VESPo: Verified Evaluation of Secret Polynomials

Jean-Guillaume Dumas, Aude Maignan, Clément Pernet, Daniel S. Roche

VESPo: Verified Evaluation of Secret Polynomials

(with application to dynamic proofs of retrievability)

Jean-Guillaume Dumas* Aude Maignan* Clément Pernet* Daniel S. Roche †

February 21, 2022

Abstract

We consider the problem of efficiently evaluating a secret polynomial at a given public point, when the polynomial is stored on an untrusted server. The server performs the evaluation and returns a certificate, and the client can efficiently check that the evaluation is correct using some pre-computed keys. Our protocols support two important features: the polynomial itself can be encrypted on the server, and it can be dynamically updated by changing individual coefficients cheaply without redoing the entire setup. As an important application, we show how these new techniques can be used to instantiate a Dynamic Proof of Retrievability (DPoR) for arbitrary outsourced data storage that achieves both low server storage size and audit complexity. Our methods rely only on linearly homomorphic encryption and pairings, and preliminary timing results indicate reasonable performance for polynomials with millions of coefficients, and efficient DPoR with for instance 1TB size databases.

1 Introduction

Verifiable computing.

Verifiable computing, first formalized by [23], consists in delegating the computation of some function to an untrusted server, who must return the result as well as a proof of its correctness. Generally, verifying a result should be much less expensive than computing it directly, and result in a provably low probability that the result is incorrect. While certified and verified computation protocols date back decades, the practical need for efficient methods is especially evident in cloud computing, wherein a low-powered device such as a mobile phone may wish to outsource expensive and critical computations to an untrusted, shared-resource commercial cloud provider.

The extensive literature on verifiable computation protocols can be divided into general-purpose computations — of an arbitrary algebraic circuit — and more limited and (hopefully) efficient special-purpose computations of certain functions. In the latter category, one of the most important problems is Verifiable Polynomial Evaluation (VPE), where a client wishes to outsource the evaluation of a univariate polynomial P at a given point x and efficiently verify the result.

Verifiable Polynomial Evaluation.

A VPE scheme is conventionally composed of three algorithms. First, a client runs $\mathsf{Setup}(P)$ to compute some public representation of P (which may be stored on the server) as well as some private information which will be used to verify later evaluations. This step may be somewhat expensive, but only needs to be performed once.

The second algorithm, $\mathsf{Eval}(x, \alpha)$, is run by the server using a public evaluation point x, as well as possibly some additional information α provided by the client. The server produces the evaluation $y = P(x)$ as well as some proof or certificate β that this evaluation is correct.

*Université Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, Grenoble INP. 700 avenue centrale, IMAG — CS 40700, 38058 Grenoble, France. \{Jean-Guillaume.Dumas,Aude.Maignan,Clement.Pernet\}@univ-grenoble-alpes.fr.

†United States Naval Academy, Annapolis, Maryland, United States. roche@usna.edu.

Finally, the third algorithm, $\mathsf{Verify}(y, \beta)$, is run by the client to check the correctness of the evaluation. This verification should be *always correct* and *probabilistically sound*, meaning that an honest server can always produce a result y and proof β that will pass the verification, whereas an incorrect evaluation y will always fail the verification with high probability. Furthermore, the Verify algorithm should be efficient, ideally much cheaper in time and/or space than the computation itself.

Additional protocol features.

In the simplest case, the considered polynomial P is static and stored in cleartext by both the server and the client. But constraints can then be added to this framework.

- **Polynomial outsourcing**. When the client device has limited storage, or to facilitate multiple clients, the polynomial and its computation must be externalized. This can always be trivially achieved by storing all client secrets on the server via symmetric encryption and a saved cryptographic hash digest; the challenge is to do so while minimizing the communication costs required for the client to verify an evaluation.

- **Secret polynomial**. In some cases, to guarantee data privacy, the polynomial has to be hidden from the server, or the client, or both. Typically, the polynomial will be stored under a fully- or partially-homomorphic encryption scheme, in such a way that the server can still compute the (necessarily encrypted) evaluation and certificate for verification. This setting has been extensively studied in the literature, with both general-purpose protocols as well as some specific for verified polynomial evaluation.

- **Public verification** The verification protocol is said to be *private* when only a party which holds the secrets derived during Setup can verify evaluations. That is, any potential verifiers (sometimes called *readers*) must be trusted not to divulge secret information to the untrusted server. In many applications, it is desirable also to have untrusted verifiers, who can check the result of an evaluation without knowing any secrets. In this *public verification* setting, the client at setup time publishes some additional information, distributed reliably but insecurely to any potential verifiers, which may be used to check evaluations and proofs issued by the server.

- **Dynamic updates**. The initial Setup protocol requires knowledge of the entire polynomial and generally is much more costly than running Verify. This creates a challenge when the client wishes to update only a few of the coefficients of the polynomial and later compute evaluations of this new, modified polynomial P'. A *dynamic* VPE protocol allows for such updates efficiently. Namely, the client and server storing polynomial P for verified evaluation can engage in an additional $\mathsf{Update}(c, d)$ protocol, which effectively updates $P(x)$ to $P(x) + cx^d$ for future evaluations, along with any secret and/or public verification information. To the best of our knowledge, no prior work in the literature discusses dynamic updates for verified polynomial evaluation, which is especially challenging when the polynomial (as well as any update) needs to be hidden from the server. The importance of allowing efficient updates is motivated by our application to verifiable data storage, which we explain next.

Proofs of Retrievability.

One important application of VC in general, and VPE in particular, is to *Proofs of Retrievability* (PoR), somewhat overlapping with the problem of *Provable Data Possession* (PDP) [25, 6]. In these settings, a client wishes to store her data on an untrusted server, then verify (without full retrieval) that the server still stores the data intact. The crucial protocol is an Audit, wherein the client issues some challenge to the server, then verifies the response using some pre-computed information to prove that the original data is still recoverable in its entirety.

A variety of tools have been employed to develop efficient PoR and PDP protocols, and some of these are based on verifiable computing, so that a PoR audit consists of some verified computation over the stored data. Retrievability is proven when any sequence of successful audits can, with high probability, be used to recover the original data, e.g., by polynomial interpolation; thus any server with a good chance to pass a random audit must hold the entire data intact.

Note that this recovery mechanism is not actually crucial except to *prove* the soundness of the audit protocol; the important feature is how cheaply the audits can be performed by a server and resource-constrained client.

1.1 Our contributions

Our contributions are the following:

- An (unencrypted) Verifiable Polynomial Evaluation (VPE) scheme with public verification which is the first to support *dynamic updates*, meaning that updating only a few coefficients of P does not require performing the whole setup phase again (Section 4 and Table 3).

  The polynomial is stored in cleartext on the server, and the technique used to provide a correct and sound protocol uses both Merkle trees and pairings. A Horner-like evaluation scheme is used to optimize the evaluation of the difference polynomial for the proof, and no secrets are required to perform the verification.

- A novel *encrypted, dynamic and private* VPE protocol (Section 5 and Table 5). That is, the polynomial is stored encrypted on the server, and efficient updates to individual coefficients can be performed.

  This is achieved by combining a linearly homomorphic cryptosystem with techniques from the first scheme. Note however, this scheme does not support public verification as this verification now requires some secrets from the client.

- A new Proof of Retrievability (PoR) scheme that is the first to simultaneously support small server storage, dynamic updates, and efficient audits (Section 6 and Table 9), based on our novel encrypted, dynamic VPE protocol.

  Previous work either had linear extra storage and poly-logarithmic audits, or sub-linear extra storage and polynomial audits; ours is the first to achieve both sub-linear extra storage and optimal $\mathcal{O}(\log n)$ client time for updates and audits.

- Experimental timings based on of our encrypted VPE and dynamic PoR protocols that indicate VPE up to millions of coefficients and PoR up to 1TB of data, both with client cost less than a few milliseconds (Tables 7 and 10).

A complete security definition of verifiable polynomial evaluation can be found in Section 2. This definition follows previous results, with the novel inclusion of an Update protocol. Then Section 3 introduces the tools for verification of polynomial evaluation. A motivating example is presented in the form of a direct extension of the bilinear pairing scheme of [26], now supporting an encrypted input polynomial (Section 3 and Table 2). Since the privacy of this protocol is not proven and it does not support neither public verifiability nor dynamic updates, it motivates the more involved contributions of Section 4 (for public verifiability and dynamicity, but on an unciphered polynomial) and of Section 5 (for dynamicity on a ciphered polynomial, but without public verifiability).

The efficiency of our protocols is measured by the computational complexity of the server-side Eval algorithm, the volume of persistent client storage, and the amount of communication and client-side complexity to perform a Verify or Audit. Improving on previously-known results, our protocols all have $\mathcal{O}(d)$ server-side computation, $\mathcal{O}(\log d)$ communication and client-side computation time, and $\mathcal{O}(1)$ client-side persistent storage. We include some practical timings in Sections 5.2 and 6.3.

In addition, our new dynamic PoR scheme requires only $o(d)$ extra server space. This improves on [36] in terms of server storage and on [5] in terms of communication and client computation complexity for Audit. For instance on a 1TB size database, with a server extra storage lower than 0.08%, and a client persistent storage less than one KB, our client can check in less than 7ms that their entire outsourced data is fully recoverable from the cloud server.

1.2 Related work

While ours is the first work we are aware of which considers verifiable polynomial computation while hiding the polynomial from the server and allowing efficient dynamic updates, there have been a number of prior works on different settings of the VPE problem.

One line of work considers *commitment schemes* for polynomial evaluation [15, 13, 30, 21, 37, 11, 33, 19]. There, the polynomial P is known only to the server, who publishes a binding commitment without revealing P itself. The verifier then confirms that a given evaluation is consistent with the pre-published commitment. The protocol of Kate *et. al.* [26], which fits in this model, introduced some of the techniques that we employ for our private verification algorithm, namely the notion of the difference polynomial (see Section 3). By contrast, our protocols aim to *hide the polynomial P from the server*.

Another line of work considers polynomial evaluation as an encrypted function, which can be evaluated at any chosen point. Function-hiding inner product encryption (IPE) [10, 28, 2] can be used to perform polynomial evaluation without revealing the polynomial P, but this inherently requires linear-time for the client, who must compute the first d powers of the desired evaluation point x.

Similarly, protocols using a Private Polynomial Evaluation (PPE) scheme have been developed in [12]. This primitive, based on an ElGamal scheme, ensures that the polynomial is protected and that the user is able to verify the result given by the server. Here the aim of the protocol is not to outsource the polynomial evaluation, but to obtain $P(x)$ and a proof without knowing anything about the polynomial. To check the proof, as with IPE the client has to produce a computation which is linear in the degree of P.

A third and more general approach which can be applied to the VPE problem is that of secure evaluation of arithmetic circuits. These protocols make use of fully homomorphic encryption (FHE) to outsource the evaluation of an arbitrary arithmetic circuit without revealing the circuit itself to the server. The VC Scheme of [23] is based on Yao's label construction. P is first transformed into an arithmetic circuit. The circuit is garbled once in a setup phase and sent to the server. To later perform a verified evaluation, the client sends an encryption of x, the server computes $P(x)$ through the garbled circuit, and the client can verify the result in time proportional to the circuit depth, which for us is $\mathcal{O}(\log d)$

Using similar techniques, Fiore et al. and Elkhiyaoui et al. [8, 18, 16] propose high-degree polynomial evaluations with a fully secure public verification solution. Very recently, Fiore et al. [19] propose a new protocol for more general circuits, using SNARKs over a quotient polynomial ring. In contrast to our work, these protocols use more expensive cryptographic primitives, and they do not consider the possibility of efficiently updating the polynomial.

Then, Proof of retrievability (PoR) and Provable data possession (PDP) protocols also have an extensive literature. PDPs generally optimize server storage and efficiency at the cost of soundness; a PDP audit may succeed even when a constant fraction of the data is unrecoverable. PoRs have stronger soundness guarantees, but at the expense of larger and more complicated server storage, often based on erasure codes and/or ORAM techniques.

State-of-the-art PoR protocols either incur a constant-factor blowup in server storage with poly-logarithmic audit cost [14, 36], or use negligible extra server storage space but require polynomial-time for audits on the client and server [35, 5]. A lower bound argument from [5] proves that some time/space tradeoff is inherent, although the proof does not distinguish between server and client computation time during audits.

2 Security properties and assumptions

A verifiable dynamic polynomial evaluation (VDPE) scheme consists of three algorithms: `Setup`, `Update`, `VEval`, between a client $\mathcal{C}$ with state $st_{\mathcal{C}}$, a server $\mathcal{S}$ with state $st_{\mathcal{S}}$ and a verifier $\mathcal{V}$ with (potentially public) state $st_{\mathcal{V}}$.

- $(st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}}) \leftarrow \texttt{Setup}(1^{\kappa}, P)$: On input of the security parameters and the polynomial P of degree d, outputs the client state $st_{\mathcal{C}}$, the verifier $st_{\mathcal{V}}$ and the server state $st_{\mathcal{S}}$.
- $\{(st'_{\mathcal{C}}, st'_{\mathcal{V}}, st'_{\mathcal{S}}), \texttt{reject}\} \leftarrow \texttt{Update}(i, \delta, st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}})$: On input of an index $i \in 0..d$, data δ, the client/verifier/server states $st_{\mathcal{C}}/st_{\mathcal{V}}/st_{\mathcal{S}}$, outputs new client/verifier/server states $st_{\mathcal{C}}'/st_{\mathcal{V}}'/st_{\mathcal{S}}'$, representing the polynomial $P + \delta X^i$, or `reject`.
- $\{z, \texttt{reject}\} \leftarrow \texttt{VEval}(st_{\mathcal{V}}, st_{\mathcal{S}}, r)$: On input of the verifier state $st_{\mathcal{C}}$, the server state $st_{\mathcal{S}}$ and an evaluation point r, outputs a successful evaluation $z = P(r)$ or `reject`.

The client may use random coins for any algorithm. This is the general setting for *public verification*, the idea being that for a *private verification*, the client will play the role of the verifier too and their states will be identical: $st_{\mathcal{V}} = st_{\mathcal{C}}$.

Adapted from [26], in order to take into account dynamicity, we propose the following security properties:

Definition 1. *(*`Setup`,`Update`,`VEval`*) is a secure publicly verifiable polynomial evaluation scheme if it satisfies the following properties:*
Correctness. *Let $d \in \mathbb{N}$, $(a_0, \ldots, a_d)$ in a ring $\mathcal{R}$ and $P(X) = \sum_{i=0}^{d} a_i X^i$, then:* $\texttt{VEval}(\texttt{Setup}(1^\kappa, P), r) = P(r)$ *and for any $\delta \in \mathcal{R}$ and $0 \leq i \leq d$:*

$$\texttt{VEval}(\texttt{Update}(i, \delta, st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}}), r) = \texttt{VEval}(st_{\mathcal{V}}, st_{\mathcal{S}}, r) + \delta r^i$$

or `reject` *has been returned by one of the protocols.*
Soundness. *The soundness requirement stipulates that the client can always detect (except with negligible probability) if any message sent by the server deviates from honest behavior. We use the following game between two observers $\mathcal{O}_1$ & $\mathcal{O}_2$ (respectively playing the roles of the client and the verifier), a potentially* malicious *server $\mathcal{A}$ and an honest server $\mathcal{S}$, with the game:*

1. *$\mathcal{A}$ chooses an initial polynomial P. $\mathcal{O}_1$ runs* `Setup` *and sends the initial server part, $st_{\mathcal{S}}$, of the memory layout to both $\mathcal{A}$ and $\mathcal{S}$; and the verifier part to $\mathcal{O}_2$.*
2. *For a polynomial number of steps $t = 1, 2, ..., poly(\kappa)$, $\mathcal{A}$ picks an operation op_t where operation op_t is either* `Update` *or* `VEval`*. $\mathcal{O}_1$ executes the* `Update` *operations with both $\mathcal{A}$ and $\mathcal{S}$, while $\mathcal{O}_2$ executes the* `VEval` *operations with also both $\mathcal{A}$ and $\mathcal{S}$*
3. *$\mathcal{A}$ is said to win the game, if any message sent by $\mathcal{A}$ differs from that of $\mathcal{S}$ and neither $\mathcal{O}_1$ nor $\mathcal{O}_2$ did output* `reject`*.*

A VDPE scheme is sound*, if no polynomial-time adversary $\mathcal{A}$ has more than negligible probability in winning the above security game.*

Privacy. *A VDPE scheme is* private*, if no polynomial-time adversary has more than negligible probability in obtaining any coefficient of P, given access to the transcript of all exchanged messages for any number of runs of* `Setup`*,* `Update` *or* `VEval`*, and the associated server parts $st_{\mathcal{S}}$, of the memory layout.*

Definition 2. *(*`Setup`,`Update`,`VEval`*) is a secure privately verifiable polynomial evaluation scheme if it verifies the **Correctness**, **Soundness** and **Privacy** requirements of Theorem 1, where the verifier state $st_{\mathcal{V}}$ is included in the client state $st_{\mathcal{C}}$ and no polynomial-time adversary $\mathcal{A}$ has more than negligible probability in winning the soundness security game when $\mathcal{O}_1$ also plays the of $\mathcal{O}_2$.*

In Section 5 we apply our new verifiable protocols to the development of a new Proof of Retrievability (PoR) scheme, provably achieving correctness, soundness, and retrievability for PoR. We follow the exact same security definition for PoR as in [5], which we will not restate here for the sake of brevity.

To prove the security of our protocols we rely on classical discrete logarithm and Diffie-Hellman like assumptions, all related to polynomial computations. The first assumption, a decisional one, is the distinct leading monomials assumption: informally it states that polynomial evaluations "in the exponents" where the polynomials have distinct leading monomials are merely indistinguishable from randomness. The formal version is recalled in Theorem 5. Then we need computational assumptions, including the hardness to compute discrete logarithms, in Theorem 3, and polynomial extensions of the hardness to produce Diffie-Hellman-like secrets even with bilinear pairings, in Theorem 4.

Definition 3 (Discrete Logarithm, DLOG, hardness assumption [27, Def. 9.63])**.** *A discrete-logarithm problem is hard relative a group $\mathbb{G}$ of group order $p \geq 2^{2\kappa}$, a generator g and a randomly sampled element h of the group, if for any probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function* negl *such that $\mathcal{P}r\left[\mathcal{A}_{DLOG}(\mathbb{G}, g, h) = x \text{ s.t. } h = g^x\right] \leq \mathsf{negl}(\kappa)$.*

In the following, We use the notation $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to denote a bilinear pairing in groups of the same prime order. Our constructions can work with any pairing types, 1, 2 or 3. If such a pairing exists then $\mathbb{G}_1$ and $\mathbb{G}_2$ are denoted as bilinear groups.

Definition 4 (t-Bilinear Strong Diffie-Hellman, t-BSDH, assumption, from [24, 26])**.** *Let $\alpha \in \mathbb{Z}_p^*$, with $p \geq 2^{2\kappa}$, and $j \in \{1,2\}$. Given as input a $(t+1)$-tuple $\left\langle g_j, g_j^{\alpha}, g_j^{\alpha^2}, \ldots, g_j^{\alpha^t} \right\rangle \in \mathbb{G}_j^{t+1}$, in a bilinear group $\mathbb{G}_j$ of order p with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, for every adversary $\mathcal{A}_{t-BSDH}$, the probability $\mathcal{P}r\left[\mathcal{A}_{t-BSDH}(g_1, g_2, g_j^{\alpha}, g_j^{\alpha^2}, \ldots, g_j^{\alpha^t}) = \left\langle c, e(g_1; g_2)^{\frac{1}{\alpha+c}} \right\rangle\right] \leq \mathsf{negl}(\kappa)$ for any value of $c \in \mathbb{Z}_p \backslash \{-\alpha\}$.*

In the following we often use groups of prime order, in order to be able to easily compute with exponents. In particular, thanks to the homomorphic property of exponentiation, we will perform some linear algebra over the group and need some notations for this. For a matrix A, g^A denotes the coefficient-wise exponentiation of a generator g to each entry in A. Similarly, for a matrix W of group elements and a matrix B of scalars, W^B denotes the extension of matrix multiplication using the group action. If we have $W = g^A$, then $W^B = (g^A)^B$. Futher, this quantity can actually be computed if needed by working in the exponents first, i.e., it is equal to $g^{(AB)}$. For example:

$$\left(g^{\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)}\right)^{\left(\begin{smallmatrix} e \\ f \end{smallmatrix}\right)} = \left(\begin{smallmatrix} g^a & g^b \\ g^c & g^d \end{smallmatrix}\right)^{\left(\begin{smallmatrix} e \\ f \end{smallmatrix}\right)} = \left(\begin{smallmatrix} g^{ae+bf} \\ g^{ce+df} \end{smallmatrix}\right) = g^{\left(\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\left(\begin{smallmatrix} e \\ f \end{smallmatrix}\right)\right)}. \tag{1}$$

For the sake of simplicity, when there is no ambiguity, we also use the associated notation shortcuts like: $e(g_1^{\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right)}; g_2^c) = e(g_1; g_2)^{\left(\begin{smallmatrix} ca \\ cb \end{smallmatrix}\right)}$.

Next is the DLM assumption that states that polynomial evaluations "in the exponents" where the polynomials have distinct leading monomials are merely indistinguishable from randomness. In [1] the assumption is given for n-multivariate polynomials with matrices of dimension $k \times k$ and projections of dimension $k \times m$ for $k \geq 2$ and $m \geq 1$. Here We will only use univariate polynomials, $n = 1$, and dimensions $k = 2$, $m = 1$. We therefore recall the assumption only for this particular case.

Definition 5 (Distinct Leading Monomial, DLM, assumption [1, Theorem 6])**.** *Let $\mathbb{G} = \langle g \rangle$ be a bilinear group of prime order p. The advantage of an adversary $\mathcal{A}$ against the $(2,1,d)$-DLM security of $\mathbb{G}$, denoted $Adv_{\mathbb{G}}^{(2,1,d)-DLM}(\mathcal{A})$, is the probability of success in the game defined in Table 1 and is negligible, with $\mathcal{A}$ being restricted to make queries $P \in Z_p[T]$ such that for any challenge P, the maximum degree in one indeterminate in P is at most d, and for any sequence $(P_1, \ldots, P_q)$ of queries, there exists an invertible matrix $M \in \mathbb{Z}_p^{q \times q}$ such that the leading monomials of $M \cdot [P_1, \ldots, P_q]^{\intercal}$ are distinct.*

Table 1: Game defining the $(2,1,d)$-DLM security for a bilinear group $\mathbb{G}$ [1]

Init	Challenge(P)	Response(b')
$r \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$	If $b == 0$	
$\beta \xleftarrow{\$} \mathbb{Z}_p^2$	Then Return $y \leftarrow g^{P(r)\cdot\beta}$	Return $b' == b$
$b \xleftarrow{\$} \{0,1\}$	Else Return $y \xleftarrow{\$} \mathbb{G}^2$	

In fact, the DLM security can also be reduced to the Matrix Diffie-Hellman assumption (MDDH) [1, Theorem 5], a generalization of the widely used decision linear assumption [22, 32, 3, 4, 7].

We will also use a public-key partially homomorphic encryption scheme where both addition and multiplication are considered. We need the following properties on the linearly homomorphic encryption function E (according to the context, we use E_{pk} or just E to denote the encryption function, similarly for the decryption function, D or D_{sk}): computing several modular additions on ciphered messages and modular multiplications but only between a ciphered message and a cleartext.

$$D(E(m_1)E(m_2)) = m_1 + m_2 \qquad \text{AND} \qquad D(E(m_1)^{m_2}) = m_1 m_2 \tag{2}$$

Remark 6. *For instance, Paillier-like cryptosystems [34, 9, 20] can satisfy these requirements,* via *multiplication in the ground ring, for addition of enciphered messages, and* via *exponentiation for ciphered multiplication.*

Note though that an implementation with Paillier cryptosystem of the evaluation $P(r)$, in a modular ring $\mathbb{Z}_m$, providing the functionalities of Equation (2), requires some care: indeed these equations are usually satisfied modulo an RSA composite number N, not equal to m.

More precisely, Paillier cryptosystem will provide $D(E(P(r))) \equiv (\sum_{i=0}^{d} p_i r^i) \mod N$. *Thus a possibility to recover the correct value, is to precompute* $r^i \mod m$, *thus use the following Algorithm 1, and require that:* $(d+1)(m-1)^2 < N$.

Algorithm 1 Homomorphic modular polynomial evaluation with a different Paillier modulus

Input: An integer $r \in [0..m-1]$;
Input: A Paillier cryptosystem (E, D) with modulus $N > (m-1)^2$.
Input: $(E(p_0), \ldots, E(p_d)) \in \mathbb{Z}_N^{d+1}$, such that $\forall i, p_i \in [0..m-1]$ and $d < \frac{N}{(m-1)^2} - 1$.
Output: $c \in \mathbb{Z}_N$ such that $D(c) \mod m \equiv P(r) \mod m \equiv \sum_{i=0}^{d} p_i r^i \mod m$.
1: let $x_0 = 1$ and $c = E(p_0)$;
2: **for** $i = 1$ **to** d **do**
3: let $x_i \equiv x_{i-1} \cdot r \mod m$; {Now $x_i \in [0..m-1]$}
4: let $c \leftarrow c \cdot E(p_i)^{x_i}$;
5: **end for**
6: **return** c.

Proof. If $0 \leq p_i \leq (m-1)$, then as x_i is considered as an integer between 0 and $m-1$, then $0 \leq \sum_{i=0}^{d} p_i x_i \leq (d+1)(m-1)^2 < N$ by the constraints on d and N. Therefore $\sum_{i=0}^{d} p_i x_i \mod N = \sum_{i=0}^{d} p_i x_i \in \mathbb{Z}$ and now $D(c) \mod m = \sum_{i=0}^{d} p_i x_i \mod m \equiv P(r)$. □

Finally, we will use a Merkle hash tree to allow verifications of updates and therefore need to use a cryptographic hash function with *collision resistance*.

Overall, since we consider the semantic security of the cryptosystem, we assume that adversaries are probabilistic polynomial time machines. More precisely we consider *Malicious adversaries*: a corrupted server controls the network and stops, forges or listens to messages in order to gain information or fool the client.

3 Tools for the verification of a polynomial evaluation

Our first step is to define a verification protocol for polynomial evaluation that supports a ciphered input polynomial over a finite ring $\mathbb{Z}_p$. For this we propose an adaptation of both [26, 18]. It seems not to be sufficient to cipher the polynomial, or to check consistency in the exponents, so we propose to use both. For this, as in the former paper, we first need to define a difference polynomial that we will use to check consistency.

Definition 7. *For a polynomial* $P(X) \in \mathbb{Z}_p[X] = \sum_{i=0}^{d} p_i X^i$ *of degree* d, *let its* subset polynomials *be:* $T_{k,P}(X) = \sum_{i=k+1}^{d} p_i X^{i-k-1} = \sum_{j=0}^{d-1-k} p_{j+k-1} X^j$.

Proposition 8. *Let* $Q_P(Y, X) = \frac{P(Y)-P(X)}{Y-X}$ *be the* difference polynomial *of a polynomial* P; *then*

$$Q_P(Y, X) = \frac{P(Y) - P(X)}{Y - X} = \sum_{i=1}^{d} p_i \sum_{k=0}^{i-1} Y^{i-k-1} X^k = \sum_{k=0}^{d-1} T_{k,P}(Y) X^k \tag{3}$$

Proof. As $Y^i - X^i = (Y-X)(\sum_{k=0}^{i-1} Y^{i-k-1} X^k)$, we obtain that $Q_P(Y, X) = \sum_{i=1}^{d} p_i \sum_{k=0}^{i-1} Y^{i-k-1} X^k = \sum_{k=0}^{d-1} X^k \left(\sum_{i=k+1}^{d} p_i Y^{i-k-1} \right)$. □

This identity relates two evaluations of P: $P(Y) = P(X) + (Y - X) Q_P(Y, X)$. This equation allows one to verify $z \stackrel{?}{=} P(r)$ by checking, for a secret s, that:

$$P(s) = z + (s - r) Q_P(s, r) \tag{4}$$

For this, let E, D be the encryption and decryption functions of a partially homomorphic cryptosystem, supporting addition of two ciphertexts and multiplication of ciphertext by a cleartext, as in Equation (2). Therefore it is possible to evaluate a ciphered polynomial at a clear evaluation point, using

powers of the evaluation point: for $x = [1, r, r^2, \ldots, r^d]$, denote by $E(P)^\intercal \boxdot x = \prod_{i=0}^{d} E(p_i)^{r^i} = E(P(r))$, the homomorphic polynomial evaluation.

Similarly, if $H = [h_i] = [g^{a_i}]$, denote by $H \odot x = \prod_{i=0} h_i^{x_i} = g^{\sum a_i x_i}$ the dot-product in the exponents. Then Table 2 shows how the server produces the evaluation via the partially homomorphic cipher and how this evaluation is bound to be correct by the consistency check in the exponents.

Table 2: Verifiable Ciphered Polynomial Evaluation

	Server	Communications	Client
`Setup`		$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ groups of order p	$P \in \mathbb{Z}_p[X]$, $1 \leq d^\circ(P) \leq d$
		pairing e	$s \xleftarrow{\$} \mathbb{Z}_p$, $W \leftarrow E(P)$, $\mathcal{K} \leftarrow g_T^{P(s)}$
		generators $g_1, g_2, g_T = e(g_1; g_2)$	$H \leftarrow [g_1^{T_{k,P}(s)}]_{k=0..d-1}$
		$\xleftarrow{W,H}$	Discard P, W, H
`VEval`	$x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$	$\xleftarrow{r}$	$r \xleftarrow{\$} \mathbb{Z}_p$
	$\zeta = W^\intercal \boxdot x$		
	$\xi = H^\intercal \odot x_{0..d-1}$	$\xrightarrow{\zeta,\xi}$	$e(\xi; g_2^{s-r}) g_T^{D(\zeta)} \stackrel{?}{=} \mathcal{K}$

Proposition 9. *The protocol of Table 2 is correct and sound (verifiable).*

Proof. **Correctness**. First, $\zeta = W^\intercal \boxdot x = \prod_{i=0}^{d} E(p_i)^{(r^i)} = E(P(r))$. Then, second, $\xi = H^\intercal \odot x = \prod_{k=0}^{d-1} g_1^{T_{k,P}(s) r^k} = g_1^{Q_P(s,r)}$, by Theorem 8. Therefore, the verification is that $g_T^{Q_P(s,r)(s-r)+P(r)} \stackrel{?}{=} g_T^{P(s)}$ and this is guaranteed by Equation (4).

Soundness. Let $\left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^t} \right\rangle \in \mathbb{G}_1^{t+1}$ be a t-BSDH instance and suppose that there exists an attack to the `VEval` protocol.

Let $[p_0, \ldots, p_t] \xleftarrow{\$} \mathbb{Z}_p^{t+1}$ for a degree t polynomial and $d = t$. Then compute $W = E(P)$, $T_{k,P} = \sum_{i=k+1}^{t} p_i Y^{i-k-1} = \sum_{j=0}^{t-1-k} t_{k,j} Y^j$. Finally, homomorphically compute $\mathcal{K} = e\left(\left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^t} \right\rangle \odot [p_0, \ldots, p_t]; g_2\right)$ and $H = [h_k]$, where $h_k = \left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^{t-1-k}} \right\rangle \odot [t_{k,0}, \ldots, t_{k,t-1-k}]$. These inputs are indistinguishable from a generic setup of the protocol of Table 2 and can thus be given to its attacker.

Finally, select a random evaluation point r and compute (ζ, ξ). The supposition if that an attacker of the `VEval` part of the protocol can get (ζ', ξ'), with some advantage, such that $(D(\zeta'), \xi') \neq (D(\zeta), \xi)$, even though both would be passing the verification. Now, on the one hand, if $D(\zeta') = D(\zeta)$, then $\xi \neq \xi'$ and it must be that $e(\xi; g_2^{s-r}) g_T^{D(\zeta)} = \mathcal{K}$ and $e(\xi'; g_2^{s-r}) g_T^{D(\zeta)} = \mathcal{K}$. Therefore, if $r \neq s$, then $e(\xi; g_2^{s-r})^{\frac{1}{s-r}} = e(\xi'; g_2^{s-r})^{\frac{1}{s-r}}$. But this contradicts the fact that $\xi \neq \xi'$, so $r = s$, and the secret can be exposed. On the other hand, if $D(\zeta') \neq D(\zeta)$, then it means that we must have the equality $(e(\xi; g_2)/(e(\xi'; g_2))^{s-r} = g_T^{D(\zeta')-D(\zeta)}$ and therefore: $\left(\frac{e(\xi;g_2)}{e(\xi';g_2)}\right)^{\frac{1}{D(\zeta')-D(\zeta)}} = g_T^{\frac{1}{s-r}}$. This proves that the adversary would solve the t-BSDH $\left\langle -r, e(g_1; g_2)^{\frac{1}{s-r}} \right\rangle$ challenge with the same advantage. □

We see here that using a decipherable partially homomorphic function for the coefficients of P is required, otherwise one could not compute the exponentiation on ξ/ξ' in the soundness proof. Several issues remain with this protocol: first it is not dynamic. Indeed, for a dynamic version, the problem is that updating only one coefficient of P requires to update up to $d-1$ coefficients of H. This work would be of the same order of magnitude as recomputing the whole setup. Second it is not fully hiding the coefficients of P as they are just put in the exponents without any masking, and we do not prove the privacy requirement[1]. Third, the protocol is not fully publicly verifiable since the decryption key of the partially homomorphic system is required. We incrementally solve the first two issues in the sequel of this paper and obtain a thus fully secure private protocol. We also are able to provide a dynamic protocol, publicly verifiable, but for an unciphered polynomial. Combining all three properties, that is, designing an efficient dynamic protocol for ciphered polynomials, but publicly verifiable, remains an open question to us.

[1] Efficient updates in similar schemes are considered, e.g., in [37] but to a protocol that verifies coefficients known to the server, *not* its evaluation at hidden coefficients

Table 3: Public and Dynamic unciphered polynomial evaluation

	Server	Communications	Client
`Setup`		$\mathbb{G}, \mathbb{G}_T$ of order p symm. pairing e gen. g, $e(g;g)$	$P \in \mathbb{Z}_p[X]$, $0 \leq d^\circ(P) \leq d$ Let $s \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{K}_1 \leftarrow e\left(g^{P(s)};g\right)$, $S \leftarrow [g^{s^k}]_{k=0..d-1}$
	$T_P \leftarrow \text{MTTree}(P)$ Store P, T_P, S	$\xleftarrow{P,S}$	$r_p \leftarrow \text{MTRoot}(P)$ Publish $\mathcal{K}_1, \mathcal{K}_2 \leftarrow g^s$; discard P, S.
`Update`		$\xleftarrow{i,\delta}$ $\xrightarrow{p_i, L_i}$	
	$(p_i, L_i) \leftarrow \text{MTLeafPath}(i, P, T_P)$ $T_P \leftarrow \text{MTupdLeaf}(i, p_i + \delta, T_P)$		$r_P \stackrel{?}{=} \text{MTpathRoot}(i, p_i, L_i)$ $r_P \leftarrow \text{MTpathRoot}(i, p_i + \delta, L_i)$ $\mathcal{K}_1 \leftarrow \mathcal{K}_1 \cdot e\left(g^{s^i \delta};g\right)$; publish $\mathcal{K}_1$
PVDUeval	Form $x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$ $\zeta \leftarrow P(r)$; $\xi \leftarrow \prod_{i=1}^{d} \prod_{k=0}^{i-1} S_{i-k-1}^{p_i x_k}$	$\xleftarrow{r}$ $\xrightarrow{\zeta, \xi}$	$r \xleftarrow{\$} \mathbb{Z}_p$ $e(\xi; \mathcal{K}_2 / g^r) e(g^\zeta; g) \stackrel{?}{=} \mathcal{K}_1$

4 Outsourced dynamic verification of the evaluation

4.1 Merkle trees for logarithmic client storage

To avoid storing the polynomial coefficients on the client side, we use a Merkle hash tree [31, 29, 5]. Then it is sufficient to store the root of the Merkle tree. For our purpose, an implementation of such trees must just provide the following algorithms:

- $T \leftarrow \text{MTTree}(X)$ creates a Merkle hash tree from a database X.
- $r \leftarrow \text{MTRoot}(X)$ computes from scratch the root of the Merkle hash tree of the whole database X.
- $(a, L) \leftarrow \text{MTLeafPath}(i, X, T)$ is an algorithm providing the client with the requested leaf element a, together with the corresponding list L of Merkle tree uncles.
- $r \leftarrow \text{MTpathRoot}(i, a, L)$ computes the root of the Merkle hash tree from a leaf element a and the associated path of uncles L.
- $T' \leftarrow \text{MTupdLeaf}(i, a, T)$ updates the whole Merkle tree T by changing the i-th leaf to be a.

The requirements are thus that:

$$\forall i, X, \text{MTRoot}(X) = \text{MTpathRoot}\left(i, \text{MTLeafPath}(i, X, \text{MTTree}(X))\right) \tag{5}$$

$$\begin{aligned} \forall i, a, X, \text{ Let } (b, L) \leftarrow \text{MTLeafPath}(i, X, \text{MTTree}(X)), \\ \text{and let } X' \leftarrow X \backslash \{(i, b)\} \bigcup \{(i, a)\}, \\ \text{then } \text{MTupdLeaf}(i, a, \text{MTTree}(X)) = \text{MTTree}(X') \end{aligned} \tag{6}$$

4.2 Public Dynamic unciphered Polynomial Evaluation

Now we give a protocol for the public verification of the evaluation of a dynamic polynomial P. It consists in three algorithms (`Setup`,`Update`,PVDUeval) detailed in Table 3 and it requires a *symmetric* pairing.

During the `Setup` algorithm, the Client sends the unciphered polynomial to the Server and deletes it to minimize the Server storage. The Client uses a random coin s to create some data to be published or to be sent to the server. We introduce a third part named the Verifier. The Verifier collects the published data and is authorized to run the `Read` and the PVDUeval algorithms. But she is not authorized to run the Init algorithm and s is not known by the Verifier.

Theorem 10. *The protocol of Table 3 is correct and sound.*

Proof. **Correctness.** For the update, the new polynomial is $P'(s) = P(s) + \delta s^i$, so that the key is updated as $\mathcal{K}_1' = \mathcal{K}_1 \cdot e(g^{\delta s^i}; g)$. Now for the evaluation, first, $\xi = \prod_{i=1}^{d} \prod_{k=0}^{i-1} S_{i-k-1}^{p_i x_k} = g^{\sum\sum s^{i-k-1} p_i x_k} = g^{Q_P(r,s)}$ and, second, we have that: $e(\xi; \mathcal{K}_2/g^r) e(g;g)^{\zeta} = e(\xi; g^{s-r}) e(g;g)^{P(r)} = e(g;g)^{Q_P(r,s)(s-r)+P(r)} = e(g;g)^{P(s)}$. Hence we see that $e(\xi; \mathcal{K}_2/g^r) e(g;g)^{\zeta} = \mathcal{K}_1$ and, therefore, the protocol is correct.

Soundness. Let $\left\langle g, g^s, g^{s^2}, \ldots, g^{s^t} \right\rangle \in \mathbb{G}^{t+1}$ be a t-BSDH instance. For the setup phase, set $d = t$ and randomly select $[p_0, \ldots, p_t] \xleftarrow{\$} \mathbb{Z}_p^{t+1}$. Then set $S = \left\langle \mathbb{G}, g, g^s, g^{s^2}, \ldots, g^{s^t} \right\rangle$ and $\mathcal{K}_1 = e\left(\left\langle g, g^s, g^{s^2}, \ldots, g^{s^t} \right\rangle \odot [p_0, \ldots, p_t]; g\right)$. These inputs are indistinguishable from generic inputs to the protocol of Table 3. For any number of update phase, randomly select δ, receive p_i and L_i from the Server, compute $\mathcal{K}_1' = \mathcal{K}_1 e(S_i^{\delta}; g)$ and refresh r_p. Finally, select a random evaluation point r, compute (ζ, ξ) and call an attacker of the PVDUeval part of the protocol to get (ζ', ξ') such that $(\zeta', \xi') \neq (\zeta, \xi)$, even though both are passing the verification. If $\zeta' = \zeta$, then as $\xi \neq \xi'$ it must be that $r = s$ and the secret is revealed; otherwise, $\zeta' \neq \zeta$ and we have both $e(\xi'; \mathcal{K}_2/g^r) e(g;g)^{\zeta'} = \mathcal{K}_1$, on the one hand, and $\mathcal{K}_1 = e(\xi; \mathcal{K}_2/g^r) e(g;g)^{\zeta}$, on the other hand. This gives $e(\frac{\xi'}{\xi}; g^{s-r}) = e(g^{\zeta-\zeta'}; g)$ and thus $e\left((\frac{\xi'}{\xi})^{s-r}; g\right) = e(g^{\zeta-\zeta'}; g)$. Finally, we have that: $e\left(\frac{\xi}{\xi'}; g\right)^{\frac{1}{\zeta'-\zeta}} = e(g;g)^{\frac{1}{s-r}}$. This proves that the adversary would solve the t-BSDH $\left\langle -r, e(g;g)^{\frac{1}{s-r}} \right\rangle$ challenge with the same advantage. □

4.3 Efficient linear-time evaluation

As a first approach to evaluate our protocols, we consider that the cardinality of the coefficient domain is a constant. Therefore, we count as arithmetic operations in the field not only the usual addition, subtraction, multiplication and inversion, but also the exponentiations that are independent of the degree of the polynomial. We thus express our asymptotic complexity bounds in Table 4, only with respect to that degree d.

Table 4: Complexity bounds for the publicly verifiable dynamic and unciphered polynomial evaluation of Table 3 for a degree d polynomial.

		Server	Communication	Client
Storage		$\mathcal{O}(d)$		$\mathcal{O}(1)$
Comput.	`Setup`	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$
	`Update`	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$
	PVDUeval	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Theorem 11. *The setup protocol of Table 3 requires $\mathcal{O}(d)$ arithmetic operations.*

The update protocol of Table 3 requires $\mathcal{O}(log(d))$ arithmetic operations.

The verification protocol of Table 3 requires $\mathcal{O}(1)$ communications and arithmetic operations for the client, and $\mathcal{O}(d)$ arithmetic operations for the Server.

Proof. For the update phase, the client computes the root of the Merkle tree from the new value $p_i + \delta$ and the path L_i given by the server in $\mathcal{O}(\log(d))$. She also has to compute an exponentiation and a product in $\mathbb{Z}_p[X]$, this is in $\mathcal{O}(1)$.

For the verification phase, communications are just 3 group elements. The client work is only 2 pairing and 2 exponentiations and 1 product.

Now for the server. First, computing ζ is $d+1$ homomorphic multiplications and d additions. Second, the server has to compute $\xi = \prod_{i=1}^{d} \prod_{k=0}^{i-1} S_{i-k-1}^{p_i x_k} = \prod_{i=1}^{d} \left(\prod_{k=0}^{i-1} S_{i-k-1}^{r^k}\right)^{p_i}$. Therefore, one can use a Horner-like prefix computation: consider $t_0 = 1$, and $t_i = S_{i-1} \cdot t_{i-1}^r$, then $t_1 = S_0$, $t_2 = S_1 S_0^r$ and $t_i = S_{i-1}(S_{i-2} \ldots (S_2 (S_1 S_0^r)^r) \ldots)^r = \prod_{k=0}^{i-1} S_{i-k-1}^{r^k}$. Thus one can use the following Algorithm 2 to compute ξ. Computing ξ then requires at most $2d$ exponentiations and $2d$ multiplications. □

Algorithm 2 Homomorphic linear prefix evaluation of the difference polynomial

Input: r, $[S_0, \ldots, S_{d-1}]$, $[p_1, \ldots, p_d]$.
Output: $\xi = \prod_{i=1}^{d} \left(\prod_{k=0}^{i-1} S_{i-k-1}^{r^k} \right)^{p_i}$.

1: $\xi = 1$; $t = 1$;
2: **for** $i = 1$ **to** d **do**
3: $\quad t \leftarrow S_{i-1} \cdot t^r$; $\qquad \{t_i = \prod_{k=0}^{i-1} S_{i-k-1}^{r^k}\}$
4: $\quad \xi \leftarrow \xi \cdot t^{p_i}$.
5: **end for**
6: **return** ξ.

5 Fully private, dynamic and ciphered protocol for polynomial evaluation

So far we have a polynomial evaluation verification, that allows efficient updates of its coefficients. We now propose a scheme which combine the polynomial evaluation with the externalization of the polynomial itself. For this, two more ingredients are added in Section 5.1: an efficient masking in the exponents in order to fulfill the hiding security property and an outsourcing of the (ciphered) polynomial itself. This latter feature allows the client to not even store the polynomial and reduces her need for storage to a small constant number of field elements. For this we use Merkle hash trees presented in Section 4.1. They ensure the authenticity of the coefficient updates, with the storage of only one hash. Finally note that the bilinear pairing need not be symmetric anymore, but need to be applied twice for the security hypothesis to hold.

5.1 Private, dynamic, ciphered protocol

Here we add a masking of the polynomial coefficients in order to make the protocol hiding. For this we use the security hypothesis of Theorem 5: indeed, DLM security states that in a group $\mathbb{G}$ of prime order, the values $(g^{P_1(A)\beta}, \ldots, g^{P_d(A)\beta})$ are indistinguishable from a random tuple of the same size, when $P_1, \ldots, P_d$ have distinct leading monomials of bounded degree and A and β are the 2×2 and 2×1 secrets. Therefore, in our modified protocol, the coefficients $g^{\Phi^i\beta}$ for a secret 2×2 matrix Φ, are indistinguishable from a random tuple (g^{Γ_i}) since the polynomials X^i, $i = 1..d$ are just distinct monomials.

We start this section with linear algebra tools and an overview and then give a full formalization and the associated proofs of security. We end the section with experiments showing the efficiency of our approach.

5.1.1 Linear algebra toolbox.

For the next protocol to hold, we need to adapt the difference polynomial to the matrix case. For instance Theorem 8 holds in the matrix case provided that the, now matrices, Y and X commute and that $Y - X$ is invertible. Let I_n be the $n\times n$ identity matrix. Then, we will for instance use $Y = sI_2$ and $X = rI_2$ with $s \neq r$.

Also to speed-up things, we need to efficiently compute geometric sums of matrices. Thanks to Fiduccia's algorithm [17], this is easily done with a number of operations logarithmic in the exponent, provided that one is not an eigenvalue of the matrix. Indeed, first, any matrix commutes with the identity so the geometric sum can be computed via one matrix exponentiation, one matrix inverse and one matrix multiplication: $\sum_{i=0}^{d} A^i = (A^{d+1} - I_n)(A - I_n)^{-1}$. Then, second, Fiduccia's algorithm computes the exponentiation modulo the characteristic polynomial, using the square and multiply fast recursive algorithm.

Lemma 12. *Algorithm 4 requires between* $40 + 8\lceil \log_2(d+1) \rceil$ *and* $40 + 11\lceil \log_2(d+1) \rceil$ *arithmetic operations.*

Proof. Counting only (modular) field operations, Algorithm 3 requires between 8 and 11 times $\lceil \log_2(d) \rceil$ additions and multiplications depending on the binary decomposition of d. Then we have 5 operations

Algorithm 3 Degree 2 modular monomial powers (2-MMP)

Input: $d \in \mathbb{Z}$, $d \geq 1$, $P = p_0 + p_1 Z + Z^2 \in \mathbb{Z}_p[Z]$ monic degree 2 polynomial.
Output: $Z^d \mod P$.

1: **if** $d == 1$ **then return** Z **end if**
2: $T \leftarrow$ 2-MMP$(\lfloor d/2 \rfloor, P)$;
3: $S \leftarrow (t_0^2 - t_1^2 p_0) + (2t_0 t_1 - t_1^2 p_1)Z$; $\{T(Z)^2 \text{ modulo } P(Z)\}$
4: **if** d is odd **then**
5: **return** $(-s_1 p_0) + (s_0 - s_1 p_1)Z$; $\{Z \cdot S(Z) \text{ modulo } P(Z)\}$
6: **else**
7: **return** S.
8: **end if**

Algorithm 4 Projected matrix geometric sum (PMGS)

Input: $k \in \mathbb{Z}$, $A = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathbb{Z}_p^{2\times 2}$, s.t. $A - I_2$ is invertible, $\beta \in \mathbb{Z}_p^2$.
Output: $\sum_{i=0}^{k} A^i \beta$.

1: Let $\pi(Z) = (ad - bc) - (a + d)Z + Z^2$; {The characteristic polynomial of A}
2: Let $F(Z) = f_0 + f_1 Z = $ 2-MMP$(k + 1, \pi)$; $\{Z^{k+1} \mod \pi(Z)$, using Algorithm 3$\}$
3: **return** $(f_1 A + (f_0 - 1)I_2)(A - I_2)^{-1}\beta$. $\{(A^{k+1} - I_2)(A - I_2)^{-1}\beta\}$

for the matrix inverse, twice 6 operations for the matrix-vector multiplications and 18 operations for the matrix polynomial evaluation. Plus 5 operations for the characteristic polynomial. □

5.1.2 Formalization of the protocol.

The dynamic externalized polynomial evaluation scheme consist of the following algorithms `Setup`, `Update` and `VEval` between a client $\mathcal{C}$ with state $st_{\mathcal{C}}$ and the server $\mathcal{S}$ of state $st_{\mathcal{S}}$. The exchanges are also summarized in Table 5.

- $(st_{\mathcal{C}}, st_{\mathcal{S}}) \leftarrow$ `Setup`$(1^\kappa, P)$: on input of the security parameters and the polynomial P, outputs the client state $st_{\mathcal{C}}$ and the server state $st_{\mathcal{S}}$, as detailed in Algorithm 5.
- $\{(st'_{\mathcal{C}}, st'_{\mathcal{S}}), \texttt{reject}\} \leftarrow$ `Update`$(i, \delta, st_{\mathcal{C}}, st_{\mathcal{S}})$: on input of an index $i \in 0..d$, the difference data δ, the client state $st_{\mathcal{C}}$ and the server state $st_{\mathcal{S}}$, outputs a new client state $st'_{\mathcal{C}}$ and a new server state $st'_{\mathcal{S}}$ (such that now the new i-th coefficient of the polynomial is $P'_i = P_i + \delta$, for P_i the previous i-th coefficient), or `reject`, as detailed in Algorithm 6.
- $\{z, \texttt{reject}\} \leftarrow$ `VEval`$(st_{\mathcal{C}}, st_{\mathcal{S}}, r)$: on input of the client state $st_{\mathcal{C}}$, the server state $st_{\mathcal{S}}$ and an evaluation point r, outputs a successful evaluation $z = P(r)$ or `reject`, as detailed in Algorithm 7.

We have now in Theorem 13, the complete result for the Dynamic Verified Evaluation of Secret Polynomials.

Theorem 13. *Under the security assumptions of Section 2, the protocol composed of Algorithms 5 to 7 (summarized in Table 5) is a fully secure verifiable polynomial evaluation scheme, as defined in Theorem 1 and the complexity bounds of its algorithms are given in Table 6.*

For the complexity bounds we still consider the cardinality of the coefficient domain to be a constant (so that, again, even exponentiations not involving the degree are considered constant) and we also consider that one encryption/decryption with the linearly homomorphic cryptosystem requires a number of arithmetic operations constant with respect to the degree.

Proof. **Correctness.** We use the left hand side of Theorem 8 and Equation (4). Applying this to $\bar{P}$, we obtain that: $\bar{\xi} = \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(\bar{H}_i; S_{i-k-1})^{x_k} = \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(g_1^{\bar{P}_i}; g_2^{s^{i-k-1}})^{r^k} = e(g_1; g_2)^{Q_{\bar{P}}(s \cdot I_2, r \cdot I_2)}$. Denote by $G(Z) = \frac{Z^{d+1}-1}{Z-1}$. Now $\bar{P}(X) = P(X)\alpha + G(X\Phi)\beta$, then $c = G(r\Phi)\beta = G(r \cdot I_2\Phi)\beta$ and thus $\bar{P}(r \cdot I_2) = D(\zeta)\alpha + c = P(r)\alpha + c$. Therefore the `VEval` verification is indeed that $g_T^{Q_{\bar{P}}(s \cdot I_2, r \cdot I_2)(s-r) + \bar{P}(r \cdot I_2)} \stackrel{?}{=}$

Algorithm 5 `Setup`$(1^\kappa, P)$

Input: 1^κ; $p \in \mathbb{P}$, $P = \sum_{i=0}^{d} p_i X^i \in \mathbb{Z}_p[X]$;
Input: a partially homomorphic cryptosystem E/D satisfying Equation (2), *for any dot-product of size* $d+1$, *modulo* p.
Output: $st_{\mathcal{S}}$, $st_{\mathcal{C}}$.

1: Client: generates order p groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ with non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators $g_1, g_2, g_T = e(g_1; g_2)$;
2: Client: generates a public/private key pair (pk, sk) for E/D;
3: Client: randomly selects $s \xleftarrow{\$} \mathbb{Z}_p \backslash \{0, 1\}$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^2$, $\Phi \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, s. t. $s\Phi - I_2$ is invertible;
4: Client: computes $\bar{P}(X) = \sum_{i=0}^{d} X^i(p_i\alpha + \Phi^i\beta)$, $W = E_{pk}(P)$, $\bar{H} = [g_1^{\bar{p}_i}]_{i=1..d} \in \mathbb{G}_1^{2\times d}$, $\bar{\mathcal{K}} = g_T^{\bar{P}(s)} \in \mathbb{G}_T^2$ and $S = [g_2^{s^k}]_{k=0..d-1} \in \mathbb{G}_2^d$;
5: Client: $r_W = \mathtt{MTRoot}(W)$; {*root of the Merkle tree*}
6: Client: sends $pk, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e, W, \bar{H}, S$ to the Server;
7: Client: **return** $st_{\mathcal{C}} \leftarrow (pk, sk, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, r_W)$;
8: Server: $T_W \leftarrow \mathtt{MTTree}(W)$; {*the Merkle tree*}
9: Server: **return** $st_{\mathcal{S}} \leftarrow (pk, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e, W, T_W, \bar{H}, S)$.

Algorithm 6 `Update`$(i, \delta, st_{\mathcal{C}}, st_{\mathcal{S}})$

Input: $i \in [0..d]$, $\delta \in \mathbb{Z}_p$, $st_{\mathcal{C}}$, $st_{\mathcal{S}}$.
Output: $st'_{\mathcal{C}}, st'_{\mathcal{S}}$ or `reject`.

1: Client: computes $e_\delta = E_{pk}(\delta)$, $\Delta = g_1^{\delta\alpha}$;
2: Client: sends i, e_δ, Δ to the Server;
3: Server: $(w_i, L_i) \leftarrow \mathtt{MTLeafPath}(i, W, T_W)$; {*gets* w_i *and its uncles from the tree*}
4: Server: $T'_W \leftarrow \mathtt{MTupdLeaf}(i, w_i \cdot e_\delta, T_W)$; {*updates the Merkle tree*}
5: Server: sends w_i, L_i to the Client;
6: Server: **return** $st'_{\mathcal{S}} \leftarrow st_{\mathcal{S}} \backslash \{T_W, w_i, \bar{h}_i\} \bigcup \{T'_W, w_i \cdot e_\delta\} \bigcup_{j=1}^{2} \{\bar{h}_i[j] \cdot \Delta[j]\}$;
7: **if** $r_W = \mathtt{MTpathRoot}(i, w_i, L_i)$ **then**
8: Client: computes $\bar{\mathcal{K}}'[j] \leftarrow e(\Delta[j]^{s^i}; g_2) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$;
9: Client: computes $r'_W = \mathtt{MTpathRoot}(i, w_i \cdot e_\delta, L_i)$;
10: Client: **return** $st'_{\mathcal{C}} \leftarrow st_{\mathcal{C}} \backslash \{\bar{\mathcal{K}}, r_W\} \bigcup \{\bar{\mathcal{K}}', r'_W\}$.
11: **else** {*else the stored root does not match the received element and uncles*}
12: Client: **return** `reject`.
13: **end if**

$g_T^{\bar{P}(s\cdot I_2)} = g_T^{\bar{P}(s)}$. Now for the `Update` operation, $P'_i = P_i + \delta$ so that $\bar{P}'(s) = \bar{P}'(s \cdot I_2) = \delta s^i \alpha + \bar{P}(s \cdot I_2)$ and $e(g_1^{\bar{P}'(s\cdot I_2)[j]}; g_2) = e(g_1^{s^i \delta\alpha[j]}; g_2) e(g_1^{\bar{P}(s\cdot I_2)[j]}; g_2) = e(\Delta[j]^{s^i}; g_2) \cdot g_T^{\bar{P}(s)[j]} = e(\Delta[j]^{s^i}; g_2) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$.

Complexity bounds. In terms of storage, apart from the public/private key pair and the groups, the client just has to store nine elements mod p, that is s, $\alpha \neq [0, 0]$, β, and Φ, together with two group elements, $\bar{\mathcal{K}}$; the server has to store the polynomial ciphered thrice, the ciphered powers of s and the Merkle tree for the ciphered polynomial: all this is $O(d)$. In terms of communications, during the `Update` phase the client sends one index and three group elements, while receiving one group element and the list of its $\log(d)$ uncles. During the `VEval` phase, only four elements are exchanged. Finally, in terms of computations, the server performs $O(d)$ operations for the Merkle tree generation at `Setup`; fetches $O(\log(d))$ uncles at `Update`; and $O(d)$ (homomorphic) operations at `VEval`, thanks to Algorithm 2. For the client, `Update` requires $O(\log(d))$ arithmetic operations to check the uncles and to compute the exponentiation s^i, together with a constant number of other arithmetic operations, independent of the degree. Similarly, computing $(r\Phi)^{d+1}$ also requires $O(\log(d))$ classical arithmetic operations thanks to Algorithm 4 and the rest is a constant number of operations that are independent of the degree.

Soundness. Let $\left\langle g_2, g_2^s, g_2^{s^2}, \ldots, g_2^{s^t} \right\rangle \in \mathbb{G}_2^{t+1}$ be a t-BSDH instance. For the setup phase, randomly select α, β, Φ and $[p_0, \ldots, p_t]$. Then compute $W = E(P)$, $\bar{H} = g^{\bar{P}}$, and let $S = \left\langle \mathbb{G}, g, g^s, g^{s^2}, \ldots, g^{s^t} \right\rangle$. Finally homomorphically compute $\bar{\mathcal{K}} = e\left(g_1; \left\langle g_2, g_2^s, g_2^{s^2}, \ldots, g_2^{s^t} \right\rangle \odot [\bar{p}_0, \ldots, \bar{p}_t]\right)$. These inputs are in-

Algorithm 7 `VEval`$(st_{\mathcal{C}}, st_{\mathcal{S}}, r)$

Input: $st_{\mathcal{C}}, st_{\mathcal{S}}$ and $r \in \mathbb{Z}_p$;
Output: $z = P(r)$ or `reject`.

1: Client: computes $r\Phi$ and $c \leftarrow \left((r\Phi)^{d+1} - I_2\right) \cdot (r\Phi - I_2)^{-1} \cdot \beta$ via Algorithm 4;
2: Client: sends r to the Server;
{*via Equation (2)*, see also, e.g., Theorem 6 and Algorithm 1}
3: Server: homomorphically computes $\zeta = W^\intercal \boxdot x = \prod_{i=0}^{d} w_i^{(r^i \mod p)}$
4: Server: $\bar{\xi} = [1_{\mathbb{G}_T}, 1_{\mathbb{G}_T}]^\intercal \in \mathbb{G}_T^2$; $t = 1_{\mathbb{G}_2}$;
5: **for** $i = 1$ **to** d **do** {*Following the ideas of Algorithm 2*}
6: Server: $t \leftarrow S_{i-1} \cdot t^r$;
7: Server: $\bar{\xi}[j] \leftarrow \bar{\xi}[j] \cdot e(\bar{H}_i[j]; t)$ for $j = 1..2$;
8: **end for**
9: Server: sends $\zeta, \bar{\xi}$ to the Client;
10: Client: computes $z = D_{sk}(\zeta) \mod p$;
11: **if** $\bar{\xi}[j]^{s-r} g_T^{z\alpha[j]+c[j]} = \bar{\mathcal{K}}[j]$ for $j = 1..2$ **then**
12: Client: **return** z.
13: **else**
14: Client: **return** `reject`.
15: **end if**

Table 5: Private, Dynamic, Ciphered and logarithmic polynomial evaluation

	Server	Communications	Client
`Setup`		$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ groups of order p pairing e to $\mathbb{G}_T$, gen. $g_1, g_2, g_T = e(g_1; g_2)$	$P \in \mathbb{Z}_p[X]$, $1 \leq d^\circ(P) \leq d$ $s \xleftarrow{\$} \mathbb{Z}_p \backslash \{0, 1\}$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^2$, $\Phi \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, s.t. $(s\Phi - I_2) \in GL_2(\mathbb{Z}_p)$ Let $\bar{P}(X) \leftarrow \sum_{i=0}^{d} X^i(p_i\alpha + \Phi^i\beta)$ $W \leftarrow E(P)$, $\bar{H} \leftarrow [g_1^{\bar{p}_i}]_{i=1..d} \in \mathbb{G}_1^{2\times d}$ $\bar{\mathcal{K}} \leftarrow g_T^{\bar{P}(s)} \in \mathbb{G}_T^2$, $S \leftarrow [g_2^{s^k}]_{k=0..d-1}$
	$T_W \leftarrow \text{MTTree}(W)$	$\xleftarrow{W, \bar{H}, S}$	$r_W \leftarrow \text{MTRoot}(W)$
	Store W, T_W, $\bar{H}$, S		discard P, $\bar{P}$, W, $\bar{H}$, S.
`Update`	$\bar{h}'_i[j] \leftarrow \Delta[j] \cdot \bar{h}_i[j]$ for $j = 1..2$	$\xleftarrow{i, e_\delta, \Delta}$	$e_\delta \leftarrow E(\delta)$, $\Delta \leftarrow g_1^{\delta\alpha}$
	$(w_i, L_i) \leftarrow \text{MTLeafPath}(i, W, T_W)$	$\xrightarrow{w_i, L_i}$	$r_W \stackrel{?}{=} \text{MTpathRoot}(i, w_i, L_i)$
	$w'_i \leftarrow w_i \cdot e_\delta$		$w'_i \leftarrow w_i \cdot e_\delta$, $\bar{\mathcal{K}}[j] \leftarrow e(\Delta[j]^{s^i}; g_2) \cdot \bar{\mathcal{K}}[j]$
	$T_W \leftarrow \text{MTupdLeaf}(i, w'_i, T_W)$		$r_W \leftarrow \text{MTpathRoot}(i, w'_i, L_i)$
`VEval`	Form $x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$	$\xleftarrow{r}$	For $r \in \mathbb{Z}_p$ s.t. $(r\Phi - I_2) \in GL_2(\mathbb{Z}_p)$
	$\zeta \leftarrow W^\intercal \boxdot x$		$c \leftarrow ((r\Phi)^{d+1} - I_2)(r\Phi - I_2)^{-1}\beta$
	$\bar{\xi} \leftarrow \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(\bar{H}_i; S_{i-k-1})^{x_k}$	$\xrightarrow{\zeta, \bar{\xi}}$	$\bar{\xi}[j]^{s-r} g_T^{D(\zeta)\alpha[j]+c[j]} \stackrel{?}{=} \bar{\mathcal{K}}[j]$ for $j = 1..2$

Table 6: Complexity bounds for verifiable dynamic and ciphered polynomial evaluation (for groups and prime fields of supposed constant order/cardinality, the asymptotics are here function of the degree d of the evaluated polynomial: storage units are given in number of group/field elements, computational operations are given in number of group/prime field arithmetic operations).

		Server	Communication	Client
Storage		$\mathcal{O}(d)$		$\mathcal{O}(1)$
Comput.	`Setup`	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$
	`Update`	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$
	`VEval`	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(d))$

distinguishable from random inputs to the protocol of Table 5. For any number of update phases, randomly select δ and compute $e_\delta = E(\delta)$ and $\Delta = g_1^{\delta\alpha}$. Also compute $\mathcal{K}' = e(g_1; S_i^{\delta\alpha}) \cdot \mathcal{K}$. Finally, select a random evaluation point r, compute $(\zeta, \bar{\xi})$ and call an attacker of the `VEval` part of the protocol to get $(\zeta', \bar{\xi}')$ such that $(D(\zeta'), \bar{\xi}') \neq (D(\zeta), \bar{\xi})$, even though both are passing the verification. This means, again, that if, one the one hand, $D(\zeta') = D(\zeta)$, then $\bar{\xi}^{(s-r)} = \bar{\xi}'^{(s-r)}$ with $\bar{\xi} \neq \bar{\xi}'$. Therefore $s = r$ and the secret is exposed. If, on the other hand, $D(\zeta') \neq D(\zeta)$ then, as $\alpha \neq [0, 0]$, set $j \in \{1, 2\}$ such that $\alpha[j] \neq 0$ and we have again: $\left(\frac{\bar{\xi}[j]}{\bar{\xi}'[j]}\right)^{\frac{1}{\alpha[j](D(\zeta')-D(\zeta))}} = e(g_1; g_2)^{\frac{1}{s-r}}$. This proves that the adversary would solve the t-BSDH $\left\langle -r, e(g_1; g_2)^{\frac{1}{s-r}} \right\rangle$ challenge.

Privacy. We show that the protocol is hiding both p_i and $\bar{p}_i$.

For $\bar{p}_i$ first. Let $B = g_1^b$ be a DLOG instance. For the setup phase, randomly select s, α, Φ, d, $[p_0, \ldots, p_d]$ and two non-zero elements $b_1, b_2 \in \mathbb{Z}_p^*$. Then compute $W = E(P)$, $\bar{h}_i = g_1^{\alpha p_i} B^{\Phi^i [b_1, b_2]^\intercal}$, $S = \left\langle g_2, g_2^s, g_2^{s^2}, \ldots, g_2^{s^t} \right\rangle$, and $\bar{\mathcal{K}} = e(g_1^{\alpha P(s)} B^{G(s\Phi)[b_1, b_2]^\intercal}; g_2)$. These inputs are indistinguishable from random inputs to the protocol of Table 5. For any update phase, randomly select δ and compute $e_\delta = E(\delta)$ and $\Delta = g_1^{\alpha\delta}$. Also compute $\bar{\mathcal{K}}'[j] = e(\Delta[j]^{s^i}; g_2) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$. Such updates are indistinguishable from random updates to the protocol of Table 5. Randomly select any number of evaluation points r and run the associated `VEval` phases, randomly alternated with update phases. Now, if an attacker can find from this transcript one coefficient $\bar{p}_i[j]$ for $j \in \{1, 2\}$, then compute $b = (\bar{p}_i[j] - p_i \alpha[j])/(\Phi^i [b_1, b_2]^\intercal)[j]$ and the DLOG is revealed.

For p_i, we proceed with a sequence of two indistinguishable games.

Under DLM security, cf. Theorem 5, the parameter $\bar{h}_i$, or more precisely, $(E(p_i), g^{p_i\alpha + \Phi^i\beta})$, is indistinguishable from $(E(p_i), g^{p_i\alpha + \Gamma_i})$ for some random 2-dimensional vectors Γ_i. Therefore the protocol of Table 5 is indistinguishable, as a whole, from the same protocol where $\Phi^i\beta$ is everywhere replaced by Γ_i, and c is (now inefficiently) computed as $\sum r^i \Gamma_i$. Now we prove that the latter is hiding. Let $Z = E(\omega)$ be the cipher of a secret ω. Randomly select d and $[u_0, \ldots, u_d] \xleftarrow{\$} \mathbb{Z}_p^{d+1}$. Compute $W_i = Z \cdot E(u_i) = E(\omega + u_i)$. Randomly select α and h_i (so that $\Gamma_i = \log_{g_1}(h_i) - (\omega + u_i)\alpha \in \mathbb{Z}_p^2$ exists, but remains unknown) for $i = 1..d$. Randomly select s and compute $\bar{\mathcal{K}} = e(H \odot [1, s, \ldots, s^d]; g_2)$. For any number of updates, randomly select δ, compute $e_\delta = E(\delta)$, so that $\delta = p_i' - p_i = (\omega + u_i') - (\omega + u_i) = u_i' - u_i$. Thus update $u_i' \leftarrow \delta + u_i$ and, therefore, compute $\Delta = g_1^{\delta\alpha}$ and $\bar{\mathcal{K}}'[j] = e(\Delta[j]^{s^i}; g_2) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$. Alternatively run such updates with random `VEval` phases; all this is indistinguishable from a normal transcript of the protocol. Now if from this transcript an attacker could find one p_j, then compute $\omega = p_j - u_j$ and the encrypted value would be revealed. □

5.2 Experiments

To assess the efficiency of our protocol, we implemented Table 5 using the following libraries[2]: `gmp-6.2.1` for modular operations, `fflas-ffpack-2.4.3` for linear algebra, `relic-0.5.0` for Paillier's cryptosystem and pairings (we used a "bn-p254" pairing over a 254-bits group).

To observe the effect of the chosen homomorphic systems (Paillier and the pairing), we ran the experiments, on a single core of a i7-6700 3.4GHz, in two sets: the first one with a low RSA modulus size of 1024 bits, and the second one with a RSA modulus size of 2048 bits.

In Table 7, we thus compare the Server time to the Client time of our protocol, to that of a simple (witness) polynomial evaluation (Horner-like) in this group. First of all, of course, the Server time, using homomorphic arithmetic, can be several orders of magnitude slower than the simple polynomial evaluation, while indeed being clearly linear. Still for a large enough degree, we can observe the logarithmic Client time to win over the linear time pure polynomial evaluation.

Second, for the protocol itself, we see that both homomorphic evaluations of the Server are quite similar, even if the Paillier cryptosystem is more expensive for large modulus. Then, on the Client side and for the considered degrees, the dominant computation is that of a single Paillier's deciphering (and that the only part non-constant in the degree is by far the most negligible).

[2] `https://gmplib.org`, `https://linbox-team.github.io/fflas-ffpack`, `https://github.com/relic-toolkit/relic`.

Table 7: Comparative behaviors of pairings and Paillier system on the Server and Client sides with a 254-bits group size for the protocol of Table 5 (on the client side, column 'pows' is the time to perform the left hand-side exponentiations (by $s - r$ and by $D(\zeta)\alpha[j] + c[j]$); column 'c' is the time to perform the matrix geometric sum (the clients's only part non constant in d); and column 'D' is the time to perform the single Paillier's deciphering; column 'Horner' is a witness simple polynomial evaluation in that group. Each experiment was performed 11 times and we report the median value, with a maximum variance lower than 14.3% between runs).

Degree	Paillier	Server Certif.		Client Verif.			Horner
		ζ	ξ	pows	c	D	
256	1024	0.05s	0.12s	0.7ms	<0.1ms	0.3ms	<0.1ms
512	1024	0.09s	0.25s	0.7ms	<0.1ms	0.3ms	0.1ms
1024	1024	0.18s	0.49s	0.7ms	<0.1ms	0.3ms	0.2ms
2048	1024	0.35s	0.97s	0.7ms	<0.1ms	0.3ms	0.4ms
4096	1024	0.71s	1.95s	0.7ms	<0.1ms	0.3ms	0.9ms
8192	1024	1.42s	3.90s	0.7ms	<0.1ms	0.3ms	1.9ms
16384	1024	2.84s	7.73s	0.7ms	<0.1ms	0.3ms	3.8ms
32768	1024	5.67s	15.72s	0.7ms	<0.1ms	0.3ms	7.6ms
65536	1024	11.32s	31.30s	0.7ms	<0.1ms	0.3ms	15.3ms
131072	1024	22.66s	62.21s	0.7ms	<0.1ms	0.3ms	30.4ms
256	2048	0.13s	0.12s	0.7ms	<0.1ms	0.9ms	<0.1ms
512	2048	0.26s	0.24s	0.7ms	<0.1ms	0.9ms	0.1ms
1024	2048	0.52s	0.48s	0.7ms	<0.1ms	0.9ms	0.2ms
2048	2048	1.02s	0.98s	0.7ms	<0.1ms	0.9ms	0.4ms
4096	2048	2.07s	1.97s	0.7ms	<0.1ms	0.9ms	0.9ms
8192	2048	4.16s	3.90s	0.7ms	<0.1ms	0.9ms	1.9ms
16384	2048	8.28s	7.82s	0.7ms	<0.1ms	0.9ms	3.8ms
32768	2048	16.56s	15.54s	0.7ms	<0.1ms	0.9ms	7.6ms
65536	2048	33.02s	31.38s	0.7ms	<0.1ms	0.9ms	15.3ms
131072	2048	66.37s	63.26s	0.7ms	<0.1ms	0.9ms	30.4ms

6 Low server storage dynamic proof of retrievability

Recall that Proofs of Retrievability (PoR) allow a client with limited storage, who has outsourced her data to an untrusted server, to confirm via an efficient Audit protocol that the data is still being stored in its entirety.

The lower bound of [5, Theorem 4] proves that a tradeoff is inevitable between low/high audit cost and high/low storage overhead. Roughly speaking, for any PoR on an N-bit database, the product of persistent storage overhead times audit computational complexity must be at least N.

The dynamic PoR schemes of [14, 36] optimize for fast audits. They incur a large $\mathcal{O}(N)$ storage overhead on the server, but can perform audits with only $(\log N)^{\mathcal{O}(1)}$ communication and computation for the client and server.

Instead, [5] optimizes for small storage overhead; their scheme has only sub-linear storage overhead of $\mathcal{O}(N/\log N)$, but a higher audit cost of $\mathcal{O}(N)$ on the server, and $\mathcal{O}\left(\sqrt{N}\right)$ client time and communication. The authors demonstrate that, for reasonable deployment scenarios on commercial cloud platforms, the higher audit cost is more than offset by the greatly reduced costs of extra persistent storage, especially if audits are only performed a few times per day.

We here further improve on the low storage overhead approach of [5], by our scheme with a small $o(N)$ storage overhead, but only $\mathcal{O}(\log N)$ communication and client computation cost for audits. That is, our new protocol still benefits from small storage overhead, while effectively pushing the higher computational cost of audits (which is inevitable from the lower bound) entirely off the client and onto the server. These savings are highlighted in Table 8.

An easy argument demonstrates that in fact our $\mathcal{O}(\log N)$ client cost for audits is a optimal. If each audit has $o(\log N)$ cost (and therefore transcript size) for audits, then the total number of possible transcripts is $o(N)$, which is a contradiction with the definition of retrievability; not every N-bit database could be recoverable via independent audit transcripts.

Table 8: Attributes of some selected Proof of Retrievability schemes

Protocol	Server			Client	
	Extra Storage	Audit Comput.	Audit Comm.	Storage	Audit Comput.
Shi et al. [36]	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$
Anthoine et al. [5]	$o(N)$	$N + o(N)$	$\mathcal{O}\left(\sqrt{N}\right)$	$\mathcal{O}(1)$	$\mathcal{O}\left(\sqrt{N}\right)$
Here	$o(N)$	$N + o(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$

6.1 Matrix based approach for audits

Here we summarize the PoR presented in [5] upon which our new scheme is based. The basic premise is to treat the data, consisting of N bits organized in machine words, as a matrix $M \in \mathbb{Z}_p^{m \times n}$, where $\mathbb{Z}_p$ is a suitable finite field of size p. Crucially, the choice of ring $\mathbb{Z}_p$ does not require any modification to the raw data itself; that is, any element of the matrix M can be retrieved in $O(1)$ time from the underlying raw data storage. The scheme is based on the commutativity of matrix-vector products. During the Setup phase, the client chooses a secret vector u of dimension m and computes $v^\intercal = u^\intercal M$; both vectors u and v are then stored by the client for later use, while the server stores the original data and hence the matrix M in the clear. Reading or updating individual entries in M can be performed efficiently with the use of Merkle hash trees and from the observation that changing one element of M only requires changing one entry in the client's secret control vector v.

To perform an audit, the client and server engage in a 1-round protocol:

1. Client chooses a random vector x of dimension n, and sends x to Server.
2. Server computes $y = Mx$ and sends the dimension-m vector y back to Client.
3. Client computes two dot products $u^\intercal y$ and $v^\intercal x$, and checks that they are equal.

The proof of retrievability relies on the fact that observing several successful audits allows, with high probability, recovery of the correct matrix M, and therefore of the entire database.

The communication costs are $\mathcal{O}(n)$ and $\mathcal{O}(m)$ in steps 1 and 2 respectively, and the client computation in step 3 is $\mathcal{O}(m+n)$, resulting in $\mathcal{O}\left(\sqrt{N}\right)$ total communication and client computation when optimizing the matrix dimensions to roughly $m = n = \sqrt{N}$. While this square-matrix setup is the basic protocol presented by [5], the authors also discuss a potential improvement in communication complexity. Instead of x being uniformly random over $\mathbb{Z}_p^n$, it can instead be a *structured* vector formed from a single random element $r \in \mathbb{Z}_p$ as $x = [r^i]_{i=1..n}$ Then the communication on step 1 is reduced to constant, and hence the total communication depends only on the row dimension $\mathcal{O}(m)$. By choosing a rectangular matrix M with few rows and many columns, the communication can be made arbitrarily small.

The tradeoff for this reduction in communication complexity is higher client storage of the control vector v as well as higher client computation cost for the n-dimensional dot product $v^\intercal x$. In [5], the authors found that the savings in communication were not worth the higher client storage and computation, and their experimental evaluation was based on the square matrix version with overhead $\mathcal{O}\left(\sqrt{N}\right)$.

6.2 Bootstrapping part of the client computation via ciphered and dynamic polynomial evaluation

Now we show how to modify the reduced communication version of the PoR protocol of [5] just presented in order to eliminate the costly client storage of $v \in \mathbb{Z}_p^n$ and computation of $v^\intercal x$ during audits. Our improved protocol is based on the observation that, when the audit challenge vector x is structured as $x = [r^i]$, then the expensive client dot product computation of $v^\intercal x$ is actually a polynomial evaluation: if the entires of v are the coefficients of a polynomial P, then $v^\intercal x$ is simply $P(r)$. We therefore eliminate the $\mathcal{O}(n)$ client persistent storage and computation cost during audits by outsourcing the (encrypted) storage of vector v and computation of $v^\intercal x = P(r)$ with our novel protocol for dynamic, encrypted,

verifiable polynomial evaluation scheme of Table 5. The obtained private-verification PoR protocol, combining the PoR of [5] with our ciphered polynomial evaluation in Section 5, is presented in Table 9.

Table 9: Private verifiable Client/server PoR protocol with low storage server

	Server	Communications	Client
`Setup`		$N = mn\log_2 p$ $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of ord. p pairing e to $\mathbb{G}_T$ gen. g_1, g_2, g_T	$\gamma, s \xleftarrow{\$} \mathbb{Z}_p^*$ form $u^\intercal \leftarrow [\gamma^i]_{i=0\ldots m-1} \in \mathbb{Z}_p^m$ $v^\intercal \leftarrow u^\intercal M \in \mathbb{Z}_p^n$ $\alpha, \beta \xleftarrow{\$} (\mathbb{Z}_p^*)^2$, $\Phi \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ $\bar{v} \leftarrow [(v_k\alpha + \Phi^k\beta)]_{k=0..n-1} \in \mathbb{Z}_p^{2\times n}$ $w \leftarrow E(v)$, $\bar{H} \leftarrow [g_1^{\bar{v}_k}]_{k=0..n-1} \in \mathbb{Z}_p^{2\times n}$ $\sigma \leftarrow [s^k]_{k=0..n-1}$, $\bar{\mathcal{K}} = g_T^{\bar{v}\sigma}$, $S \leftarrow g_2^\sigma$
	$T_M \leftarrow \text{MTTree}(M)$ $T_w \leftarrow \text{MTTree}(w)$ Store $M, T_M, w, T_w, \bar{H}, S$	$\xleftarrow{M,w,\bar{H},S}$	$r_M \leftarrow \text{MTRoot}(M)$ $r_w \leftarrow \text{MTRoot}(w)$ Discard M, v, $\bar{v}$, w, $\bar{H}$, σ, S. Store $\gamma, r_M, \alpha, \beta, s, \Phi, \bar{\mathcal{K}}, r_w$
`Update`	$(M_{ik}, L_{M_{ik}}) \leftarrow \text{MTLeafPath}(k + i\cdot n, M, T_M)$ $(w_k, L_{w_k}) \leftarrow \text{MTLeafPath}(k, w, T_w)$	$\xleftarrow{i,k}$ $\xrightarrow{M_{ik}, L_{M_{ik}}, w_k, L_{w_k}}$	$r_w \stackrel{?}{=} \text{MTpathRoot}(k, w_k, L_{w_k})$ $r_M \stackrel{?}{=} \text{MTpathRoot}(k + i\cdot n, M_{ik}, L_{M_{ik}})$
	$M_{ik} \leftarrow M'_{ik}$, $w_k \leftarrow w_k \cdot e_\delta$ $\bar{H}_k[j] \leftarrow \Delta[j]\bar{H}_k[j]$ for $j = 1..2$ $T_w \leftarrow \text{MTupdLeaf}(k, w_k, T_w)$ $T_M \leftarrow \text{MTupdLeaf}(k + i\cdot n, M'_{ik}, T_M)$	$\xleftarrow{M'_{ik}, e_\delta, \Delta}$	$\delta \leftarrow \gamma^i(M'_{ik} - M_{ik})$, $e_\delta \leftarrow E(\delta)$, $\Delta \leftarrow g_1^{\delta\alpha}$ $\bar{\mathcal{K}}[j] \leftarrow e(\Delta[j]^{s^k}; g_2)\cdot\bar{\mathcal{K}}[j]$ for $j = 1..2$ $r_w \leftarrow \text{MTpathRoot}(k, w_k e_\delta, L_{w_k})$ $r_M \leftarrow \text{MTpathRoot}(k + i\cdot n, M'_{ik}, L_{M_{ik}})$
`Audit`	form $x \leftarrow [r^k]_{k=0\ldots n-1}^\intercal \in \mathbb{Z}_p^n$ $y \leftarrow Mx$, $\zeta = w^\intercal \boxdot x$ $\bar{\xi} = \prod_{i=1}^{n-1}\prod_{k=0}^{i-1} e(\bar{H}_i; S_{i-k-1})^{x_k}$	$\xleftarrow{r}$ $\xrightarrow{y,\zeta,\bar{\xi}}$	$r \xleftarrow{\$} \mathbb{Z}_p^*$ s.t. $(r\Phi - I_2) \in GL_2(\mathbb{Z}_p)$ $c \leftarrow ((r\Phi)^{d+1} - I_2)(r\Phi - I_2)^{-1}\beta$ $\bar{\xi}[j]^{s-r} g_T^{D(\zeta)\alpha[j]+c[j]} \stackrel{?}{=} \bar{\mathcal{K}}[j]$ for $j = 1..2$ $u^\intercal y \stackrel{?}{=} D(\zeta)$

Theorem 14. *The protocol of Table 9 is correct and sound.*

Proof. For the sake of simplicity, we here only consider the case $t = 1$, that is a single control vector.

Correctness. Assume that all the parties are honest. After each update phase, thanks to the correctness of the Merkle hash tree algorithms $w^\intercal = E(u^\intercal M)$ and $\bar{\mathcal{K}} = e(g_1^{\bar{v}\sigma}; g_2)$. To see this, suppose a modification of the database at indices i and k, and let $M' = M + (M'_{ik} - M_{ik})\mathcal{E}_{ik}$ where $\mathcal{E}_{ik}$ is the single entry matrix with 1 at position (i, k). We have $u^\intercal M' = u^\intercal M + u^\intercal(M'_{ik} - M_{ik})\mathcal{E}_{ik} = u^\intercal M + \gamma^i e_k(M'_{ik} - M_{ik})$ where e_k is the k-th canonical vector. Thus, $v' = v + \gamma^i(M'_{ik} - M_{ik})e_k = v + \delta e_k$ satisfies $u^\intercal M' = v'^\intercal$. Only the k-th coefficients are different in v and v', and in w and w' as well. For the latter, $w'_k = E(v'_k) = E(v_k + \delta) = E(v_k)E(\delta) = w_k E(\delta)$. The server thus computes w' such that $w' = E(u^\intercal M')$. Moreover, for $j = 1..2$, $\bar{v}'[j] = \bar{v}[j] + \delta\alpha[j]e_k$, so that, similarly, $\bar{H}'_k[j] = \Delta[j]\bar{H}_k[j]$ with $\Delta = g_1^{\delta\alpha}$, and $\bar{\mathcal{K}}'[j] = e(g_1^{\bar{v}'[j]\sigma}; g_2) = e(g_1^{\bar{v}[j]\sigma} g_1^{\delta\alpha[j]e_k\sigma}; g_2) = \bar{\mathcal{K}}[j]\cdot e(g_1^{\delta\alpha[j]s^k}; g_2) = \bar{\mathcal{K}}[j]\cdot e(\Delta[j]^{s^k}; g_2)$. Now, concerning the audit phase. Since we consider the polynomial evaluation as a dotproduct, the application of Proposition 8 to our notations gives: $(s-r)\left(\sum_{i=1}^{n-1}\sum_{k=0}^{i-1}\bar{v}_i s^{i-k-1}r^k\right) + \sum_{i=0}^{n-1}\bar{v}_i r^i = \sum_{i=0}^{n-1}\bar{v}_i s^i$. Thus the audit phase is: $\bar{\xi} = \prod_{i=1}^{n-1}\prod_{k=0}^{i-1} e(\bar{H}_i; S_{i-k-1})^{x_k} = \prod_{i=1}^{n-1}\prod_{k=0}^{i-1} e(g_1^{\bar{v}_i}; g_2^{s^{i-k-1}})^{r^k} = e(g_1; g_2)^{\sum_{i=1}^{n-1}\sum_{k=0}^{i-1}\bar{v}_i s^{i-k-1}r^k}$. Moreover, $\alpha D(\zeta) + c = \alpha v x + ((r\Phi)^{d+1} - I_2)(r\Phi - I_2)^{-1}\beta = \alpha v x + \sum_{k=0}^{n-1} r^k\Phi^k\beta = \bar{v}x$. Thus we have that $\bar{\xi}[j]^{s-r} g_T^{D(\zeta)\alpha[j]+c[j]} = g_T^{(s-r)(\sum_{i=1}^{n-1}\sum_{k=0}^{i-1}\bar{v}_i[j]s^{i-k-1}r^k) + \bar{v}[j]x} = g_T^{\bar{v}[j]\sigma} = \bar{\mathcal{K}}[j]$ and, finally, $u^\intercal y = u^\intercal M x = v^\intercal x$.

Soundness. An attacker to the protocol must provide (y', ζ', ξ') such that $(y', \zeta', \xi') \neq (y, \zeta, \xi)$, but still $u^\intercal y' = D_{sk}(\bar{\zeta}')$, with a non negligible advantage ϵ. There are two cases: if $(D_{sk}(\bar{\zeta}'), \xi') \neq (D_{sk}(\bar{\zeta}), \xi)$ then the attacker had to break the polynomial evaluation; otherwise, it must be that $u^\intercal y' = u^\intercal y$ with $y' \neq y$.

For the first case, Theorem 13 assesses the security of the polynomial evaluation. For the second case, we consider $T = E_{pk}(t)$ the cipher of a secret t by the homomorphic scheme. Here, as in Theorem 13, we use the fact that the protocol of Table 9 is indistinguishable as a whole from the same protocol where, within the polynomial evaluation of, $\Phi^i\beta$ is everywhere replaced by a random Γ_i. Further, this

is indistinguishable from a third protocol where, at each `Update` of index i, a new Γ'_i is also randomly redrawn and replaces Γ_i in the client state. We thus continue the proof with this third game setting. Now, using e_ℓ the ℓ-th canonical vector of $\mathbb{Z}_p^m$, we can (abstractly) consider $\tilde{u} = u + te_\ell$ and $\tilde{v}^\intercal = \tilde{u}^\intercal M = (u^\intercal + te_\ell)M = v + tM_{\ell,*}$. Then, for the `Setup` phase, we can randomly select m, n and $\ell \leq m$. Then also $M \in \mathbb{Z}_p^{m\times n}$, $u \in \mathbb{Z}_p^m$, and compute $v^\intercal = u^\intercal M$. From this, compute $w_k = E(v_k)T^{M_{\ell k}} = E(v_k + tM_{\ell k}) = E(\tilde{v}_k)$. We also randomly select s, α and $\bar{h}_k$ (so that $\Gamma_k = \log_{g_1}(\bar{h}_k) - \tilde{v}_k\alpha$ exists, but is unknown). For any `Update` phases, compute $w'_k = w_k T^{M'_{\ell k} - M_{\ell k}}$ and select randomly a Δ (so that $\bar{h}'_k[j] = \bar{h}_k[j]\Delta[j]$ for $j = 1..2$ now correspond to a new $\Gamma_k = \log_{g_1}(h'_k) - \alpha\tilde{v}'_k$ still unknown).

Finally, the attacker provides a vector y' such that both $\tilde{u}^\intercal(y' - y) = 0$ and $y' \neq y \mod p$. Since ℓ is randomly chosen from $1..m$, the probability that the vectors are distinct at index ℓ, in other words that $y'_\ell \neq y_\ell \mod p$, is at least $1/m$. If this is the case, then, denoting $z = y' - y$, we have that $z_\ell \neq 0 \mod p$. Now, $\tilde{u}^\intercal z = 0$ implies that $u^\intercal z + tz_\ell = 0$ so that the secret can be computed as $t \equiv -{z_\ell}^{-1} \cdot (u^\intercal z) \mod p$ and the homomorphic cryptosystem is subject to an attack with advantage ϵ/m. □

6.3 Experiments

We now compare our modification of the PoR protocol with the one in [5], publicly available there: https://github.com/dsroche/la-por.

Table 10 has three blocks of experiments, each for four database sizes ranging from 1GB to 1TB. The first block of experiments is a run of the original statistically secure PoR protocol with two dotproducts for the verification, considering the matrix as 56 bits elements modulo a 57-bits prime. The second block of experiments is our new modification, but still using close to square matrices. Subject now to computational security, we have to use a larger coefficient domain, namely here a 254-bits prime (with associated bilinear groups and a 2048-bits Paillier modulus, both estimated equivalent to a 112-bit computational security). We separate the timings of the `Update` phase in two phases, the remaining linear algebra phase and the new polynomial evaluation phase. In the third block of experiments we use a more rectangular matrix, trying to reduce communications while not increasing too much the Server computational effort.

Overall, we see first in Table 10, that changing the coefficient domain size increases the computational effort of the server in the linear algebra phase. Still, reducing the dimension of the dotproduct for the client, as shown in he third block, allows the client to be faster for databases larger than 100GB. In any case, the client audit computational effort is never larger than a few milliseconds and thus the dominant part is most certainly communications. On this aspect, we see that our modification allows for large reductions in both the Client storage (even with square matrices) and the overall communications. Indeed, the client private state is the vector dimension, the Paillier's private key, twelve group elements and two Merkle tree roots; while the communications are mostly one vector of modular integers in the smallest dimension.

The price to pay is from about a factor of four (large database) to an order of magnitude (tiny database) for the server computations. But the persistent client storage is going from dozens of MB to less than one KB, and the communication volume can be decreased by more than two orders of magnitude.

Acknowledgments

We thank Gaspard Anthoine for providing us with some preliminary comparisons for the verification of ciphered polynomials using the PBC and libpaillier libraries.

7 Conclusion

We have presented a protocol verifying the outsourced evaluation of secret polynomials. Client verification is of the order of a few milliseconds and is faster than direct polynomial evaluation over a small finite field, as soon as the degree of the polynomial is larger than a few thousand.

This enables us to reduce by several orders of magnitude the communications, Client storage and Client computations for state-of-the-art low server-storage dynamic proofs of retrievability.

Table 10: Modification of the PoR audit protocol, with 254-bits groups, 2048-bits Paillier, on 1 core i7-6700 3.40GHz & 64 GB ram (CPU time are the median values for a single run; each experiment was performed 11 times; in all cases, the maximum relative difference between the runs was at most 3.6%).

Database	1GB	10GB	100GB	1TB
Private-verified audit using 57-bits prime [5, Figure 1]				
Matrix view	12339×12432	39131×39200	123831×123872	396281×396368
Server extra storage	<0.01%	<0.01%	<0.01%	<0.01%
Client Storage	169KB	535KB	1 693KB	5 418KB
Server Audit	0.2s	2.4s	158.6s	1 503.1s
Communications	169KB	535KB	1 693KB	5 418KB
Client Audit	0.2ms	0.5ms	7.6ms	18.8ms
Dynamic-ciphered delegated polynomial evaluation with 254-bits groups of Table 9				
Matrix view	5815×5816	18390×18390	58154×58154	186092×186093
Server extra storage	0.12%	0.04%	0.01%	<0.01%
Client storage	0.94KB	0.94KB	0.94KB	0.94KB
Server Audit				
(matrix-vector part)	4.3s	43.2s	433.2s	4 441.7s
(polynomial part)	5.6s	17.6s	55.5s	176.1s
Communications	181KB	571KB	1 803KB	5 770KB
Client Audit				
(dotproduct part)	1.7ms	20.3ms	56.8ms	79.9ms
(polynomial part)	1.7ms	1.7ms	1.7ms	1.7ms
Dynamic-ciphered delegated polynomial evaluation with 254-bits groups of Table 9				
Matrix view	6599×5125	7265×46551	7929×426519	8600×4026778
Server extra storage	0.11%	0.10%	0.09%	0.08%
Client storage	0.94KB	0.94KB	0.94KB	0.94KB
Server Audit				
(matrix-vector part)	4.3s	42.1s	434.0s	4 534.0s
(polynomial part)	5.0s	44.7s	412.7s	3 849.7s
Communications	205KB	226KB	246KB	267KB
Client Audit				
(dotproduct part)	1.9ms	2.1ms	2.3ms	2.5ms
(polynomial part)	1.7ms	1.7ms	1.7ms	1.7ms

We mention as possible future work the potential to parallelize the polynomial part of the Server's computation. The matrix-vector product part was already parallelized in [5, Table 6], a Server auditing the 1TB database in a few minutes. For us, as the dimensions become more rectangular, as we can see in Table 10, the Server's polynomial part is sometimes not negligible anymore, and could thus also benefit from some parallelization. For this, we would need to parallelize both the Paillier's homomorphic dot-product and the Horner-like pairings. On the one hand, the former operations, line 3 in Algorithm 7, could be blocked in independant exponentiations and a final multiplications in a binary tree. On the other hand, for the latter operations, a standard "baby steps / giant steps" approach could be employed for the iteration of lines 5-8 in Algorithm 7:

- First, for steps of size k, compute t^{r^k}, then $t^{r^{kj}}$ for $j = 1..d/k$ as a parallel prefix; then iterates the multiplications by the coefficients of S in parallel for the d/k blocks.
- Second, then all the pairings could be computed in parallel and their final multiplications performed again with a binary tree.

This parallelism could be used to further reduce the Server latency for large databases, allow faster multi-user queries, and make the scheme even more practically relevant.

References

[1] M. Abdalla, F. Benhamouda, and A. Passelègue. An algebraic framework for pseudorandom functions and applications to related-key security. In R. Gennaro and M. Robshaw, editors, *CRYPTO 2015*, pages 388–409, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-662-47989-6_19.

[2] M. Abdalla, F. Bourse, H. Marival, D. Pointcheval, A. Soleimanian, and H. Waldner. Multi-client inner-product functional encryption in the random-oracle model. In C. Galdi and V. Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, Amalfi, Italy, September 14-16, 2020*, volume 12238 of *LNCS*, pages 525–545. Springer, 2020. doi:10.1007/978-3-030-57990-6_26.

[3] S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In M. Robshaw and J. Katz, editors, *CRYPTO 2016*, pages 333–362, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-662-53015-3_12.

[4] M. Ambrona, G. Barthe, and B. Schmidt. Generic transformations of predicate encodings: Constructions and applications. In J. Katz and H. Shacham, editors, *CRYPTO 2017*, pages 36–66, Cham, 2017. Springer. doi:10.1007/978-3-319-63688-7_2.

[5] G. Anthoine, J.-G. Dumas, M. Hanling, M. de Jonghe, A. Maignan, C. Pernet, and D. S. Roche. Dynamic proofs of retrievability with low server storage. In *30th USENIX Security Symposium, August 11-13*, pages 537–554, Aug. 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/anthoine.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *14th ACM CCS*, pages 598–609. ACM, 2007. doi:10.1145/1315245.1315318.

[7] N. Attrapadung and J. Tomida. Unbounded dynamic predicate compositions in abe from standard assumptions. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020*, pages 405–436, Cham, 2020. Springer. doi:10.1007/978-3-030-64840-4_14.

[8] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *CRYPTO 2011, Santa Barbara, CA, USA, August 14-18, 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, 2011. doi:10.1007/978-3-642-22792-9_7.

[9] J. Benaloh. Dense probabilistic encryption. In *First Annual Workshop on Selected Areas in Cryptography*, pages 120–128, Kingston, ON, May 1994. URL: http://sacworkshop.org/proc/SAC_94_006.pdf.

[10] A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Auckland, New Zealand, November 29 - December 3, 2015, Part I*, volume 9452 of *LNCS*, pages 470–491. Springer, 2015. doi:10.1007/978-3-662-48797-6_20.

[11] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *IACR Cryptol. ePrint Arch.*, 2020:81, 2020. URL: https://eprint.iacr.org/2020/081.

[12] X. Bultel, M. L. Das, H. Gajera, D. Gérault, M. Giraud, and P. Lafourcade. Verifiable private polynomial evaluation. In T. Okamoto, Y. Yu, M. H. Au, and Y. Li, editors, *Provable Security*, pages 487–506, Cham, 2017. Springer. doi:10.1007/978-3-030-41702-4_4.

[13] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Auckland, New Zealand, November 29 - December 3, 2015, Part II*, volume 9453 of *LNCS*, pages 262–288. Springer, 2015. doi:10.1007/978-3-662-48800-3_11.

[14] D. Cash, A. Küpçü, and D. Wichs. Dynamic proofs of retrievability via oblivious RAM. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013, Athens, Greece, May 26-30, 2013*, volume 7881 of *LNCS*, pages 279–295. Springer, 2013. doi:10.1007/978-3-642-38348-9_17.

[15] D. Catalano and D. Fiore. Vector commitments and their applications. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography - PKC 2013, Nara, Japan, February 26 - March 1, 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, 2013. doi:10.1007/978-3-642-36362-7_5.

[16] K. Elkhiyaoui, M. Önen, M. Azraoui, and R. Molva. Efficient techniques for publicly verifiable delegation of computation. In X. Chen, X. Wang, and X. Huang, editors, *AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 119–128. ACM, 2016. doi:10.1145/2897845.2897910.

[17] C. M. Fiduccia. An efficient formula for linear recurrences. *SIAM J. Comput.*, 14(1):106–112, 1985. doi:10.1137/0214007.

[18] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM CCS*, pages 501–512, New York, NY, USA, 2012. ACM. doi:10.1145/2382196.2382250.

[19] D. Fiore, A. Nitulescu, and D. Pointcheval. Boosting verifiable computation on encrypted data. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 124–154, Cham, 2020. Springer. doi:10.1007/978-3-030-45388-6_5.

[20] L. Fousse, P. Lafourcade, and M. Alnuaimi. Benaloh's dense probabilistic encryption revisited. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011, Dakar, Senegal, July 5-7, 2011*, volume 6737 of *LNCS*, pages 348–362. Springer, 2011. doi:10.1007/978-3-642-21969-6_22.

[21] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019. URL: https://eprint.iacr.org/2019/953.

[22] R. Gay, D. Hofheinz, E. Kiltz, and H. Wee. Tightly CCA-secure encryption without pairings. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016*, pages 1–27, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-662-49890-3_1.

[23] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010, Santa Barbara, CA, USA, August 15-19, 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, 2010. doi:10.1007/978-3-642-14623-7_25.

[24] V. Goyal. Reducing trust in the PKG in identity based cryptosystems. In A. Menezes, editor, *CRYPTO 2007, Santa Barbara, CA, USA, August 19-23, 2007*, volume 4622 of *LNCS*, pages 430–447. Springer, 2007. doi:10.1007/978-3-540-74143-5_24.

[25] A. Juels and B. S. Kaliski Jr. Pors: Proofs of retrievability for large files. In *14th ACM CCS*, pages 584–597. ACM, 2007. doi:10.1145/1315245.1315317.

[26] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In M. Abe, editor, *ASIACRYPT 2010, Singapore, December 5-9, 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

[27] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2020. URL: https://books.google.fr/books?id=RsoOEAAAQBAJ.

[28] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu. Function-hiding inner product encryption is practical. In D. Catalano and R. D. Prisco, editors, *Security and Cryptography for Networks, SCN 2018, Amalfi, Italy, September 5-7, 2018*, volume 11035 of *LNCS*, pages 544–562. Springer, 2018. doi:10.1007/978-3-319-98113-0_29.

[29] B. Laurie, A. Langley, E. Kasper, and Google. Certificate Transparency. RFC 6962, IETF, June 2013. URL: https://tools.ietf.org/html/rfc6962.

[30] B. Libert, S. C. Ramanna, and M. Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 30:1–30:14. Dagstuhl, 2016. doi:10.4230/LIPIcs.ICALP.2016.30.

[31] R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *CRYPTO '87*, pages 369–378, 1988. doi:10.1007/3-540-48184-2_32.

[32] P. Morillo, C. Ràfols, and J. L. Villar. The kernel matrix Diffie-Hellman assumption. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016*, pages 729–758, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-662-53887-6_27.

[33] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh. Scaling verifiable computation using efficient set accumulators. In S. Capkun and F. Roesner, editors, *29th USENIX Security Symposium, August 12-14*, pages 2075–2092, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/ozdemir.

[34] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT '99, Czech Republic, May 2-6, 1999*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999. doi:10.1007/3-540-48910-X_16.

[35] H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *ASIACRYPT 2008, Melbourne, Australia, December 7-11, 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, 2008. doi:10.1007/978-3-540-89255-7_7.

[36] E. Shi, E. Stefanov, and C. Papamanthou. Practical dynamic proofs of retrievability. In *ACM CCS*, pages 325–336, New York, NY, USA, 2013. ACM. URL: http://elaineshi.com/docs/por.pdf, doi:10.1145/2508859.2516669.

[37] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In C. Galdi and V. Kolesnikov, editors, *Security and Cryptography for Networks, SCN 2020, Amalfi, Italy, September 14-16, 2020*, volume 12238 of *LNCS*, pages 45–64. Springer, 2020. doi:10.1007/978-3-030-57990-6_3.