathcal{P}, \sigma)$ be the j-th verification query. According to whether $\sigma = (y_0, y_1)$ or $\sigma = \Lambda$, we have only two possible cases for B_j to occur:

1. $z = \rho - y_0 - xy_1 = 0 \pmod{p}$.
2. $Z = g^{\rho - m}\Lambda^{-1} = 1$

where in both cases ρ is computed by using at least one value $r_{\tau^*} \in \mathbb{Z}_p$ such that $(\tau^*, \cdot) \notin T$.

For $1 \leq j \leq Q$, let z_j (resp. Z_j) be the value computed in the j-th query. Let $\mathtt{NotZero_j}$ be the event "$\neg B_1 \wedge \cdots \wedge \neg B_{j-1}$", and notice that for $1 \leq i \leq j-1$, $\neg B_i$ may mean either $z_i \neq 0$ or $Z_i \neq 1$. Therefore, we have:

$$\Pr[B_j | \neg B_1 \wedge \cdots \wedge \neg B_{j-1}] \leq \Pr[z_j = 0 \mid \mathtt{NotZero_j}] + \Pr[Z_j = 1 \mid \mathtt{NotZero_j}] \tag{15}$$

where the probability is taken over the random choice of r_{τ^*}.

Using an argument similar to that in the proof of Claim 5, it is possible to show that, information-theoretically, the probability that any adversary guesses correctly the value r_{τ^*} at the j-th verification query (conditioned on the event $\mathtt{NotZero_j}$) cannot be better than $1/(p - D(j-1))$.

Moreover, the value ρ can be thought of as a univariate polynomial $\eta(r_{\tau^*})$ in the variable r_{τ^*} of degree at most D (where $D = 1$ if $\sigma = (y_0, y_1)$). Since $\mathcal{P}$ is not well defined, the polynomial η is non-constant, and thus:

$$\Pr[z_j = 0 \mid \mathtt{NotZero_j}] \leq \frac{1}{p - D(j-1)} \tag{16}$$

$$\Pr[Z_j = 1 \mid \mathtt{NotZero_j}] \leq \frac{D}{p - D(j-1)} \tag{17}$$

Therefore, by applying equations (16) and (17) to equation (15), we obtain:

$$\Pr[\mathsf{Bad}_2] \leq \frac{Q(D+1)}{p - D(Q-1)}$$

which proves the Claim. □

Claim 8 $|\Pr[G_2] - \Pr[G_3]| \leq \frac{DQ^2}{p}$ *where p is the order of the group used in the scheme and Q is the number of verification queries made by the adversary $\mathcal{A}$ during the experiment.*

Proof. Game 2 and Game 3 differ only in the sampling of the value r_τ in authentication queries. In particular, notice that there are at most Q of such queries. For each of these authentication queries, say (τ, m), assume that there were Q prior verification queries involving label τ. Using the argument in the previous lemma, the number of possible values of r_τ is at least $p - DQ$, and the games will differ only if sampling the fresh r'_τ will hit one of the DQ values that are excluded conditioning on $\mathtt{NotZero}_Q$. However, this happens with probability DQ/p. Thus the Claim follows by union bound. □

Claim 9 $\Pr[G_3] \equiv \Pr[G_4 \wedge \neg\mathsf{Bad}_4]$.

Proof. Let Bad_4 be the event that bad' is set true in Game 4. If the event Bad_4 does not occur, then we claim that Game 4 is identical to Game 3. The only change is in the way the challenger answers verification queries $(m, \mathcal{P}, \sigma)$ for a program $\mathcal{P}$ that is well defined on T.

Let $(m, \mathcal{P}, \sigma)$ be a verification query with $\sigma = (y_0, y_1)$ or $\sigma = \Lambda$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T, and let us first consider the case in which $\forall i = 1, \ldots, n$ it holds that $(\tau_i, m_i) \in T$ and a tag σ_i had already been computed. Recall that the challenger computes $\hat{\sigma}$ using the Eval algorithm. If we consider the answer provided by the challenger in this case, then we have:

1. $\sigma = \hat{\sigma}$*:* the answer is correct by correctness of the scheme.
2. $(y_0, y_1) \neq (\hat{y}_0, \hat{y}_1)$. Let ρ be the value computed by the verification algorithm to check equation (12), and observe that ρ must be the same when running both $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ and $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \hat{\sigma})$ as the same values r_τ's are used.
   By correctness of $\hat{\sigma}$ we have that $\rho = \hat{y}_0 + \hat{y}_1 \cdot x$. In order for $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ to output 1, it must hold $\rho = y_0 + y_1 \cdot x$. So, returning 1 only if $z = (y_0 - \hat{y}_0) + x(y_1 - \hat{y}_1) = 0$ is the same as returning the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$.
3. $\Lambda \neq \hat{\Lambda}$. Let ρ be the value computed by the verification algorithm to check equation (12), and observe that ρ must be the same when running both $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ and $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \hat{\sigma})$ as the same values r_τ's are used.
   By correctness of $\hat{\sigma}$ we have that $g^\rho = g^{\hat{m}} \hat{\Lambda}$. In order for $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ to output 1, it must hold $g^\rho = g^m \Lambda$. So, returning 1 only if $Z = g^{m - \hat{m}}(\Lambda / \hat{\Lambda}) = 1$ is the same as returning the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$.

Now, let us consider the case in which $\mathcal{P}$ is well defined on T but there exists some $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin T$. By definition of well defined program, this means that if we fix the input values of all wires labeled with τ where $(\tau, \cdot) \in T$, then the circuit f always returns the same output whatever are the values of the input wires with label $\tilde{\tau}$ such that $(\tilde{\tau}, \cdot) \notin T$. In other words, the value corresponding to the input wire τ_i is irrelevant when it comes to evaluating f. Of course, this remains true even in the homomorphic evaluations of f: the one using y in Eval and the other one using the r_τ's in Ver. This means that for all wires labeled with $\tilde{\tau}$ (for $(\tilde{\tau}, \cdot) \notin T$) the dummy tags chosen for such indices do not contribute to the computation of $\hat{\Lambda}$, and the same holds for the random values $r_{\tilde{\tau}}$ with respect to ρ. Therefore the above argument for the case when $(\tau_i, m_i) \in T, \forall i = 1, \ldots, n$ applies here as well. □

Claim 10 $\Pr[\mathsf{Bad}_4] \leq Q \cdot \mathbf{Adv}^{DHI}_{\mathcal{B}}(\lambda)$ *where Q is the number of verification queries made by the adversary $\mathcal{A}$ in Game 4.*

Proof. For sake of presentation, we prove the claim using a slightly different assumption: the adversary $\mathcal{A}$ is given a tuple $(g^x, \ldots, g^{x^\ell})$ for randomly chosen $g \in \mathbb{G}$ and $x \in \mathbb{Z}_p$, and it is required to compute g. It is not hard to see that this is only a rewriting of the $(\ell - 1)$-DHI assumption defined above. Indeed a tuple $(g^x, \ldots, g^{x^\ell})$ can be rewritten as $(h, h^x, \ldots, h^{x^{\ell-1}})$ by letting $h = g^x$.

Assume by contradiction that there exists an adversary $\mathcal{A}$ such that, when run in Game 4, we have $\Pr[\mathsf{Bad}_4] \geq \epsilon(\lambda)$ for some non-negligible function ϵ. Then, we show how to build an efficient simulator $\mathcal{B}$ that breaks the $(D - 1)$-DHI assumption with advantage $\mathbf{Adv}^{DHI}_{\mathcal{B}}(\lambda) \geq \epsilon(\lambda)/Q$.

If Bad_4 occurs in Game 4, then there must exist an index $1 \leq \mu \leq Q$ such that $\mathsf{bad}' \leftarrow \mathsf{true}$ is set in the μ-th verification query.

$\mathcal{B}$ takes as input a tuple $(g^x, \ldots, g^{x^D})$. Let Q be an upper bound on the number of verification queries made by the adversary. $\mathcal{B}$ chooses $\mu^* \xleftarrow{\$} \{1, \ldots, Q\}$ uniformly at random as a guess for the index μ in which bad' is updated to true. Next, it runs $\mathcal{A}$ on input $\mathsf{ek} = (g^x, \ldots, g^{x^D})$ answering queries as follows.

Authentication Queries. Given an authentication query (m, τ) it chooses $y_1 \xleftarrow{\$} \mathbb{Z}_p$ at random, sets $y_0 = m$ and returns $\sigma = (y_0, y_1)$. Notice that this alternative generation of the tag without using x generates the same distribution of tags as the one in Game 4, in which a fresh value r'_τ is used in every authentication query.

Verification Queries. On input the j-th verification query $(m, \mathcal{P}, \sigma)$, $\mathcal{B}$ proceeds as follows:

- If $\mathcal{P}$ is not well defined on T, output 0 (reject). Notice that this answer is correct by the change introduced in Game 2.
- If $\mathcal{P}$ is well defined compute $\hat{\sigma}$ using the Eval algorithm (exactly as the challenger in Game 4). If $\sigma = \hat{\sigma}$ output 1 (accept).
- If $\mathcal{P}$ is well defined, $\sigma \neq \hat{\sigma}$ and $j \neq \mu^*$, then output 0 (reject).
- If $\mathcal{P}$ is well defined, $\sigma \neq \hat{\sigma}$ and $j = \mu^*$, then output 0 (reject), and proceed as follows:
  - If $\sigma = (y_0, y_1)$ (and $\hat{\sigma} = (\hat{y}_0, \hat{y}_1)$), compute $x' = (y_0 - \hat{y}_0)/(\hat{y}_1 - y_1) \pmod p$, and $g' = (g^x)^{-1/x'}$.
  - If $\sigma = \Lambda$ (and $\hat{\sigma} = \hat{\Lambda}$), compute $g' = (\hat{\Lambda}/\Lambda)^{(m-\hat{m})^{-1}}$.

At the end of the simulation $\mathcal{B}$ outputs g'. If $\mathcal{B}$ correctly guessed the index $\mu^* = \mu$ in which the adversary queries the forgery, then $\mathcal{B}$ clearly succeeds in finding the correct $g' = g$ with probability at least $\epsilon(\lambda)$. Therefore, we have:

$$\mathbf{Adv}_{\mathcal{B}}^{DHI}(\lambda) = \Pr[\mathsf{Bad}_4 \wedge \mu^* = \mu] \geq \Pr[\mu^* = \mu] \Pr[\mathsf{Bad}_4] \geq \frac{\epsilon(\lambda)}{Q}$$

which concluded the proof of the Claim. □

To conclude the proof of the Theorem, observe that $\Pr[G_4] = 0$ as all verification queries to Type 1 and Type 2 forgeries are answered with 0. So, if we put together the results of the above claims we obtain

$$\mathbf{Adv}_{\mathcal{A},\mathsf{HomMAC}}^{\mathsf{HomUF-CMA}}(\lambda) \leq \mathbf{Adv}_{\mathcal{B},F}^{PRF}(\lambda) + \frac{Q}{2^\lambda} + \frac{Q(D+1)}{p - D(Q-1)} + \frac{DQ^2}{p} + Q \cdot \mathbf{Adv}_{\mathcal{B}}^{DHI}(\lambda).$$

Since $p \approx 2^\lambda$ and both D and Q are $\mathsf{poly}(\lambda)$, if the PRF is secure and the $(D-1)$-DHI assumption holds (i.e., $\mathbf{Adv}_{\mathcal{B},F}^{PRF}(\lambda) = \mathsf{negl}(\lambda)$ and $\mathbf{Adv}_{\mathcal{B}}^{DHI}(\lambda) = \mathsf{negl}(\lambda)$), then $\mathcal{A}$ has at most negligible advantage of breaking the unforgeability of our construction.

Acknowledgements The authors would like to thank Valerio Pastro and Daniel Wichs for helpful discussions on this work.

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. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 1–20. Springer, Mar. 2012.

3. 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.
4. 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.
5. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, Aug. 2011.
6. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, Jan. 2012.
7. 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/2012/095`.
8. 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.
9. 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.
10. 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.
11. X. Boyen. The uber-assumption family (invited talk). In S. D. Galbraith and K. G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, Sept. 2008.
12. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, May 2013.
13. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Mar. 2013.
14. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 207–223. Springer, May 2011.
15. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 680–696. Springer, May 2012.
16. K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Aug. 2010.
17. K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 151–168. Springer, Aug. 2011.
18. R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.
19. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 501–512. ACM Press, Oct. 2012.
20. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, May 2012.
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, 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.
23. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. Cryptology ePrint Archive, Report 2012/290, 2012. `http://eprint.iacr.org/2012/290`.
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, 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.
27. 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.

28. J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.
29. S. Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, Nov. 1994.
30. S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Transactions on Fundamentals*, E85-A(2):481–484, 2002.
31. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Mar. 2012.
32. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.
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. A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.
35. P. Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In R. Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Mar. 2008.
36. R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *EUROSM '79*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

A A compact homomorphic MAC with an additional level of multiplication

Here we describe an extension of the scheme given in Section 4. The new scheme allows to further apply homomorphic operations on the tags returned by Eval. In particular, the extended homomorphic evaluation allows for circuits of degree up to 2, i.e., at most one multiplication and an unbounded number of additions.

The intuitive idea of this construction is based on the following facts. Given two tags $\Lambda^{(1)}, \Lambda^{(2)} \in \mathbb{G}$ (as in the construction of Section 4) one could still apply homomorphic operations if these were only additions: just compute $\Lambda = \Lambda^{(1)} \cdot \Lambda^{(2)}$. To compute a multiplication, we can use the bilinear property of the pairing and compute $\Omega = e(\Lambda^{(1)}, \Lambda^{(2)})$. The way it is described here, the scheme is not fully correct, however we show below that some appropriate modifications allow to turn this basic idea into a scheme with the desired property.

The security of the new scheme relies on the Bilinear version of the Diffie-Hellman Inversion assumption used in Section 4. We recall it below.

Definition 2 (ℓ-BDHI). *Let $\lambda \in \mathbb{N}$ be the security parameter, and $\mathbb{G}$ be a group of order $p > 2^\lambda$. Let $\mathbb{G}, \mathbb{G}_T$ be groups with an efficient bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. For randomly chosen generators $g, \gamma \in \mathbb{G}$ and a randomly chosen $x \xleftarrow{\$} \mathbb{Z}_p$ we define the advantage of an adversary $\mathcal{A}$ in solving the ℓ-Parallel Diffie-Hellman Inversion problem as*

$$\mathbf{Adv}_{\mathcal{A}}^{BDHI}(\lambda) = \Pr[\mathcal{A}(g, g^x, \ldots, g^{x^\ell}) = e(g,g)^{1/x}]$$

and we say that the ℓ-BDHI assumption holds in $\mathbb{G}, \mathbb{G}_T$ if for every PPT $\mathcal{A}$ and for $\ell = \mathsf{poly}(\lambda)$, the advantage $\mathbf{Adv}_{\mathcal{A}}^{BDHI}(\lambda)$ is at most negligible in λ.

Our Construction. The description of our scheme follows.

$\mathsf{KeyGen}(1^\lambda, D)$. Let λ be the security parameter and $D = \mathsf{poly}(\lambda)$ be an upper bound so that the scheme can support the homomorphic evaluation of circuits of degree at most D. The key generation works as follows.

Generate bilinear groups $\mathbb{G}, \mathbb{G}_T$ of order p such that p is a prime of roughly λ bits and there exists an efficient and non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Choose a random generator $g \xleftarrow{\$} \mathbb{G}$, a random value $\alpha \in \{0,1\}^\lambda$, two random values $a, x \xleftarrow{\$} \mathbb{Z}_p$, and a seed K of a pseudorandom function $F_K : \{0,1\}^* \to \mathbb{Z}_p$. For $i = 1$ to D compute $h_i = g^{x^i}, \gamma_i = g^{ax^i}$, and set $\gamma = g^a$. Set $\sigma_U = (1, (r_\alpha - 1)/x)$ for $r_\alpha = F_K(\alpha)$ (σ_U is essentially the MAC of 1 under this special and secret label α).

Output $\mathsf{sk} = (K, g, h, x, \alpha), \mathsf{ek} = (\gamma, h_1, \gamma_1, \ldots, h_D, \gamma_D, \sigma_U)$ and let the message space $\mathcal{M}$ be $\mathbb{Z}_p$.

$\mathsf{Auth}(\mathsf{sk}, \tau, m)$**.** The tagging algorithm is the same as the one of the construction in Section 3. To authenticate a message $m \in \mathbb{Z}_p$ with label $\tau \in \{0,1\}^\lambda$, compute $r_\tau = F_K(\tau)$, set $y_0 = m$, $y_1 = (r_\tau - m)/x \bmod p$, and output $\sigma = (y_0, y_1)$.

$\mathsf{Eval}_1(\mathsf{ek}, f, \boldsymbol{\sigma})$**.** The homomorphic evaluation algorithm Eval_1 allows to compute a MAC for a circuit f of degree at most D starting from MACs obtained by evaluating circuits of degree 1. The algorithm is almost the same as the Eval algorithm in the construction of Section 4 and works as follows. It takes as input the evaluation key ek, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathbb{Z}_p^2$ (i.e., it is a tag for a degree-1 polynomial).

First, proceed exactly as in the construction of Section 3 to compute the coefficients $(y_0, \ldots, y_d)$. If $d = 1$ (i.e., the circuit f computes a degree-1 polynomial), then return $\sigma = (y_0, y_1)$. Otherwise, compute

$$\Lambda = \prod_{i=1}^{d} h_i^{y_i}, \quad \Gamma = \prod_{i=1}^{d} \gamma_i^{y_i}$$

and return $\sigma = (y_0, \Lambda, \Gamma)$.

$\mathsf{Eval}_2(\mathsf{ek}, \phi, \boldsymbol{\sigma})$**.** The homomorphic evaluation algorithm Eval_2 allows to further applies homomorphic operations on tags that were already obtained by using the algorithm Eval_1, namely tags for circuits of degree at most D. Precisely, Eval_2 allows to evaluate any circuit of degree at most 2. Thus, having Eval_1 and Eval_2 enables us to obtain a scheme that supports the evaluation of circuits of degree up to $2D$, and whose tags have constant size.

The algorithm takes as input the evaluation key ek, an arithmetic circuit $\phi : \mathbb{Z}_p^n \to \mathbb{Z}_p$ of degree ≤ 2 and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$ so that $\sigma_i \in \mathbb{Z}_p^2$ (i.e., it is a tag for a degree-1 polynomial), or $\sigma_i \in \mathbb{Z}_p \times \mathbb{G}^2$ (i.e., it is a tag obtained by Eval_1). In particular, we assume that at least one of the σ_i's is in $\mathbb{Z}_p \times \mathbb{G}^2$ (otherwise, one can use Eval_1).

First, for any tag $\sigma_i = (y_0^{(i)}, y_1^{(i)}) \in \mathbb{Z}_p^2$ the algorithm transforms σ_i into a tag $\tilde{\sigma}_i = (y_0^{(i)}, \Lambda^{(i)}, \Gamma^{(i)}) \in \mathbb{Z}_p \times \mathbb{G}^2$ as follows:

$$\Lambda^{(i)} = h_1^{y_1^{(i)}}, \quad \Gamma^{(i)} = \gamma_1^{y_1^{(i)}}$$

Next, Eval_2 proceeds on the circuit ϕ gate-by-gate as follows. At each gate g, given two tags σ_1, σ_2 (or a tag σ_1 and a constant $c \in \mathbb{Z}_p$), it runs one of the procedures described below (according to the case). It obtains a new tag σ and passes this on as input to the next gate in the circuit.

When the computation reaches the last gate of the circuit ϕ, Eval_2 outputs the tag σ obtained by evaluating such last gate.

- $\mathsf{Add}_1(\mathsf{ek}, \sigma_1, \sigma_2)$. This takes as input two tags $\sigma_1 = (y_0^{(1)}, \Lambda^{(1)}, \Gamma^{(1)})$ and $\sigma_2 = (y_0^{(2)}, \Lambda^{(2)}, \Gamma^{(2)})$ and outputs a tag $\sigma = (y_0, \Lambda, \Gamma)$ which is computed as follows:

$$y_0 = y_0^{(1)} + y_0^{(2)}, \quad \Lambda = \Lambda^{(1)} \cdot \Lambda^{(2)}, \quad \Gamma = \Gamma^{(1)} \cdot \Gamma^{(2)}$$

- $\mathsf{ConstMult_1}(\mathsf{ek}, \sigma_1, c)$. This takes as input a tag $\sigma_1 = (y_0^{(1)}, \Lambda^{(1)}, \Gamma^{(1)})$ and a constant $c \in \mathbb{Z}_p$ and outputs the tag $\sigma = (y_0, \Lambda, \Gamma)$ which is computed as follows:
$$y_0 = c \cdot y_0^{(1)}, \quad \Lambda = (\Lambda^{(1)})^c, \quad \Gamma = (\Gamma^{(1)})^c$$
- $\mathsf{Mult_1}(\mathsf{ek}, \sigma_1, \sigma_2)$. This takes as input two tags $\sigma_1 = (y_0^{(1)}, \Lambda^{(1)}, \Gamma^{(1)})$ and $\sigma_2 = (y_0^{(2)}, \Lambda^{(2)}, \Gamma^{(2)})$ and outputs a tag $\sigma = \Omega$ for the multiplication, computed as follows:
$$\Omega = e(\Lambda^{(1)}, \Gamma^{(2)}) \cdot e(\Lambda^{(1)}, \gamma^{y_0^{(2)}}) \cdot e(\Lambda^{(2)}, \gamma^{y_0^{(1)}})$$
- $\mathsf{Shift_{1\to 2}}(\mathsf{ek}, \sigma_1)$. This takes as input a tag $\sigma_1 = (y_0^{(1)}, \Lambda^{(1)}, \Gamma^{(1)})$ and outputs $\sigma = \Omega$ computed as $\sigma \leftarrow \mathsf{Mult_1}(\mathsf{ek}, \sigma_1, \sigma_U)$. Here we are using multiplication by 1 to obtain a tag that is valid for the same message, but that it is of the $\mathbb{G}_T$ form. The only case in which this algorithm is needed is before running the following algorithm $\mathsf{Add_2}$ (i.e., when one needs to compute an addition after the last multiplication) and one of the two input tags σ_1, σ_2 is not of the $\mathbb{G}_T$ form.
- $\mathsf{Add_2}(\mathsf{ek}, \sigma_1, \sigma_2)$. This takes as input two tags $\sigma_1 = \Omega^{(1)} \in \mathbb{G}_T$ and $\sigma_2 = \Omega^{(2)} \in \mathbb{G}_T$ and outputs $\sigma = \Omega$ which is computed as:
$$\Omega = \Omega^{(1)} \cdot \Omega^{(2)}$$
- $\mathsf{ConstMult_2}(\mathsf{ek}, \sigma_1, c)$. This takes as input a tag $\sigma_1 = \Omega^{(1)}$ and a constant $c \in \mathbb{Z}_p$ and outputs $\sigma = \Omega$ which is computed as:
$$\Omega = (\Omega^{(1)})^c$$

$\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$**.** Let $\mathcal{P} = (f, \tau_1, \dots, \tau_n)$ be a labeled program, $m \in \mathbb{Z}_p$ and σ be a tag of either the form $(y_0, y_1) \in \mathbb{Z}_p^2$, or $(y_0, \Lambda, \Gamma) \in \mathbb{Z}_p \times \mathbb{G}^2$, or $\sigma = \Omega$.

First, proceed as in the construction of Section 3 to compute $\rho = f(r_{\tau_1}, \dots, r_{\tau_n})$.

Next, according to the form of σ make the following checks:

1. If $\sigma = (y_0, y_1)$, then output 1 only if
$$\rho = y_0 + y_1 \cdot x \ \wedge \ y_0 = m.$$
2. If $\sigma = (y_0, \Lambda, \Gamma)$, then output 1 only if
$$g^\rho = g^{y_0} \cdot \Lambda \ \wedge \ y_0 = m \ \wedge \ e(\Lambda, \gamma) = e(\Gamma, g)$$
3. If $\sigma = \Omega$, then output 1 only if
$$e(g, \gamma)^{\rho - m} = \Omega$$

Correctness. The correctness basically follows from the correctness of the scheme described in Section 4. The only less trivial fact to observe is what happens in the procedure $\mathsf{Mult_1}$ which computes the multiplication of two tags $\sigma_1 = (y_0^{(1)}, \Lambda^{(1)}, \Gamma^{(1)})$, $\sigma_2 = (y_0^{(2)}, \Lambda^{(2)}, \Gamma^{(2)})$ (that we assume to be valid for messages m_1 and m_2 respectively).

By definition we have that $\Lambda^{(i)} = g^{(y^{(i)}(x) - y^{(i)}(0))}$ while $\Gamma^{(i)} = (\Lambda^{(i)})^a$.

$$\begin{aligned}
\Omega &= e(\Lambda^{(1)}, \Gamma^{(2)}) \cdot e(\Lambda^{(1)}, \gamma^{y_0^{(2)}}) \cdot e(\Lambda^{(2)}, \gamma^{y_0^{(1)}}) \\
&= e(g, \gamma)^{(y^{(1)}(x) - y^{(1)}(0))(y^{(2)}(x) - y^{(2)}(0)) + y^{(2)}(x) y^{(1)}(0) + y^{(1)}(x) y^{(2)}(0) + y^{(1)}(0) y^{(2)}(0)} \\
&= e(g, \gamma)^{y^{(1)}(x) y^{(2)}(x) - y^{(1)}(0) y^{(2)}(0)} \\
&= e(g, \gamma)^{\rho_1 \rho_2 - m_1 m_2}
\end{aligned}$$

where the last equality follows from the correctness of the tags σ_1, σ_2.

A.1 Proof of Security

Theorem 5. *If F is a PRF and the D-Bilinear Diffie Hellman Inversion Assumption holds in $\mathbb{G}, \mathbb{G}_T$, then the homomorphic MAC scheme described in Section A is secure.*

To prove the security of our scheme we define the following hybrid games. We denote with G_i the event that the experiment Game i, run with the adversary $\mathcal{A}$, outputs 1.

Game 0: this is the same as the real $\mathsf{HomUF-CMA}$ experiment, except that in every verification query $(m, \mathcal{P}, \sigma)$, in order to check whether $\mathcal{P}$ is well-defined or not, the challenger uses the probabilistic test of Proposition 1.
Thus we have that for all adversaries $\mathcal{A}$ making at most Q verification queries we have

$$|\Pr[\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HomMAC}}(\lambda)] - \Pr[G_0(\mathcal{A})]| \leq Q \cdot 2^{-\lambda} \tag{18}$$

Game 1: this is the same as Game 0, except that the PRF is replaced with a truly random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$: basically, each value r_τ is generated uniformly at random in $\mathbb{Z}_p$.

Game 2 : This is the same as Game 1, except for the following changes. Let bad_2 be a flag value which is initially set to false. Here we set bad_2 to true if we ever receive a query (either a verification query or an authentication one) containing the label α.

Game 3: This is the same as Game 2 except for a change in answering verification queries. Let bad_3 be a flag value which is initially set to false. For all verification queries $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is *not* well defined on T, the challenger answers 0 (reject) and proceeds as follows.
First, for every τ_i such that $(\tau_i, \cdot) \notin T$ it takes $r_{\tau_i} \leftarrow \mathcal{R}(\tau_i)$. Next, it computes ρ using the internal procedure of Ver.
If $\sigma = (y_0, y_1)$, compute $z = \rho - y_0 - x \cdot y_1$. If $z = 0 \pmod p$, then set $\mathsf{bad}_3 \leftarrow \mathsf{true}$.
If $\sigma = (y_0, \Lambda, \Gamma)$ compute $Z = g^{\rho - m} \Lambda^{-1}$. If $Z = 1$, then set $\mathsf{bad}_3 \leftarrow \mathsf{true}$.
If $\sigma = \Omega$ compute $Z = e(g, \gamma)^{\rho + m} \Omega^{-1}$. If $Z = 1$, then set $\mathsf{bad}_3 \leftarrow \mathsf{true}$.

Game 4: this is the same as Game 3 except for the following change in answering authentication queries. Given a query (τ, m) such that $(\tau, m) \notin T$, if $r_\tau = \mathcal{R}(\tau)$ was previously used to answer a verification query, then resample a fresh $r'_\tau \xleftarrow{\$} \mathbb{Z}_p$ to create the tag, and from now on use r'_τ in every verification query involving label τ.

Game 5: this is the same as Game 4 except for the following change in answering verification queries. Let bad_5 be a flag value which is initially set to false. For every verification query $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T, the challenger proceeds as follows.
For every index $i \in \{1, \ldots, n\}$ such that $(\tau_i, \cdot) \notin T$ (i.e., $\mathcal{P}$ contains a new label) it chooses a dummy tag σ_i (e.g., for a random message).
Let $\sigma_1, \ldots, \sigma_n$ be the tags associated to labels $\tau_1, \ldots, \tau_n$ respectively. If $\sigma = (y_0, y_1)$ or $\sigma = (y_0, \Lambda, \Gamma)$, then the challenger uses $\mathsf{Eval}_1(\mathsf{ek}, f, (\sigma_1, \ldots, \sigma_n))$ to homomorphically compute $\hat{\sigma}$ (which is either $(\hat{y}_0, \hat{y}_1)$, or $\hat{\Lambda}$ respectively). If $\hat{\sigma} = \hat{\Omega}$, instead it computes $\hat{\sigma}$ as follows. First, it computes $y_0, \ldots, y_d$ as in Eval_1. Next, it goes back to the last multiplication level before the end of the evaluation of f. Let $y^{(1)}, \ldots, y^{(t)}$ be the tags that were obtained immediately before reaching the last multiplication level. The challenger first "transforms" such tags into tags of the form $(y_0^{(i)}, \Lambda^{(i)}, \Gamma^{(i)})$ and then it runs Eval_2 on such tags to obtain $\hat{\sigma} = \hat{\Omega}$.
Let $\hat{\sigma}$ be the tag obtained by the honest homomorphic evaluation as described before. If $\sigma = \hat{\sigma}$, then the challenger outputs 1 (accept). Otherwise, it outputs 0 (reject).
Moreover,

1. If $(y_0, y_1) \neq (\hat{y}_0, \hat{y}_1)$, compute $z = (y_0 - \hat{y}_0) + x(y_1 - \hat{y}_1)$. If $z = 0 \pmod{p}$, then set $\mathsf{bad}_5 \leftarrow \mathsf{true}$.
2. If $\Lambda \neq \hat{\Lambda}$, compute $Z = (\Lambda/\hat{\Lambda})g^{(m-\hat{m})}$. If $Z = 1$, then set $\mathsf{bad}_5 \leftarrow \mathsf{true}$.
3. If $\Omega \neq \hat{\Omega}$, compute $Z = e(g, \gamma)^{m-\hat{m}}(\hat{\Omega}/\Omega)$. If $Z = 1$, then set $\mathsf{bad}_5 \leftarrow \mathsf{true}$.

To prove the theorem we prove the following claims. Most of them have proofs similar to those used in the proof of Theorem 4.

Claim 11 $|\Pr[G_0] - \Pr[G_1]| \leq \mathbf{Adv}^{PRF}_{\mathcal{B},F}(\lambda)$.

The proof can be obtained via a straightforward reduction to the security of the PRF.

Claim 12 $|\Pr[G_1] - \Pr[G_2]| \leq \frac{nQ}{2^\lambda}$ *where n is the maximum number of allowed input labels per labeled program and Q is the number of queries made by the adversary $\mathcal{A}$ during the experiment.*

Proof. First notice that the first change introduced in this game is merely syntactical and does not affect the quality of the verification procedure. As for the second modification the stated bound can be easily be obtained by simply counting the number of maximum different labels that $\mathcal{A}$ can use in its queries. This is at most a polynomial number n for each of the Q queries. Also, notice that even tough the value $\mathcal{R}(\alpha) = r_\alpha$ is only computationally hidden by the values contained in ek, the label α remains information theoretically hidden even given r_α (as described in Game 2, we are now using a truly random function to generate r_α from α). □

Claim 13 $|\Pr[G_2] - \Pr[G_3]| \leq \frac{Q(d+1)}{p-d(Q-1)}$ *where p is the order of the group used in the scheme and Q is the number of verification queries made by the adversary $\mathcal{A}$ during the experiment.*

The proof is essentially the same as the proof of Claim 7.

Claim 14 $|\Pr[G_3] - \Pr[G_4]| \leq \frac{DQ^2}{p}$ *where p is the order of the group used in the scheme and Q is the number of verification queries made by the adversary $\mathcal{A}$ during the experiment.*

The proof is essentially the same as the proof of Claim 8.

Claim 15 $\Pr[G_4] \equiv \Pr[G_5 \wedge \neg\mathsf{Bad}_5]$.

Proof. Let Bad_5 be the event that bad_5 is set true in Game 5. If the event Bad_4 does not occur, then we claim that Game 4 is identical to Game 3. The only change is in the way the challenger answers verification queries $(m, \mathcal{P}, \sigma)$ for a program $\mathcal{P}$ that is well defined on T, and we claim that such change is only syntactic, and thus it does not change the adversary's view of the game.

Let $(m, \mathcal{P}, \sigma)$ be a verification query. If $\sigma = (y_0, y_1)$ then by the same argument in the proof of Claim 10 the answer is correct. If $\sigma = (y_0, \Lambda, \Gamma) \neq (\hat{y}_0, \hat{\Lambda}, \hat{\Gamma}) = \hat{\sigma}$, then observe that:

- if $(y_0, \Lambda) \neq (\hat{y}_0, \hat{\Lambda})$, then the answer is correct by the same argument in Claim 9;
- if $(y_0, \Lambda) = (\hat{y}_0, \hat{\Lambda})$ and $\Gamma \neq \hat{\Gamma}$, then $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ would output 1 only if $e(\Gamma, g) = e(\Lambda, \gamma)$. Since $\Lambda = \hat{\Lambda}$, we have that $e(\Lambda, \gamma) = e(\hat{\Lambda}, \gamma) = e(\hat{\Gamma}, g)$. Hence, returning 1 only if $\Gamma = \hat{\Gamma}$ (i.e., $e(\Gamma, g) = e(\hat{\Gamma}, g)$) is the same as the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$.

We consider the remaining case in which the queried tag σ is of the form $\sigma = \Omega \in \mathbb{G}_T$. Similarly to the proof of Claim 9 we only consider the case when $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well defined on T and $\forall i = 1, \ldots, n$, $(\tau_i, m_i) \in T$. Indeed, as we have shown, the case in which $\mathcal{P}$ is well defined and there is some $(\tau_i, \cdot) \notin T$ is essentially the same.

Now, if we analyze how the challenger computes $\hat{\sigma} = \hat{\Omega}$ (which is also the way $\mathcal{B}$ does), then this is basically a way to compute a valid tag (of the $\mathbb{G}_T$ form) for the circuit f. By correctness, it must be $e(g, \gamma)^{\rho + \hat{m}} = \hat{\Omega}$ where ρ is the value computed by the verification algorithm as $f(r_{\tau_1}, \ldots, r_{\tau_n})$. ρ is the same when running both $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ and $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \hat{\sigma})$ as the very same r_τ's are used. In order for $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$ to output 1, it must hold $e(g, \gamma)^{\rho + m} = \Omega$. So, returning 1 only if $Z = e(g, \gamma)^{m - \hat{m}}(\hat{\Omega}/\Omega) = 1$ is the same as returning the output of $\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \sigma)$. □

Claim 16 $\Pr[\mathsf{Bad}_5] \leq Q \cdot \mathbf{Adv}_{\mathcal{B}}^{BDHI}(\lambda)$ *where Q is the number of verification queries made by the adversary $\mathcal{A}$ in Game 5.*

Proof. Assume by contradiction that there exists an adversary $\mathcal{A}$ such that, when run in Game 4, we have $\Pr[\mathsf{Bad}_5] \geq \epsilon(\lambda)$ for some non-negligible function ϵ. Then, we show how to build an efficient simulator $\mathcal{B}$ that breaks the D-BDHI assumption with advantage $\mathbf{Adv}_{\mathcal{B}}^{BDHI}(\lambda) \geq \epsilon(\lambda)/Q$.

If Bad_5 occurs in Game 5, then there must exist an index $1 \leq \mu \leq Q$ such that $\mathsf{bad}_5 \leftarrow \mathsf{true}$ is set in the μ-th verification query.

$\mathcal{B}$ takes as input a tuple $(\eta, \eta^x, \ldots, \eta^{x^\ell})$ and its goal is to compute $e(\eta, \eta)^{1/x}$. $\mathcal{B}$ picks a random $\beta \xleftarrow{\$} \mathbb{Z}_p$ and defines $\gamma = \eta^\beta$. Next, for $i = 1$ to D, it sets $\gamma_i = (\eta^{x^i})^\beta$ and $h_i = \eta^{x^{i-1}}$. It also simulates the tag σ_U for the value 1 using "1" as label. This is done by using the same procedure described below for the simulation of authentication queries. Let Q be an upper bound on the number of verification queries made by the adversary. $\mathcal{B}$ chooses $\mu^* \xleftarrow{\$} \{1, \ldots, Q\}$ uniformly at random as a guess for the index μ in which bad_5 is updated to true. Finally, $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{ek} = (\gamma, h_1, \gamma_1, \ldots, h_D, \gamma_D, \sigma_U)$ and it answers queries as follows.

Authentication Queries. Given an authentication query (m, τ) it chooses $y_1 \xleftarrow{\$} \mathbb{Z}_p$ at random, sets $y_0 = m$ and returns $\sigma = (y_0, y_1)$. Notice that this alternative generation of the tag without using x generates the same distribution of tags as the one in Game 5, in which a fresh value r'_τ is used in every authentication query.

Verification Queries. On input the j-th verification query $(m, \mathcal{P}, \sigma)$, $\mathcal{B}$ proceeds as follows:

- If $\mathcal{P}$ is not well defined on T, output 0 (reject). Notice that this answer is correct by the change introduced in Game 2.
- If $\mathcal{P}$ is well defined compute $\hat{\sigma}$ as the challenger in Game 5, i.e., by using either Eval_1 or the modified version of Eval_2. If $\sigma = \hat{\sigma}$ $\mathcal{B}$ outputs 1 (accept).
- If $\mathcal{P}$ is well defined, $\sigma \neq \hat{\sigma}$ and $j \neq \mu^*$, then output 0 (reject).
- If $\mathcal{P}$ is well defined, $\sigma \neq \hat{\sigma}$ and $j = \mu^*$, then output 0 (reject), and proceed as follows:
  - If $\sigma = (y_0, y_1)$ (and $\hat{\sigma} = (\hat{y}_0, \hat{y}_1)$), compute $x' = (y_0 - \hat{y}_0)/(\hat{y}_1 - y_1) \pmod p$, and $U = e(\eta, \eta)^{1/x'}$
  - If $\sigma = (y_0, \Lambda, \Gamma)$ (and $\hat{\sigma} = (\hat{y}_0, \hat{\Lambda}, \hat{\Gamma})$), compute $\eta' = (\hat{\Lambda}/\Lambda)^{(y_0 - \hat{y}_0)^{-1}}$ and $U = e(\eta', \eta)$.
  - If $\sigma = \Omega$ (and $\hat{\sigma} = \hat{\Omega}$), compute $U = (\Omega/\hat{\Omega})^{(\beta(m - \hat{m}))^{-1}}$

At the end of the simulation $\mathcal{B}$ outputs U. If $\mathcal{B}$ correctly guessed the index $\mu^* = \mu$ in which the adversary queries the forgery, then it is not hard to verify that $\mathcal{B}$ is correctly computing

$U = e(\eta, \eta)^{1/x}$ with probability at least $\epsilon(\lambda)$ (i.e., $\mathcal{A}$'s success probability). Therefore, we have:

$$\mathbf{Adv}_{\mathcal{B}}^{BDHI}(\lambda) = \Pr[\mathsf{Bad}_5 \wedge \mu^* = \mu] \geq \Pr[\mu^* = \mu] \Pr[\mathsf{Bad}_5] \geq \frac{\epsilon(\lambda)}{Q}$$

which concludes the proof of the Claim. □

To conclude the proof of the Theorem, observe that $\Pr[G_5] = 0$ as all verification queries to Type 1 and Type 2 forgeries are answered with 0. So, if we put together the results of the above claims we obtain

$$\mathbf{Adv}_{\mathcal{A},\mathsf{HomMAC}}^{\mathsf{HomUF-CMA}}(\lambda) \leq \mathbf{Adv}_{\mathcal{B},F}^{PRF}(\lambda) + \frac{(n+1)Q}{2^\lambda} + \frac{Q(D+1)}{p - D(Q-1)} + \frac{DQ^2}{p} + Q \cdot \mathbf{Adv}_{\mathcal{B}}^{BDHI}(\lambda).$$

Since $p \approx 2^\lambda$ and D, n and Q are $\mathsf{poly}(\lambda)$, if the PRF is secure and the D-BDHI assumption holds then $\mathcal{A}$ has at most negligible advantage of breaking the unforgeability of our construction.