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 †

March 13, 2023

Abstract

Proofs of Retrievability are protocols which allow a Client to store data remotely and to efficiently ensure, via audits, that the entirety of that data is still intact. Dynamic Proofs of Retrievability (DPoR) also support efficient retrieval and update of any small portion of the data. We propose a novel protocol for arbitrary outsourced data storage that achieves both low remote storage size and audit complexity. A key ingredient, that can be also of intrinsic interest, reduces to efficiently evaluating a secret polynomial at given public points, when the (encrypted) polynomial is stored on an untrusted Server. The Server performs the evaluations and also returns associated certificates. A Client can check that the evaluations are correct using the certificates and some pre-computed keys, more efficiently than re-evaluating the polynomial. 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. Our methods rely on linearly homomorphic encryption and pairings, and our implementation shows good performance for polynomial evaluations with millions of coefficients, and efficient DPoR with terabytes of data. For instance, for a 1TB database, compared to the state of art, we can reduce the Client storage by 5000x, communication size by 20x, and client-side audit time by 2x, at the cost of one order of magnitude increase in server-side audit time.

1 Introduction

With a constant growth in the amount of produced data, it becomes more and more important to use remote facilities to store this data. Users and organizations using such outsourcing need to ensure the *integrity* of their data.

In this setting, 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. This is the field of **Proofs of Retrievability** (PoR), somewhat overlapping with the problem of *Provable Data Possession* (PDP) [40, 6].

A variety of tools have been employed to develop efficient PoR and PDP protocols, see for instance [40, 6, 60, 22, 61, 5] and references therein. 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 single 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.

Some of these protocols are based on verifiable computing, so that a PoR audit consists of some verified computation over the stored data [36]. Generally speaking, verifiable computing consists in delegating the computation of a function to an untrusted Server. This Server returns the result as well

*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.

as a proof of its correctness, and verifying a result should be less expensive than computing it directly. While certified and verified computation protocols date back decades, the practical need for efficient methods is especially evident in cloud computing, wherein again a low-powered device, such as a mobile phone, may wish to outsource expensive and critical computations to an untrusted, shared-resource, commercial cloud. The literature on verifiable computation protocols can be divided into general-purpose computations — of an arbitrary algebraic circuit — and more limited but more efficient special-purpose computations of certain functions (see, e.g., [65, 26] and references therein). In the latter category, one problem is Verifiable Polynomial Evaluation (VPE), where a Client wishes to outsource the evaluation of a univariate polynomial P on an untrusted Server at given public points and efficiently verify the result.

Existing VPE protocols usually do not consider dynamicity, at least not efficiently: even for the modification of a single coefficient, most of the time the whole protocol has to be reinitialized. Also, previous PoR protocols would either have a low audit complexity but a storage size several times that of the database; or have low remote storage but a less scalable audit complexity. In this paper, we propose a novel protocol for arbitrary outsourced dynamic data storage that achieves both low remote storage size and audit complexity. A key ingredient of our protocol is to be able to perform a dynamic VPE, in order to efficiently handle updates of the database.

A verifiable polynomial evaluation scheme is conventionally composed of three main algorithms. First, a Client runs $\texttt{Setup}(P)$ to compute some public representation of the (potentially secret) polynomial 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, $\texttt{Eval}(x)$, is run by the Server using a public evaluation point x provided by the Client. The Server produces the evaluation $y = P(x)$ as well as some proof (or certificate) Π that this evaluation is correct. Finally, the third algorithm, $\texttt{Verify}(y, \Pi)$, 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 Π that will pass the verification, whereas an incorrect evaluation y will fail the verification with high probability for any purported proof Π. Furthermore, the `Verify` algorithm should be efficient, ideally much cheaper in time and/or space than the computation itself.

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, when needed:

- **Polynomial outsourcing**. When the Client device has limited storage, or to facilitate evaluations for multiple Clients, both the polynomial storage and its computation must be externalized. Besides evaluation and verification, an additional `Read` protocol is often provided to allow random access to some polynomial coefficients. The challenge is for the Client to obtain the polynomial evaluation while minimizing the communication costs required to verify it.

- **Secret polynomial**. To guarantee data privacy, the polynomial could 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 ones for verified polynomial evaluation; see, e.g., [31, 38, 8, 19, 50, 32, 12, 47, 57, 14].

- **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. A *dynamic* VPE protocol allows for such updates efficiently. Namely, the Client and Server storing polynomial $P = \sum_{i=0}^{d} p_i X^i$ for verified evaluation can engage in an additional $\texttt{Update}(i, p'_i)$ protocol, which effectively updates $P(x)$ to $P(x) + (p'_i - p_i)x^i$ for future evaluations. To the best of our knowledge, no prior work in the literature discusses dynamic updates for verified polynomial evaluation. When the polynomial (as well as any update) needs to be hidden from the Server, the difficulty is in general to preserve both secrecy and verifiability while allowing those efficient partial updates. The importance of allowing efficient updates is motivated by our application to verifiable data storage, where a Client outsourcing storage of a large database wishes to make small changes efficiently.

- **Private/public verification**. The verification protocol is said to be *private* when only the 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. Sometimes, 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 Verifiers, which may be used to check evaluations and proofs issued by the Server.

1.1 Our contributions

Our contributions are the following:

- An (unencrypted) Verifiable Polynomial Evaluation (VPE) scheme with public verification, supporting *secured dynamic updates* (Section 4 and Table 4). 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 12). 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 Dynamic Proofs of Retrievability (DPoR) 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 poly-logarithmic time audits and linear extra storage, or sub-linear extra storage and polynomial-time audits; ours is the first to achieve both sub-linear extra storage and optimal $\mathcal{O}(\log n)$ Client time for updates and audits. This could be beneficial especially in blockchain settings such as FileCoin where the proof and verification must be done on-chain [56].
- A full implementation and experimental timings based on our encrypted VPE and dynamic PoR protocols that indicate VPE up to millions of coefficients and DPoR up to terabytes of data, both with Client cost less than a few milliseconds (Tables 7 and 10).

These contributions are organized as follows. 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 [42], now supporting an encrypted input polynomial (Section 3 and Table 3). Since the privacy of this protocol is not proven and it supports 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`. Improving on previously-known results, our VPEs protocols all have $\mathcal{O}(d)$ (parallelizable) Server-side computation, $\mathcal{O}(1)$ communication and Client-side computation time, and $\mathcal{O}(1)$ Client-side persistent storage. We include some practical timings in Sections 5.4 and 6.3 and Appendix D. In addition, our new dynamic proofs of retrievability require only $o(d)$ extra Server space. This improves on [61] 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 [23, 20, 48, 34, 64, 15, 54, 32, 47]. There, the polynomial P is known to the Server, who publishes a binding commitment. The Verifier then confirms that a given evaluation is consistent with the pre-published commitment. 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) [13, 44, 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 [18]. 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 perform 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 [36] 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. [9, 30, 27] propose high-degree verified polynomial evaluations. The major issue for these works is that they were not meant to be dynamic: they use some structured masking that must be updated together with the polynomial update (otherwise updates leak some secrets). But then the update is not efficient anymore as the structure impacts all of the polynomial coefficient masking.

More recently, Fiore et al. [31, 32, 14] propose a new protocol for more general circuits, using succinct non-interactive arguments of knowledge (SNARKs) or probabilistically checkable proofs (PCPs) 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 – while preserving the security properties. A summary of how our protocols compare to the state of the art is given in Table 1.

Table 1: Comparing verifiable computation schemes for polynomial evaluation of degree d. See also [50, Table 1] or [57, Table 1] (Most of the time dynamicity is not considered in the literature).

Protocol	Server	Comm.	Verif.	Dyn.	LHE	P-Q
BGV11 [9]	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✗	✓	✗
FG12 [30]	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✗	✗	✗
libsnark [38]	$\mathcal{O}(d \log d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✗	MT	✗
bulletproof [19]	$\mathcal{O}(d)$	$\mathcal{O}(\log d)$	$\mathcal{O}(d)$	✗	✓	✗
FGP14 [31]	$\mathcal{O}(d \log d)$	$\mathcal{O}(1)$	$\mathcal{O}(\log d)$	✗	FHE	✗
libiop [8]	$\mathcal{O}(d \log d)$	$\mathcal{O}(\log^2 d)$	$\mathcal{O}(d)$	✗	✗	✓
ligero++ [12]	$\mathcal{O}(d \log d)$	$\mathcal{O}(\log^2 d)$	$\mathcal{O}(d)$	✗	✗	✓
FNP20 [32]	$\mathcal{O}(d \log d)$	$\mathcal{O}(1)$	$\mathcal{O}(\log d)$	✗	FHE	✗
DORY [47]	$\mathcal{O}(d \log d)$	$\mathcal{O}(\log d)$	$\mathcal{O}(\log d)$	✗	✓	✗
BCFK21 [14]	$\mathcal{O}(d)$	$\mathcal{O}(\log^2 d)$	$\mathcal{O}(\log^2 d)$	✗	FHE	✓
VESPo, Table 12	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✓	✓	✗

From this table, we see that many instances, like SNARKS, need $O(d \log d)$ operations on the Server side, where VESPo remains linear, $O(d)$ in the input size. Also dynamicity is usually not considered in the literature. For us, the difficulty is to be able to modify a small part of the input, without having to replay the whole `Setup` phase, while not compromising security. A salient point is that many schemes cannot directly handle the *encrypted* setting. In some cases a solution could be to simulate the whole encryption as arithmetic circuits, but this drastically affects performance. For instance we tried libsnark over a Paillier encryption, this rapidly exhausted the RAM of our server (i.e. even with degrees as small as 20), and thus denote this exhaustion by MT (memory thrashing). Finally, we mention if the protocol

is feasibly post-quantum (P-Q) secure in the 'P-Q' column ([8, 12] are P-Q-secure, [14] do not mention it but seems P-Q-secure, all the others, including us, use bilinear pairings). In Appendix E, we abstract the requirements of our protocols to see if they could be modified to use only post-quantum secure routines. Our preliminary results there show that this might be possible but that using quantum-safe routines in our case would still be several orders of magnitude slower.

In fact, efficiency, linearity, dynamicity and encryption, are all four of paramount importance for instance for our particular application, as detailed next.

Proof of retrievability (PoR) and Provable data possession (PDP) protocols also have an extensive literature [6, 28, 40, 59, 60, 63, 22, 61, 21, 5]. PDPs, first introduced by Ateniese et al., generally optimize Server storage and efficiency at the cost of soundness: a PDP audit only guarantees (probabilistically) that a *large fraction* of the data was not altered; a single block deletion or alteration is likely to go undetected in an audit.

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.

PoR methods based on block erasure encoding are a class of methods which guarantee with a high probability that the client's entire data can be retrieved. The idea is to check the authenticity of a number of erasure encoding blocks during the data recovery step but also during the audit algorithm. Those approaches will not detect a small amount of corrupted data. But the idea is that if there are very few corrupted blocks, they could be easily recovered via the error correcting code [46]. Now, state-of-the-art, dynamic, PoR protocols either incur a constant-factor blowup in Server storage with poly-logarithmic audit cost [22, 21, 61], or use negligible extra Server storage space but require polynomial-time audits on the Client and Server [60, 5]. We refer, e.g., to [5, § 7] for a more detailed comparison between PoR and PDP schemes. In fact, a lower bound argument from [5, Theorem 4] proves that some time/space tradeoff is inherent. 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. We show in Section 6 that with VESPo, we let the Server perform most of the computations (but this remains quite fast), so that we still need only negligible extra Server storage, but drastically reduce the Client communication, storage and computations.

2 Security properties and assumptions

2.1 Preliminaries

Pairings. 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. If such a pairing exists then $\mathbb{G}_1$ and $\mathbb{G}_2$ are denoted as bilinear groups. We often use groups of prime order, in order to be compute within the exponents. In particular, thanks to the homomorphic property of exponentiation, we will perform some linear algebra over the group and need 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 $W^B = (g^A)^B$. Further, 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: $e(g_1^{\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right)}; g_2^c) = e(g_1; g_2)^{\left(\begin{smallmatrix} ca \\ cb \end{smallmatrix}\right)}$.

Linearly Homomorphic Encryption (LHE). 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 homomorphic additions on ciphered messages and homomorphic multiplications but only between a

ciphered message and a cleartext.

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

Remark 1. *For instance, Paillier-like cryptosystems [55, 10, 33] 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 $r^i \mod m$ and require that: $(d+1)(m-1)^2 < N$. This way one can actually homomorphically compute over $\mathbb{Z}$ and use the modulo m only after decryption. See Appendix C for more details.

Merkle Hash Trees. Finally, we will use a Merkle hash tree to allow verifications of updates. A Merkle hash tree is a tree in which every leaf is labelled with the cryptographic hash of a data block, and every other node is labelled with the hash of the labels of its child nodes [51, 45, 5]. By just storing the root of the tree, one can check the presence of a given leaf in the tree with only a logarithmic number of additional nodes (*uncles*) and hash computations. More details on how we use them are given in Section 4.1.

2.2 Verifiable scheme

A verifiable dynamic polynomial evaluation (VDPE) scheme consists of five algorithms: `Setup`, `Read`, `Update`, `Eval` and `Verify` 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 $st_{\mathcal{V}}$. The algorithms can `reject`, when specified, if some inconsistencies are detected.

- $(st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}}) \leftarrow \texttt{Setup}(1^\kappa, P)$: On input of the security parameter κ and the polynomial P of degree $d^\circ(P) = d$, outputs the Client state $st_{\mathcal{C}}$, the Verifier $st_{\mathcal{V}}$ and the Server state $st_{\mathcal{S}}$. We denote by $\mathscr{S}_\kappa(P) = \{\texttt{Setup}(1^\kappa, P)\}$ the set of admissible states for a given polynomial (dependent on the different random choices).
- $\{p_i, \texttt{reject}\} \leftarrow \texttt{Read}(i, st_{\mathcal{V}}, st_{\mathcal{S}})$: On input of an index $i \in 0..d$, the Verifier/Server states $st_{\mathcal{V}}/st_{\mathcal{S}}$, outputs the i^{th} coefficient of P or `reject`.
- $\{(st'_{\mathcal{C}}, st'_{\mathcal{V}}, st'_{\mathcal{S}}), \texttt{reject}\} \leftarrow \texttt{Update}(i, p'_i, st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}})$: On input of an index $i \in 0..d$, data p'_i, 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 $P + (p'_i - p_i)X^i$, or `reject`. A variant of this algorithm, δ`Update`, takes as input the difference data $\delta = p'_i - p_i$ instead of p'_i.
- $\{\zeta, \bar{\xi}\} \leftarrow \texttt{Eval}(st_{\mathcal{S}}, r)$: On input of the Server state $st_{\mathcal{S}}$ and an evaluation point r, outputs ζ the encrypted value of $P(r)$ and a proof $\bar{\xi}$.
- $\{z, \texttt{reject}\} \leftarrow \texttt{Verify}(st_{\mathcal{V}}, r, \zeta, \bar{\xi})$: On input of the Verifier state $st_{\mathcal{V}}$, the evaluation point r, the encrypted value ζ of $P(r)$ and the proof $\bar{\xi}$, 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: $st_{\mathcal{V}} = st_{\mathcal{C}}$.

2.3 Security properties

Adapted from both [42, 31], in order to take into account dynamicity, we propose the following security game and the associated security properties:

Definition 2. *(`Setup`, `Read`, `Update`, `Eval`, `Verify`) is a secure publicly verifiable polynomial evaluation scheme if it satisfies the following three properties:*

Figure 1: VDPE soundness security 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 $\mathcal{S}$

1. $\mathcal{A}$ chooses an initial polynomial P.
2. $\mathcal{O}_1$ runs `Setup`, keeps $st_{\mathcal{C}}$ and sends the initial Server part, $st_{\mathcal{S}}$, of the memory layout to both $\mathcal{A}$ and $\mathcal{S}$; and the Verifier part, $st_{\mathcal{V}}$, to $\mathcal{O}_2$.
3. 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`, `Read`, `Eval` or `Verify`. $\mathcal{O}_1$ executes the `Update` operations with both $\mathcal{A}$ and $\mathcal{S}$, while $\mathcal{O}_2$ executes the `Read`, `Eval` or `Verify` operations also with both $\mathcal{A}$ and $\mathcal{S}$.
4. $\mathcal{A}$ is said to win the game, if any cleartext sent by $\mathcal{A}$ differs from that of $\mathcal{S}$, or, if any ciphered message sent by $\mathcal{A}$ does not deciphers like that of $\mathcal{S}$, and neither $\mathcal{O}_1$ nor $\mathcal{O}_2$ did witness `reject`.

(i) **Correctness.** *Let $d \in \mathbb{N}$, $P(X) = \sum_{i=0}^{d} p_i X^i$, with $(p_0, \ldots, p_d)$ in a ring $\mathcal{R}$ and $(st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}}) \in \mathscr{S}_\kappa(P)$, then for any $0 \leqslant i \leqslant d$ and $p'_i \in \mathcal{R}$:*

$$\texttt{Read}(i, st_{\mathcal{V}}, st_{\mathcal{S}}) = p_i \tag{3}$$

$$\texttt{Update}(i, p'_i, st_{\mathcal{C}}, st_{\mathcal{V}}, st_{\mathcal{S}}) \in \mathscr{S}_\kappa(P + (p'_i - p_i)X^i) \tag{4}$$

$$\texttt{Verify}(st_{\mathcal{V}}, r, \texttt{Eval}(st_{\mathcal{S}}, r)) = P(r) \tag{5}$$

(ii) **Soundness.** *The soundness requirement stipulates that the Client (or the Verifier) always* `reject` *(except with negligible probability) if any message sent by the Server deviates from the honest (correct) behavior*[1]*. A VDPE scheme is* sound*, if no polynomial-time adversary has more than negligible probability in winning the security game of Figure 1.*

(iii) **Privacy.** *We use now the following variant game: $\mathcal{A}$ chooses two initial polynomials P_0, P_1; $\mathcal{O}_1$ randomly chooses one bit $b \in \{0, 1\}$; the players run steps (2-3) of Figure 1 on P_b. A VDPE scheme is* private*, if no polynomial-time adversary has more than negligible probability in obtaining b.*

Definition 3. *(*`Setup`*,* `Read`*,* `Update`*,* `Eval`*,* `Verify`*) is a secure* privately *verifiable polynomial evaluation scheme if it verifies the* ***Correctness****,* ***Soundness*** *and* ***Privacy*** *requirements of Definition 2, 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 either the soundness or the privacy security games when $\mathcal{O}_1$ also plays the role of $\mathcal{O}_2$.*

In Section 5 we apply our new verifiable protocols to create a new Dynamic Proofs of Retrievability (DPoR) scheme, provably achieving correctness, soundness, and retrievability for DPoR. We follow the exact same security definition for DPoR as in [5], adapted from [61], which we will not restate here for the sake of brevity.

2.4 Assumptions

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 Definition 6. We also need computational assumptions, including the hardness to compute discrete logarithms, in Definition 4, and polynomial extensions of the hardness to produce Diffie-Hellman-like secrets even with bilinear pairings, in Definition 5.

Definition 4 (Discrete Logarithm, **DLOG**, hardness assumption [43, Def. 9.63])**.** *For a computational security parameter $\kappa \in \mathbb{N}$, a discrete-logarithm problem is hard relatively to a group $\mathbb{G}$ of group*

[1] One might also ask for *knowledge soundness* (see, e.g., [49]) on the coefficients of the outsourced polynomial, but this is easily achieved for any VDPE scheme by definition: the Client can simply interpolate from the evaluations.

order $p \geqslant 2^{2\kappa}$*, a generator* g *and a randomly sampled element of the group,* $h \xleftarrow{\$} \mathbb{G}$*, if for any probabilistic polynomial-time (ppt) algorithms* $\mathcal{A}_{DLOG}$*, there exists a negligible function* negl *such that* $\mathcal{P}r\left[\mathcal{A}_{DLOG}(\mathbb{G}, g, h) = x \text{ s.t. } h = g^x\right] \leqslant \mathsf{negl}(\kappa)$.

Definition 5 (t-Bilinear Strong Diffie-Hellman, **t-BSDH**, assumption, from [37, 42])**.** *For a computational security parameter* $\kappa \in \mathbb{N}$*, let* $\alpha \in \mathbb{Z}_p^*$*, with* $p \geqslant 2^{2\kappa}$*, 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 ppt-adversary* $\mathcal{A}_{t-BSDH}$ *and for any value of* $c \in \mathbb{Z}_p \backslash \{-\alpha\}$*, we have the probability:*

$$\mathcal{P}r\left[\mathcal{A}_{t-BSDH}(g_1, g_2, g_j^{\alpha}, g_j^{\alpha^2}, \ldots, g_j^{\alpha^t}) = \left\langle c, e(g_1; g_2)^{\frac{1}{\alpha+c}} \right\rangle\right] \leqslant \mathsf{negl}(\kappa)$$

Next is the distinct leading monomial (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 $k \geqslant 2$ and $m \geqslant 1$. 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 6 (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 2 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 2: $(2, 1, d)$-DLM security game 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 [35, 53, 3, 4, 7].

Eventually, when we use Merkle Hash Tree, we need to apply a *Collision Resistant Hash Function* (**CRHF**), so that finding different hash trees with the same root is hard.

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 start with ideas mostly from [42, 9], in order to highlight the difficulties in our setting: adding dynamicity and encryption; that is allowing to modify only parts of the inputs at a low cost, while dealing with covert inputs and preserving a proven security. More precisely, our modifications allow the adaptation of the security proof in order to incorporate the updates, and require some algorithmic tricks to preserve the linearity of the Server computations.

First, we 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:* $T_{k,P}(X) = \sum_{i=k+1}^{d} p_i X^{i-k-1} = \sum_{j=0}^{d-1-k} p_{j+k-1} X^j$.

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

$$Q_P(Y,X) = \sum_{i=1}^{d} p_i \sum_{k=0}^{i-1} Y^{i-k-1}X^k = \sum_{k=0}^{d-1} T_{k,P}(Y)X^k \tag{6}$$

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

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:

$$P(s) = z + (s-r)Q_P(s,r) \tag{7}$$

Table 3: Verifiable Ciphered Polynomial Evaluation

	Server	Communications	Client
`Setup`		$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ groups of order p pairing e, gen. $g_1, g_2, g_T = e(g_1;g_2)$	$P \in \mathbb{Z}_p[X]$, $1 \leqslant d^\circ(P) \leqslant d$; let $s \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ $W \leftarrow E_{pk}(P)$, $\mathcal{K} \leftarrow g_T^{P(s)}$, $H \leftarrow [g_2^{T_{k,P}(s)}]_{k=0}^{d-1}$
	Output : $st_{\mathcal{S}} = \{pk, \mathbb{G}_{1,2,T}, W, H\}$	W, H ⟵	$st_{\mathcal{C}} = \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, \mathcal{K}, s\}$
`Eval/Verify`	Form $x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$	r ⟵	$r \stackrel{\$}{\leftarrow} \mathbb{Z}_p$
	$\zeta = W^\intercal \boxdot x$; $\xi = H^\intercal \odot x_{0..d-1}$	ζ, ξ ⟶	$e(g_1^{s-r};\xi)g_T^{D_{sk}(\zeta)} \stackrel{?}{=} \mathcal{K}$ **Output** : $D_{sk}(\zeta)$ or **reject**

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 $H = [h_0, \ldots, h_d] = [g^{a_0}, \ldots, g^{a_d}]$, denote by $H \odot x = \prod_{i=0}^{d} h_i^{x_i} = g^{\sum a_i x_i}$ the dot-product in the exponents. Then Table 3 shows how the Server produces the evaluation via the partially homomorphic cipher and the subset polynomials (in this table, and in the following protocols presentations, time passes from top to bottom only, driven by the "Communications" column). Then this evaluation is bound to be correct by the consistency check in the exponents.

Proposition 9 (A proof is given in Appendix B)**.** *The protocol of Table 3 is correct and sound under the d-BSDH assumption.*

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[2]. 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 remainder of this paper and obtain a 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 a publicly verifiable dynamic protocol for ciphered polynomials, preserving a good efficiency while still being secure, remains an open question to us (usually when adapting a static protocol, either dynamicity involves too much recomputation or the security is compromised by the updates).

[2]Efficient updates in similar schemes are considered, e.g., in [64] but to a protocol that verifies coefficients known to the Server, *not* its evaluation at hidden coefficients

4 Outsourced dynamic verification of the evaluation

In order to be able to deal with updates, a classical tool is to add Merkle trees that are updated along with the polynomial parts. Checking the root of the Merkle tree allows for logarithmic verifications and updates of any coefficient of the polynomial. Modifications of the polynomial coefficients are also included in the Client state so that old polynomials cannot be used for the verification of `Eval`. The difficulty then is to preserve a linear time Server with a fast and light Client; we show next how to achieve this.

4.1 Merkle trees for logarithmic Client storage

In order to avoid storing the polynomial coefficients on the Client side, we thus use a Merkle hash tree [51, 45, 5]. Then it is sufficient to store the root of the Merkle tree: under the CRHF assumption, a malicious Server cannot give back different polynomial coefficients. For our purpose, an implementation of such trees must just provide the following algorithms:

- $T \leftarrow \mathrm{MTTree}(X)$ creates a Merkle hash tree from a database X.
- $r \leftarrow \mathrm{MTRoot}(X)$ computes from scratch the root of the Merkle hash tree of the whole database X.
- $L \leftarrow \mathrm{MTUncles}(i, T)$ retrieves a list of "uncle" node hashes along the path in the tree to index i.
- $r \leftarrow \mathrm{MTpathRoot}(i, a, L)$ computes the root of the Merkle hash tree from a leaf element a and the associated path of uncles L.
- $T' \leftarrow \mathrm{MTupdLeaf}(i, a, T)$ updates the whole Merkle tree T by changing the i-th leaf to be a.

The correctness requirements are that, for any index i and databases X, Y which are identical except possibly for the i'th index index (i.e., $\forall j \neq i, x_j = y_j$), we have

$$\mathrm{MTRoot}(X) = \mathrm{MTpathRoot}\left(i, x_i, \mathrm{MTUncles}\left(i, \mathrm{MTTree}(X)\right)\right) \quad (8)$$

$$\mathrm{MTTree}(X) = \mathrm{MTupdLeaf}(i, x_i, Y) \quad (9)$$

And the soundness requirement is that no P.P.T. adversary can compute a tuple (X, i, b, L) such that

$$x_i \neq b \quad \text{and} \quad \mathrm{MTRoot}(X) = \mathrm{MTpathRoot}(i, b, L). \quad (10)$$

4.2 Public dynamic unciphered polynomial evaluation

Thanks to these additional Merkle-tree operations, we can now give a protocol for the public verification of the evaluation of a dynamic polynomial P. It consists in five algorithms (`Setup`, `Read`, `Update`, `Eval`, `Verify`) detailed in Table 4 and it requires, for now, a *symmetric* pairing.

During the `Setup` algorithm, the Client sends the unciphered polynomial to the Server and deletes it to minimize its storage. The Client uses a random coin s to create some data to be published or to be sent to the Server. The Verifier collects the published data and is authorized to run the `Read` and the `Verify` algorithms. But she is not authorized to run the `Setup` and `Update` algorithms (she does not know s). At any point the Client can take the role of a Verifier. This is shown in Table 4, where the different states are as follows: $st_{\mathcal{V}} = \{\mathbb{G}, \mathbb{G}_T, g, e, \mathcal{K}_1, \mathcal{K}_2, r_P\}$, $st_{\mathcal{C}} = st_{\mathcal{V}} \cup \{s\}$ and $st_{\mathcal{S}} = \{\mathbb{G}, g, P, T_P, S\}$.

Proposition 10 (A proof is given in Appendix B)**.** *The protocol of Table 4 is correct and sound under the d-BSDH and CRHF assumptions.*

One difficulty is to preserve a linear-time Server. We show next that this is indeed possible here.

Table 4: Public and Dynamic unciphered polynomial evaluation

	Server	Communications	Client/Verifier
`Setup`	$T_P \leftarrow \text{MTTree}(P)$ **Output** : $st_{\mathcal{S}} = \{\mathbb{G}, P, T_P, S\}$	$\mathbb{G}, \mathbb{G}_T$ of order p, gen. g symm. pairing e P, S ⟵	$P \in \mathbb{Z}_p[X]$, $1 \leqslant d^\circ(P) \leqslant d$; let $s \xleftarrow{\$} \mathbb{Z}_p$ $\mathcal{K}_1 \leftarrow e\left(g^{P(s)}; g\right)$, $\mathcal{K}_2 \leftarrow g^s$, $S \leftarrow [g^{s^k}]_{k=0}^{d-1}$ $r_P \leftarrow \text{MTRoot}(P)$ $st_{\mathcal{V}} = \{\mathbb{G}, \mathbb{G}_T, g, e, \mathcal{K}_1, \mathcal{K}_2, r_P\}$, $st_{\mathcal{C}} = st_{\mathcal{V}} \cup \{s\}$
`Read`	$L_i \leftarrow \text{MTUncles}(i, P, T_P)$	i ⟵ p_i, L_i ⟶	$r_P \stackrel{?}{=} \text{MTpathRoot}(i, p_i, L_i)$ **Output** : p_i or **reject**
`Update`	$L_i \leftarrow \text{MTUncles}(i, P, T_P)$ $T_P \leftarrow \text{MTupdLeaf}(i, p'_i, T_P)$; $p_i \leftarrow p'_i$ **Output** : $st_{\mathcal{S}} = \{\mathbb{G}, P, T_P, S\}$	i, p'_i ⟵ p_i, L_i ⟶	$r_P \stackrel{?}{=} \text{MTpathRoot}(i, p_i, L_i)$, $r'_P \leftarrow \text{MTpathRoot}(i, p'_i, L_i)$ $\mathcal{K}'_1 \leftarrow \mathcal{K}_1 \cdot e\left(g^{s^i(p'_i - p_i)}; g\right)$ $st_{\mathcal{V}} = \{\mathbb{G}, \mathbb{G}_T, g, e, \mathcal{K}'_1, \mathcal{K}_2, r'_P\}$, $st_{\mathcal{C}} = st_{\mathcal{V}} \cup \{s\}$ or **reject**
`Eval/Verify`	Form $x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$ $\zeta \leftarrow P(r)$; $\xi \leftarrow \prod_{i=1}^{d} \prod_{k=0}^{i-1} S_{i-k-1}^{p_i x_k}$	r ⟵ ζ, ξ ⟶	$r \xleftarrow{\$} \mathbb{Z}_p$ $e(\xi; \mathcal{K}_2 / g^r) e(g^\zeta; g) \stackrel{?}{=} \mathcal{K}_1$ **Output** : $D(\zeta)$ or **reject**

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 5, only with respect to that degree d. The main idea is to evaluate the polynomial of Equation (6) (with a priori a quadratic number of monomials) in a Horner-like fashion, so that it requires only a linear number of operations.

Table 5: Complexity bounds for the publicly verifiable dynamic and unciphered polynomial evaluation of Table 4 for a degree d polynomial.

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

Proposition 11 (A proof is given in Appendix B)**.** *In Table 4, the setup protocol requires $\mathcal{O}(d)$ arithmetic and hashing operations; the update protocol requires $\mathcal{O}(log(d))$ arithmetic and hashing operations; the verification protocol requires $\mathcal{O}(1)$ communications and arithmetic operations for the Client, and $\mathcal{O}(d)$ arithmetic operations for the Server.*

In the next Section, we then propose a novel fully private protocol, combining and formalizing the ideas from the encrypted one and the dynamic one.

5 Fully private, dynamic and ciphered polynomial evaluation

So far we have a polynomial evaluation verification, that allows efficient updates of its coefficients. We now propose a scheme which combines 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

permanent 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.

We start the section with the security tools and then the linear algebra algorithms we will use 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 Security requirements

Here we add a secret masking of the polynomial coefficients in order to make the protocol hiding. For this we use the security hypothesis of Definition 6: indeed, DLM security states that in a bilinear 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.

5.2 Linear algebra toolbox

For the next protocol to hold, we need to adapt the difference polynomial to the matrix case. For instance Lemma 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 with the DLM masks, we need to efficiently compute geometric sums of matrices. Thanks to Fiduccia's algorithm [29], this is easily done with a number of operations logarithmic in the exponent, provided that 1 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. This is summarized in Algorithms 1 and 2 and analyzed in Appendix B.

Algorithm 1 Degree 2 modular monomial powers (2-MMP)

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

1: **if** $d == 1$ **then return** Z
2: $T \leftarrow$ 2-MMP$(\lfloor d/2 \rfloor, P)$;
3: $S \leftarrow (t_0^2 - t_1^2 p_0) + (2t_0t_1 - t_1^2 p_1)Z$; {$T(Z)^2$ modulo $P(Z)$}
4: **if** d is odd **then**
5: **return** $(-s_1p_0) + (s_0 - s_1p_1)Z$; {$Z \cdot S(Z)$ modulo $P(Z)$}
6: **else**
7: **return** S.
8: **end if**

Algorithm 2 Projected matrix geometric sum (PMGS)

Input: $k \in \mathbb{Z}$, $A = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathbb{Z}_p^{2\times2}$, 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 1}
3: **return** $(f_1A + (f_0 - 1)I_2)(A - I_2)^{-1}\beta$. {$(A^{k+1} - I_2)(A - I_2)^{-1}\beta$}

Lemma 12. *Algorithm 2, computing the matrix geometric sum, requires between* $40 + 8\lceil\log_2(d_p + 1)\rceil$ *and* $40 + 11\lceil\log_2(d_p + 1)\rceil$ *arithmetic operations.*

Proof. Counting only (modular) field operations, Algorithm 1 requires between 8 and 11 times $\lceil \log_2(d_p) \rceil$ additions and multiplications depending on the binary decomposition of d_p. Then we have 5 operations 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. □

5.3 Formalization of the protocol

The dynamic externalized polynomial evaluation scheme consist of the following algorithms `Setup`, `Read`, `Update`, `Eval` and `Verify` between a Client $\mathcal{C}$ with state $st_{\mathcal{C}}$ and the Server $\mathcal{S}$ of state $st_{\mathcal{S}}$. Following the definition of a VDPE scheme of Section 2.2, `Setup` is detailed in Algorithm 3; `Read` is detailed in Algorithm 4; `Update` is detailed in Algorithm 5; `Eval` is detailed in Algorithm 6; and `Verify` is detailed in Algorithm 7. Finally, a lighter variant of `Setup` and `Update` (detailed in Algorithm 8) dedicated to the DPoR protocol is proposed. The exchanges are summarized in Table 12 of Appendix A.

Algorithm 3 `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 $\bar{P}(X) = \sum_{i=0}^{d} X^i(p_i\alpha + \Phi^i\beta)$, $W = E_{pk}(P) = [E(p_i)]_{i=0}^{d}$, $S = [g_1^{s^k}]_{k=0}^{d-1} \in \mathbb{G}_1^d$, $\bar{H} = [g_2^{\bar{p}_i}]_{i=1}^{d} \in \mathbb{G}_2^{2\times d}$, $\bar{\mathcal{K}} = g_T^{\bar{P}(s)} \in \mathbb{G}_T^2$ and $d_p = d \bmod \varphi(p) \equiv d \bmod p-1$;
5: Client: $r_W = \mathtt{MTRoot}(W)$; {root of the Merkle tree}
6: Client: sends $pk, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e, W, S, \bar{H}$ to the Server;
7: Client: **return** $st_{\mathcal{C}} \leftarrow \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, r_W, d_p\}$;
8: Server: $T_W \leftarrow \mathtt{MTTree}(W)$; {*the Merkle tree*}
9: Server: **return** $st_{\mathcal{S}} \leftarrow \{pk, \mathbb{G}_{1,2,T}, e, W, T_W, S, \bar{H}\}$.

Algorithm 4 `Read`$(ist_{\mathcal{C}}, st_{\mathcal{S}})$

Input: $i \in [0..d]$, $(st_{\mathcal{C}}, st_{\mathcal{S}}) = \mathtt{Setup}(1^\kappa, P)$.
Output: p_i the value of the i^{th} coefficient of P.

1: Client: sends i;
2: Server: $L_i \leftarrow \mathtt{MTUncles}(i, W, T_W)$;
3: Server: sends w_i, L_i to the Client;
4: **if** $r_W \neq \mathtt{MTpathRoot}(i, w_i, L_i)$ **then**
{*the stored root does not match the received element and uncles*}
5: Client: **return** `reject`.
6: **else**
7: Client: computes $p_i = D(w_i)$;
8: **end if**

Remark 13. *We show next how to use a dynamic VPE protocol inside a DPoR scheme. There, the client updates a polynomial coefficient p_i by sending an encryption of only the difference $\delta = p_i' - p_i$ without needing to know the value of p_i. In this variant, the value of p_i does not have to be checked and the hash tree is superfluous.*

We thus consider δ`Setup`, a variant of `Setup` *where the client does not need* $\mathtt{MTRoot}(W)$ *(line 5 of Algorithm 3) and the server does not compute the tree at all (remove* $\mathtt{MTTree}(W)$*, line 8). $st_{\mathcal{C}}$ and $st_{\mathcal{S}}$ are therefore reduced to $st_{\mathcal{C}} = \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, d_p\}$ and $st_{\mathcal{S}} = \{pk, \mathbb{G}_{1,2,T}, e, W, S, \bar{H}\}$ (note that this prevents using the* `Read` *operation on the polynomial). In addition, we also consider a*

Algorithm 5 `Update`$(i, p'_i, st_{\mathcal{C}}, st_{\mathcal{S}})$

Input: $i \in [0..d]$, $p'_i \in \mathbb{Z}_p^*$, $(st_{\mathcal{C}}, st_{\mathcal{S}}) = \texttt{Setup}(1^\kappa, P)$.
Output: $(st'_{\mathcal{C}}, st'_{\mathcal{S}}) = \texttt{Setup}(1^\kappa, P + (p'_i - p_i)X^i)$ or `reject`.

1: Client: gets $st_{\mathcal{C}} = (pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, r_W, d_p)$,
2: Client: computes $w'_i = E(p'_i)$,
3: Client: computes $\bar{H}'_i \leftarrow g_2^{p'_i\alpha + \Phi^i\beta}$
4: Client: sends i, w'_i if $(i > 0)$ sends $\bar{H}'_i$;
5: Server: $L_i \leftarrow \text{MTUncles}(i, W, T_W)$;
6: Server: $T'_W \leftarrow \text{MTupdLeaf}(i, w'_i, T_W)$; {*updates the Merkle tree*}
7: Server: sends w_i, L_i to the Client;
8: Server: $st^*_{\mathcal{S}} \leftarrow st_{\mathcal{S}} \backslash \{T_W, w_i\} \bigcup \{T'_W, w'_i\}$
9: **if** $i == 0$ **then** Server: **return** $st'_{\mathcal{S}} \leftarrow st^*_{\mathcal{S}}$
else Server: **return** $st'_{\mathcal{S}} \leftarrow st^*_{\mathcal{S}} \backslash \{\bar{H}_i\} \bigcup_{j=1}^{2} \{\bar{H'}_i[j]\}$
10: **if** $r_W \neq \text{MTpathRoot}(i, w_i, L_i)$ **then**
{*the stored root does not match the received element and uncles*}
11: Client: **return** `reject`.
12: **else**
13: Client: computes $\Delta \leftarrow g_2^{(p'_i - p_i)\alpha}$;
14: Client: computes $\bar{\mathcal{K}}'[j] \leftarrow e(g_1; \Delta[j]^{s^i}) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$;
15: Client: computes $r'_W = \text{MTpathRoot}(i, w'_i, L_i)$;
16: Client: **return** $st'_{\mathcal{C}} \leftarrow st_{\mathcal{C}} \backslash \{\bar{\mathcal{K}}, r_W\} \bigcup \{\bar{\mathcal{K}}', r'_W\}$.
17: **end if**

Algorithm 6 `Eval`$(st_{\mathcal{S}}, r)$

Input: $st_{\mathcal{S}}$ and a evaluation point $r \in \mathbb{Z}_p$;
Output: ζ the encrypted evaluation of $P(r)$ and a proof $\bar{\xi}$.

1: Server: computes $\zeta = W^\intercal \boxdot x = \prod_{i=0}^{d} w_i^{(r^i \mod p)}$
{*via Equation (2)*, see also, e.g., Remark 1 and Algorithm 10}
2: Server: $\bar{\xi} = [1_{\mathbb{G}_T}, 1_{\mathbb{G}_T}]^\intercal \in \mathbb{G}_T^2$; $t = 1_{\mathbb{G}_1}$;
3: **for** $i = 1$ **to** d **do** {Following the ideas of Section 4.3}
4: Server: $t \leftarrow S_{i-1} \cdot t^r$;
5: Server: $\bar{\xi}[j] \leftarrow \bar{\xi}[j] \cdot e(t; \bar{H}_i[j])$ for $j = 1..2$;
6: **end for**
7: Server: **return** $\zeta, \bar{\xi}$.

Algorithm 7 `Verify`$(st_{\mathcal{C}}, r, \zeta, \bar{\xi})$

Input: $st_{\mathcal{C}}$, the evaluation point $r \in \mathbb{Z}_p$, its encrypted evaluation ζ and a proof $\bar{\xi}$;
Output: $z = P(r)$ or `reject`.

1: Client: computes $r\Phi$ and $c \leftarrow \left((r\Phi)^{d_p+1} - I_2\right) \cdot (r\Phi - I_2)^{-1} \cdot \beta$ {via Algorithm 2}
2: Client: computes $z = D_{sk}(\zeta) \mod p$;
3: **if** $\bar{\xi}[j]^{s-r} g_T^{z\alpha[j]+c[j]} = \bar{\mathcal{K}}[j]$ for $j = 1..2$ **then**
4: Client: **return** z.
5: **else**
6: Client: **return** `reject`.
7: **end if**

variant of the `Update` *algorithm, which takes* $\delta = p'_i - p_i$ *as input instead of* p_i*, as detailed in Algorithm 8. The corresponding dynamic (from the difference) externalized polynomial evaluation scheme is then reduced to the algorithms* δ`Setup`, δ`Update`, `Eval` *and* `Verify`.

We have now our main result for the Dynamic Verified Evaluation of Secret Polynomials.

Theorem 14 (A proof is given in Appendix B)**.** *Under the d-BSDH, DLOG, CRHF and DLM security assumptions of Section 2, the protocol composed of Algorithms 3 to 8 (summarized in Table 12) is a fully*

Algorithm 8 δ`Update`$(i, \delta, st_{\mathcal{C}}, st_{\mathcal{S}})$

Input: $i \in [0..d]$, $\delta \in \mathbb{Z}_p^*$, $(st_{\mathcal{C}}, st_{\mathcal{S}}) =$ `Setup`$(1^\kappa, P)$.
Output: $(st'_{\mathcal{C}}, st'_{\mathcal{S}}) =$ `Setup`$(1^\kappa, P + \delta X^i)$ or `reject`.

1: Client: computes $e_\delta = E_{pk}(\delta)$, $\Delta = g_2^{\delta\alpha}$;
2: Client: sends i, e_δ, Δ to the Server;
3: Server: sends w_i to the Client;
4: **if** $i == 0$ **then** Server: **return** $st'_{\mathcal{S}} \leftarrow st^*_{\mathcal{S}}$
else Server: **return** $st'_{\mathcal{S}} \leftarrow st^*_{\mathcal{S}} \backslash \{\bar{H}_i\} \bigcup_{j=1}^{2} \{\bar{H}_i[j] \cdot \Delta[j]\}$
5: Client: computes $\bar{\mathcal{K}}'[j] \leftarrow e(g_1; \Delta[j]^{s^i}) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$;
6: Client: **return** $st'_{\mathcal{C}} \leftarrow st_{\mathcal{C}} \backslash \{\bar{\mathcal{K}}\} \bigcup \{\bar{\mathcal{K}}'\}$.

secure verifiable polynomial evaluation scheme, as defined in Definition 2 and the complexity bounds of its algorithms are given in Table 6.

Table 6: Complexity bounds for verifiable dynamic and ciphered polynomial evaluation (function of the degree d of the polynomial, for groups of supposed constant cardinality: number of group elements/arithmetic operations).

		Server	Communication	Client
	Storage	$\mathcal{O}(d)$		$\mathcal{O}(1)$
Comput.	`Setup`	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$
	`Read`	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$
	`Update`	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$	$\mathcal{O}(\log(d))$
	δ`Update`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
	`Eval/ Verify`	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

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.

5.4 Experiments

To assess the efficiency of our protocol, we implemented Table 12 using the following libraries[3]: `gmp-6.2.1` for modular operations, `fflas-ffpack-2.4.3` for linear algebra, `relic-0.6.0` for Paillier's cryptosystem and pairings (we used a "bn-p254" pairing), `libsnark` (commit `2af4402`) for baseline polynomial evaluation verification. Our source code to perform these experiments is available via the following GitHub repository: https://github.com/jgdumas/vespo.

To observe the effect of the chosen homomorphic systems (Paillier with an RSA modulus size of 2048 bits and the pairing), we ran the experiments, on a single core of an intel Gold 6126 2.6GHz for the Client and Horner computations and on one or four cores for the Server (the parallelization of the prefix-like Server part of Algorithm 6 is given in Appendix D). 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 and of an unciphered static polynomial evaluation with a SNARK (a ciphered evaluation with these SNARK would require to arithmetize the Homomorphic cryptosystem and seems still out of reach).

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. Second, for the protocol itself, we see that both homomorphic evaluations of the Server are quite similar, even if

[3] https://gmplib.org, https://linbox-team.github.io/fflas-ffpack, https://github.com/scipr-lab/libsnark.git, 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 12 (column 'pows' is the time to perform the lhs exponentiations (by $s - r$ and by $D(\zeta)\alpha[j] + c[j]$); column 'c' times the matrix geometric sum; and column 'D' times the single Paillier's deciphering; below are some baseline comparisons: 'Horner' is a witness direct evaluation in that group, '`libsnark`' is an unciphered and static polynomial evaluation verification. Each experiment was performed 11 times and we report the median value, with a maximum variance lower than 16.4% between runs).

Degree	Server Certification				Client Verification		
	1 core		4 cores		1 core		
	ζ	ξ	ζ	ξ	D	c	pows
256	0.12s	0.08s	0.04s	0.03s	0.9ms	<0.1ms	0.7ms
512	0.24s	0.15s	0.07s	0.05s			
1024	0.48s	0.30s	0.13s	0.10s			
2048	0.95s	0.61s	0.26s	0.18s			
4096	1.90s	1.22s	0.51s	0.35s			
8192	3.82s	2.44s	1.01s	0.70s			
16384	7.58s	4.87s	2.02s	1.40s			
32768	15.24s	9.78s	4.05s	2.75s			
65536	30.55s	19.58s	8.06s	5.45s			
131072	60.82s	39.02s	16.15s	10.90s			

	Client	Server (1 core)				Proof
	1 core	d°	256	1024	8192	131072	size
Horner (no verif., no crypt.)		<0.1ms	0.2ms	1.6ms	32.0ms	-
`libsnark` (no crypt.)	3.8ms		0.04s	0.12s	0.74s	10.57s	287B
Here (v. & c. & dyn.)	1.6ms		0.20s	0.78s	6.26s	99.84s	320B

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 in practice potentially non-constant in the degree is by far the most negligible). Third our Client is even faster than an unciphered one (we use less pairings than `libsnark`) and for a large enough degree, we can observe the Client time to win over the linear time pure polynomial evaluation. Also, our ciphered Server slowdown remains within a factor close to four (or only two without Paillier) when compared to the static and unciphered one.

6 Low Server storage dynamic PoR

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. The dynamic PoR schemes of [22, 61] 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; 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 $\mathcal{O}(\sqrt{N})$ 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.

Table 8: Attributes of some selected DPoR schemes

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

Table 9: Private verifiable Client/Server DPoR protocol with low storage Server

	Server	Communications	Client
`Init`			**Input**: p prime, cryptosystem E/D **Input**: $M \in \mathbb{Z}_p^{m\times n}$ $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$, $u^\intercal \leftarrow [\gamma^i]_{i=0}^{m-1} \in \mathbb{Z}_p^m$ $v^\intercal \leftarrow u^\intercal M \in \mathbb{Z}_p^n$
		$\overline{Vect\mathbf{2}Poly(v)}^4$	
		δ`Setup`	
	$st_{\mathcal{S}}$		$st_{\mathcal{C}}$
		M	
	$T_M \leftarrow \text{MTTree}(M)$ **Output:** $st_{\mathcal{S}}, M, T_M$		$r_M \leftarrow \text{MTRoot}(M)$ **Output:** $st_{\mathcal{C}}, \gamma, r_M$
`Write`	$L_{M_{ik}} \leftarrow \text{MTUncles}(k + i\cdot n, M, T_M)$	i, k	**Input:** i, k, M'_{ik}
		$M_{ik}, L_{M_{ik}}, w_k$	$r_M \overset{?}{=} \text{MTpathRoot}(k + i\cdot n, M_{ik}, L_{M_{ik}})$
			$\delta \leftarrow \gamma^i(M'_{ik} - M_{ik})$
	$st_{\mathcal{S}}$	δ`Update`	$\bar{\delta}, st_{\mathcal{C}}$
	$st_{\mathcal{S}}$		$st_{\mathcal{C}}$
		M'_{ik}	
	$M_{ik} \leftarrow M'_{ik}$, $T_M \leftarrow \text{MTupdLeaf}(k + i\cdot n, M'_{ik}, T_M)$ **Output:** $st_{\mathcal{S}}, M, T_M$		$r_M \leftarrow \text{MTpathRoot}(k + i\cdot n, M'_{ik}, L_{M_{ik}})$ **Output:** $st_{\mathcal{C}}, r_M$ or `reject`
`Audit`	form $x \leftarrow [r^k]_{k=0}^{n-1\intercal} \in \mathbb{Z}_p^n$, then $y \leftarrow Mx$	r	$r \xleftarrow{\$} \mathbb{Z}_p^*$ s.t. $(r\Phi - I_2) \in GL_2(\mathbb{Z}_p)$
	$st_{\mathcal{S}}, r$	$y, \zeta, \bar{\xi}$	$st_{\mathcal{C}}, r, \zeta, \bar{\xi}$
	`Eval`		`Verify`
	$\zeta, \bar{\xi}$		$D_{sk}(\zeta)$ *or* `reject`
			$u^\intercal y \overset{?}{=} D_{sk}(\zeta)$
			Output: `accept` or `reject`

An easy argument demonstrates that our $\mathcal{O}(\log N)$ Client cost for audits is optimal. If each audit has $o(\log N)$ cost (and thus transcript size), 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.

6.1 Matrix based approach for audits

Here we summarize the DPoR of [5] upon which our new scheme is based. The 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 an `Init` 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 (`Read`, and `Write` protocols in the DPoR case), 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 $\mathcal{O}(\sqrt{N})$ total communication and Client computation when optimizing the matrix dimensions to roughly $m = n = \sqrt{N}$.

Table 10: Modification of the DPoR audit protocol, with 254-bits groups, 2048-bits Paillier, on a Gold 6126 2.6GHz & 10 GB/core (real time are median values for a single run; each experiment was performed 11 times; 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 & Tables 5-6-7][5]				
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 (1/12 cores)	0.29s/0.04s	2.68s/0.30s	29.04s/3.36s	219.7s/41.48s
Communications	169KB	535KB	1 693KB	5 418KB
Client Audit (1 core)	0.6ms	1.7ms	5.3ms	18.3ms
Square Dynamic-ciphered delegated polynomial evaluation with 254-bits groups of Table 9[6]				
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 (1/12 cores): matrix-vector step	1.1s/0.2s	11.3s/1.3s	113.4s/12.9s	1 152.5s/131.1s
Server Audit (1/12 cores): polynomial step	4.4s/0.5s	13.5s/1.4s	42.6s/4.2s	141.7s/13.4s
Communications	181KB	571KB	1 803KB	5 770KB
Client Audit (1 core): dotproduct step	3.2ms	8.4ms	13.1ms	37.9ms
Client Audit (1 core): polynomial step	1.7ms	1.7ms	1.7ms	1.7ms
Rectangular Dynamic-ciphered delegated polynomial evaluation with 254-bits groups of Table 9[6]				
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 (1/12 cores): matrix-vector step	1.1s/0.2s	11.3s/1.3s	113.2s/12.8s	1 147.9s/130.7s
Server Audit (1/12 cores): polynomial step	3.8s/0.4s	35.5s/3.6s	324.1s/30.6s	3 064.8s/283.6s
Communications	205KB	226KB	246KB	267KB
Client Audit (1 core): dotproduct step	3.7ms	4.0ms	4.4ms	4.8ms
Client Audit (1 core): polynomial step	1.7ms	1.7ms	1.7ms	1.7ms

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 $\mathcal{O}(\sqrt{N})$.

[4] Converts the vector v into the polynomial $P(x) = \sum_{i=0}^{n-1} v_i x^i$.

6.2 Bootstrapping Client via VESPo

Now we show how to modify the reduced communication version of the DPoR 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 entire 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 12. The obtained private-verification DPoR protocol, combining that of [5] with our ciphered polynomial evaluation in Section 5, is presented in Table 9.

Theorem 15 (A proof is given in Appendix B)**.** *The protocol of Table 9 is correct and sound under the d-BSDH, DLOG, CRHF and DLM security assumptions.*

6.3 Experiments

We now compare our modification of the DPoR protocol with the one in [5]. 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 DPoR 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 `Write` phase in two phases, the remaining linear algebra phase and the new polynomial evaluation phase (δ`Update`). 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 (more limited losses in the more realistic case where the Server can use multiple cores). In any case, 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.

7 Conclusion

We have presented a protocol verifying publicly a dynamic unciphered polynomial evaluation and then a protocol verifying privately a dynamic ciphered polynomial evaluation. Now, combining efficient and proven dynamicity for ciphered polynomial with public verifiability raises security issues and reminds an open problem. Still, we have also 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 in turn 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.

[5] https://github.com/dsroche/la-por
[6] https://github.com/jgdumas/vespo

Acknowledgments

We thank Gaspard Anthoine for providing us with some preliminary comparisons with the PBC and libpaillier libraries and Anthony Martinez for the libsnark baseline benchmarking. We thank Jean-Louis Roch for fruitful exchanges about the parallelization of the Server side and for pointing out [62]. Finally we thank the anonymous referees who greatly helped improve the paper. This material is based on work supported in part by the Agence Nationale pour la Recherche under Grants ANR-21-CE39-0006 Sangria and ANR-15-IDEX-0002.

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. Springer, 2015. 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, *SCN 2020, Amalfi, Italy, September 14-16*, 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. Springer, 2016. 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 *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, 2020. doi:10.1007/978-3-030-64840-4_14.

[8] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *ICALP 2018, July 9-13, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

[9] 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.

[10] 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.

[11] D. J. Bernstein, N. Heninger, P. Lou, and L. Valenta. Post-quantum RSA. In T. Lange and T. Takagi, editors, *PQCrypto 2017, Utrecht, The Netherlands*, volume 10346 of *LNCS*, pages 311–329. Springer, 2017. doi:10.1007/978-3-319-59879-6_18.

[12] R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, and Y. Zhang. *Ligero++: A New Optimized Sublinear IOP*, pages 2025–2038. ACM, New York, NY, USA, 2020. URL: https://doi.org/10.1145/3372297.3417893.

[13] 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.

[14] A. Bois, I. Cascudo, D. Fiore, and D. Kim. Flexible and efficient verifiable computation on encrypted data. In J. A. Garay, editor, *PKC 2021, May 10-13*, volume 12711 of *LNCS*, pages 528–558. Springer, 2021. doi:10.1007/978-3-030-75248-4_19.

[15] 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.

[16] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014. doi:10.1145/2633600.

[17] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, apr 1974. doi:10.1145/321812.321815.

[18] 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, *ProvSec 2017, Provable Security*, pages 487–506, Cham, 2017. Springer. doi:10.1007/978-3-319-68637-0_29.

[19] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018. doi:10.1109/SP.2018.00020.

[20] 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, Nov. 29–Dec. 3*, volume 9453 of *LNCS*, pages 262–288. Springer, 2015. doi:10.1007/978-3-662-48800-3_11.

[21] D. Cash, A. Küpçü, and D. Wichs. Dynamic proofs of retrievability via oblivious RAM. *J. Cryptol.*, 30(1):22–57, Jan. 2017. doi:10.1007/s00145-015-9216-2.

[22] 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.

[23] 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.

[24] I. Chillotti, D. Ligier, J. Orfila, and S. Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Singapore*, volume 13092 of *LNCS*, pages 670–699. Springer, 2021. doi:10.1007/978-3-030-92078-4_23.

[25] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Sofia, Bulgaria*, volume 9056 of *LNCS*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

[26] J.-G. Dumas. Proof-of-work certificates that can be efficiently computed in the cloud. In V. P. Gerdt, W. Koepf, W. M. Seiler, and E. V. Vorozhtsov, editors, *CASC'18, Lille, France*, volume 11077 of *LNCS*, pages 1–17. Springer, 2018. doi:10.1007/978-3-319-99639-4_1.

[27] 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.

[28] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. *ACM Trans. Inf. Syst. Secur.*, 17(4):15:1–15:29, Apr. 2015. doi:10.1145/2699909.

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

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

[31] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *CCS*, pages 844–855. ACM, 2014. doi:10.1145/2660267.2660366.

[32] D. Fiore, A. Nitulescu, and D. Pointcheval. Boosting verifiable computation on encrypted data. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020*, volume 12111 of *LNCS*, pages 124–154. Springer, 2020. doi:10.1007/978-3-030-45388-6_5.

[33] L. Fousse, P. Lafourcade, and M. Alnuaimi. Benaloh's dense probabilistic encryption revisited. In A. Nitaj and D. Pointcheval, editors, *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.

[34] 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.

[35] 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. Springer, 2016. doi:10.1007/978-3-662-49890-3_1.

[36] 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.

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

[38] J. Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326. Springer, 2016.

[39] S. Halevi and V. Shoup. Bootstrapping for helib. *J. Cryptol.*, 34(1):7, 2021. doi:10.1007/s00145-020-09368-7.

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

[41] W. Kahan and R. J. Fateman. Symbolic computation of divided differences. *SIGSAM Bull.*, 33(2):7–28, 1999. doi:10.1145/334714.334716.

[42] 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*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

[43] 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.

[44] 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, *SCN 2018, Amalfi, Italy, September 5-7*, volume 11035 of *LNCS*, pages 544–562. Springer, 2018. doi:10.1007/978-3-319-98113-0_29.

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

[46] J. Lavauzelle and F. L. dit Vehel. New proofs of retrievability using locally decodable codes. In *2016 IEEE ISIT*, pages 1809–1813, July 2016. doi:10.1109/ISIT.2016.7541611.

[47] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In K. Nissim and B. Waters, editors, *TCC 2021, Raleigh, NC, USA, November 8-11*, volume 13043 of *LNCS*, pages 1–34. Springer, 2021. doi:10.1007/978-3-030-90453-1_1.

[48] 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, *ICALP 2016, July 11-15, Rome, Italy*, volume 55 of *LIPIcs*, pages 30:1–30:14. Dagstuhl, 2016. doi:10.4230/LIPIcs.ICALP.2016.30.

[49] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptol.*, 16(3):143–184, 2003. doi:10.1007/s00145-002-0143-7.

[50] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *CCS 2019, London, UK, November 11-15*, pages 2111–2128. ACM, 2019. doi:10.1145/3319535.3339817.

[51] 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.

[52] Microsoft Research. SEAL (release 4.0). https://github.com/Microsoft/SEAL, Mar. 2022.

[53] 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, 2016. doi:10.1007/978-3-662-53887-6_27.

[54] 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.

[55] 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.

[56] Protocol Labs. Filecoin: A decentralized storage network, July 2017. URL: https://filecoin.io.

[57] C. Ràfols and A. Zapico. An algebraic framework for universal and updatable snarks. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *LNCS*, pages 774–804. Springer, 2021. doi:10.1007/978-3-030-84242-0_27.

[58] J. Roch, D. Traoré, and J. Bernard. On-line adaptive parallel prefix computation. In W. E. Nagel, W. V. Walter, and W. Lehner, editors, *Euro-Par 2006, Dresden, Germany, Aug. 28 - Sep. 1*, volume 4128 of *LNCS*, pages 841–850. Springer, 2006. doi:10.1007/11823285_88.

[59] F. Sebé, J. Domingo-Ferrer, A. Martínez-Ballesté, Y. Deswarte, and J.-J. Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Transactions on Knowledge and Data Engineering*, 20:1034–1038, 2008. doi:10.1109/TKDE.2007.190647.

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

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

[62] M. Snir. Depth-size trade-offs for parallel prefix computation. *J. Algorithms*, 7(2):185–201, 1986. doi:10.1016/0196-6774(86)90003-9.

[63] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *28th ACSAC*, pages 229–238, 2012. doi:10.1145/2420950.2420985.

[64] 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, *SCN 2020, Amalfi, Italy, September 14-16*, volume 12238 of *LNCS*, pages 45–64. Springer, 2020. doi:10.1007/978-3-030-57990-6_3.

[65] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, January 2015. doi:10.1145/2641562.

A Overview of VESPo exchanges

We here recall in Table 12, the summary of the exchanges of Algorithms 3 to 8. This gives an overview of our verifiable & dynamic evaluation of ciphered polynomials.

With this summary we can refine in Table 11 the results of Table 6 if we are using Paillier for the LHE (a Paillier encryption is 1 modular exponentiation and 3 modular multiplications, a Paillier decryption is 1 exponentiation and 1 multiplication, homomorphic multiplication is an exponentiation and homomorphic addition is a multiplication; then we approximate Algorithm 1 with 6 exponentiationsand 40 modular operations and we approximate the application of the pairing bilinear map with 1 exponentiation).

Table 11: Dominant terms in operations counts for Table 12 using Paillier (a value of x approximates in fact $x + o(x)$; then "Hash" counts calls to the cryptographic hash function, "mexp" is for modular exponentiations, "group" is for the other arithmetic operations).

Alg.	Server			Client		
	group	mexp	Hash	group	mexp	Hash
3	0	0	$2d$	$17d$	$6d$	$2d$
4	0	0	0	1	1	$\lceil \log_2(d) \rceil$
5	0	0	$\lceil \log_2(d) \rceil$	18	16	$2\lceil \log_2(d) \rceil$
8	3	0	0	8	8	0
6/7	$3d$	$4d$	0	52	11	0

B Proofs of the propositions and theorems

Now, we give the proofs of the propositions in Sections 3 and 4 and of our main theorems for our private and dynamic ciphered polynomials evaluation protocol and our low Server storage and audit complexity DPoR.

Proposition 9 (From page 9)**.** *The protocol of Table 3 is correct and sound under the d-BSDH assumption.*

Proof. **Correctness**. First, $\zeta = W^\intercal \boxdot x = \prod_{i=0}^{d} E(p_i)^{(r^i)} = E(P(r))$. Then, second, $\xi = H^\intercal \odot x = \prod_{k=0}^{d-1} g_2^{T_{k,P}(s)r^k} = g_2^{Q_P(s,r)}$, by Lemma 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 (7).

Table 12: Private & Dynamic, Ciphered polynomial evaluation, summarizing Algorithms 3 to 8.

	Server	Communications	Client
`Setup` Alg. 3		$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ groups of order p pairing e to $\mathbb{G}_T$, gen. $g_1, g_2, g_T = e(g_1; g_2)$	$P \in \mathbb{Z}_p[X]$, $1 \leqslant d^\circ(P) \leqslant d$ $s \xleftarrow{\$} \mathbb{Z}_p \backslash \{0,1\}$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^2$, $\Phi \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, s.t. $(s\Phi - I_2) \in GL_2(\mathbb{Z}_p)$ Let $\bar{P}(X) \leftarrow \sum_{i=0}^{d} X^i(p_i\alpha + \Phi^i\beta)$ $W \leftarrow E_{pk}(P)$, $S \leftarrow [g_1^{s^k}]_{k=0}^{d-1} \in \mathbb{G}_1^d$ $\bar{\mathcal{K}} \leftarrow g_T^{\bar{P}(s)} \in \mathbb{G}_T^2$, $\bar{H} \leftarrow [g_2^{p_i}]_{i=1}^{d} \in \mathbb{G}_2^{2\times d}$
	$T_W \leftarrow \texttt{MTTree}(W)$ **Output** : $st_{\mathcal{S}} = \{pk, \mathbb{G}_{1,2,T}, e, W, T_W, S, \bar{H}\}$	$W, \bar{H}, S$ ⟵	$d_p \leftarrow d \mod \varphi(p)$, $r_W \leftarrow \texttt{MTRoot}(W)$ $st_{\mathcal{C}} = \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, r_W, d_p\}$
`Read` Alg. 4		i ⟵	
	$L_i \leftarrow \texttt{MTUncles}(i, W, T_W)$	w_i, L_i ⟶	$r_W \overset{?}{=} \texttt{MTpathRoot}(i, w_i, L_i)$ **Output** : $p_i \leftarrow D_{sk}(w_i)$ or **reject**
`Update` Alg. 5		$i, w_i', if\ (i > 0)\ \bar{H}_i'$ ⟵	$w_i' \leftarrow E_{pk}(p_i')$, $\bar{H}_i' \leftarrow g_2^{p_i'\alpha + \Phi^i\beta}$
	$L_i \leftarrow \texttt{MTUncles}(i, W, T_W)$ $T_W \leftarrow \texttt{MTupdLeaf}(i, w_i', T_W)$ $w_i \leftarrow w_i'$ **Output** : $st_{\mathcal{S}} = \{pk, \mathbb{G}_{1,2,T}, e, W, T_W, S, \bar{H}\}$	w_i, L_i ⟶	$r_W \overset{?}{=} \texttt{MTpathRoot}(i, w_i, L_i)$ $r_W \leftarrow \texttt{MTpathRoot}(i, w_i', L_i)$ $\Delta \leftarrow g_2^{(p_i' - p_i)\alpha}$, $\bar{\mathcal{K}}[j] \leftarrow e(g_1, \Delta[j]^{s^i}) \cdot \bar{\mathcal{K}}[j]$ $st_{\mathcal{C}} = \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, r_W, d_p\}$ or **reject**
`δUpdate` Alg. 8	If $i > 0, \bar{H}_i'[j] \leftarrow \Delta[j] \cdot \bar{H}_i[j]$ for $j = 1..2$ $w_i \leftarrow w_i \cdot e_\delta$ **Output** : $st_{\mathcal{S}} = \{pk, \mathbb{G}_{1,2,T}, e, W, S, \bar{H}\}$	i, e_δ, Δ ⟵	$e_\delta \leftarrow E_{pk}(\delta)$, $\Delta \leftarrow g_2^{\delta\alpha}$ $\bar{\mathcal{K}}[j] \leftarrow e(g_1; \Delta[j]^{s^i}) \cdot \bar{\mathcal{K}}[j]$ $st_{\mathcal{C}} = \{pk, sk, \mathbb{G}_{1,2,T}, g_{1,2,T}, e, s, \alpha, \beta, \Phi, \bar{\mathcal{K}}, d_p\}$
`Eval/Verify` Alg. 6/7	Form $x \leftarrow [1, r, r^2, \ldots, r^d]^\intercal$ $\zeta \leftarrow W^\intercal \boxdot x$	r ⟵	For $r \in \mathbb{Z}_p$ s.t. $(r\Phi - I_2) \in GL_2(\mathbb{Z}_p)$ $c \leftarrow ((r\Phi)^{d_p+1} - I_2)(r\Phi - I_2)^{-1}\beta$
	$\bar{\xi}[j] \leftarrow \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(S_{i-k-1}; \bar{H}_i[j])^{x_k}$ for $j = 1..2$	$\zeta, \bar{\xi}$ ⟶	$\bar{\xi}[j]^{s-r} g_T^{D_{sk}(\zeta)\alpha[j] + c[j]} \overset{?}{=} \bar{\mathcal{K}}[j]$ for $j = 1..2$ **Output** : $D_{sk}(\zeta)$ or **reject**

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 and suppose that there exists an attack to the `Audit` protocol.

Let $[p_0, \ldots, p_t] \xleftarrow{\$} \mathbb{Z}_p^{t+1}$ for a degree t polynomial and $d = t$. Then compute directly $W = E(P)$, $T_{k,P} = \sum_{i=k+1}^{t} p_i Y^{i-k-1} = \sum_{j=0}^{t-1-k} t_{k,j} Y^j$ and homomorphically compute:

$$\mathcal{K} = e\left(g_1; \left\langle g_2, g_2^s, g_2^{s^2}, \ldots, g_2^{s^t} \right\rangle \odot [p_0, \ldots, p_t]\right),$$

together with $H = [h_k]$, where $h_k = \left\langle g_2, g_2^s, g_2^{s^2}, \ldots, g_2^{s^{t-1-k}} \right\rangle \odot [t_{k,0}, \ldots, t_{k,t-1-k}]$. These inputs are indistinguishable from a generic setup of the protocol of Table 3 and can thus be given to its attacker.

Finally, select a random evaluation point r and compute (ζ, ξ). The supposition is that an attacker of the `Audit` 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 $e(g_1^{s-r}; \xi) g_T^{D(\zeta)} = \mathcal{K}$ and $e(g_1^{s-r}; \xi') g_T^{D(\zeta)} = \mathcal{K}$. Therefore, if $r \neq s$, then $e(g_1^{s-r}; \xi) = e(g_1^{s-r}; \xi')$ 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 $(e(g_1; \xi)/(e(g_1; \xi'))^{s-r} = g_T^{D(\zeta') - D(\zeta)}$ and therefore: $\left(\frac{e(g_1;\xi)}{e(g_1;\xi')}\right)^{\frac{1}{D(\zeta')-D(\zeta)}} = g_T^{\frac{1}{s-r}}$. 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. □

From this proof, one can see that using a decipherable partially homomorphic function for the coefficients of P is required for the soundness (otherwise one could not compute the exponentiation on ξ/ξ').

Proposition 10 (From page 10)**.** *The protocol of Table 4 is correct and sound under the d-BSDH and CRHF assumptions.*

Proof. **Correctness.** First, (8) gives the correctness of `Read`. For `Update`, (9) provides the correctness of the hash tree. Then, with $\delta = p'_i - p_i$, 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. First for the `Read`/`Update` parts. Suppose an attacker can provide $p'_i \neq p_i$ that passes the Merkle root check. This would violate the soundness property of Equation (10), which is derived from the collision resistance of the underlying hash function.

Second, for the `Eval`/`Verify` parts. 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, just set $d = t$ and then 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 4. For any number of update phase, randomly select p'_i (or 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 `Eval` 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. □

Proposition 11 (From page 11)**.** *In Table 4, the setup protocol requires $\mathcal{O}(d)$ arithmetic and hashing operations; the update protocol requires $\mathcal{O}(log(d))$ arithmetic and hashing operations; the verification protocol requires $\mathcal{O}(1)$ communications and arithmetic operations for the Client, and $\mathcal{O}(d)$ arithmetic operations for the Server.*

Proof. The setup phase requires the Client to perform one polynomial evaluation and d exponentiations for $O(d)$ arithmetic operations, together with the computation of the Merkle tree on both sides, for $O(d)$ hashing operations.

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 [41]: 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 therefore $t_i = S_{i-1}(S_{i-2} \ldots (S_2(S_1 S_0^r)^r)^r \ldots)^r = \prod_{k=0}^{i-1} S_{i-k-1}^{r^k}$. Thus one can use the following Algorithm 9 to compute ξ.

Computing ξ then requires at most $2d$ exponentiations and $2d$ multiplications. □

Theorem 14 (From page 14)**.** *Under the d-BSDH, DLOG, CRHF and DLM security assumptions of Section 2, the protocol composed of Algorithms 3 to 8 (summarized in Table 12) is a fully secure verifiable polynomial evaluation scheme, as defined in Definition 2 and the complexity bounds of its algorithms are given in Table 6.*

Algorithm 9 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.

Proof. First of all, we have that:

$$\begin{cases} H'_i = H_i.\Delta \\ w'_i = w_i.e_\delta \end{cases} \Leftrightarrow \begin{cases} \Delta = H'_i.H_i^{-1} \\ e_\delta = w'_i.w_i^{-1} \end{cases}$$

Therefore, it is equivalent to consider the protocols using only Algorithm 5 or only Algorithm 8 or any combinations of both. Also, the `Read` part is identical to that of Table 4 and so are the associated security proofs.

Correctness. For the `Update` operation, $\bar{P}'(s) = \bar{P}'(s \cdot I_2) = (p'_i - p_i)s^i\alpha + \bar{P}(s \cdot I_2)$ and $e(g_1; g_2^{\bar{P}'(s \cdot I_2)[j]}) = e(g_1; g_2^{s^i(p'_i - p_i)\alpha[j]}) \cdot e(g_1; g_2^{\bar{P}(s \cdot I_2)[j]}) = e(g_1; (\bar{H}'_i[j] \cdot \bar{H}_i[j]^{-1})^{s^i}) \cdot g_T^{\bar{P}(s)[j]} = e(g_1; (\bar{H}'_i[j] \cdot \bar{H}_i[j]^{-1})^{s^i}) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$. Finally, We use the left hand side of Lemma 8 and Equation (7). Applied to $\bar{P}$, this is: $\bar{\xi} = \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(S_{i-k-1}; \bar{H}_i)^{x_k} = \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(g_1^{s^{i-k-1}}; g_2^{\bar{P}_i})^{r^k}$ so that $\bar{\xi} = e(g_1; g_2)^{Q_{\bar{P}}(s \cdot I_2, r \cdot I_2)}$. 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 verification in `Eval`/`Verify` is indeed that $g_T^{Q_{\bar{P}}(s \cdot I_2, r \cdot I_2)(s-r) + \bar{P}(r \cdot I_2)} \stackrel{?}{=} g_T^{\bar{P}(s \cdot I_2)} = g_T^{\bar{P}(s)}$.

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 `Eval`/`Verify` 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 `Verify`, thanks to Algorithm 9. For the Client, `Update` requires $O(\log(d))$ arithmetic operations to check the uncles and to compute the exponentiation s^i and Φ^i, together with a constant number of other arithmetic operations, independent of the degree. Similarly, computing $(r\Phi)^{d_p+1}$ also requires $O(min\{\log(d), \log(p)\})$ classical arithmetic operations thanks to Algorithm 2. This is $\mathcal{O}(1)$ if p is considered constant and the rest is also a constant number of operations that are independent of the degree.

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. For the setup phase, randomly select α, β, Φ and $[p_0, \ldots, p_t]$. Then compute $W = E(P)$, $\bar{H} = g_2^{\bar{P}}$, and let $S = \left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^t} \right\rangle$. Finally homomorphically compute:

$$\bar{\mathcal{K}} = e\left(\left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^t} \right\rangle \odot [\bar{p}_0, \ldots, \bar{p}_t]; g_2 \right).$$

These inputs are indistinguishable from random inputs to the protocol of Table 12. For any number of update phases, randomly select i and p'_i and compute $w'_i \leftarrow E(p'_i)$, $\bar{H}'_i \leftarrow g_2^{p'_i\alpha + \Phi^i\beta}$ and $\Delta = g_2^{\delta\alpha}$. Also compute $\mathcal{K}' = e(S_i^{(p'_i - p_i)\alpha}; g_2) \cdot \mathcal{K}$. Finally, select a random evaluation point r, compute $(\zeta, \bar{\xi})$ and call an attacker of the `Eval` 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, on 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 $B = g_2^b$ 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 $W = E(P)$, $\bar{H}_i = g_2^{\alpha p_i} B^{\Phi^i [b_1,b_2]^\intercal}$, $S = \left\langle g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^t} \right\rangle$, and $\bar{\mathcal{K}} = e(g_1; g_2^{\alpha P(s)} B^{G(s\Phi)[b_1,b_2]^\intercal})$. These inputs are indistinguishable from random inputs to the protocol of Table 12. For any update phase, randomly select i and p'_i and compute $w'_i \leftarrow E(p'_i)$, $\bar{H}'_i \leftarrow g_2^{p'_i \alpha + \Phi^i \beta}$ and $\Delta = g_2^{\alpha\delta}$. Also compute $\bar{\mathcal{K}}'[j] = e(g_1; \Delta^{s^i}) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$. Such updates are indistinguishable from random updates to the protocol of Table 12. Randomly select any number of evaluation points r and run the associated `Eval` 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. Definition 6, the parameter $\bar{H}_i$, or more precisely, the pair $(E(p_i), g_2^{p_i \alpha + \Phi^i \beta})$, is indistinguishable from $(E(p_i), g_2^{p_i \alpha + \Gamma_i})$ for some random 2-dimensional vectors Γ_i. Therefore the protocol of Table 12 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 $\Gamma_i = \log_{g_2}(h_i) - (\omega + u_i)\alpha \in \mathbb{Z}_p^2$ exists, but remains unknown) for $i = 1..d$. Randomly select s and compute $\bar{\mathcal{K}} = e(g_1; H \odot [1, s, \ldots, s^d])$. For any number of updates, randomly select p'_i, compute $w'_i \leftarrow E(p'_i)$ 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 $\Delta = g_2^{\delta\alpha}$ and $\bar{\mathcal{K}}'[j] = e(g_1; \Delta[j]^{s^i}) \cdot \bar{\mathcal{K}}[j]$ for $j = 1..2$. Alternatively run such updates with random `Eval` 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. □

Theorem 15 (From page 19)**.** *The protocol of Table 9 is correct and sound under the d-BSDH, DLOG, CRHF and DLM security assumptions.*

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, we have $w^\intercal = E(u^\intercal M)$ and $\bar{\mathcal{K}} = e(g_1; g_2^{\bar{v}\sigma})$. 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 $\Delta = g_2^{\delta\alpha}$, and $\bar{\mathcal{K}}'[j] = e(g_1; g_2^{\bar{v}'[j]\sigma}) = e(g_1; g_2^{\bar{v}[j]\sigma} g_2^{\delta\alpha[j] e_k \sigma}) = \bar{\mathcal{K}}[j] \cdot e(g_1; g_2^{\delta\alpha[j] s^k}) = \bar{\mathcal{K}}[j] \cdot e(g_1; \Delta[j]^{s^k})$. 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 we have: $\bar{\xi} = \prod_{i=1}^{n-1} \prod_{k=0}^{i-1} e(S_{i-k-1}; \bar{H}_i)^{x_k}$ so that also $\bar{\xi} = \prod_{i=1}^{n-1} \prod_{k=0}^{i-1} e(g_1^{s^{i-k-1}}; g_2^{\bar{v}_i})^{r^k} = e(g_1; g_2)^{\sum_{i=1}^{n-1} \sum_{k=0}^{i-1} \bar{v}_i s^{i-k-1} r^k}$.

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 $\bar{\mathcal{K}}[j] = g_T^{\bar{v}[j]\sigma} = g_T^{(s-r)(\sum_{i=1}^{n-1} \sum_{k=0}^{i-1} \bar{v}_i [j] s^{i-k-1} r^k) + \bar{v}[j] x}$. From the setup, this means that $\bar{\mathcal{K}}[j] = \bar{\xi}[j]^{s-r} g_T^{D(\zeta)\alpha[j] + c[j]}$ 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 14 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, we use again 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 `Write` 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 `Init` phase, we can randomly select m, n and $\ell \leqslant m$. 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 $\Gamma_k = \log_{g_2}(\bar{H}_k) - \tilde{v}_k\alpha$ exists, but is unknown). For any `Write` 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 $\Gamma'_k = \log_{g_2}(\bar{H}'_k) - \tilde{v}'_k\alpha$ 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. □

C Paillier's cryptosystem as the linearly homomorphic primitive

Paillier's homomorphic system works modulo some RSA composite number N. Now it is possible to use it to compute evaluations modulo a different m (for instance a prime), provided that m is small enough: consider the modulo m operations to be over $\mathbb{Z}$, perform the homomorphic operations, and use m only to reduce *after* decryption. This is illustrated in Algorithm 10.

Algorithm 10 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 $c_0 = E(p_0)$;
2: **for** $i = 1$ **to** d **do**
3: $\quad x_i \leftarrow x_{i-1} \cdot r \mod m$; {Now $x_i \in [0..m-1]$}
4: $\quad c_i \leftarrow c_{i-1} \cdot E(p_i)^{x_i}$; {Now $c = E(\sum_{k=0}^i p_i x_i)$}
5: **end for**
6: **return** $c = c_d$.

Lemma 16. *Algorithm 10 is correct.*

Proof. If $0 \leqslant p_i \leqslant (m-1)$, then as $x_i \equiv r^i \mod m$ is considered as an integer between 0 and $m-1$, then $0 \leqslant \sum_{i=0}^d p_i x_i \leqslant (d+1)(m-1)^2 < N$ 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 $D(c) \mod m = \sum_{i=0}^d p_i x_i \mod m \equiv P(r)$. □

D Parallel prefix-like algorithm for the Server

We here provide the parallelization we used for the Server audits in our experiments. For the DPoR, the matrix-vector product part was already parallelized in [5, Table 6], a Server auditing the 1TB database in a few minutes. For the polynomial part, as the dimensions become more rectangular, as we can see in Table 10, the Server's polynomial part is sometimes not negligible anymore, thus also benefits from some parallelization. For this, we would need to parallelize both the homomorphic dot-product and the Horner-like pairings. On the one hand, the former operations, line 1 in Algorithm 6, can be blocked in independent exponentiations and final multiplications in a binary tree. On the other hand, for the latter operations, a standard "baby steps / giant steps" approach can be employed for the iteration of lines 3-6 in Algorithm 6:

- 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 is exposed in Algorithm 11.

Algorithm 11 Parallel Server `Eval`

Input: Group order p, polynomial degree d, evaluation point r and vectors W, S, $\bar{H}[j]$, all as in Algorithm 6.
Input: Cutting parameter q (e.g. can be the number of threads).
Output: SERVER ζ, $\bar{\xi}[j]$ for $j = 1..2$.

Let $(b, \eta) \in \mathbb{N}^2$ s.t. $d + 1 = bq + \eta$, with $0 \leqslant \eta < q$;
Set $b_k \leftarrow \begin{cases} k(b+1) & k = 0..(\eta - 1) \\ kb + \eta & k = \eta..q \end{cases}$ {q blocks of size $b+1$ or b}
{PHASE A: $r^i \mod p$, for $i = 0..d$}
$\rho_0 \leftarrow 1$, $\rho_1 \leftarrow r$,$i \leftarrow 1$;
while $i \leqslant d$ **do** {$\lceil \log_2(d) \rceil$ parallel steps}
  parfor $k = 1..min(i; d - i)$ **do**
    $\rho_{i+k} \leftarrow \rho_i \cdot \rho_k \mod p$;
  end parfor
  $i \leftarrow 2i$;
end while
{PHASE B: $\zeta = W^\intercal \boxdot x = \prod_{i=0}^{d} w_i^{(r^i \mod p)}$}
parfor $k = 1..q$ **do** {q blocks of size b or $b+1$ in parallel}
  $\zeta_k \leftarrow \prod_{i=b_{k-1}}^{b_k - 1} w_i^{\rho_i}$
end parfor
$\zeta \leftarrow \prod_{k=1}^{q} \zeta_k$ {$\lceil \log_2(q) \rceil$ parallel steps}
{PHASE C: $u_\ell = \prod_{k=0}^{\ell} S_{\ell-k}^{r^k}$, for $\ell = 0..(d-1)$}
$u_0 \leftarrow S_0$;
for $k = 1$ **to** $q - 1$ **do** {q parallel steps}
  $u_{b_k} \leftarrow u_{b_{k-1}}^{\rho_{b_k - b_{k-1}}} \prod_{\ell=b_{k-1}+1}^{b_k} S_\ell^{\rho_{b_k - \ell}}$;
end for
parfor $k = 0..(q-1)$ **do** {q blocks of size b or $b-1$ in parallel}
  for $\ell = 0$ **to** $b_{k+1} - b_k - 1$ **do**
    $u_{b_k+\ell+1} \leftarrow S_{b_k+\ell+1} \cdot u_{b_k+\ell}^r$;
  end for
end parfor
{PHASE D: $\bar{\xi} = \prod_{i=1}^{d} \prod_{k=0}^{i-1} e(S_{i-1-k}; \bar{H}_i)^{r^k}$}
$\bar{\xi} = [1_{\mathbb{G}_T}, 1_{\mathbb{G}_T}]^\intercal \in \mathbb{G}_T^2$;
for $j = 1$ **to** 2 **do**
  parfor $k = 1..q$ **do** {q blocks of size b or $b+1$ in parallel}
    $\bar{\xi}_k[j] \leftarrow \prod_{\ell=b_{k-1}}^{b_k - 1} e(u_\ell; \bar{H}_{\ell-1}[j])$
  end parfor
  $\bar{\xi}[j] \leftarrow \prod_{k=1}^{q} \bar{\xi}_k[j]$ {$\lceil \log_2(q) \rceil$ parallel steps}
end for

Lemma 17. *Algorithm 11 is correct, work-optimal with work $W_q = O(d)$ and runs in time $W_q/q + o(W_q)$ on q processors.*

Proof. Correctness of phases A, B and D stems directly from the correctness of Algorithm 6. Phase C is correct since the new variables u_ℓ satisfy $\{u_0 = S_0, u_{\ell+1} = S_{\ell+1} u_\ell^r\}$.

Then, p is the prime group order, and for any homomorphic system satisfying Equation (2) we have:

- Phase A: requires d multiplications modulo p with depth $O(\log(d))$ and the parallelism is thus only bounded by Brent's law [17, Lemma 2];

- Phase B: requires $d+1$ homomorphic exponentiations and d homomorphic multiplications with a depth of $b = d/q$ such operations and the parallelism is thus only bounded by Brent's law;
- Phase C: requires d exponentiations and multiplications in $\mathbb{G}_1$. But this is implemented in parallel with a depth of $b = d/q$ such operations, only after precomputing $q-1$ times b operations each with a depth of $\log(b)$;
- Phase D: requires d pairings and $d-1$ multiplications in $\mathbb{G}_T$ with a depth of $b = d/q$ such operations and the parallelism is thus only bounded by Brent's law.

So only Phase C requires more operations in parallel than in sequence. And that number of operations is $d + b(q-1)$ exponentiations and multiplications if ran on q processors. This latter work is in fact optimal for prefix-like computations as shown in [62, Corollary 4] (see also [58]): indeed consider a family of binary gates $\theta_{\rho_i}(a,b)$ that on inputs a and b compute $a \cdot b^{\rho_i}$, that is one multiplication and one exponentiation. They satisfy the conditions of [62, Corollary 4] and thus computing all the u_ℓ is lower bounded by $d(2-1/q)$ calls to that gate when ran on q processors. □

Remark 18. *The accumulated* independent *exponentiations/pairings of lines 11, 16 and 26 of Algorithm 11 can in fact be gathered in small batches, where each batch can factorize some computations (e.g. using a generalized Shamir trick with multiple exponentiations in $\mathbb{G}_1$, or using NAF windows, etc.). Therefore, on the one hand, with respect to a purely sequential computation, the extra work required by Phase C (when used with more than 2 processors) is in fact batched. On the other hand, the other part of Phase C cannot benefit from these batches and is therefore dominant, but is more parallel. Therefore, as shown also in Table 13, this allows us to reach, on multiple cores, pretty good overall practical speed-ups.*

Table 13: Parallel Server-side VESPo

Degree	5816	18390	58154	186093	426519	4026778
1 core	4.4s	13.5s	42.6s	141.7s	324.1s	3 064.8s
4 cores	1.2s	3.8s	11.8s	38.3s	87.8s	831.6s
8 cores	0.7s	2.0s	6.3s	19.9s	45.4s	428.9s
12 cores	0.5s	1.4s	4.2s	13.4s	30.6s	283.6s

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

E Post-quantum homomorphic routines

The use of linearly homomorphic encryption (e.g., Paillier) and pairings means that, as implemented, our protocols are not resistant to quantum attacks. In response to recent recommendations by NIST and other standards organizations that all cryptographic solutions be made quantum-resistant, we considered the impacts of replacing these primitives with fully homomorphic encryption (FHE) primitives which are believed to be quantum-resistant.

Unfortunately, there are two reasons why further work is needed before we could recommend using FHE in our protocols. First, as we discuss in detail below, our preliminary implementation results are prohibitively slow using state of the art FHE libraries, due apparently to the inherent non-linear nature of polynomial evaluation on encrypted evaluation points. Second, our proof of security as written reduces the soundness guarantee to the t-BSDH problem, which has no analogue in FHE cryptosystems, and it is not clear what different assumption on FHE primitives could be used in its place.

We now detail our preliminary investigation into using FHE in our protocols, to better explain the shortcomings mentioned above and hopefully encourage future work in this direction.

We need two systems. First, where we use Paillier's cryptosystem, our protocols were already abstracted by the requirements of Equation (2). It is thus possible to use instead any quantum-safe linearly homomorphic primitives. There, Paillier's routine with larger parameters might be a possibility, see e.g. [11]. Other possibilities for now is to use quantum-safe fully homomorphic encryption, like BGV [16], here without bootstrapping.

Second, we need to modify the parts where we use pairings, in order to replace them with quantum-safe routines. For this we first abstract the requirements. Denote by $\mathcal{E}$, and resp. $\mathcal{D}$, the homomorphic encryption, resp. decryption, functions. We want those to support homomorphic addition, homomorphic multiplication between a ciphered message and a cleartext, together with depth-1 homomorphic multiplication between two ciphertexts and with homomorphic equality testing (in a private setting, this latter requirements can also for instance be implemented by decryption and direct equality testing). We can notate these requirements as follows:

$$\begin{aligned} \mathcal{D}(\mathcal{E}(m_1) \oplus \mathcal{E}(m_2)) &= m_1 + m_2 \\ \mathcal{D}(\mathcal{E}(m_1)^{m_2}) &= m_1 \times m_2 \\ \mathcal{D}(\mathcal{E}(m_1) \otimes \mathcal{E}(m_2)) &= m_1 \times m_2 \\ \mathcal{E}(m_1) \stackrel{?}{\ominus} \mathcal{E}(m_2) &\iff m_1 \stackrel{?}{=} m_2 \end{aligned}$$

The pairings parts in Table 12 are now transformed as in Table 14:

Table 14: Abstraction of the pairings functionalities

`Setup`	$\bar{\mathcal{K}} \leftarrow \mathcal{E}(\bar{P}(s))$, $S \leftarrow [\mathcal{E}(s^k)]_{k=0}^{d-1}$, $\bar{H} \leftarrow [\mathcal{E}(p_i\alpha + \Phi^i\beta]_{i=1}^{d}$
`Update`	$\bar{\mathcal{K}} \leftarrow \mathcal{E}\left(\alpha(p'_i - p_i)s^i\right) \oplus \bar{\mathcal{K}}$
`Eval`	$\bar{\xi} \leftarrow \oplus_{i=1}^{d} \left(\oplus_{k=0}^{i-1} S_{i-k-1}^{x_k}\right) \otimes \bar{H}_i$
`Verify`	$\bar{\xi}^{s-r} \oplus \mathcal{E}(D(\zeta)\alpha + c) \stackrel{?}{\ominus} \bar{\mathcal{K}}$

The two important issues are then the security analysis and the performance. First, the security analysis we have performed depends on assumptions of pairings (namely the hardness of t-BSDH). For Table 14 we would need to use some other assumption on the chosen FHE primitives. Second, our protocol efficiency crucially depends on efficient ciphertext-cleartext multiplication. We here report on some attempts with the BGV system implemented with the SEAL [52] and the HElib [39] libraries.

We were able to make our protocol work with these LWE-like implementations but for now there is a prohibitive performance price to pay, for two reasons:

1. A first constraint in SEAL and HElib is the size of the cleartext modulus which can usually not yet be very large, in practice at most some small fraction of a machine word.

2. A second limitation for these libraries, is that the ciphertext-cleartext multiplication is not much more efficient than ciphertext-ciphertext, since the noise in the polynomials is similarly increasing in both cases.

More precisely, for the computation of our coefficient ζ, we were able to use batched arithmetic with both SEAL and HElib and this is quite efficient, but works only for very small primes.

Differently, this is for the computation of our second coefficient, ξ, that the price to pay is much too prohibitive, even for very small primes and (too) low security parameters. Indeed, to compute ξ, our Server scheme involves computations of the form $S_3 \oplus S_2^r \oplus S_1^{r^2} \oplus S_0^{r^3}$, where r is a cleartext and the S_i are ciphertexts. On the one hand, if the r^k are precomputed, this is of constant multiplicative depth 1, even when counting ciphertext-cleartext multiplications, but then the overall double-loop scheme of Lemma 8 is quadratic-time. On the other hand, if ξ is instead homomorphically computed with the linear prefix-like Algorithm 9, the computations now involve in fact computations of the form $S_3 \oplus (S_2 \oplus (S_1 \oplus (S_0^r))^r)^r$. As mentioned, even though the involved multiplications are only ciphertext-cleartext, in the available libraries the noises increase linearly, much closer to a linear multiplicative depth. Bootstrapping is thus required a linear number of times. For instance, the latency of a BGV bootstrapping operation costs at least several dozen seconds [39][4]. We provide in Table 15, evaluations of our scheme using either SEAL and the quadratic, depth-1 version, or HElib and the linear, but bootstrapped version. Comparing with Table 7, we see that quantum-safe routines are for now still several orders of magnitude slower.

The dominant cost in these experiments is in fact the bootstrapping. Future work thus might be:

[4] In contrast, some other libraries, such as FHEW [25] and TFHE [24], may have faster bootstrapping operations but require to re-implement the homomorphic arithmetic with boolean circuits.

Table 15: Post-quantum prototypes (SEAL modulo 1032193, with 123.1 eq. security, 4096-batched ζ, and quadratic-time ξ; HElib modulo 31, with 39.5 eq. security, 24-batched ζ, and linear-time but bootstrapped ξ).

	Deg.	Server			Client
		ζ	ξ	bootstrap.	
SEAL	32	<0.01s	2.03s	0	6.57ms
	64	<0.01s	5.45s	0	
	128	<0.01s	20.95s	0	
	256	<0.01s	82.14s	0	
	512	<0.01s	325.87s	0	
	1024	<0.01s	1 294.88s	0	
	2048	<0.01s	5 171.84s	0	
	4096	<0.01s	20 667.99s	0	
HElib	32	0.01s	7.26s	0	283.44ms
	64	0.01s	80.02s	13	
	128	0.02s	257.20s	45	
	256	0.03s	613.83s	109	
	512	0.05s	1 334.52s	238	
	1024	0.10s	2 765.61s	493	
	2048	0.20s	5 643.51s	1005	
	4096	0.39s	11 382.50s	2030	

- Designing a post-quantum linearly homomorphic encryption with efficient ciphertext-cleartext multiplication
- Transforming the computation of ξ so that it is more batchable (a strategy could be to start by adapting the parallelization presented in Appendix D, so that more identical operations could be performed simultaneously)

For instance, phase C in Algorithm 11 is solely responsible for the multiplicative depth. Then we see that line 16 can be performed with a depth of q, while line 20 can be performed with q depth-b operations, with $d = bq$. With some FHE implementations (as reflected in Table 15) the first aggregated multiplications require less bootstrapping. Thus, depending on their respective costs and the actual architecture, some choices of b (and q) might reduce the overall required bootstraps. By looking at Tables 7 and 15, we see that even such a (small) constant gain in bootstrapping is not yet sufficient to compete with the pairings.