Fherret: Proof of FHE Correct-and-Honest Evaluation with Circuit Privacy from MPCitH

Janik Huth[1,2], Antoine Joux[1], and Giacomo Santato[1,2]

[1] CISPA – Helmholtz Center for Information Security, Saarbrücken, Germany
janik.huth@cispa.de ,joux@cispa.de, giacomo.santato@cispa.de
[2] Saarland University, Saarbrücken, Germany

Abstract. The major Fully Homomorphic Encryption (FHE) schemes guarantee the privacy of the encrypted message only in the honest-but-curious setting, when the server follows the protocol without deviating. However, various attacks in the literature show that an actively malicious server can recover sensitive information by executing incorrect functions, tampering with ciphertexts, or observing the client's reaction during decryption.
Existing integrity solutions for FHE schemes either fail to guarantee circuit privacy, exposing the server's computations to the client, or introduce significant computational overhead on the prover by requiring proofs of FHE operations on ciphertexts.
In this work, we present Fherret, a novel scheme leveraging the MPC-in-the-Head (MPCitH) paradigm to provide a proof of correct-and-honest homomorphic evaluation while preserving circuit privacy. This proof guarantees that the client can safely decrypt the ciphertext obtained from the server without being susceptible to reaction-based attacks, such as verification and decryption oracle attacks. Additionally, this proof guarantees that the server's evaluation maintains correctness, thereby protecting the client from $\mathsf{IND\text{-}CPA^D}$-style attacks.
Our solution achieves a prover overhead of 4λ homomorphic evaluations of random functions from the function space $\mathcal{F}$, while retaining a competitive verifier overhead of 2λ homomorphic evaluations and a communication size proportional to $\sqrt{2\lambda}$ times the size of a function from $\mathcal{F}$.
Furthermore, Fherret is inherently parallelizable, achieving a parallel computation overhead similar to a homomorphic evaluation of a random function from $\mathcal{F}$ for both the prover and the verifier.

Keywords: Fully Homomorphic Encryption, Reaction-Based attacks, $\mathsf{IND\text{-}CPA^D}$ security, MPC-in-the-Head

1 Introduction

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that enables computation on encrypted data without requiring decryption. Specifically, a client encrypts a message m and sends the ciphertext to a server. The server,

having chosen a function f, evaluates it homomorphically on the ciphertext and returns a result that decrypts to $f(m)$.

The major FHE schemes ([BGV12],[FV12],[DM15],[CGGI20]) guarantee the privacy of the encrypted message only in the honest-but-curious setting, when the server follows the protocol without deviating. However, various attacks in the literature have shown that an actively malicious server can recover sensitive information by executing an incorrect function or tampering with ciphertexts.

In this setting, security concerns arise when a dishonest server attempts to extract information by observing the client's reaction during decryption. These attacks can be described by giving a verification oracle to the adversary.

Verification oracle attacks are well-known in the FHE literature: in [LMSV12] and [ZPS12] an adversary can construct a decryption oracle by observing the reactions of the decrypting party. In [CGG16] and [CCCM22], the secret key gets recovered in a similar way. Analogous attacks were studied for the case of client-aided outsourced computations in [AGHV22] and [AV21]. Here, the attacker substitutes the ciphertexts with freshly encrypted ones during the interactions with the client to learn the encrypted message without any decryption happening.

Another attack strategy involves breaking the correctness of the computation. Li and Micciancio [LM21] present a key recovery attack on certain approximate FHE schemes that do not satisfy the IND-CPA$^{\mathsf{D}}$ security definition they introduce. This definition is theoretically equivalent to IND-CPA for exact schemes. While their attack only applies to approximate schemes like CKKS, this attack has recently been extended to exact schemes [CCP+24,CSBB24] where adversaries compromise correctness by manipulating function evaluations. These new IND-CPA$^{\mathsf{D}}$-style attacks against exact schemes can be viewed as reaction-based attacks, exploiting the client's response when correctness is violated.

A common countermeasure to reaction-based attacks is to require a proof of the correct behavior of the server. This allows the user to refuse to decrypt unverified ciphertexts. However, most existing proofs require that the function f which is evaluated homomorphically by the server is also known by the client.

In many FHE applications, such as private set intersection, privacy-preserving neural network inference, and genomic data analysis, the server must keep the evaluated function secret. While standard FHE ensures the confidentiality of user inputs, it does not inherently protect f. In other words, the ciphertext returned by the server may leak information about the function. To prevent this, specialized variants of FHE enforce circuit privacy.

Circuit privacy is a crucial property in FHE, ensuring that the ciphertext produced by the server reveals no information about f beyond the fact that it decrypts to $f(m)$.

Currently, the only known techniques for proving correct FHE evaluation while preserving circuit privacy rely on zk-SNARKs. However, this approach suffers from significant overhead, as the prover must account for all FHE operations that do not directly affect the plaintext but only the ciphertext (e.g.,

bootstrapping, modulus switching, noise flooding, and relinearization). As a result, the computational cost of generating the proof becomes prohibitively high.

1.1 Our Contributions

In this paper, we present Fherret (FHE coRREcT-and-Honest Evaluation Proof), a new scheme designed to protect FHE schemes against reaction-based attacks in settings where the evaluated functions need to remain confidential. Our contributions can be summarized as follows:

A Novel Approach to FHE Integrity: We present a new method for ensuring the integrity of homomorphic computations while preserving circuit privacy. Unlike previous methods that relied on zk-SNARKs of the homomorphic evaluation circuit, we leverage the MPC-in-the-Head (MPCitH) paradigm among N parties using τ rounds. Our approach operates in the random oracle model (ROM) and can be applied to any circuit-private FHE scheme without additional assumptions.

Operations are performed in the plaintext space: Rather than proving the evaluation of the FHE circuit itself, Fherret constructs proofs directly within the FHE scheme. This approach allows us to bypass the need for proving computation-heavy tasks that do not affect the encrypted message, such as bootstrapping, modulus switching, and noise flooding. These operations are simply executed as part of the protocol, significantly improving efficiency.

Scalability with FHE efficiency: The efficiency of proof generation and verification scales directly with the performance of the underlying FHE scheme. As a result, our approach benefits from any hardware acceleration designed to optimize FHE computations. Furthermore, Fherret is inherently parallelizable, enabling both the prover and verifier to achieve computational performance comparable to that of homomorphically evaluating a random function from $\mathcal{F}$.

Scheme Universality: Fherret is agnostic to the specific FHE scheme used and can be applied to any non-approximate FHE scheme. This also means it does not interfere with packing, SIMD, modified bootstrapping, NTT/RNS optimizations, or scheme switching. The only requirements are correctness and circuit privacy.

Security against reaction-based attacks: Fherret provides security against verification oracle attacks, ensuring that reaction-based adversaries cannot exploit the decryption process to recover sensitive information. More than that, it guarantees that correctness is preserved through all the server's homomorphic evaluation, preventing the FHE scheme from becoming approximate and protecting it from $\mathsf{IND\text{-}CPA^D}$-style attacks.

Publicly Verifiable: The verification of the proof can be performed by any user who has access to the public key of the FHE scheme, to the input ciphertext of the client, and the proof itself. Even without the secret key and without learning anything about the function that was evaluated or about the result of the evaluation, any third party can still verify the proof.

Implementation: We provide a proof of concept implementation of our scheme in C++ applied to the BGV-RNS scheme, as implemented in the OpenFHE

library. We also discuss implementation-related optimizations that allows to strongly reduce the number of FHE evaluations needed in the scheme.

1.2 Related Work

Verifiable computation A significant line of research focuses on enhancing FHE schemes with additional guarantees. One prominent direction is the integration of verifiable computation (VC) into FHE.

These methods ensure that the decrypted message corresponds to the honest homomorphic evaluation of a function that is *known* by the verifier, making them incompatible with circuit privacy. Examples include Homomorphic Message Authentication Codes ([GW13], [CF13], [FGP14], [CKP+24]), Zero-Knowledge Proofs ([GGW24], [CCC+25], [ABPS24]), and Trusted Execution Environments ([NLDD21]). A more comprehensive survey can be found in [VKH23].

Some ZK-based VC schemes allow the function to be partially or fully private. While these could be analyzed within our framework, our focus differs fundamentally from VC. VC aims to minimize verifier costs, often achieving sublinear or even constant verification complexity. In contrast, our approach prioritizes protection against a dishonest server, accepting a linear verification cost.

The majority of the ZK-based VC schemes for FHE that support the evaluation of a private function ([FNP20], [BCFK21], [GNS21], [ABPS24]) require the prover to prove the correct execution of the FHE evaluation circuit itself. As previously discussed, operating in the ciphertext space significantly increases computational complexity due to the inclusion of FHE maintenance operations (e.g., bootstrapping, modulus switching, and relinearization). Consequently, prover overhead remains high even for simple functions. For instance, in [VKH23], functions with a homomorphic evaluation time of 10–15 milliseconds require proof generation times of 5–7 minutes in the most optimized cases. Moreover, to achieve reasonable efficiency, most VP constructions are tailored to specific FHE schemes.

Finally, some ZK-based VC schemes for FHE support the evaluation of a private function while operating in the plaintext space ([ACGS24], [GGW24], [GBK+24]). Unfortunately, these schemes rely on evaluating a zk-SNARK homomorphically within the FHE scheme. This approach makes the constructions vulnerable to reaction-based attacks, as the proof must be decrypted before verification, which renders these schemes susceptible to attacks from a dishonest server. A detailed discussion on verification attacks against such constructions is provided in [ZWL+25], highlighting their incompatibility with our security model.

Achieving stronger security definitions. While IND-CCA2 security is unattainable for malleable encryption schemes like FHE, researchers have explored several intermediate security notions. Numerous studies have investigated IND-CCA1 security for FHE schemes. Recently, Manulis and Nguyen showed that IND-CCA1 security can be achieved even in the presence of a bootstrapping key [MN24].

Although IND-CCA1 security allows an attacker access to a decryption oracle, this oracle is unavailable during the challenge phase. As a result, reaction-based attacks remain effective against IND-CCA1 schemes (see [DSA13] for further details).

A stronger definition, vCCA, which provides security against verification oracle attacks, was introduced in [MN24]. Their construction embeds the FHE scheme into an IND-CCA2-secure framework for robustness and applies SNARKs to the homomorphic evaluation for integrity. However, this approach is primarily of theoretical interest and remains impractical.

2 Preliminaries

2.1 Notations

We denote the target security level of our scheme by λ. For any positive integer $n \in \mathbb{Z}^+$, we denote the set $\{0, \ldots, n-1\}$ by $[n]$. For $n > 1$, we denote the set $\{1, \ldots, n-1\}$ by $[n]^*$. For any finite set D, we use the notation $s \leftarrow_\$ D$ to indicate that s is sampled uniformly at random from D. We place ourselves in the random oracle model and assume that each hash function $\mathcal{H}$ is modeled by a random oracle. When dealing with a nondeterministic algorithm $A(\mathsf{input})$, we sometimes write $A(\sigma; \mathsf{input})$ to consider explicitly the random coins σ of the algorithm and treat it as deterministic. Throughout the paper, we denote the base-2 logarithm by log.

2.2 Fully Homomorphic Encryption ([Riv87])

We recall the definition of Fully Homomorphic Encryption in the public key setting.

Definition 1 (Fully Homomorphic Encryption). *We define a fully homomorphic encryption scheme* FHE *as a tuple of four algorithms* FHE = (KeyGen, Enc, Eval, Dec) *with the following syntax.*

- $\mathtt{KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$**:** *Given a security parameter* λ*, returns a secret key* sk *and a public key* pk*.*
- $\mathtt{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$**:** *Given a public key* pk *and a message* m*, returns a ciphertext* ct*.*
- $\mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct}) \to \mathsf{ct}_{\mathsf{out}}$**:** *Given a public key* pk*, a function* f*, and a vector of ciphertexts* ct*, returns a ciphertext* $\mathsf{ct}_{\mathsf{out}}$*.*
- $\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$**:** *Given a secret key* sk *and a ciphertext* ct*, returns a message* m*.*

A FHE *scheme is said to be correct if, for any pair* $(\mathsf{sk}, \mathsf{pk})$ *output by* KeyGen*, for any function* f *and for any vector of plaintexts* m*, where the number of elements in the vector is equal to the number of inputs of* f*, the following holds:*

$$\mathtt{Dec}(\mathsf{sk}, (\mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct}))) = f(m),$$

where ct *is the vector of the encryptions of elements of* m*, i.e.,* $\mathsf{ct}_i \leftarrow \mathtt{Enc}(\mathsf{pk}, m_i)$*.*

2.3 Security Definitions for FHE

We recall the definition of IND-CPA security.

Definition 2 (IND-CPA security). *Let* $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$ *be a fully homomorphic encryption scheme. We define the* IND-CPA *game as the experiment* $\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}$*, where* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is an adversary. The experiment is defined as follows:*

$$\begin{aligned}\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda):\ & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KeyGen}(\lambda)\\ & (m_0, m_1, s) \leftarrow \mathcal{A}_1(\mathsf{pk})\\ & b \leftarrow_{\$} \{0,1\}, c^* \leftarrow \mathtt{Enc}(\mathsf{pk}, m_b)\\ & b' \leftarrow \mathcal{A}_2(\mathsf{pk}, s, c^*)\\ & \textbf{\textit{return}}\ b = b'\end{aligned}$$

We say that a FHE *scheme is* IND-CPA*-secure if any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *has a negligible advantage. The advantage here is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda) = \left|\Pr[\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda) = 1] - \frac{1}{2}\right|.$$

Definition 3 (Circuit Privacy). *An* FHE *scheme is called Circuit Private on* $\mathcal{F}$ *if there is a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathsf{CP}}$*, such that, for any vector of valid ciphertexts* ct*,*

$$\{\mathsf{sk}, \mathcal{S}_{\mathsf{CP}}(\mathsf{pk}, m_{\mathsf{out}})\} \overset{c}{\approx} \{\mathsf{sk}, \mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct})\},$$

where $f \in \mathcal{F}$*,* $m_{\mathsf{out}} \leftarrow f(\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}))$ *and* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KeyGen}(\lambda)$*.*

2.4 Multi-Party Computation

This section introduces the notations we use for *Multi-Party Computation* (MPC) protocols. Throughout this paper, we use additive sharings of polynomials over rings. Let N be the number of parties that interact in the MPC protocol. An $N-$*sharing* of a finite ring element $x \in R$ is an N-tuple

$$[\![x]\!] = \left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$$

such that

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]} \quad (\text{in } R).$$

We call $x^{[\![i]\!]}$ the i-th *share* of x. In the protocols we consider, each party $\mathcal{P}_i$ for $i \in [N]$, receives one share $x^{[\![i]\!]}$ for each shared value x. Using their shares, the parties can then perform computations independently.

In practice, a sharing $[\![x]\!]$ is usually obtained by computing $N-1$ random values $x^{[\![0]\!]}, \ldots, x^{[\![N-2]\!]}$, and setting

$$x^{[\![N-1]\!]} = x - \sum_{i=0}^{N-2} x^{[\![i]\!]}$$

afterwards to obtain a valid sharing $[\![x]\!]$ of x.

During the MPC protocol, the parties can perform different computations with their respective shares to compute the output $g(x)$ for some function g. In this setup, $\mathcal{P}_i$ outputs its own value of g, which we denote by g_i.

With these sharings, the parties can perform different computations independently. Assume that party $\mathcal{P}_i$ receives the shares $x^{[\![i]\!]}$ and $y^{[\![i]\!]}$ corresponding to sharings of x and y, and let $\alpha \in R$ be a constant. With these shares, the parties can perform the following computations:

- **Addition**: They can compute $[\![x+y]\!]$ by locally setting

  $$(x+y)^{[\![i]\!]} = x^{[\![i]\!]} + y^{[\![i]\!]}$$

  for $i \in [N]$.
- **Multiplication by a constant**: They can compute $\alpha[\![x]\!] = [\![\alpha x]\!]$ by locally setting

  $$(\alpha x)^{[\![i]\!]} = \alpha x^{[\![i]\!]}$$

  for $i \in [N]$.

2.5 The MPCitH Paradigm

The construction of our protocol relies on the *MPC-in-the-Head* (MPCitH) paradigm, which was introduced in [IKOS07]. We consider an MPC protocol between N parties $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$, that securely and correctly computes the output of a function g, given a secret input x. In this setting, the secret x is given by a sharing $[\![x]\!]$, where party i receives the i-th share $x^{[\![i]\!]}$. The function g outputs either 1 or 0, corresponding to accept or reject, respectively. In our protocol, we require that the MPC protocol is $(N-1)$-private, meaning that the views of any $N-1$ out of N parties reveal no information about the secret x. With this type of MPC protocol, a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that she knows a witness x with $g(x) = 1$ by constructing a Zero-Knowledge protocol as follows:

- $\mathcal{P}$ generates a random sharing $[\![x]\!]$ of x.
- $\mathcal{P}$ simulates ("in the head") all parties of the MPC protocol and sends commitments to each party's view to the verifier.
- $\mathcal{V}$ randomly selects $N-1$ parties whose views the prover must reveal.
- The verifier checks whether these revealed views are consistent with an honest execution of the MPC protocol and the prover's commitments.

Because the MPC protocol is $(N-1)$-private, opening the views of all but one party reveals no information about the secret x. Since the choice of the $N-1$ opened parties is random, a malicious prover can cheat with probability $\frac{1}{N}$ by corrupting the computation of the unopened party. Thus, the *soundness error* of a protocol based on the MPCitH paradigm is (at least) $\frac{1}{N}$. To amplify the soundness to the required security level of λ bits, we use τ repetitions of the protocol, resulting in a soundness error of $N^{-\tau}$.

2.6 Hypercube Folding

To reduce computational overhead, MPCitH protocols commonly use the *hypercube technique*, introduced in [AGH+23].

Assume the number of parties in the MPC protocol is a power of two, i.e., $N = 2^D$. The hypercube technique transforms one instance of an MPC protocol between N parties into D instances of the MPC protocol between 2 parties.

By computing D instances in parallel, the total soundness error of $\frac{1}{N} = \frac{1}{2^D} = (\frac{1}{2})^D$ is the same as in the original protocol between N parties.

The process of converting a sharing between $N = 2^D$ parties into D sharings between 2 parties is called *folding* of the shares. Consider a sharing of x of the form $x = \sum_{i=0}^{N-1} x^{[\![i]\!]}$. Let $B_j(i)$ denote the j-th bit of the binary decomposition of an integer i. For any fixed $j \in [D]$, we see that

$$x = \sum_{B_j(i)=0} x^{[\![i]\!]} + \sum_{B_j(i)=1} x^{[\![i]\!]} \tag{1}$$

is a sharing of x between 2 parties. In this setting, the first party receives the share

$$x_j^{[\![0]\!]} = \sum_{B_j(i)=0} x^{[\![i]\!]},$$

while the second party receives the share

$$x_j^{[\![1]\!]} = \sum_{B_j(i)=1} x^{[\![i]\!]}.$$

This holds for every $j \in [D]$, resulting in D sharings of x of the form $x = x_j^{[\![0]\!]} + x_j^{[\![1]\!]}$. We refer to this construction by

$$\left(x_j^{[\![0]\!]}, x_j^{[\![1]\!]}\right)_{j\in[D]} \leftarrow \mathsf{Folding}\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right).$$

In practice, we can use the folding algorithm with running time $\mathcal{O}(N)$ introduced in [HJ24a, Section 6].

2.7 Reducing the Communication Cost

To reduce the communication cost of our protocol, we use a state-of-the-art technique applied to various MPCitH protocols, called *puncturable pseudorandom functions* (PPRFs).

Definition 4. *A family G of puncturable PRFs over $[N]$ is a PRF family G indexed by a key K with domain $[N]$, satisfying the following conditions:*

- *For each key K and index $i^* \in [N]$, there exists a punctured key K_{i^*} and an algorithm $\mathcal{A}$ such that*

$$\text{for each } j \in [N] \setminus \{i^*\} : \mathcal{A}(K_{i^*}, j) = G_K(j).$$

- *The punctured key K_{i^*} reveals no information about $G_K(i^*)$.*

GGM trees. A common method to instantiate PPRFs in practice is the tree-based PRF construction by Goldreich, Goldwasser and Micali, known as *GGM trees* [GGM86]. This construction builds a binary tree of depth $\lceil \log(N) \rceil$ with N leaves, where the root node is labeled with a master root seed. Each node's left and right children are labeled inductively using a length-doubling *pseudorandom generator* (PRG) on the respective parent node. In our scheme, we use a (salted) hash function with domain separation to instantiate this descent function. The N leaves of this tree represent the N shares of the value x that we use in the MPCitH protocol. Given a root seed r, we denote the described expansion of this root into N leaves by $\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right) \leftarrow \mathrm{GGM(r, N)}$. To reveal all N leaves except one leaf i^* in a GGM tree, it suffices to reveal the labels of the siblings of the nodes on the path from the root to the leaf i^*. This way, all leaves except the leaf at position i^* can be reconstructed by communicating $\lceil \log(N) \rceil$ nodes, instead of $N - 1$ leaves.

To obtain a valid sharing of the value x, we need an *offset* δ_x if we use this basic GGM tree construction. This offset is computed as

$$\delta_x = x - \sum_{i=0}^{N-1} x^{[\![i]\!]}$$

and is broadcast to all parties.

Consistency for large values. In MPCitH schemes, the bitsize of the tree nodes is typically set to λ, while also using a salt of 2λ bits in the derivation function. Given $\mathrm{salt} \leftarrow \{0,1\}^{2\lambda}$ and a hash function $\mathcal{H}$, we define $\mathcal{H}^{\mathrm{salt}}(\mathrm{msg}) := \mathcal{H}(\mathrm{salt} \,\|\, \mathrm{msg})$ for any $\mathrm{msg} \in \{0,1\}^*$. If we use the salted hash function for the GGM tree derivation, we use the notation $\mathsf{GGM}_{\mathrm{salt}}$ instead. For our scheme, we require sharings of functions $f \in \mathcal{F}$. We can use the GGM tree construction for f by using internal nodes of the tree with the same bitsize as elements in $\mathcal{F}$. Since this bitsize is often much larger than the desired security parameter

λ, a GGM construction using such large inner nodes is expensive. Instead, we can use the standard GGM tree construction described above with inner nodes of just λ bits. For these trees, we can expand each leaf to more than λ bits using a pseudorandom generator to achieve the desired bitsize of elements in $\mathcal{F}$. These additional bits require offsets and we have to ensure that the same value is consistently shared among all trees. Indeed, if we use τ rounds of the MPCitH protocol, we obtain τ such trees T_j, each having one offset value δ_{f_j} for $j \in [\tau]$. The leaves of each such tree, together with the offset value, correspond to a function

$$f_j = \sum_{i=0}^{N-1} f_j^{[\![i]\!]} + \delta_{f_j}.$$

However, from the verifier's perspective, these f_j do not have to be equal across rounds. A state-of-the-art method for ensuring the desired consistency of sharings across multiple rounds was introduced by Baum et al. in [BBD+23]. The idea is to use a non-interactive version of the SoftspokenOT technique [Roy22] to ensure that the prover provides offsets resulting in a sharing of the same element across multiple rounds based on a probabilistic check. This technique is utilized in many schemes that employ a similar paradigm to MPCitH, known as VOLEitH (Vector Oblivious Linear Evaluation in the Head), such as in [CLY+24], [BFG+24], [Bui24].

In our scheme, we use a subroutine for the consistency check based on the techniques introduced in [HJ24b]. We explain this consistency check in detail in Section 3.3.

3 The Scheme

In this chapter, we construct a proof of the correct-and-honest homomorphic evaluation of a private function from $\mathcal{F}$ on a known ciphertext.

In Section 3.1, we introduce a basic version of the Fherret scheme to illustrate its fundamental mechanics, although this version is neither secure nor optimized. In Sections 3.2 and 3.3, we explain the attacks against the basic version and describe the modifications required to make it secure. In Section 3.4, we present the adapted scheme constructed to prevent attacks from Sections 3.2 and 3.3 and prove its security.

3.1 A Simple but Insecure Version of the Scheme

We construct an interactive protocol to prove the correct-and-honest homomorphic evaluation of a private function $f \in \mathcal{F}$ on a vector of ciphertexts ct. The protocol runs between a prover/client $\mathcal{P}$ and a verifier/server $\mathcal{V}$, using the MPCitH paradigm. This initial version of the scheme is neither secure nor optimized but is designed to demonstrate the core principles.

The prover computes a random sharing $[\![f]\!]$ of f between N parties and then evaluates each share $f^{[\![i]\!]}$ on the ciphertext ct, obtaining ct_i for $i \in [N]$. After

Inputs:

- Public key pk, function f, vector of ciphertexts ct (Prover)
- Secret key sk, public key pk, vector of ciphertexts ct (Verifier)

Step 1: Commitment

1. Let f be a polynomial in $\mathcal{F}$
2. For $j \in [\tau]$:
   - For $i \in [N-1]$: Sample ${f_j}^{[\![i]\!]} \leftarrow_{\$} \mathcal{F}$
   - Compute the missing share ${f_j}^{[\![N]\!]} \leftarrow f - \sum_{i\in[N-1]} {f_j}^{[\![i]\!]}$
   - For $i \in [N]$: Compute $\mathsf{ct}_{i,j} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![i]\!]}, \mathsf{ct})$ and $h_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \mathsf{ct}_{i,j})$
   - Compute $X_j \leftarrow \bigoplus_{i\in[N]} \mathsf{ct}_{i,j}$
3. Compute $\mathsf{com} \leftarrow \mathcal{H}\left((h_{i,j})_{(i,j)\in[N]\times[\tau]}\right)$
4. Send $(\mathsf{com}, X_0, \ldots, X_{\tau-1})$ to the verifier

Step 2: Challenge

1. For $j \in [\tau]$: sample $i^*_j \leftarrow_{\$} [N]$
2. Send $(i^*_0, \ldots, i^*_{\tau-1})$ to the prover

Step 3: Response

1. Send $({f_j}^{[\![i]\!]}, \sigma_{i,j})$ for all $(i,j) \in [N] \times [\tau]$ except for the pairs (i^*_j, j) for all $j \in [\tau]$ to the verifier

Step 4: Verification

1. For $j \in [\tau]$:
   - For $i \in [N] \setminus \{i^*_j\}$: Compute $\mathsf{ct}_{i,j} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![i]\!]}, \mathsf{ct})$ and $h'_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \mathsf{ct}_{i,j})$.
   - Compute $\mathsf{ct}_{i^*_j,j} \leftarrow \bigoplus_{i\neq i^*_j} \mathsf{ct}_{i,j} \bigoplus X_j$ and $h'_{i^*_j,j} \leftarrow \mathcal{H}(i^*_j \parallel j \parallel \mathsf{ct}_{i^*_j,j})$
   - Compute $m_{\mathsf{out},j} \leftarrow \sum_{i\in[N]} \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_{i,j})$
2. Compute $h' \leftarrow \mathcal{H}\left((h'_{i,j})_{(i,j)\in[N]\times[\tau]}\right)$
3. If $h' = \mathsf{com}$ and all $m_{\mathsf{out},j}$ for $j \in [\tau]$ are equal, output `accept`. Otherwise, output `reject`

Fig. 1. Illustration of the basic ideas of the Fherret scheme (insecure)

that, $\mathcal{P}$ commits to them by hashing all of these ciphertexts and then computes X as the bitwise XOR of all the ct_i.

After receiving the commitment and X, $\mathcal{V}$ samples a random index $i^* \in [N]$ and sends it to $\mathcal{P}$. The prover responds by revealing every share of f, and the related random coins σ_i, except ${f}^{[\![i^*]\!]}$ and σ_{i^*} to the verifier. Since the use of random shares ensures that the protocol is $N-1$ private, $\mathcal{V}$ gains no information about f.

Using these shares and these random coins, the verifier recomputes the evaluations of $f^{[\![i]\!]}$ on the ciphertext ct, obtaining ct_i for $i \neq i^*$. Using X, $\mathcal{V}$ can also recompute the homomorphic evaluation of $f^{[\![i^*]\!]}$ on ct. After doing this, $\mathcal{V}$ checks if the hashes of the ct_i match the commitment and obtains the result of the decryption of the homomorphic evaluation by decrypting all the ct_i and summing them together. A description of the τ round basic scheme is provided in Figure 1.

A malicious prover $\mathcal{P}'$ who attempts to use an invalid sharing must cheat in at least one position. This remains undetected by the verifier only if the cheating position equals i^*. By repeating this protocol for multiple rounds, we can decrease the cheating probability until it is negligible.

While this protocol already limits the ability of a malicious prover by checking the honesty of many of its computations, the verifier is still vulnerable to weaker versions of reaction-based attacks. In the following sections, we describe these attacks and outline how to fully protect the scheme against them.

3.2 Simple Error Correction During Decryption

The basic scheme in Figure 1 is still vulnerable to verification oracle attacks. The weakness that an attacker can exploit arises from the construction of the verification process.

In the final step, the user checks whether all $m_{\mathsf{out},j}$ for $j \in [\tau]$ are equal. If they are, the verifier accepts; otherwise, verification fails. Thus, for the prover to modify the outcome, they need to alter only one value among all $m_{\mathsf{out},j}$, which can be done with a non-negligible probability of $\frac{1}{N}$. Consequently, this scheme only reduces the effectiveness of the verification oracle by a factor of N while still permitting the same attacks (such as [ZPS12], [CCP+24], [CSBB24]) based on the user's reaction during decryption.

To see this, consider an FHE scheme with RLWE ciphertexts $\mathsf{ct} = (a, b)$. Assume a server is executing the scheme with f being the identity function for $\tau - 1$ rounds. In the last round, after computing all the $\mathsf{ct}_{i,\tau-1}$ for $i \in [N]$, the server modifies one of them, say the I-th, by introducing some extra noise e to the b component and computing $\mathsf{ct}'_{I,\tau-1} = \mathsf{ct}_{I,\tau-1} + (0, e)$. It also adjusts $X_{\tau-1}$ and $h_{I,\tau-1}$ accordingly. With probability $\frac{1}{N}$, the challenge $i^*_{\tau-1}$ is equal to I. This implies that the final hash check succeeds because the user does not recompute $\mathsf{ct}_{i^*_{\tau-1},\tau-1}$ but retrieves it from $X_{\tau-1}$. As a result, the final output of the verification depends only on the unanimity check, which succeeds if and only if $\mathtt{Dec}(\mathsf{ct}'_{I,\tau-1}) = \mathtt{Dec}(\mathsf{ct}_{I,\tau-1})$. This allows the attacker to gather information about the magnitude of the RLWE error of $\mathsf{ct}_{I,\tau-1}$, potentially compromising the security of the FHE scheme.

To counteract these types of attacks, we modify the final check such that the verifier accepts if and only if more than half of the $m_{\mathsf{out},j}$ are equal. This forces an adversary to manipulate at least $\frac{\tau}{2}$ rounds to alter the verification result. To achieve this without failing the hash commitment check, the attacker would need to guess the verifier's challenge correctly for each of these rounds, which

happens with a negligible probability of $\left(\frac{1}{N}\right)^{\frac{\tau}{2}}$. This implies that we can safely omit the equal majority check, because if the commitment check succeeds, the probability of the second check failing is negligible. Therefore, we can always expect that more than half of the $m_{\mathsf{out},j}$ are equal with probability almost 1.

To express this concept more easily in the protocols, we define the majority of any finite list $\mathcal{L}$ as:

$$\mathsf{maj}(\mathcal{L}) = \begin{cases} \text{the element in } \mathcal{L} \text{ that appears more than } |\mathcal{L}|/2 \text{ times} \\ \perp \ \text{, if such an element does not exist.} \end{cases}$$

3.3 Adding a Consistency Check

Another problem with the scheme in Figure 1 is that the verifier cannot check if the prover is always computing shares of the same function during different rounds. Potentially, the prover could use different functions for different rounds without detection.

Even if the prover does not know the message m that is encrypted in ct, having access to a verification oracle could still allow the prover to obtain additional information about m. In particular, even considering the majority check from Section 3.2, the prover could choose two functions f_0 and f_1 and perform $\frac{\tau}{2}$ rounds of the protocol by computing shares of f_0 and $\frac{\tau}{2}$ rounds using shares of f_1. If the verification oracle outputs $\mathtt{accept}$, this reveals that $f_0(m) = f_1(m)$, providing additional information about m.[3]

Hypercube version of the scheme. We can reduce the computational overhead for the prover and verifier by using the hypercube version of the scheme. In particular, we can transform τ parallel instances of the protocol between $N = 2^D$ parties into $\tau \cdot D$ parallel instances of the protocol between 2 parties. In the basic version of the scheme, the prover has to do $\tau \cdot N$ evaluations on ct, while the verifier has to do $\tau \cdot (N-1)$ such evaluations. By considering the hypercube version instead, we can reduce this to $2\tau D$ evaluations on ct for the prover and $\tau \cdot D$ evaluations on ct for the verifier.

To achieve this, we use the hypercube folding algorithm on the leaves of each GGM tree we computed before to obtain $\tau \cdot D$ sharings of f between two parties. Since we need to send one XOR X_j of all $\mathsf{ct}_{i,j}$ per sharing to the verifier, the communication for the hypercube version increases by a factor of $D = \log(N)$. Indeed, we need to send $\tau \cdot D$ values $X_{i,j}$ in the hypercube version instead of τ values X_j in the base scheme from Figure 1.

VOLE consistency check As discussed in Section 2.7, we reduce computational overhead by using internal GGM tree nodes of size λ, initializing each tree

[3] When working on a field, the set where two polynomials are equal is small by the Schwartz–Zippel lemma. Instead, when working modulo a power of 2 this set might grow noticeably and allow for easier attacks.

Inputs:

- Public key pk, function f, vector of ciphertexts ct (Prover)
- Secret key sk, public key pk, vector of ciphertexts ct (Verifier)

Step 1: Commitment (Prover)

1. Let f be a polynomial in $\mathcal{F}$ of degree d with k variables
2. For $(i,j) \in [D] \times [\tau]$: Sample random coins $\sigma_{i,j}$ for the FHE $\texttt{Eval}$ and compute $h^{\sigma}_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \sigma_{i,j})$
3. Compute $h^{\sigma} \leftarrow \mathcal{H}\left(\left(h^{\sigma}_{i,j}\right)_{i,j \in [D] \times [\tau]}\right)$
4. Sample salt $\leftarrow_{\$} \{0,1\}^{2\lambda}$ for the GGM tree derivation
5. For $j \in [\tau]$:
   - Sample $\vec{u}_j \leftarrow_{\$} \{0,1\}^{\lambda}$
   - Compute $T_{\vec{u}_j}$: $\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right) \leftarrow \mathsf{GGM}_{\text{salt}}\left(\vec{u}_j, N\right)$
   - For $i \in [N]$: Compute $\vec{f}_j^{[\![i]\!]} \leftarrow \mathrm{PRG}_0\left(\vec{u}_j^{[\![i]\!]}\right)$ and $z_j{}^{[\![i]\!]} \leftarrow \mathrm{PRG}_1\left(\vec{u}_j^{[\![i]\!]}\right)$
   - $\left(\vec{f}_j^{[\![0]\!]_i}, \vec{f}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{f}_j^{[\![0]\!]}, \ldots, \vec{f}_j^{[\![N-1]\!]}\right)$
   - $\left(z_j{}^{[\![0]\!]_i}, z_j{}^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![N-1]\!]}\right)$
6. Compute $z_0 \leftarrow \sum_{i=0}^{N-1} z_0{}^{[\![i]\!]}$
7. For $j \in [\tau]$:
   - Compute $\delta_{\vec{f}_j} \leftarrow \vec{f} - \sum_{i=0}^{N-1} \vec{f}_j^{[\![i]\!]}$ and $\delta_{z_j} = z_0 - \sum_{i=0}^{N-1} z_j{}^{[\![i]\!]}$
   - For $i \in [D]$:
     - Let $f_j{}^{[\![0]\!]_i} \leftarrow \vec{f}_j^{[\![0]\!]_i}$ and $f_j{}^{[\![1]\!]_i} \leftarrow \vec{f}_j^{[\![1]\!]_i} + \delta_{\vec{f}_j}$
     - Compute $\mathsf{ct}^{(0)}_{i,j} \leftarrow \texttt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, f_j{}^{[\![0]\!]_i}, \mathsf{ct}\right)$ and $\mathsf{ct}^{(1)}_{i,j} \leftarrow \texttt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, f_j{}^{[\![1]\!]_i}, \mathsf{ct}\right)$
     - Compute $h^{(0)}_{i,j} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}^{(0)}_{i,j}\right)$ and $h^{(1)}_{i,j} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}^{(1)}_{i,j}\right)$
     - Compute $X_{i,j} \leftarrow \mathsf{ct}^{(0)}_{i,j} \oplus \mathsf{ct}^{(1)}_{i,j}$
8. Compute

$$\mathsf{com} \leftarrow \mathcal{H}\left(\text{salt} \parallel h^{\sigma} \parallel \left(h^{(0)}_{i,j}, h^{(1)}_{i,j}\right)_{(i,j) \in [D] \times [\tau]} \parallel \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]} \parallel \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$$

9. Send $\text{commit} \leftarrow \left(\mathsf{com}, \text{salt}, h^{\sigma}, \left(X_{i,j}\right)_{(i,j) \in [D] \times [\tau]}, \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]}, \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$ to the verifier

Fig. 2. Commitment of the hypercube Fherret scheme including the consistency check

T_j with $u_j \leftarrow_{\$} \{0,1\}^{\lambda}$. After expanding each tree to N leaves, we apply a PRG $G\colon \{0,1\}^{\lambda} \to \mathcal{F}$ to obtain shares in $\mathcal{F}$, then compute offsets δ_{f_j} to correct the sum to match a common f. To ensure the consistency of the functions across all τ subtrees to the verifier and to avoid the attack explained above, the prover

Step 2: First Challenge (Verifier)

1. Sample $\vec{\mu} \leftarrow_{\$} (\mathbb{F}_{p^l})^{|\mathcal{F}|}$
2. For $m \in [\tau \cdot D]$: Sample $\alpha_m \leftarrow_{\$} \mathbb{F}_{p^l}$
3. Send $\vec{\mu}, (\alpha_m)_{m \in [\tau \cdot D]}$ to the prover

Step 3: First Response (Prover)

1. Compute $c \leftarrow z_0 + \vec{f} \cdot \vec{\mu}$
2. For $j \in [\tau]$:
   - Compute $\vec{A}^{(j)}_{G_{\vec{f}}} \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \, \vec{f}_j^{[\![0]\!]_i}$
   - Compute $\vec{A}^{(j)}_{G_z} \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \, z_j^{[\![0]\!]_i}$
   - Compute $a^{(j)} \leftarrow \vec{A}^{(j)}_{G_z} + \vec{A}^{(j)}_{G_{\vec{f}}} \cdot \vec{\mu}$
3. Send $c, \left(a^{(j)}\right)_{j \in [\tau]}$ to the verifier

Step 4: Second Challenge (Verifier)

1. For $(i,j) \in [D] \times [\tau]$: Compute $b^*_{i,j} \leftarrow_{\$} \{0,1\}$
2. Send $\left(b^*_{i,j}\right)_{(i,j) \in [D] \times [\tau]}$ to the prover

Step 5: Second Response (Prover)

1. Compute the PPRF key $K \leftarrow K_{\left(b^*_{i,j}\right)_{(i,j) \in [D] \times [\tau]}}$
2. Send K, $(\sigma_{i,j})_{i,j}$ for all $(i,j) \in [D] \setminus \{i^*_0, \ldots, i^*_{\tau-1}\} \times [\tau]$ and the hashes $\left(h^{\sigma}_{i^*_j, j}\right)_{j \in [\tau]}$ to the verifier

Fig. 3. Interactive version of the Challenges and Responses

needs to do some extra work to convince the verifier by adding a consistency check. To do that, we adopt a VOLE-in-the-head–style check [BBD+23,HJ24b].

Consider the function $\vec{f} \in \mathcal{F}$ with $\log(|\mathcal{F}|) > \lambda$. We use vectorial notation to be consistent with VOLE literature. Let $\vec{u}_j \leftarrow_{\$} \{0,1\}^{\lambda}$. First, we compute a GGM tree family $(T_0, \ldots, T_{\tau-1})$ for the small value $\vec{u}_j$ as described in Section 2.7 to obtain the leaves

$$\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right) \leftarrow \mathrm{GGM}_{\mathrm{salt}}\left(\vec{u}_j, N\right) \text{ for } j \in [\tau].$$

After computing the hypercube folding of each such sharing, we obtain

$$\left(\vec{u}_j^{[\![0]\!]_i}, \vec{u}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right)$$

for each $j \in [\tau]$, where $D = \log(N)$. In total, we obtain $\tau \cdot D$ such hypercube sharings. For each such sharing, the prover knows both shares, while the verifier learns only one of the two from the PPRF key. Indeed, the verifier learns all

Step 2: First Challenge (Prover)

1. Compute $\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]} \leftarrow \mathcal{H}(\text{commit})$

Step 3: First Response (Prover)

1. Compute $c \leftarrow z_0 + \vec{f}\cdot\vec{\mu}$
2. For $j \in [\tau]$:
   - Compute $\vec{A}^{(j)}_{G_{\vec{f}}} \leftarrow \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, \vec{f}_j^{[\![0]\!]_i}$
   - Compute $\vec{A}^{(j)}_{G_z} \leftarrow \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, {z_j}^{[\![0]\!]_i}$
   - Compute $a^{(j)} \leftarrow \vec{A}^{(j)}_{G_z} + \vec{A}^{(j)}_{G_{\vec{f}}}\cdot\vec{\mu}$

Step 4: Second Challenge (Prover)

1. For $(i,j) \in [D]\times[\tau]$:
   $b^*_{i,j} \leftarrow \mathcal{H}\left(\text{commit} \parallel \vec{\mu} \parallel (\alpha_m)_{m\in[\tau\cdot D]} \parallel c \parallel \left(a^{(j)}\right)_{j\in[\tau]}\right)$

Step 5: Second Response (Prover)

1. Compute the PPRF key $K \leftarrow K_{\left(b^*_{i,j}\right)_{(i,j)\in[D]\times[\tau]}}$
2. Send K, c and $\left(a^{(j)}\right)_{j\in[\tau]}$, the random coins $(\sigma_{i,j})_{i,j}$ for all $(i,j) \in [D] \setminus \{i^*_0,\ldots,i^*_{\tau-1}\}\times[\tau]$ and the hashes $\left(h^{\sigma}_{i^*_j,j}\right)_{j\in[\tau]}$ to the verifier

Fig. 4. Non-interactive version of the Challenges and Responses

leaves of T_j except one leaf $\vec{u}_j^{[\![i^*]\!]}$. Therefore, by using the folding algorithm, the verifier cannot recompute the hypercube shares that contain $\vec{u}_j^{[\![i^*]\!]}$ in the summation (1), depending on the bit decomposition of the index i^*.

To derive the shares corresponding to $\vec{f}$, we use a PRG on the leaves of each subtree. In particular, we derive these shares using

$$\vec{f}_j^{[\![i]\!]} \leftarrow \mathrm{PRG}_0\left(\vec{u}_j^{[\![i]\!]}\right)$$

for each $(i,j) \in [N]\times[\tau]$, where $\mathrm{PRG}_0\colon \{0,1\}^\lambda \to (\mathbb{F}_p)^{|\mathcal{F}|}$ is a pseudorandom generator. To correct these pseudorandom values in order to obtain a sharing of $\vec{f}$, we compute the offset values

$$\delta_{\vec{f}_j} = \vec{f} - \sum_{i=0}^{N-1} \vec{f}_j^{[\![i]\!]}$$

for each $j \in [\tau]$. For the consistency check described in [HJ24b], we also need to derive an additional value from the leaves. In particular, we derive ${z_j}^{[\![i]\!]} \leftarrow$

Step 6: Verification and Decryption (Verifier)

1. Use K and salt to recompute and fold all trees (with one missing leaf each) and expand the leaves using $\mathrm{PRG}_0, \mathrm{PRG}_1$
2. For $(i,j) \in [D] \setminus \{i_0^*, \ldots, i_{\tau-1}^*\} \times [\tau]$: Compute $h_{i,j}^{\sigma} \leftarrow \mathcal{H}\left(i \parallel j \parallel \sigma_{i,j}\right)$
3. Compute $h^{\sigma'} \leftarrow \mathcal{H}\left(\left(h_{i,j}^{\sigma}\right)_{i,j \in [D] \times [\tau]}\right)$
4. For $(i,j) \in [D] \times [\tau]$:
   - Denote the known bit position of the respective hypercube sharing of $\vec{f}_j^{[\![\cdot]\!]_i}$ by $b_{i,j} \leftarrow 1 - b_{i,j}^*$
   - Recompute $f_j{}^{[\![b_{i,j}]\!]_i} \leftarrow \vec{f}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\, \delta_{\vec{f}_j}$
   - $\mathsf{ct}_{i,j}^{(b_{i,j})} \leftarrow \mathtt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, f_j{}^{[\![b_{i,j}]\!]_i}, \mathsf{ct}\right)$
   - $h_{i,j}^{(b_{i,j})} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}_{i,j}^{(b_{i,j})}\right)$
   - $\mathsf{ct}_{i,j}^{(1-b_{i,j})} \leftarrow \mathsf{ct}_{i,j}^{(1-b_{i,j})} \oplus X_{i,j}$
   - $h_{i,j}^{(1-b_{i,j})} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}_{i,j}^{(1-b_{i,j})}\right)$
5. Set $\delta_{z_0} \leftarrow 0$
6. For $j \in [\tau]$:
   - Compute $\Delta_j^* \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i}\, b_{i,j}$
   - Compute $\vec{G}_{\vec{f}}^{(j)}(\Delta_j^*) \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \left(\vec{f}_j^{[\![b_{i,j}]\!]} + b_{i,j}\, \delta_{\vec{f}_j}\right)$
   - Compute $\vec{G}_z^{(j)}(\Delta_j^*) \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \left(z_j{}^{[\![b_{i,j}]\!]} + b_{i,j}\, \delta_{z_j}\right)$
7. Compute $h' \leftarrow \mathcal{H}\left(\mathrm{salt} \parallel h^{\sigma'} \parallel \left(h_{i,j}^{(0)}, h_{i,j}^{(1)}\right)_{(i,j) \in [D] \times [\tau]} \parallel \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]} \parallel \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$
8. If $\vec{G}_z^{(j)}(\Delta_j^*) + \vec{G}_{\vec{f}}^{(j)}(\Delta_j^*) \cdot \vec{\mu} = a^{(j)} + c\, \Delta_j^*$ for all $j \in [\tau]$, $h^{\sigma'} = h^{\sigma}$ and $h' = \mathsf{com}$ output `accept`. Otherwise, output `reject`.
9. Compute $m_{\mathsf{out},i,j} \leftarrow \mathtt{Dec}\left(\mathsf{sk}, \mathsf{ct}_{i,j}^{(b_{i,j})}\right) + \mathtt{Dec}\left(\mathsf{sk}, \mathsf{ct}_{i,j}^{(1-b_{i,j})}\right)$
10. Output $m_{\mathsf{maj}} \leftarrow \mathsf{maj}\left(\{m_{\mathsf{out},i,j}\}_{(i,j) \in [D] \times [\tau]}\right)$

Fig. 5. Verification of the hypercube Fherret scheme including the consistency check

$\mathrm{PRG}_1\left(\vec{u}_j^{[\![i]\!]}\right)$ for each $(i,j) \in [N] \times [\tau]$, where $\mathrm{PRG}_1 \colon \{0,1\}^{\lambda} \to \mathbb{F}_{p^l}$ is a pseudorandom generator. We discuss the required size of l depending on p and λ in Section 4. We set $z_0 = \sum_{i=0}^{N-1} z_0{}^{[\![i]\!]}$ and compute the offset values $\delta_{z_j} = z_0 - \sum_{i=0}^{N-1} z_j{}^{[\![i]\!]}$ for $j \in [\tau]^*$.

By construction, we obtain τ values of $\vec{f}_j$ and z_j from the GGM tree family and the offset values, one for each of the τ subtrees. In particular, we have that

$$\vec{f}_j = \sum_{i=0}^{N-1} \vec{f}_j^{[\![i]\!]} + \delta_{\vec{f}_j} \text{ and } z_j = \sum_{i=0}^{N-1} z_j{}^{[\![i]\!]} + \delta_{z_j}$$

for $j \in [\tau]$, where $\delta_{z_0} = 0$. The idea of the consistency check is to convince the verifier that the values of $(\vec{f}_j, z_j)$ are equal across all rounds $j \in [\tau]$. In particular,

we want that $(\vec{f}_j, z_j) = (\vec{f}, z)$ for $j \in [\tau]$. The purpose of z is to act as a one time pad to hide the value of $\vec{f}$ in the equality test. In the hypercube setting, each subtree T_j induces a vectorial function of the form

$$\vec{g}_{i,j}(b) = \left(\vec{f}_j^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + b \cdot \left(\vec{f}_j, z_j\right),$$

where the verifier chooses $b_{i,j}$ and learns $\vec{g}_{i,j}(b_{i,j}) = \left(\vec{f}_j^{[\![b_{i,j}]\!]_i}, {z_j}^{[\![b_{i,j}]\!]_i}\right)$. Using this representation, we can pick random coefficients $\alpha_m \in \mathbb{F}_{2^l}$ for $m \in [\tau \cdot D]$ and define the function

$$\begin{aligned}\vec{G}^{(j)}_{\vec{f},z}(\Delta_j) &= \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, \vec{g}_{i,j}(b_{i,j}) \\ &= \sum_{i=0}^{D-1} \alpha_{j\cdot D+i} \left(\vec{f}_j^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + \Delta_j \cdot \left(\vec{f}_j, z_j\right),\end{aligned}$$

corresponding to each the subtree T_j, where $\Delta_j = \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, b_{i,j}$. We can write this function as

$$\vec{G}^{(j)}_{\vec{f},z}(\Delta_j) = \vec{A}^{(j)}_{G_{\vec{f},z}} + \Delta_j \cdot (\vec{f}_j, z_j), \tag{2}$$

where $\vec{A}^{(j)}_{G_{\vec{f},z}} = \vec{G}^{(j)}_{\vec{f},z}(0)$. Note that each $\vec{G}^{(j)}_{\vec{f},z}$ is an affine function known by the prover, while the verifier chooses the evaluation point Δ^*_j and learns the value $\vec{G}^{(j)}_{\vec{f},z}(\Delta^*_j)$ using the PPRF key. After the prover has committed to the GGM tree family, the verifier computes a random vector $\vec{\mu} \in (\mathbb{F}_{p^l})^{|\mathcal{F}|}$ and sends it to the prover. The consistency check relies on the following Lemma:

Lemma 1 ([HJ24b]). *Let $\vec{f}, \vec{f'} \in (\mathbb{F}_p)^{|\mathcal{F}|}$ and $z, z' \in \mathbb{F}_{p^l}$ with $(\vec{f}, z) \neq (\vec{f'}, z')$ and let $\vec{\mu} \leftarrow_{\$} (\mathbb{F}_{p^l})^{|\mathcal{F}|}$. Then $\Pr[z + \vec{\mu}\cdot\vec{f} = z' + \vec{\mu}\cdot\vec{f'}] \leq p^{-l}$, where the probability is on the choice of $\vec{\mu}$.*

Multiplying (2) by $(\vec{\mu}, 1)$, we obtain

$$\begin{aligned}\vec{G}^{(j)}_{\vec{f},z}(\Delta_j) \cdot (\vec{\mu}, 1) &= \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) + \Delta_j(\vec{f}_j, z_j) \cdot (\vec{\mu}, 1) \\ &= \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) + \Delta_j[z_j + \vec{f}_j \cdot \vec{\mu}]\end{aligned}$$

Note that, if all $(\vec{f}_j, z_j)$ are indeed equal for all rounds, then the values $z_j + \vec{f}_j \cdot \vec{\mu}$ are identical for all $j \in [\tau]$. Therefore, the prover responds by sending the coefficients $a^{(j)} = \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) \in \mathbb{F}_{p^l}$ for $j \in [\tau]$ and just one coefficient $c = z + \vec{f} \cdot \vec{\mu}$ to the verifier. With the PPRF key, the verifier can recompute Δ^*_j and recompute the values of $\vec{G}^{(j)}_{\vec{f},z}(\Delta^*_j)$ for $j \in [\tau]$. Finally, the verifier checks if

$$\vec{G}^{(j)}_{\vec{f},z}(\Delta^*_j) \cdot (\vec{\mu}, 1) = a^{(j)} + c\Delta^*_j$$

for all $j \in [\tau]$. If this consistency check succeeds, the verifier proceeds with the protocol, otherwise the verifier rejects.

As shown in [HJ24b], the false positive rate of this consistency check is bounded by $\tau^2 p^{-l}$. We provide the pseudocode for the Fherret scheme including the consistency check. In Figure 2, we describe the adapted Commitment scheme including the consistency check. In Figure 3, we describe the respective Challenge and Response algorithms between prover and verifier. In Figure 4, we provide the non-interactive version of theses algorithms by using the random oracle $\mathcal{H}$ with proper domain separation to derive the challenges. The Verification algorithm including the consistency check is given in Figure 5.

Remark 1. (Using AES in the tree derivation.) As explained above, using internal nodes of just λ bits reduces the size of the PPRF key of our scheme compared to a version using big internal nodes of size $|\mathcal{F}|$. For $\lambda = 128$, using internal nodes of size λ bits has another advantage: We can use an internal tree derivation function based on the AES block cipher instead of using a (salted) hash function, which is a technique introduced in [BCC$^+$24]. This allows us to take advantage of the AES instruction set used in many recent CPU architectures, resulting in an improvement of the running time of our scheme. For the adapted tree derivation based on AES, we need two AES keys K_0 and K_1 to derive the children of a node on a given level of the tree in the cGGM setting. The left child of a node Y is set to $\mathrm{AES}_{K_0}(Y) \oplus \mathrm{AES}_{K_1}(Y)$, while the right child is given by $Y \oplus \mathrm{AES}_{K_0}(Y) \oplus \mathrm{AES}_{K_1}(Y)$ to achieve a XOR preserving construction. Instead of sending the salt of size 2λ bits, we send the two keys K_0, K_1 of size λ bits each in this version. Therefore, the communication cost remains the same.

3.4 A Secure Version of the Scheme

In this section, we prove the security of the scheme.

Prover privacy In many works on verifiable computation (VC) over fully homomorphic encryption (FHE), circuit privacy has been modeled as a context-hiding property of the VC scheme [BCFK21,GNS23]. While this model is useful for describing security when applying zk-SNARKs in the ciphertext space, recent VC works that operate in the plaintext space [ACGS24] have introduced the notion of honest-verifier prover privacy (HVPP) to describe the security of the prover's function.

We adapt this definition to prove that, any information a semi-honest verifier $\mathcal{V}$ can compute by participating in the protocol, $\mathcal{V}$ could compute using only its input and prescribed output. To show this we provide a simulator $\mathcal{S}$ that is able to produce an output computationally indistinguishable from a transcript of the real protocol using only elements that are already known to $\mathcal{V}$. We remark that considering a semi-honest verifier is in line with the majority of the works on circuit privacy. [4]

[4] This definition (and the security of our scheme) can be adapted to the malicious verifier setting by considering malicious circuit private FHE schemes ([DD22,OPP14]).

Definition 5 (Honest-verifier prover-privacy (HVPP), adapted from [ACGS24]). *A protocol is said to satisfy honest-verifier prover privacy if there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ such that for every function $f \in \mathcal{F}$, the following distributions are computationally indistinguishable:*

$$\{sk, \mathcal{S}(\lambda, pk, m, \mathsf{ct}, f(m))\} \approx_{\mathsf{c}} \{sk, \boldsymbol{View}_{\mathcal{V}}(\lambda, sk, pk, m, \mathsf{ct}, f)\}$$

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\lambda)$ *and* $\boldsymbol{View}_{\mathcal{V}}$ *denotes the view of the* $\mathcal{V}$ *during an execution of the protocol on client's input* (m, ct)*, where* $\mathsf{ct} \leftarrow \mathtt{Enc}(\mathsf{sk}, m)$*, on server's input* f *and security parameter* λ*.*

Theorem 1. *The scheme from Figure 2 satisfies HVPP.*

Proof. Deferred to Appendix A.

Verifier security. To protect the verifier from malicious behavior, we must prove that an adversary cannot exploit the decryption or verification process extract sensitive information. In particular, such attacks require the adversary to create a dependency between the verifier's output and private components, such as the verifier's secret key sk or the message m encrypted in the ciphertext ct.

In cryptographic literature, reaction-based attacks are typically modeled using ideal oracles which provide abstract interfaces to the verifier's behavior. These oracles allow us to reason about what an attacker could learn under well-defined access patterns.

To establish the security of our scheme against such attacks, we show that the outputs of these oracles can be efficiently simulated using only public information and the expected output of the protocol. This implies that an adversary cannot obtain any additional useful information beyond what is provided as input to the simulator.

To support these claims, we begin by proving the following theorem on the scheme from Figure 2 .

Theorem 2 (Function Commitment). *Let the verifier's output be not* $\perp$*. Then, in the Random Oracle Model (ROM), there exists a polynomial-time extractor that can recover the function* $f \in \mathcal{F}$ *used by the server with negligible failure probability. Furthermore, except with negligible probability, the output* m_{maj} *computed by the verifier satisfies:*

$$m_{\mathsf{maj}} = f(m),$$

where m *is the message encrypted in the input ciphertext,* f *is the extracted function.*

Proof. Deferred to Appendix B.

This theorem establishes two essential properties:

1. The prover is bound to a specific function $f \in \mathcal{F}$ in order for the verifier to produce a valid output.
2. The protocol's output consistently corresponds to the evaluation of f on the original message m.

Verifier security against IND-CPA$^{\mathsf{D}}$-style attacks and verification oracle attacks. Correctness oracles reveal to the attacker when the result of the homomorphic evaluation of a function f on a ciphertext encrypting m gets decrypted to something different from $f(m)$.

Verification oracles are used to describe when the ciphertext returned by the prover is tampered with in a way that might change the verifier's output of the protocol.

We show that both of these oracles can be simulated from elements already available to the prover.

Theorem 3. *Considering the protocol from Figure 2, there exist two probabilistic polynomial-time (PPT) simulators $\mathcal{S}_{\mathsf{Ver}}$ and $\mathcal{S}_{\mathsf{Corr}}$, such that*

$$\mathcal{O}_{\mathsf{Ver}}(\mathsf{sk}, \mathcal{T}) \overset{c}{\approx} \mathcal{S}_{\mathsf{Ver}}(\mathsf{pk}, \mathcal{T}) \quad \textit{and} \quad \mathcal{O}_{\mathsf{Corr}}(\mathsf{sk}, \mathcal{T}) \overset{c}{\approx} \mathcal{S}_{\mathsf{Corr}}(\mathsf{pk}, \mathcal{T}),$$

where we define the transcript of a protocol as

$$\mathcal{T} \leftarrow \boldsymbol{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f),$$

and $\mathcal{O}_{\mathsf{Ver}}, \mathcal{O}_{\mathsf{Corr}}$ as the verification oracle and as the correctness oracle.

Proof. Deferred to Appendix C.

To illustrate this more concretely, we refer to the attack strategies in [CCP+24], [CSBB24]. The adversary encrypts zero and then maliciously evaluates a function homomorphically. For example, the adversary homomorphically multiplies the ciphertext for a large power of two without bootstrapping, despite the FHE evaluation algorithm requiring it. This breaks correctness, yielding a nonzero result.

However, in our scheme, the verifier detects inconsistencies by comparing the expected ciphertexts from honest homomorphic evaluation against those produced by the adversary's modified evaluation. So, the verifier rejects these ciphertexts without decrypting them.

Thus, with these two oracles, an attacker gains no information about the secret key sk or the original message m.

Verifier security against Decryption Oracle attacks Decryption oracles reveal to the attacker the output of the protocol, typically the decryption of the ciphertext returned by the prover, asking the attacker to infer additional information from this, such as attempting to recover the challenger's secret key sk.

We show that, in our scheme, we can describe in detail the leakage deriving from a decryption oracle.

Theorem 4. *Considering the protocol from Figure 2, there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}_{\mathsf{Dec}}$, such that*

$$\mathcal{O}_{\mathsf{Dec}}(\mathsf{sk}, \mathcal{T}) \overset{c}{\approx} \mathcal{S}_{\mathsf{Dec}}(\mathsf{pk}, \mathcal{T}, f(m)),$$

where we define the transcript of a protocol as

$$\mathcal{T} \leftarrow \boldsymbol{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f),$$

f as the (extractable) function used by the prover, m as the message encrypted in the input ciphertext from the verifier and $\mathcal{O}_{\mathsf{Dec}}$ as the decryption oracle.

Proof. Deferred to Appendix C.

As expected in such an attack, this reveals $f(m)$ and provides some information about m. However, crucially, the output remains independent of the secret key sk.

Public Verifiability. We observe that the verification protocol (Figure 5) can be carried out almost entirely, specifically points 1 to 8, without requiring access to sensitive elements from either the verifier or the prover. In particular, the verifier's secret key sk, input message m, and the prover's function f are not needed.

This property allows the verifier to delegate the computationally expensive portion of the verification process to an external trusted party. The trusted party can independently verify the prover's honest behavior, compute the intermediate ciphertexts up to step 8, and then return these ciphertexts to the verifier. The verifier can then complete the protocol by simply decrypting the final ciphertexts.

4 Parameters and Implementation

Fherret is not scheme-specific and can be applied to any (non-approximate) FHE scheme. While we make particular choices regarding schemes and parameters, these choices are not mandatory for implementing Fherret. In this proof of concept, we focus on estimating the runtime for both the prover and the verifier, and we discuss implementation-related optimizations.

4.1 Setting Fherret Parameters

Depending on the efficiency of the hash function used in the tree derivation, many MPCitH schemes can feasibly compute sharings for up to $N = 2^{16}$ parties within a reasonable computation time. To reduce the communication cost of Fherret, it is desirable to use a large number of parties N and a small number of rounds τ. In contrast, the computational overhead for both the prover and the verifier in constructing the trees is reduced when using a smaller number of parties N and a larger number of rounds τ.

To achieve a desired security level of λ bits, we must choose N and τ such that $\log(N) \cdot \frac{\tau}{2} \geq \lambda$. Therefore, depending on the desired trade-offs in efficiency and communication, the user can select these parameters accordingly.

Including the consistency check, we must choose parameters $N = 2^D$ and τ such that the overall security of the scheme is at least λ bits.

Table 1. Running times of Fherret for $\lambda = 130$ and $(D, \tau) = (13, 20)$ using twin AMD EPYC 9374F processors running at 3.85 GHz.

	Commitment (Prover)						Verification (Verifier)					
	MPCitH			FHE		total	MPCitH			FHE		total
$\mathcal{F}_{d,k}$	GGM	PRG + Fold	$\mathcal{H}$	Monomials	Coeff		GGM	PRG + Fold	$\mathcal{H}$	Monomials	Coeff	
Single core												
$\mathcal{F}_{2,4}$	0.4 s	3.2 s	4.2 s	0.12 s	29.4 s	37.5 s	0.4 s	3.0 s	4.2 s	0.12 s	14.7 s	22.5 s
$\mathcal{F}_{2,10}$	0.4 s	15.7 s	4.3 s	0.67 s	130 s	151 s	0.4 s	14.9 s	4.4 s	0.67 s	65 s	86 s
$\mathcal{F}_{2,20}$	0.4 s	67 s	4.6 s	2.5 s	458 s	534 s	0.4 s	49 s	4.7 s	2.6 s	235 s	291 s
$\mathcal{F}_{3,4}$	0.4 s	7.5 s	4.4 s	0.4 s	80 s	93 s	0.4 s	7.8 s	4.5 s	0.4 s	38 s	51 s
$\mathcal{F}_{4,4}$	0.4 s	14.6 s	4.4 s	1.1 s	166 s	187 s	0.4 s	13.9 s	4.4 s	1.1 s	84 s	104 s
$\mathcal{F}_{5,4}$	0.4 s	24.7 s	4.5 s	2.5 s	323 s	356 s	0.4 s	22.6 s	4.4 s	2.5 s	160 s	189 s
$\mathcal{F}_{6,4}$	0.4 s	41.6 s	8.6 s	9.3 s	1576 s	1639 s	0.4 s	34.4 s	8.6 s	9.3 s	659 s	714 s
Using 128 threads in parallel for the FHE evaluation												
$\mathcal{F}_{2,4}$	0.4 s	3.2 s	4.2 s	0.12 s	1.8 s	9.6 s	0.4 s	3.0 s	4.2 s	0.12 s	0.93 s	8.8 s
$\mathcal{F}_{2,10}$	0.4 s	15.7 s	4.3 s	0.67 s	7.1 s	28 s	0.4 s	14.9 s	4.4 s	0.67 s	3.7 s	25 s
$\mathcal{F}_{2,20}$	0.4 s	67 s	4.6 s	2.5 s	24.4 s	103 s	0.4 s	49 s	4.7 s	2.6 s	12.4 s	72 s
$\mathcal{F}_{3,4}$	0.4 s	7.5 s	4.4 s	0.4 s	4.9 s	17.7 s	0.4 s	7.8 s	4.5 s	0.4 s	2.5 s	15.6 s
$\mathcal{F}_{4,4}$	0.4 s	14.6 s	4.4 s	1.1 s	11.7 s	32.6 s	0.4 s	13.7 s	4.4 s	1.1 s	6.0 s	25.9 s
$\mathcal{F}_{5,4}$	0.4 s	24.7 s	4.5 s	2.5 s	24.9 s	58 s	0.4 s	22.6 s	4.4 s	2.5 s	12.3 s	43 s
$\mathcal{F}_{6,4}$	0.4 s	41.6 s	8.6 s	9.3 s	74 s	135 s	0.4 s	34.4 s	8.6 s	9.3 s	37 s	92 s
$\mathcal{F}_{8,4}$	0.4 s	94 s	8.7 s	31 s	239 s	379 s	0.4 s	75 s	9.0 s	31 s	123 s	244 s
$\mathcal{F}_{10,4}$	0.4 s	200 s	9.0 s	72 s	587 s	879 s	0.4 s	159 s	9.3 s	71 s	303 s	553 s

By the union bound, the cheating probability of a malicious prover is upper bounded by $\frac{1}{N^{\tau/2}} + \tau^2 p^{-l}$, where p is the plaintext modulus of the FHE scheme. For $\lambda = 128$, we can, for instance, set $l = \lceil 256/\log(p) \rceil$ to achieve a negligible failure probability in the consistency check. The total communication cost of the Fherret scheme is given by

$$\text{size} \approx 9\lambda + 4\tau\lambda + 8\lambda^2 + 2\lambda \cdot |\mathsf{ct}| + 2\tau \cdot |f|.$$

For a computation of this size, see Appendix D.

A comparison of the resulting bit-security level and communication size for various values of D and τ is provided in Table 2.

4.2 Choosing $\mathcal{F}$

In the definition of circuit privacy, the choice of the function space from which the evaluated function is drawn is crucial. Most works on circuit privacy consider function spaces consisting of all polynomials with a fixed degree d and a fixed number of variables k. We denote this space as $\mathcal{F}_{d,k}$.

Table 2. Examples of parameter choices for D and τ and the resulting bit security and communication size of the Fherret protocol using $l = \lceil 2\lambda/\log(p) \rceil$.

D	τ	bit security level	size (bits)
15	18	135	$270 \cdot \lvert\mathsf{ct}\rvert + 36 \cdot \lvert f \rvert + 156735$
14	19	133	$266 \cdot \lvert\mathsf{ct}\rvert + 38 \cdot \lvert f \rvert + 152817$
13	20	130	$260 \cdot \lvert\mathsf{ct}\rvert + 40 \cdot \lvert f \rvert + 146770$
12	22	132	$264 \cdot \lvert\mathsf{ct}\rvert + 44 \cdot \lvert f \rvert + 152196$
11	24	132	$264 \cdot \lvert\mathsf{ct}\rvert + 48 \cdot \lvert f \rvert + 153252$
10	26	130	$260 \cdot \lvert\mathsf{ct}\rvert + 52 \cdot \lvert f \rvert + 149890$

In $\mathcal{F}_{d,k}$, each function contains $\binom{d+k+1}{k+1}$ monomials. Therefore, the size of its description is proportional to the number of monomials, multiplied by the bit length of the message space (that in our case is the bit size of the plaintext modulus).

In specific applications, the topology of the function (i.e., the structure of the monomials) may not need to remain hidden, and only the coefficients are considered sensitive. In such cases, the additive sharing can be performed over a significantly smaller function space $\mathcal{F}$. This allows the performance of homomorphic evaluation over a random function in $\mathcal{F}$ to better approximate the actual cost of evaluating the prover's function.

However, we stress that this optimization is not suitable for all scenarios. It can only be safely applied to function spaces that are large enough to resist attacks capable of recovering the function with a limited number of queries.

4.3 Improving the Timings for Repeated FHE Evaluations

In Step (7) of the commitment phase (Figure 2) and Step (4) of the verification phase (Figure 5), we require the homomorphic evaluation of a total of 4λ and 2λ random functions from $\mathcal{F}$, respectively.

We can significantly reduce the runtime of these evaluations by leveraging the structure of the function space $\mathcal{F}_{d,k}$. Specifically, each polynomial in $\mathcal{F}_{d,k}$ can be expressed as a sum of monomials of the form $x_1^{i_1} \dots x_k^{i_k}$, for $0 \leq i_1 + \dots + i_k \leq d$, each multiplied by its corresponding coefficient $c_{i_1,\dots,i_k}$.

Crucially, the homomorphic evaluation of these monomials is independent of the coefficients and thus common to all the functions being evaluated. This allows us to compute all relevant monomials once and reuse them across all function evaluations.

This reuse strategy becomes especially impactful when $\mathcal{F}$ includes polynomials of high degree or a large number of variables. In such cases, amortizing the cost of monomial evaluations significantly reduces the overall overhead associated with repeated FHE evaluations.

4.4 Implementation and Running Times

We provide the running times of our scheme based on a proof-of-concept implementation in C++. As the underlying FHE scheme, we use the BGV-RNS scheme, as implemented in the OpenFHE library [BAB+22]. We use a plaintext modulus $p = 65537$ and perform a packing of 12 messages per ciphertext.

Currently, the major FHE libraries supporting BGV and BFV do not provide implementations that guarantee circuit privacy. Therefore, **our proof-of-concept implementation ensures security only for the verifier**. Nonetheless, since Fherret's overhead is proportional to the cost of FHE evaluations, the *relative* impact on performance would remain comparable even when circuit privacy is enforced.

Our implementation targets the $\lambda = 128$ bit security level, leveraging the optimizations discussed in Section 3.3. Among the various parameter choices listed in Table 2, the configuration $(D, \tau) = (13, 20)$ offered the best trade-off between tree generation and FHE evaluation costs. This choice resulted in the shortest overall running times for both the prover and the verifier.

In Table 1, we report the running times of Fherret for various choices of the function space $\mathcal{F}$. Specifically, we consider executions that guarantee circuit privacy over function spaces $\mathcal{F}_{d,k}$, defined by polynomials of fixed depth d and a fixed number of variables k.

The table includes partial timings to illustrate the asymptotic scaling behavior of different components of the protocol. Notably, the tree derivation process depends only on the security parameter and is independent of $\mathcal{F}$. In contrast, the PRG expansion and hypercube folding steps scale linearly with the number of monomials in $\mathcal{F}$. The column labeled $\mathcal{H}$ aggregates the timings for all steps occurring after the FHE evaluations; this includes hashing, the generation (or verification) of the VOLE proof, and the construction of the PPRF key. Among these, the hashing time is the dominant factor and also scales linearly with the number of monomials.

We also break down the FHE evaluation timings into two components: the evaluation of monomials and the evaluation of the 4λ sets of coefficients (2λ for the verifier). By examining single-core timings, we quantify the impact of the optimization introduced in Section 4.3. Without this optimization, monomial evaluation would be redundantly performed 520 times by the prover and 260 times by the verifier. By sharing monomial computations across evaluations, we reduce the total homomorphic evaluation time by approximately 67% to 80%, with the savings increasing as the size of $\mathcal{F}$ grows.

Finally, we also report timings using 128 parallel threads for FHE evaluations, demonstrating the scalability of Fherret. This parallelization enables efficient evaluation even for functions with large depth or high number of variables.

5 Conclusion

In this work, we introduced Fherret, a novel proof system for FHE schemes that leverages the MPC-in-the-Head (MPCitH) paradigm to certify that the

homomorphic evaluation was performed correctly and honestly. Fherret protects the verifier from reaction-based attacks and preserves circuit privacy for the evaluated function.

References

ABPS24. Shahla Atapoor, Karim Baghery, Hilder V. L. Pereira, and Jannik Spiessens. Verifiable FHE via lattice-based SNARKs. *IACR Communications in Cryptology*, 1(1), 2024.

ACGS24. Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part V*, volume 15488 of *LNCS*, pages 302–334. Springer, Singapore, December 2024.

AGH+23. Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 564–596. Springer, Cham, April 2023.

AGHV22. Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 70–99. Springer, Cham, November 2022.

AV21. Adi Akavia and Margarita Vald. On the privacy of protocols based on CPA-secure homomorphic encryption. Cryptology ePrint Archive, Report 2021/803, 2021.

BAB+22. Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022.

BBD+23. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

BCC+24. Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. Faster signatures from MPC-in-the-head. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 396–428. Springer, Singapore, December 2024.

BCFK21. Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 528–558. Springer, Cham, May 2021.

BFG+24. Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. Cryptology ePrint Archive, Paper 2024/541, 2024. `https://eprint.iacr.org/2024/541`.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

Bui24. Dung Bui. Shorter VOLEitH signature from multivariate quadratic. Cryptology ePrint Archive, Paper 2024/465, 2024. `https://eprint.iacr.org/2024/465`.

CCC+25. Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. Cryptology ePrint Archive, Paper 2025/286, 2025.

CCCM22. Bhuvnesh Chaturvedi, Anirban Chakraborty, Ayantika Chatterjee, and Debdeep Mukhopadhyay. A practical full key recovery attack on TFHE and FHEW by inducing decryption errors. Cryptology ePrint Archive, Report 2022/1563, 2022.

CCP+24. Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA$^{\mathrm{D}}$ security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2505–2519. ACM Press, October 2024.

CF13. Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Berlin, Heidelberg, May 2013.

CGG16. Ilaria Chillotti, Nicolas Gama, and Louis Goubin. Attacking FHE-based applications by software fault injections. Cryptology ePrint Archive, Report 2016/1164, 2016.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

CKP+24. Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. VERITAS: Plaintext encoders for practical verifiable homomorphic encryption. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2520–2534. ACM Press, October 2024.

CLY+24. Hongrui Cui, Hanlin Liu, Di Yan, Kang Yang, Yu Yu, and Kaiyi Zhang. ReSolveD: Shorter signatures from regular syndrome decoding and VOLE-in-the-head. Cryptology ePrint Archive, Paper 2024/040, 2024. `https://eprint.iacr.org/2024/040`.

CSBB24. Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPA^{D} security of "exact" and threshold FHE schemes and libraries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 3–33. Springer, Cham, August 2024.

DD22. Nico Döttling and Jesko Dujmovic. Maliciously circuit-private FHE from information-theoretic principles. In Dana Dachman-Soled, editor, *ITC 2022*, volume 230 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl, July 2022.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

DSA13. Angsuman Das, Dutta Sabyasachi, and Adhikari Avishek. Indistinguishability against chosen ciphertext verification attack revisited: The complete picture. *Provable Security: 7th International Conference, ProvSec*, 2013.

FGP14. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 844–855. ACM Press, November 2014.

FNP20. Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Cham, May 2020.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

GBK+24. Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. Cryptology ePrint Archive, Paper 2024/1684, 2024.

GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

GGW24. Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 449–487. Springer, Cham, August 2024.

GNS21. Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. Cryptology ePrint Archive, Report 2021/322, 2021.

GNS23. Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. *Journal of Cryptology*, 36(4):41, October 2023.

GW13. Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Berlin, Heidelberg, December 2013.

HJ24a. Janik Huth and Antoine Joux. MPC in the head using the subfield bilinear collision problem. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 39–70. Springer, Cham, August 2024.

HJ24b. Janik Huth and Antoine Joux. VOLE-in-the-head signatures from subfield bilinear collisions. Cryptology ePrint Archive, Paper 2024/1537, 2024.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, October 2021.

LMSV12. Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, August 2012.

MN24. Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 63–93. Springer, Cham, May 2024.

NLDD21. Deepika Natarajan, Andrew Loveless, Wei Dai, and Ronald Dreslinski. CHEX-MIX: Combining homomorphic encryption with trusted execution environments for two-party oblivious inference in the cloud. Cryptology ePrint Archive, Paper 2021/1603, 2021.

OPP14. Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 536–553. Springer, Berlin, Heidelberg, August 2014.

Riv87. Ronald L Rivest. A method for obtaining digital signature and public-key cryptosystems. *ACM*, 21:2, 1987.

Roy22. Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022.

VKH23. Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption. *arXiv preprint arXiv:2301.07041*, 2023.

ZPS12. Zhenfei Zhang, Thomas Plantard, and Willy Susilo. Reaction attack on outsourced computing with fully homomorphic encryption schemes. In Howon Kim, editor, *ICISC 11*, volume 7259 of *LNCS*, pages 419–436. Springer, Berlin, Heidelberg, November / December 2012.

ZWL+25. Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, and Xianhui Lu. FHE-SNARK vs. SNARK-FHE: From analysis to practical verifiable computation. Cryptology ePrint Archive, Paper 2025/302, 2025.

Appendix

A Proof of Theorem 1

We write explicitly $\mathbf{View}_{\mathcal{V}}(\lambda, sk, pk, m, \mathsf{ct}, f)$ as

$$\left(\mathsf{pprf\text{-}keys}, \mathsf{offsets}, (X_j)_{j\in[\tau]}, \mathsf{com}\right),$$

where

$$\mathsf{pprf\text{-}keys} = \{(i^*_j)_{j\in[\tau]}, K_{i^*_{pre}}, K_{(i^*_0,\ldots,i^*_{\tau-1})}, \mathrm{salt}\},$$

and

$$\mathsf{offsets} = \{(\delta_f)_{j\in[\tau]}, (\delta_z)_{j\in[\tau]^*}, (a^{(j)})_{j\in[\tau]}, c\}.$$

We can construct $\mathcal{S}(\lambda, pk, m, \mathsf{ct}, f(m))$ in the following way:

1. Sample $g \leftarrow_{\$} \mathcal{F}$ and compute $\bar{g} \leftarrow g + (f(m) - g(m))$.
2. Follow honestly the protocol by using $\mathsf{pk}, \mathsf{ct}, \bar{g}$ up to step (6).
3. In step (7), looking at the second challenge $(b^*_{i,j})_{(i,j)\in[D]\times[\tau]}$, substitute the ciphertexts that are hidden to the verifier by using the simulator from circuit privacy
$$\mathsf{ct}_{i,j}^{b^{\bar{*}}_{i,j}} \leftarrow \mathcal{S}_{\mathsf{CP}}(\mathsf{pk}, f_j^{[\![b^{\bar{*}}_{i,j}]\!]_i}(m)).$$
4. Complete the rest of the protocol honestly.

Having a look at the outputs of the simulator we have that:

1. The $\mathsf{pprf\text{-}keys}$ and $\mathsf{offsets}$ behave like random elements in the ROM.
2. $(X_j)_{j\in[\tau]}$ and com are computationally indistinguishable from a real execution of the protocol because of the circuit privacy simulator.

B Proof of Theorem 2

Having a meaningful output means that the VOLE check and the hash checks are successful. If the VOLE check is true, the function

$$f_j^{[\![0]\!]_i} + f_j^{[\![1]\!]_i} + \delta_{f_j}$$

is always the same for each $j \in [\tau]$ and $i \in [D]$, where the $f_j^{[\![b]\!]_i}$ are the functions reconstructed during verification from $\mathsf{pprf\_keys}$. Since each $f_j^{[\![b]\!]_i}$ is computed by using a PRF, that we model as a random oracle, the probability of constructing a working VOLE proof compatible with $\mathsf{pprf\_keys}$ without querying the random oracle to obtain the $f_j^{[\![b]\!]_i}$ is negligible.

This means that we can recover all of the $f_j^{[\![b]\!]_i}$ by reading the random oracle memory. Now, since the hash checks verify, we know that the $\mathsf{offsets}$ sent by the verifier are the same ones used in the VOLE proof. Since this proof also verifies we can extract the prover's function as the common value

$$f := f_j^{[\![0]\!]_i} + f_j^{[\![1]\!]_i} + \delta_{f_j}.$$

Finally, thanks to the hash checks, we know that all the ciphertexts queried in a challenge are computed correctly as

$$\mathsf{ct}_{i,j}^{b^*_{i,j}} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, f_j^{[\![b_{i,j}]\!]_i}, \mathsf{ct}),$$

therefore the probability of modifying the majority of these ciphertexts without the hash check failing is negligible.

This implies that at least half of the decrypted ciphertexts will correctly decrypt to $f(m)$.

C Proof of Theorems 3 and 4

We observe that the verification protocol (Figure 5) can be carried out almost entirely, specifically steps (1) to (8), without requiring access to sensitive elements from either the verifier or the prover. In particular, the verifier's secret key sk, input message m, and the prover's function f are not needed.

We refer to this partial verification, which performs only the VOLE and hash checks, as $\mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T})$, where we define $\mathcal{T}$ as the transcript of the protocol

$$\mathcal{T} \leftarrow \mathbf{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f).$$

We define the three oracles as follows:

$$\mathcal{S}_{\mathsf{Ver}}(\mathsf{pk}, \mathcal{T}) = \mathcal{S}_{\mathsf{Corr}}(\mathsf{pk}, \mathcal{T}) = \begin{cases} \mathsf{true} & \text{if} \quad \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true} \\ \mathsf{false} & \text{if} \quad \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{false} \end{cases}$$

$$\mathcal{S}_{\mathsf{Dec}}(\mathsf{pk}, \mathcal{T}, f(m)) = \begin{cases} \mathsf{f(m)} & \text{if} \quad \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true} \\ \perp & \text{if} \quad \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{false} \end{cases}$$

Theorem 2 grants us that, except for a negligible probability, the output of the protocol is $f(m)$ if $\mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true}$ and is $\perp$ otherwise. This means that these simulators will behave in the same way as the oracles they simulate.

D Communication Cost

The total communication cost of Fherret includes:

- a global commitment hash com of size 2λ
- a salt of size 2λ
- a commitment to the random coins of size 2λ
- $\tau \cdot D = 2\lambda$ ciphertexts $\mathsf{ct}_{i,j}$
- τ offsets $\vec{f_j}$ and $(\tau - 1)$ offsets z_j of size $\tau \cdot |f| + (\tau - 1) \cdot 2\lambda$
- the vector $\vec{\mu} \in (\mathbb{F}_{p^l})^{|\mathcal{F}|}$ can be recomputed using a PRG on a seed of size λ
- the elements $(\alpha_m)_{m \in [\tau \cdot D]}$ of size $\tau \cdot D \cdot 2\lambda \approx 4\lambda^2$
- the vectors $\left(a^{(j)}\right)_{j \in [\tau]}$ of size $\approx \tau \cdot |f|$
- one element c of size 2λ
- a PPRF key of size $\tau \cdot \lambda D \approx 2\lambda^2$
- the random coins $\sigma_{i,j}$ of size $(\tau - 1) \cdot D \cdot \lambda \approx 2\lambda^2$
- the hashes $(h^{\sigma}_{i^*_j, j})_{j \in [\tau]}$ of size $\tau \cdot 2\lambda$

In all computations, we use that $\tau D \approx 2\lambda$. This yields a total communication size in bits of

$$\text{size} \approx 9\lambda + 4\tau\lambda + 8\lambda^2 + 2\lambda \cdot |\mathsf{ct}| + 2\tau \cdot |f|.$$