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] Max Planck Institute for Software Systems (MPI-SWS), Germany
fiore@mpi-sws.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

⋆ Work done while Postdoctoral researcher at NYU

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 $m_1, \ldots, m_n$ to a server, and later asks the server to run a program $\mathcal{P}$ over $(m_1, \ldots, m_n)$. The server computes $m \leftarrow \mathcal{P}(m_1, \ldots, m_n)$ 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 $m_1, \ldots, m_n$ to the client, who can then compute/check $m = \mathcal{P}(m_1, \ldots, m_n)$ by itself. This however vanishes the advantages of the outsourcing and is too costly in terms of 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, 17, 3] and memory delegation [18].

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* [11]. 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 [13, 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). 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

[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).

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 the full version of this paper 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.

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

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

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 [10, 22, 12, 14] and in the standard model [1, 4, 15, 16, 20, 5, 6]. Linearly-homomorphic message authenticators have been considered also for proofs of retrievability for outsourced storage [32]. 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 [11]. 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) [8]. 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. A drawback of 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 [34, 9].

Other Related Work. The notion of homomorphic authenticators is also (somewhat) related to the notion of verifiable computation [28, 29, 26, 21, 17, 3, 7, 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

[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.

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 [18]. This primitive allows a client to outsource large amounts of data to a server so that he can later verify computations 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. For lack of space, all proofs will appear in the full version of this paper.

2 Background and Definitions

Arithmetic Circuits. Here we provide a very brief overview of arithmetic circuits. The interested reader is referred to [33] 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 $\times$ 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.

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.

[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.

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 as follows: by fixing all inputs in T one writes the computation as a new multivariate polynomial whose variables are only the inputs not in T. Then, one checks whether this polynomial is a constant function.

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 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 pseudo-random function $F_K : \{0,1\}^* \to \mathbb{Z}_p$ and a random value $x \xleftarrow{\$} \mathbb{Z}_p$. 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[7].

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 $O(d)$.

Correctness. Very roughly, correctness follows from the special property of the polynomials y generated by Auth, i.e., that $y(0) = m$ and $y(x) = r_\tau$. In particular, this property is preserved when evaluating the circuit f over tags $y^{(1)}, \ldots, y^{(n)}$. We give a formal proof of correctness in the full version of this paper.

Security. The security of our scheme is established by the following theorem (again the proof is deferred to the full version of this paper).

Theorem 1. *If F is a PRF, then the homomorphic MAC scheme described in Section 3 is secure.*

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.

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

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 [13, 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 $\mathbf{Adv}^{DHI}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(g, g^x, \ldots, g^{x^\ell}) = g^{1/x}]$ 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 $\mathbf{Adv}^{DHI}_{\mathcal{A}}(\lambda)$ 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 $\rho = y_0 + y_1 \cdot x$ and $y_0 = m$. Otherwise, use g to check whether the following equation holds:

$$g^\rho = g^m \cdot \Lambda \tag{2}$$

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 (2) 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 the full version of this paper 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.

Security. Security follows from the following theorem (whose proof is postponed to the full version of this paper).

Theorem 2. *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.*

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, Paris-Rocquencourt, France, June 2–5, 2009. Springer, Berlin, Germany.
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, Taormina, Sicily, Italy, Mar. 19–21, 2012. Springer, Berlin, Germany.
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, Bordeaux, France, July 6–10, 2010. Springer, Berlin, Germany.
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, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.
5. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 367–385. Springer, Berlin, Germany, Dec. 2012.

6. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC 2013*, volume 7778 of *LNCS*, pages 386–404. Springer, Berlin, Germany, 2013.
7. 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, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
8. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS '12: Proceedings of the 3rd Symposium on Innovations in Theoretical Computer Science*, 2012.
9. 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/.
10. 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, Irvine, CA, USA, Mar. 18–20, 2009. Springer, Berlin, Germany.
11. 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, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
12. 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, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.
13. 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, Egham, UK, Sept. 1–3, 2008. Springer, Berlin, Germany.
14. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one way functions and their applications. In *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Berlin, Germany, 2013.
15. 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, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
16. 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, Darmstadt, Germany, May 21–23, 2012. Springer, Berlin, Germany.
17. 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, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
18. 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, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
19. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *2012 ACM Conference on Computer and Communication Security*. Full version available at `http://eprint.iacr.org/2012/281`. ACM Press, October 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, Darmstadt, Germany, May 21–23, 2012. Springer, Berlin, Germany.
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, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
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, Paris, France, May 26–28, 2010. Springer, Berlin, Germany.
23. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. Cryptology ePrint Archive, Report 2012/290, 2012. http://eprint.iacr.org/.
24. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, Maryland, USA, May 31 – June 2, 2009. ACM Press.
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, San Jose, California, USA, June 6–8, 2011. ACM Press.
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, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.
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, San Jose, CA, USA, Feb. 18–22, 2002. Springer, Berlin, Germany.
28. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *24th ACM STOC*, pages 723–732, Victoria, British Columbia, Canada, May 4–6, 1992. ACM Press.
29. S. Micali. Cs proofs. In *35th FOCS*, Santa Fe, New Mexico, Nov. 20–22, 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, Taormina, Sicily, Italy, Mar. 19–21, 2012. Springer, Berlin, Germany.
32. H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107, Melbourne, Australia, Dec. 7–11, 2008. Springer, Berlin, Germany.
33. 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.
34. 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, San Francisco, CA, USA, Mar. 19–21, 2008. Springer, Berlin, Germany.