Cryptanalysis on Lightweight Verifiable Homomorphic Encryption

Jung Hee Cheon[1,2], Daehyun Jang[1]

[1] Seoul National University, Seoul, Republic of Korea
{jhcheon, jadh0309}@snu.ac.kr
[2] CryptoLab Inc., Seoul, Republic of Korea

Abstract. Verifiable Homomorphic Encryption (VHE) is a cryptographic technique that integrates Homomorphic Encryption (HE) with Verifiable Computation (VC). It serves as a crucial technology for ensuring both privacy and integrity in outsourced computation, where a client sends input ciphertexts ct and a function f to a server and verifies the correctness of the evaluation upon receiving the evaluation result $f(\mathsf{ct})$ from the server.

Chatel et al. [CKP+24] (CCS'24) introduced two VHE schemes: Replication Encoding (REP) and Polynomial Encoding (PE). A similar approach to REP was used by Albrecht et al. [ADDG24] (EUROCRYPT'24) to develop a Verifiable Oblivious PRF scheme (vADDG). A key approach in these schemes is to embed specific secret information within ciphertexts and use them to verify homomorphic evaluations.

This paper presents efficient forgery attacks against the verifiability guarantees of these VHE schemes. We introduce two attack strategies. The first targets REP and vADDG, extracting secret information in encrypted form from input ciphertexts and leveraging it to forge output ciphertexts without being detected by the verification algorithm. The second targets PE, exploiting its secret embedding structure to forge output ciphertexts that remain valid on input values for verification, yet violate the verifiability property.

Our forgery attack on vADDG demonstrates that the proposed 80-bit security parameters provide at most 10 bits of concrete security. Our attack on REP and PE achieves a probability 1 attack with linear time complexity when using fully homomorphic encryption.

Keywords: Verifiable Homomorphic Encryption · Verifiable OPRF · Forgery Attacks · Cryptanalysis

1 Introduction

Verifiable Computation (VC) is a technique that guarantees the correctness of the result of a computation outsourced by a client [GGP10]. This technique allows the client to detect and prevent erroneous or malicious computations by the server. This ensures the integrity of the computations. On the other hand, Homomorphic Encryption (HE) allows computations to be performed on encrypted

data without having to decrypt. HE allows the privacy of sensitive information to be maintained while complex calculations are outsourced to the server. Overall, in outsourced computing, VC ensures the integrity of the computational results, while HE guarantees the privacy of the client's data. Verifiable Homomorphic Encryption (VHE) is the combination of these two technologies. A naive approach to VHE is the use of the SNARK or other VC techniques for homomorphic computation on encrypted data [ABPS24]. However, this approach results in a highly inefficient solution due to several non-arithmetic operations required in HE. These operations do not integrate seamlessly with existing VC techniques.

Alternatively, other lines of research [CKP+24, ADDG24] have explored the design of efficient verifiable homomorphic encryption schemes without combining VC techniques with HE schemes. These schemes aim to achieve VHE at lower computational cost by concealing specific secret information within ciphertexts and precomputed values, and using them to ensure verifiability. In this paper, we review such *'lightweight'* verifiable homomorphic encryption schemes and demonstrate that these approaches have been insufficient, highlighting the need for further investigation and improvement.

1.1 Lightweight Verifiable Homomorphic Encryptions

vADDG. An Oblivious Pseudorandom Function (OPRF) is a two-party protocol between a server $\mathcal{A}$ holding the secret PRF key $\mathbf{k}$, and a client $\mathcal{B}$ holding its private input $\mathbf{x}$. At the end of the protocol, $\mathcal{B}$ obtains $F_{\mathbf{k}}(\mathbf{x})$, while $\mathcal{A}$ learns nothing about x and $\mathcal{B}$ learns nothing about $\mathbf{k}$. HE provides a direct solution for constructing an OPRF: $\mathcal{B}$ encrypts the inputs under an HE scheme and sends the ciphertexts to $\mathcal{A}$. Then, $\mathcal{A}$ homomorphically evaluates the PRF using its private key and returns the result to $\mathcal{B}$. Finally, $\mathcal{B}$ decrypts the received ciphertext to obtain the output.

Albrecht *et al.* [ADDG24] leveraged this idea to propose a new candidate for an OPRF, referred to as the ADDG scheme, based on the TFHE homomorphic encryption scheme [CGGI20]. They further extended it to a Verifiable OPRF (VOPRF), which we refer to as the vADDG scheme. For the verifiability extension, vADDG adopts the following method: First, $\mathcal{A}$ publishes a set of verification values (i.e., challenges) $\{\mathbf{x}_k^\star\}_{1\le k\le \kappa}$ together with their PRF evaluations $\{\mathbf{z}_k^\star\}_{1\le k\le \kappa}$ and corresponding zero-knowledge proofs. Suppose that $\mathcal{B}$ wishes to evaluate the OPRF $F_{\mathbf{k}}$ on private inputs $\mathbf{x}_i$ $(1 \le i \le \alpha)$ with verification. $\mathcal{B}$ forms a tuple consisting of ν copies of each $\mathbf{x}_i$ and β verification values from the pre-published set $\{\mathbf{x}_k^\star\}_{1\le k\le \kappa}$. This tuple, of length $\alpha\nu+\beta$, is then randomly permuted, encrypted, and finally sent to $\mathcal{A}$ and computed.

To verify the integrity of the result after outsourcing, $\mathcal{B}$ first decrypts and recovers the random permutation. Then $\mathcal{B}$ checks whether the evaluation of $\mathbf{x}_k^\star$'s matches that of $\mathbf{z}_k^\star$'s. It also checks that the copies of each $\mathbf{z}_i$ all have the same value. Then $\mathcal{B}$ accepts the $\mathbf{z}_i$'s as honest results. A notable feature of this (v)ADDG scheme is that the homomorphic computation of this PRF circuit requires only one level of bootstrapping depth by removing the key-switching key.

VERITAS. Chatel *et al.* [CKP+24] proposed a novel VHE scheme, called Veritas, which supports all operations in BFV and BGV. Compared to the baseline HE schemes, Veritas introduces an overhead ranging from a factor of 1 to a two-digit factor, depending on the characteristics of the circuit. The main idea of these schemes is similar to that of vADDG scheme. For a given circuit f, the client holds random verification values v_i's and their precomputed results $f(v_i)$'s. The client then encrypts a message along with the verification values using specially designed encoders, namely Replication Encoding (REP) and Polynomial Encoding (PE). If the homomorphic computation is correct, the decrypted result will contain both the precomputed results and the desired computational result. To verify the integrity of the result, the client checks whether the computed values match the previously known values. If they do, along with additional verifications, the client accepts the result as valid.

Replication Encoding. REP encodes given messages and verification values as follows: Let n be a power of two. Among the slots indexed from 1 to n, the verification values v_i's are placed in certain slots indexed by $S \subset \{1, \ldots, n\}$ with $|S| = n/2$. Meanwhile, the other slots indexed by $S^c = \{1, \ldots, n\} \setminus S$ are repeatedly filled with the message m. For example, if $n = 4$ and $S = \{1, 4\}$, the client creates a vector (v_1, m, m, v_2) for a message m and challenge values v_1, v_2. We refer to the former as the verification slots and the latter as the computation slots. If the computation is correctly performed, the resulting vector will be $(f(v_1), f(m), f(m), f(v_2))$. The client verifies as follows: Check that all values in the verification slots match the precomputed values $f(v_i)$'s and that the values in the computation slots are identical. If both conditions are satisfied, the client accepts the computation result as $f(m)$.

Polynomial Encoding and ReQuadratization. PE, the other encoding method in [CKP+24], encodes a message $\mathbf{m} \in \mathbb{Z}_t^N$ as follows: the client randomly chooses $\alpha \leftarrow \mathbb{Z}_t^\times$ and a verification value $\mathbf{v} \leftarrow \mathbb{Z}_t^N$. Next, the client interpolates $\mathbf{m}, \mathbf{v}$ at $Y = 0$ and $Y = \alpha$, respectively. As a result, the client obtains a linear polynomial $\mathbf{m} + \left(\frac{\mathbf{v}-\mathbf{m}}{\alpha}\right) Y \in \mathbb{Z}_t^N[Y]$. Then, this polynomial is coefficientwise encrypted to a ciphertext polynomial $\mathsf{ct}_0 + \mathsf{ct}_1 Y \in \mathcal{R}_q^2[Y]$. Here, $\mathcal{R}_q^2$ is the BFV ciphertext space and $\mathcal{R}_q^2[Y]$ is the set of polynomials in the variable Y with coefficients in $\mathcal{R}_q^2$, and the operations on the coefficients correspond to homomorphic operations. For verification, the client checks whether $\mathsf{Dec}(\mathsf{F}(\alpha)) = f(\mathbf{v})$, and if they match, the constant term $\mathsf{Dec}(\mathsf{F}(0))$ is accepted as the desired computation result $f(\mathbf{m})$.

However, the computation cost of PE increases exponentially with the multiplication depth: squaring $\mathsf{ct}_0 + \mathsf{ct}_1 Y$ yields $\mathsf{ct}_0' + \mathsf{ct}_1' Y + \mathsf{ct}_2' Y^2$, and squaring it again results in $\mathsf{ct}_0'' + \cdots + \mathsf{ct}_4'' Y^4$. This results in an exponential performance degradation. To address this issue, [CKP+24] proposed the ReQuadratization (ReQ), a client-aided protocol that transforms a quartic ciphertext polynomial into a quadratic ciphertext polynomial. See Section V.D and Appendix E of [CKP+24] for more details.

1.2 Attack Description on Lightweight VHE Schemes

A fundamental requirement of a VHE scheme is that the server must homomorphically evaluate the designated function f on the client's ciphertexts. If the server can instead produce a valid ciphertext corresponding to another function $g \neq f$ that still passes the verification, the verifiability guarantee of the scheme is broken. In our forgery attacks on lightweight VHE schemes, the server does not need to know the embedded secret information in plaintext. Rather, these attacks leverage the capabilities of the HE schemes to obliviously recover the secret information in its *encrypted form*, and then use it to forge valid ciphertexts.

Attack on vADDG. For the vADDG scheme, the essential step of the forgery attack is to homomorphically recover the positions in the encrypted state where identical values appear. However, since the ADDG scheme utilizes the TFHE scheme which supports only a single level of bootstrapping depth, it is difficult to carry out the forgery attack using the published verification values. To address this, we propose a method for extracting positions by evaluating a characteristic function. Specifically, the adversary can construct a characteristic function that takes each TFHE ciphertext string as input and outputs an encryption of 0 or 1 and then adds that output to the ciphertext string itself. The forgery attack succeeds if this characteristic function outputs 0 at the verification value and outputs 1 at one of the computation values. Using this approach, we obtain a circuit that achieves a forgery probability of roughly 2^{-10}, showing that the concrete security of the 80-bit parameter set in [ADDG24] is significantly weaker in practice.

Attack on Replication Encoding. Similar to the previous forgery attack on vADDG, in REP the server can forge the ciphertexts without knowing the verification slot, but only with their information in the encrypted state by utilizing homomorphic computation. The essential step of this attack is to identify the computation slots S^c in the encrypted state.

The adversary can evaluate a cheating circuit to extract the positions of common values among the n slots. For instance, when $n = 4$, the circuit

$$(v_1, m, m, v_2) \mapsto (0, 1, 1, 0)$$

enables the adversary to identify the verification slots in the encrypted state. We design a simple circuit that extracts this information from a fresh ciphertext using homomorphic comparison over $\mathbb{Z}_t$. After recovering the verification slots in the encrypted state, the adversary can forge the computational result. For example, the following vector

$$(f(v_1), f(m), f(m), f(v_2)) \odot (1, 0, 0, 1) \; + \; (g(v_1), g(m), g(m), g(v_2)) \odot (0, 1, 1, 0)$$

places the verification values $f(v_i)$ in the verification slots and the malicious values $g(m)$ in the computation slots, where g is any circuit different from f, possibly malicious.

Attack on Replication Encoding with Multiple Secret Keys. The above attack circuit utilizes homomorphic rotations to identify the positions of repeated values across different slots. A straightforward countermeasure is to prevent such rotations by withholding the rotation key of index 1 from the server; however, this not only disables bootstrapping—often unacceptable in practice—but also depends on the heuristic assumption that the rotation key of index 1 cannot be derived from other materials. Instead, the attack can be mitigated by employing *multiple secret keys*, thereby preventing computations across different slots. This method allows bootstrapping to remain available, while simultaneously blocking the adversary from carrying out the previous attack. We denote this variant of REP as $\mathsf{REP}^{\mathsf{msk}}$.

However, $\mathsf{REP}^{\mathsf{msk}}$ can also be attacked in the same manner as the attack on vADDG. By evaluating a characteristic function on each slot for with a randomly chosen support A, if a value in a slot belongs to the set $A \subset \mathbb{Z}_t$, then after evaluation that slot will contain 1; otherwise, it will include 0. Therefore, if A contains only the message and not the verification value, this forgery attack will succeed. We present a pseudorandom characteristic function with a cost of $O(\log t)$ homomorphic multiplication. When the adversary implements this characteristic function, the attack success probability becomes greater than $(e(1 + n/2))^{-1}$, which is non-negligible. In contrast to the cryptanalysis on vADDG, where security could be ensured by adjusting parameters, in this case, an attack success probability of $O(n^{-1})$ always occurs regardless of the parameter choice.

Attack on Polynomial Encoding. We now present attacks on PE assuming access to the ReQ protocol, which enables the server to perform a circuit with large multiplicative depth. In the previous attack, the verification slots S, the secret of REP, were recovered in an encrypted form and exploited in a forgery attack. In PE, α is the secret, and similarly, if α is recovered in encrypted form, a forgery attack can be mounted. However, it is difficult to obtain a ciphertext ct containing information about α through homomorphic computation, since α is only used for interpolating random values and hence even the plaintexts reveal nothing about α. [3].

Nevertheless, unlike the previous case, there is another way to forge it: There exists a vulnerability in the encoding structure. Since the encoding method of PE is an interpolation at $Y = 0$ and $Y = \alpha$, by leveraging the algebraic properties of $\alpha \in \mathbb{Z}_t^\times$, it is possible to interpolate the desired computed ciphertexts at $Y = 0$ and $Y = \alpha$ without obtaining any information about α or its encryption.

Let $\mathsf{F}(Y) \in \mathcal{R}_q^2[Y]$ denote the honest output requested by the client, and let $\mathsf{G}(Y) \in \mathcal{R}_q^2[Y]$ denote any malicious output. We define $\mathsf{H}(Y)$ as follows:

$$\mathsf{H}(Y) = \mathsf{G}(Y) + \Big(\mathsf{F}(Y) - \mathsf{G}(Y)\Big) \cdot Y^{\phi(t)}.$$

[3] However, one might regard $\mathsf{Enc}(0, \ldots, 0) + \mathsf{Enc}(1, \ldots, 1)Y \in \mathcal{R}_q^2[Y]$ as an encryption of α. See Discussion in Section 7.

We design a polynomial circuit that operates on $\mathcal{R}_q^2[Y]$ and deterministically outputs $\mathsf{H}(Y)$ (up to the ReQ protocol) in $O(\log t)$. Consequently, the output $\mathsf{H}(Y)$ will pass the client's verification, that is, $\mathsf{Dec}(\mathsf{H}(\alpha)) = \mathsf{Dec}(\mathsf{F}(\alpha)) = f(\mathbf{v})$, while $\mathsf{H}(0) = \mathsf{G}(0)$ is decrypted to yield a forged result.

1.3 Methodology: Homomorphic Cryptanalysis

The attacks presented in this paper rely on neither deep mathematics nor sophisticated arguments, but instead maximize the potential power of homomorphic computation to exploit the weakness of the schemes. It is valuable to examine our methodology in a more abstract way, as it may offer further applications in the design of VHE.

The fundamental approach to achieving verifiability in these lightweight schemes is to hide precomputed values via additional secrets alongside the secret key of HE. Both vADDG and REP adopt an index set S as a secret value, which determines the slot or ciphertext position. Similarly, in the case of PE, the element α from $\mathbb{Z}_t^\times$ serves as the secret value. If an adversary obtains knowledge of these secret values, i.e., S and α, a forgery attack becomes trivial by manipulating the secret encryption. Therefore, these schemes conceal the secret values within ciphertexts.

Our approach to attack the schemes is to obtain the secret values *encrypted states* and use them to generate a cheating circuit. For vADDG and REP, the central idea of the attack is to homomorphically test whether two slots encrypt the same message. This enables the adversary to produce an encryption of 1 at positions corresponding to equal plaintexts via homomorphic operations. In contrast, for PE, the security hinges on the secrecy of α, as verification is carried out by evaluating Y at α. While the ReQuadratization employs a re-randomization procedure to obscure α and prevent its computation, our attacks bypass this defense by analyzing the encoding structure of PE and devising a tailored circuit based on interpolation at $Y = 0$ and $Y = \alpha$.

In any case, even without knowing the secret value itself, the adversary can homomorphically implement a cheating circuit by leveraging the functionalities of homomorphic encryption.

1.4 Related Works

There are two major security concerns in information security: integrity and confidentiality. As data utilization expands beyond mere storage to computational applications, these concerns have evolved into the notions of Verifiable Computation and Homomorphic Encryption, respectively. To achieve VC, various cryptographic primitives have been proposed, including SNARKs [LHW+25], Homomorphic MACs [CF13, GW13], and Homomorphic Signatures [CCRS24]. Meanwhile, since Gentry's groundbreaking discovery [Gen09], HE has seen continuous advancements in both functionality and efficiency. Notable schemes such as BGV, BFV, TFHE, and CKKS [BGV12, FV12, CGGI20, CKKS17] have significantly improved HE's practical applicability.

A natural follow-up question is whether VC and HE can be integrated, allowing both concerns to be addressed simultaneously. There are two major approaches to constructing Verifiable HE [ZWL+25]. One approach is to apply integrity mechanisms to the ciphertext operations of HE [ABPS24, BCFK21, FGP14, FNP20, GNSV23]. Another approach is to achieve verifiability by generating SNARK-based proofs in the encrypted domain [ACGaSV24, GGW24, GBK+24]. However, adding verifiability to HE is a non-trivial task. Both approaches suffer from significant efficiency limitations. Compared to native HE computations, this overhead is larger by four to five orders of magnitude.

For the former, conventional integrity mechanisms such as SNARKs are incompatible with the non-arithmetic ciphertext operations of HE. As a result, these operations must be emulated by arithmetic operations, which incurs significant overhead when proving them. For the latter approach, the plaintext space must be suitable for generating proofs, which leads to inefficiency in parameter selection, and the process of homomorphically evaluating proofs introduces substantial overhead.

The verifiable homomorphic encryption scheme targeted in this paper follows a lightweight approach different from the two major existing methods. The idea is to achieve verifiability by hiding verification secrets under the semantic security of HE and by altering the message encoding of the underlying scheme accordingly [CKP+24, ADDG24, SXLS24]. However, there is no well-established hardness assumption or security proof that justifies a reduction from the confidentiality of HE to its integrity, making the connection between the two notions unclear.

Our attack leverages the homomorphic properties of HE to forge the underlying encoding structure. This idea was implicitly suggested in [CHLR18], and later formalized in the construction of vADDG [ADDG24], where the authors proposed a naive hardness assumption that constructing such a cheating circuit within shallow depth is infeasible. This suggests the need for a deeper exploration of what kinds of computations are possible in various homomorphic encryption settings.

2 Preliminaries

In this section, we present the formal definitions and necessary background required for this paper. In particular, regarding the formal definition and security of the verifiable homomorphic encryption, we follow the definition from [VKH23] with the necessary modifications.

2.1 Homomorphic Encryption

Let $\mathcal{M}$ be a plaintext space and let $\mathcal{F} \subset \{f_\alpha \mid f_\alpha : \mathcal{M}^{n_\alpha} \to \mathcal{M}\}$ be a set of functions on tuples of $\mathcal{M}$. Then Homomorphic Encryption (over $\mathcal{F}$) is stated as follows.

Definition 1 (Homomorphic Encryption). *A Homomorphic Encryption over $\mathcal{F}$ is a tuple of PPT algorithms*

$$\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$$

satisfying the following:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F})$*: pk includes evaluation keys evk.*
- $\mathsf{ct}_x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x)$ *for* $x \in \mathcal{M}$
- $\mathsf{ct}_y \leftarrow \mathsf{Eval}_{f,\mathsf{evk}}(\mathsf{ct}_x)$ *for* $y = f(x)$ *where* $f \in \mathcal{F}$.
- $y \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_y)$.

Definition 2 (δ-Correcteness). *Let Π be a Homomorphic Encryption scheme. Π is* δ-Correct *if the evaluation on the ciphertext decrypts to the correct result. For all f and x, it should be satisfied that*

$$\Pr\left[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_y) = f(x) : \begin{array}{r}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F}) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x) \\ \mathsf{ct}_y \leftarrow \mathsf{Eval}_{f,\mathsf{evk}}(\mathsf{ct}_x)\end{array}\right] \geq 1 - \delta.$$

Bootstrapping. In many homomorphic encryption schemes, errors accumulate with each operation. Bootstrapping is a technique used to remove these errors and refresh the ciphertext. If computations can be performed without limit, the scheme is called Fully Homomorphic Encryption (FHE); otherwise, it is known as Somewhat Homomorphic Encryption (SHE). In many cases, bootstrapping transforms SHE into FHE.

TFHE. TFHE [CGGI20] is a FHE scheme that supports gate-level bootstrapping, that the (v)ADDG scheme is instantiated with. Its plaintext space is $\mathbb{Z}_Q$ and the message space $\mathbb{Z}_P$ is encoded into $\mathbb{Z}_Q$ for $P < Q$. Thereafter a plaintext is encrypted into the ciphertext space $\mathcal{C} = \mathbb{Z}_Q^{n+1}$ where n is determined by the security parameter.

TFHE bootstrapping requires a bootstrapping key btk for evaluation, which is an encryption of the secret key sk under a different secret key sk'. This bootstrapping key transforms a ciphertext under sk into a ciphertext under sk', and a ciphertext under sk' can be switched back to one under sk by using the key-switching key ksk. The TFHE bootstrapping has an additional feature beyond error refreshing. A lookup table evaluation can be implemented during the bootstrapping process. This bootstrapping technique is called Programmable Bootstrapping (PBS) or functional bootstrapping. PBS enables the homomorphic evaluation of non-linear functions in TFHE. For more details, see [Joy22].

BFV. BFV is an RLWE-based HE scheme that supports the computation of vectors of integers [Bra12, FV12], and it is the scheme in which VERITAS is instantiated. Its message space is $\mathbb{Z}_t^N$, where N is a power of 2 and $t = p^r$ is a

power of a prime p. For $q > t$, the ciphertext space is $\mathcal{R}_q^2 = \left(\mathbb{Z}[X]/(q, X^M+1)\right)^2$ where M a power of 2 and $N = M/d$ where d is a multiplicative order of $p \in \mathbb{Z}_M^\times$. Multiplication is given by the Hadamard product, i.e., element-wise multiplication.

A notable feature of BFV is that it supports rotations $\sigma_i : \mathbb{Z}_t^N \to \mathbb{Z}_t^N$, which shift vectors in $\mathbb{Z}_t^N$ with the aid of the rotation key rtk_i. In addition, ciphertext–ciphertext multiplication requires the relinearization key rlk. As far as known, BFV bootstrapping requires the evaluation keys rtk_1 and rlk. For further technical details, refer to Appendix A.

2.2 Verifiable Homomorphic Encryption

Verifiable Homomorphic Encryption is stated as follows. Many works in the literature introduce a tag as part of the output of Enc and Eval, but we will treat it as part of the ciphertext, when it exists. Let $\mathcal{M}$ be a plaintext space and let $\mathcal{F} \subset \{f_\alpha \mid f_\alpha : \mathcal{M}^{n_\alpha} \to \mathcal{M}\}$ be a set of functions on tuples of $\mathcal{M}$.

Definition 3. *A Verifiable Homomorphic Encryption over $\mathcal{F}$ is a tuple of PPT algorithms*

$$\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Verify}, \mathsf{Dec})$$

satisfying the following:

- $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F})$*:*
- $\mathsf{ct}_x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x)$ *for* $x \in \mathcal{M}$*.*
- $\mathsf{ct}_y \leftarrow \mathsf{Eval}_{f,\mathsf{evk}}(\mathsf{ct}_x)$ *for* $y = f(x)$ *where* $f \in \mathcal{F}$*.*
- $b \leftarrow \mathsf{Verify}_{\mathsf{sk}}(\mathsf{ct}_y)$*, the client accepts if* $b = 0$ *and rejects if* $b = 1$*.*
- $y \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_y)$*.*

Definition 4 (δ-Completeness). *A Verifiable Homomorphic Encryption scheme Π is* δ-complete *if the client always accepts a correct output. For all f and x, it should be satisfied that*

$$\Pr\left[0 \leftarrow \mathsf{Verify}_{\mathsf{sk}}(\mathsf{ct}_y) : \begin{array}{r} \mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F})) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x) \\ \mathsf{ct}_y \leftarrow \mathsf{Eval}_{f,\mathsf{evk}}(\mathsf{ct}_x) \end{array}\right] \geq 1 - \delta$$

Malicious Adversary Model. We now formally define the security of VHE. We assume that the underlying scheme Π satisfies both δ-correctness and δ-completeness for some negligible δ. Throughout this work, we consider two types of adversaries: malicious and covert. The following definition for malicious adversaries is a modification of the corresponding definition in [VKH23]. Specifically, the modification is the removal of the tag, as noted above.

Definition 5 (Security against Malicious Adversary, modified from [VKH23]). *A Verifiable Homomorphic Encryption scheme Π is* sound in the presence of malicious adversaries *if the client rejects an incorrect output with overwhelming*

probability in λ for any malicious PPT adversary $\mathcal{A}_{\mathsf{mal}}$. For all f, it should be satisfied that

$$\Pr\left[\begin{array}{c} \\ 0 \leftarrow \mathsf{Verify}_{\mathsf{sk}}(ct_y) \\ \mathsf{Dec}_{sk}(ct_y) \neq f(x) \\ \\ \end{array} : \begin{array}{c} pk, sk \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F})) \\ x \leftarrow \mathcal{A}_{\mathsf{mal}}^{evk} \\ ct_x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x) \\ ct_y \leftarrow \mathcal{A}_{\mathsf{mal}}^{evk}(ct_x) \end{array}\right] \textit{ is negligible.}$$

Covert Adversary. Originally introduced in the context of secure multiparty computation in [AL07], the covert adversary model can be extended to the VHE setting, given that HE can be regarded as a special case of two-party computation, and was adopted as the adversary model in [CKP+24]. The ε-covert adversary model, assumes that any adversarial deviation is detected with probability at least ε. A customary choice of the deterrence factor ε is a noticeable probability, e.g., $1/\mathsf{poly}(\lambda)$. Unless otherwise stated, we use the term *covert adversary* to refer to a $1/\mathsf{poly}(\lambda)$-covert adversary. In this paper, we provide an informal definition of the covert adversary model. For a formal treatment, we refer the reader to Definition 3.4 from [AL07].

Definition 6 (Security against Covert Adversaries, informal). *Let $\varepsilon(\lambda) \in [0, 1]$. A Verifiable Homomorphic Encryption scheme Π is* sound in the presence of ε-covert adversaries *if for any PPT adversary $\mathcal{A}$, any deviation from the prescribed protocol is detected with probability at least ε.*

Access to Oracles. We assume that the adversary can access to the evaluation keys, but not to the decryption, verification, or encryption oracles. In the attack on REP and PE, we always assume that the adversary has rlk, and in the attack on vADDG, we assume that the adversary has btk. The only exception is that in the case of PE, we assume that the adversary can access the ReQ protocol. We denote adversary models as follows: for example, $\mathcal{A}_{\mathsf{mal}}^{\mathsf{rtk}_1, \mathcal{O}_{\mathsf{ReQ}}}$ denotes a malicious adversary who can access rtk_1, rlk, and ReQ, while $\mathcal{A}_{\mathsf{cov}}$ denotes an $1/\mathsf{poly}(\lambda)$-covert adversary who can access only rlk. (Since we always assume that all adversaries can access rlk, we omit it from the notation.)

3 Cryptanalysis of vADDG

3.1 (v)ADDG Scheme

In [ADDG24], Albrecht et al. presented an Oblivious PRF candidate and its extension to a verifiable OPRF, denoted as the ADDG and vADDG schemes, respectively. The ADDG scheme utilizes TFHE to construct an OPRF candidate: the client outsources homomorphic encryption of a bit string $\mathbf{x}$, and the server homomorphically computes $\mathbf{z} = F(\mathbf{k}, \mathbf{x})$ with its PRF key $\mathbf{k}$ and returns it to the client.

We briefly explain how the PRF $F(\mathbf{k}, \mathbf{x})$ is homomorphically calculated in ADDG. For the TFHE ciphertext space $\mathcal{C}$, we denote by $[x]_P \in \mathcal{C}$ a TFHE encryption of $x \in \mathbb{Z}_P$. For $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{Z}_P^k$, we denote

$$[\mathbf{x}]_P = ([x_1]_P, [x_2]_P, \ldots, [x_k]_P) \in \mathcal{C}^k.$$

First, the client inputs an encryption of a 128-bit string $\mathbf{x} \in \mathbb{Z}_2^{128}$. Then, using only homomorphic additions and constant multiplications without bootstrapping, the server homomorphically evaluates $\mathbf{y} \in \mathbb{Z}_2^{256}$ using the PRF key $\mathbf{k}$. Next, the server performs $\mathsf{CPPBS}_{(2,3)}$ (Circuit Privacy Programmable Bootstrapping), a programmable bootstrapping with functionalities

$$\mathsf{CPPBS}_{(2,3)} = \begin{cases} [0]_2 \mapsto [0]_3, \\ [1]_2 \mapsto [1]_3, \end{cases}$$

to obtain an encryption of $\mathbf{y}$ over $\mathbb{Z}_3^{256}$. Finally, again using only homomorphic additions and constant multiplications, the server computes an encryption of an 82-trit string $\mathbf{z} \in \mathbb{Z}_3^{82}$, which is the output of the homomorphic evaluation of the PRF. A notable feature of the (v)ADDG scheme is that it consumes only a bootstrapping depth of one: By the removal of the key-switching key, this scheme prevents further bootstrapping executions after the CPPBS.

$$\boxed{\mathsf{Eval}_{F_{\mathbf{k}}} : [\mathbf{x}]_2^{\mathsf{sk}} \in \mathcal{C}^{128} \mapsto [\mathbf{y}]_2^{\mathsf{sk}} \in \mathcal{C}^{256} \overset{\mathsf{CPPBS}_{(2,3)}}{\mapsto} [\mathbf{y}]_3^{\mathsf{sk}'} \in \mathcal{C}^{256} \mapsto [\mathbf{z}]_3^{\mathsf{sk}'} \in \mathcal{C}^{82}}$$

Fig. 1: Workflow of the Homomorphic PRF suggested in [ADDG24]. The superscripted sk (resp. sk') denotes that the secret key is sk (resp. sk'). $\mathbf{k}$ is the PRF key.

Verifiability through Replication. For the extension from ADDG to vADDG for verifiability, [ADDG24] suggested the following method: assume that a client wishes to evaluate $F(\mathbf{k}, \mathbf{x}_i)$ for distinct inputs $\mathbf{x}_i$ where $i = 1, \ldots, \alpha$, and $\mathbf{k}$ is the server's secret PRF key.

1. First, the server publishes κ verification values
$$\mathfrak{R} = \{(\mathbf{x}_k^\star, \mathbf{z}_k^\star)\}_{1 \le k \le \kappa} \quad (\mathbf{z}_k^\star = F(\mathbf{k}, \mathbf{x}_k^\star))$$
in plaintext, along with zero-knowledge proofs that enable the client to verify the integrity of these results.
2. The client prepares and encrypts a vector of strings
$$\Big(\overbrace{\mathbf{x}_1, \ldots, \mathbf{x}_1}^{\nu \text{ copies}}, \ldots, \overbrace{\mathbf{x}_\alpha, \ldots, \mathbf{x}_\alpha}^{\nu \text{ copies}}, \overbrace{\mathbf{x}_{k_1}^\star, \ldots, \mathbf{x}_{k_\beta}^\star}^{\beta \text{ verification values}} \Big) \in (\mathbb{Z}_2^{128})^\gamma$$
where $(\mathbf{x}_{k_j}^\star, \mathbf{z}_{k_j}^\star) \xleftarrow{\$} \mathfrak{R}$ and $\gamma = \alpha\nu + \beta$.

3. Permute the γ ciphertexts using a random permutation $\rho : \{1, \ldots, \gamma\} \to \{1, \ldots, \gamma\}$, and then send them to the server. After the server evaluate the PRF $F(\mathbf{k}, \mathbf{x}_i)$ on each input $\mathbf{x}_i$, apply the inverse permutation ρ^{-1} and decrypt the ciphertexts to obtain γ 82-trit strings $Z = (\mathbf{z}_s) \in \left(\mathbb{Z}_3^{82}\right)^{\gamma}$ for $1 \le s \le \gamma$.
4. Client verifies whether the followings are correct.
   (a) For sets $\{\mathbf{z}_1, \ldots, \mathbf{z}_\nu\}, \{\mathbf{z}_{\nu+1}, \ldots, \mathbf{z}_{2\nu}\}, \ldots, \{\mathbf{z}_{(\alpha-1)\nu+1}, \ldots, \mathbf{z}_{\alpha\nu}\}$, each set contains a common single value, and all these α values are distinct.
   (b) $\mathbf{z}_{\alpha\nu+j} = \mathbf{z}^{\star}_{k_j}$

The suggested parameters for 80-bit security for verifiability is $(\alpha, \beta, \nu) = (105, 10, 11)$.

3.2 Attack on vADDG

The vADDG scheme is designed to prevent forgery attacks by leveraging the assumed hardness of building deep circuits under depth restrictions [CHLR18], and by restricting bootstrapping to a single level. In [ADDG24], the authors propose a concrete attack strategy under this model and determine secure parameters to resist such attacks, leading to the choice of $(\alpha, \beta, \nu) = (105, 10, 11)$. However, we show that a successful forgery remains possible under this parameter set with non-negligible probability.

The idea behind this attack is to create a pseudorandom characteristic function and implement it homomorphically. To build such a function, we follow the following procedure. Let the adversarial server receives the inputs $\mathsf{ct}_s := [\mathbf{x}_s]_2$ for all $1 \le s \le \gamma$ where each $\mathbf{x}_s = (x_{s,u})_{1 \le u \le 128} \in \mathbb{Z}_2^{128}$ is a 128-bit string. Now design a homomorphic characteristic function $\chi_u : \mathcal{C}^{128} \to \mathcal{C}, \mathsf{ct}_s \mapsto \overline{\mathsf{ct}_{s,u}}$ as follows: First, let an index $u \in \{1, \ldots, 128\}$ be fixed. Next, for each ciphertext string ct_s, evaluate

$$\overline{\mathsf{ct}_{s,u}} := \mathsf{CPPBS}_{(2,3)}(\mathsf{ct}_{s,u}).$$

Then, this characteristic function χ_u is a homomorphic pseudorandom function which outputs $[1]_3$ with probability approximately $1/2$, otherwise $[0]_3$. With these γ outputs $(\overline{\mathsf{ct}_{s,u}})_{1 \le s \le \gamma}$, the adversary can try to forge the PRF F by adding $(\overline{\mathsf{ct}_{s,u}})_{1 \le s \le \gamma}$ to the output strings $\left(\mathsf{Eval}_{F(\mathbf{k},-)}(\mathsf{ct}_s)\right)_{1 \le s \le \gamma}$. For example, for $\mathbf{t} = (1, 0, \ldots, 0)$, we can forge the PRF by evaluating

$$\mathsf{Eval}_{F(\mathbf{k},-)}(\mathsf{ct}_s) + \mathbf{t} \cdot \chi_u(\mathsf{ct}_s).$$

Note that $\chi_u(\mathsf{ct}_s)$ is the same for the same inputs and so this forgery is not detected if $\chi_u(\mathsf{ct}_s)$ is 0 for the all β verification values. The success probability is given in Theorem 1.

Theorem 1. *For a given parameter (α, β, ν), the expected probability that the bootstrapping-depth one circuit in Algorithm 1 successfully forges the output is*

$$\left(1 - (1/2)^{\alpha}\right) \cdot (1/2)^{\beta}.$$

over the all random inputs.

Algorithm 1 Attack on vADDG

procedure $\mathsf{vADDG_Attack}((\mathsf{ct}_s)_{1\le s\le\gamma}, u)$ ▷ $u \in \{1,\dots,128\}$
$\mathsf{ct}_s = [\mathbf{x}_s]_2 = ([x_{s,1}]_2,\dots,[x_{s,128}]_2)$ for $1 \le s \le \gamma$.
  $\mathbf{t} \leftarrow \mathbb{Z}_3^{82}$ for $\mathbf{t} \neq (0,0,\dots,0)$ ▷ Select the trits to forge.
  for $s = 1$ **to** γ **do**
    $\overline{\mathsf{ct}_{s,u}} \leftarrow \mathsf{CPPBS}_{(2,3)}([x_{s,u}]_2)$
    $[\mathbf{z}_s]_3 \leftarrow \mathsf{Eval}_{F(\mathbf{k},-)}(\mathsf{ct}_s)$ ▷ $[\mathbf{z}_s]_3 = ([z_{s,1}]_3,\dots,[z_{s,82}]_3)$.
    $\mathsf{ct}_s^{\mathsf{forged}} \leftarrow [\mathbf{z}_s]_3 + \overline{\mathsf{ct}_{s,u}} \cdot \mathbf{t}$
  end for
  return $\left(\mathsf{ct}_s^{\mathsf{forged}}\right)_{1\le s\le\gamma}$
end procedure

Proof. The forgery succeeds when $\overline{\mathsf{ct}_{s,u}} = [0]_3$ for all s correspond to the β verification values and at least one $\overline{\mathsf{ct}_{s,u}} = [1]_3$ for s corresponds to α message values. Let p be the probability that $\overline{\mathsf{ct}_{s,u}}$ became an encryption of 1. In Algorithm 1, the expected value of p over all input values is $1/2$. The attack success probability is obtained by subtracting the probability that all values become zero, $(1-p)^{\alpha+\beta}$, from the probability that all s corresponding to β become zero, $(1-p)^{\beta}$. Thus, the success probability is given by $Q(p) := (1-p)^{\beta} - (1-p)^{\alpha+\beta}$. Substituting $p = 1/2$, we have the desired result $(1-(1/2)^{\alpha})\cdot(1/2)^{\beta}$. □

Corollary 1. *The* vADDG *scheme with parameters* $(\alpha,\beta,\nu) = (105,10,11)$ *provides at most 10-bit security against a malicious adversary.*

In the covert adversary setting, this forgery is detected if at least one of the β verification values is altered, which occurs with probability $1-(1/2)^{\beta}$. Therefore, a covert adversary cannot successfully carry out this attack.

Parameter Selection for vADDG. The success probability of this attack does not change with an increase in the parameter ν. Also, the probability does not change significantly with a decrease in the parameter α, since $Q(p) = (1-(1/2)^{\alpha})(1/2)^{\beta}$ has a lower bound of $(1/2)\cdot(1/2)^{\beta}$ and an upper bound of $(1/2)^{\beta}$, which are both independent of α. Thus in our discussion of parameters, we only concerns β. To ensure 80-bit security, β must exceed 80.

In [ADDG24], other cheating strategies are mentioned with success probabilities of $1/\binom{\gamma}{\beta}$ and $\alpha/\binom{\gamma}{\nu}$. By setting $\beta = 80$, we obtain the following two new parameter sets that satisfy 80-bit security:

$$(\alpha,\beta,\nu) = (28440,80,6), \text{ or } (189,80,10).$$

The first parameter has an overhead size $((\alpha\nu+\beta)/\alpha)$ of approximately $\times 6.0$ but requires a larger set of inputs, meanwhile the second parameter has an overhead size of approximately $\times 10.4$ but requires a smaller set of inputs.

4 Cryptanalysis on Replication Encoding

We recall the Replication Encoding (REP) and its verification procedure described in [CKP+24]. For the convenience of presentation, we describe our attack assuming the evaluation of a single-input and single-output polynomial $f : \mathbb{Z}_t \to \mathbb{Z}_t$ on a ciphertext encrypted under the HE scheme $\mathsf{Enc} : \mathbb{Z}_t^N \to \mathcal{R}_q^2$, where $t = p$ is a prime. However, it is straightforward to extend our attack to a multivariate polynomial circuit $f : \mathbb{Z}_t^\mu \to \mathbb{Z}_t^\nu$ for $\mu, \nu \in \mathbb{Z}^+$, where an element of $\mathbb{Z}_t^\mu$ is regarded as an extended slot element. For the case $t = p^r$ with $r > 1$, a similar idea can be applied to construct an analogous attack. We defer the details for $t = p^r$ to Appendix B.

4.1 Replication Encoding

Let $f : \mathbb{Z}_t \to \mathbb{Z}_t$ be a circuit that the client requested to evaluate. The REP proceeds as follows.

1. The client randomly selects a subset $S \subset \{1, \ldots, n\}$ with size $|S| = n/2$ for a positive integer n dividing N.
2. The slots indexed by $S^c = \{1, \ldots, n\} \setminus S$ are filled with a same message m, and the complement slots indexed by S are filled with $n/2$ random values $v_i \xleftarrow{\$} \mathbb{Z}_t$.

We call the slots indexed by S as *verification slots*, and the slots indexed by S^c as *computation slots*. After evaluating f to the ciphertext, its decryption contains $f(v_i)$'s in the verification slots and $f(m)$ in the computation slots. For verification, it is assumed that the client knows $f(v_i)$'s and their integrity in advance, either by outsourcing the computation of $f(v_i)$'s without encryption to a third party using existing verifiable computation techniques or by securely computing them himself. Note that the verification values are not published, unlike vADDG. The client verifies as follows:

1. Decrypt the ciphertext to get a vector (z_i) for $i = 1, \ldots, n$.
2. Check whether $z_i = f(v_i)$ for verification slots $i \in S$.
3. Check whether z_i are equal for computation slots $i \in S^c$.

If these conditions are satisfied, the common value in the computation slots is accepted as $f(m)$. See [CKP+24] for more details.

4.2 Attacks on REP

Learning or guessing $S \subset \{1, \ldots, n\}$ is not easy. Instead of recovering S, we take a similar approach to the attack on vADDG: we learn S in *encrypted state* by computing $\mathsf{Enc}(J_S)$ for $J_S := (j_1, \ldots, j_n)$ where $j_i = 1$ if $i \in S$ and $j_i = 0$

otherwise. Once the adversary recovers $\mathsf{Enc}(J_S)$[4], the ciphertext can be easily forged by computing the following:

$$\mathsf{ct}_{\mathsf{forged}} := \mathsf{Mult}(\mathsf{Enc}(J_S), \mathsf{Eval}_f(\mathsf{ct})) + \mathsf{Mult}(\mathsf{Enc}(J_{S^c}), \mathsf{Eval}_g(\mathsf{ct})), \tag{1}$$

where f is the circuit that the client requested to evaluate, and g is any modified circuit by the adversary. Then $\mathsf{ct}_{\mathsf{forged}}$ provides the client with a malicious result $g(m)$ but successfully passes the verification step.

To recover $\mathsf{Enc}(J_{S^c})$, we designed a circuit to extract the position vector of the common values. Figure 2 shows the functionality of the desired algorithm; If the ith slot is chosen, the output of this algorithm is the position vector of the slots equal to this value. We do not consider the case where $v_i = m$ for some $i \in S$, as in such case the client would already have attained $f(v_i) = f(m)$.

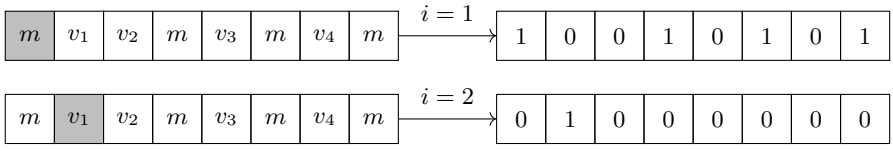


Fig. 2: Finding Common Values among n Slots.

We first define two functions $\mathsf{Duplicate}_i$ and $\mathsf{Compare}$ as follows:

- $\mathsf{Duplicate}_i : \mathbb{Z}_t^n \to \mathbb{Z}_t^n$ is a function that maps $(x_1, \ldots, x_i, \ldots, x_n) \mapsto (x_i, \ldots, x_i)$
- $\mathsf{Compare}(x, y) : \mathbb{Z}_t^2 \to \mathbb{Z}_t$ is 1 if $x = y$ and 0 if $x \neq y$. We can extend it to act on vectors componentwise manner, $\mathsf{Compare} : \mathbb{Z}_t^{2n} \to \mathbb{Z}_t^n$

Also we extend $\mathsf{Compare}$ to act on vectors componentwise manner. We note that Fermat's little theorem guarantees $1 - (x - y)^{t-1} = \mathsf{Compare}(x, y)$ when t is a prime, which needs $\lceil \log t \rceil$ multiplications to evaluate. Now we introduce Algorithm 2 to construct the desired circuit and describe its properties.

Algorithm 2 Finding Common Value Slots

1: **Input:** A ciphertext ct encrypted via REP and an index i.
2: **Output:** A ciphertext $\mathsf{ct}_{\mathsf{CVS},i}$, possibly an encryption of the vector J_{S^c} if $i \in S^c$.
3: **procedure** $\mathsf{CVS}(\mathsf{ct}, i)$
4: $\quad \mathsf{ct}' \leftarrow \mathsf{Eval}_{\mathsf{Duplicate}_i}(\mathsf{ct})$ ▷ $\mathsf{ct}' = \mathsf{Enc}((x_i, \ldots, x_i))$
5: $\quad \mathsf{ct}_{\mathsf{CVS},i} \leftarrow \mathsf{Eval}_{\mathsf{Compare}}(\mathsf{ct}', \mathsf{ct})$
6: $\quad$ **return** $\mathsf{ct}_{\mathsf{CVS},i}$
7: **end procedure**

[4] Note that $J_{S^c} = (1, \ldots, 1) - J_S$, thus recovering $\mathsf{Enc}(J_{S^c})$ is equivalent to recovering $\mathsf{Enc}(J_S)$.

Theorem 2. *For uniformly random* $i \xleftarrow{\$} \{1, \ldots, n\}$, $\mathsf{CVS}(\mathsf{ct}, i)$ *in Algorithm 2 outputs* $\mathsf{Enc}(J_{S^c})$ *with probability of* $|S|/n = 1/2$ *in* $O(\log t)$ *homomorphic multiplications and rotations.*

Proof. Let ct be an encryption of a vector that has m in computational slots S^c. If $i \in S^c$, then $\mathsf{ct}' \leftarrow \mathsf{Eval}_{\mathsf{Duplicate}_i}(\mathsf{ct})$ is an encryption of $(m, \ldots, m)$. Consequently, $\mathsf{Eval}_{\mathsf{Compare}}(\mathsf{ct}', \mathsf{ct})$ returns $\mathsf{Enc}(J_{S^c})$.

$\mathsf{Eval}_{\mathsf{Duplicate}_i}$ requires one masking by the i-th elementary vector and a partial rotation sum within each message slot of length n that takes $\lceil \log n \rceil$ homomorphic rotation. Also we showed that $\mathsf{Eval}_{\mathsf{Compare}}$ needs $O(\log t)$ homomorphic multiplications. Since $n \leq \phi(t)$, the total complexity is $O(\log t)$ homomorphic multiplications/rotations. □

Thus, REP is not sound in the presence of the malicious adversary with rotation key with index 1.

Corollary 2. *REP is not sound in the presence of* $\mathcal{A}_{\mathsf{mal}}^{\mathsf{rtk}_1}$.

If the attack fails, (i.e. $i \in S$) then the output of $\mathsf{ct}_{\mathsf{forged}}$ of the forgery fomular 1 where $\mathsf{Enc}(J_{S^c}) \leftarrow \mathsf{CVS}(\mathsf{ct}, i)$ is detected by the client. Thus this attack does not imply that the REP is not sound in the presence of the covert adversary.

Increasing Attack Success Probability. Now we show how to increase the success probability of the attack by repeating Algorithm 2 for different indices i. For any fixed i, if we sum up the values in the n slots of $\mathsf{ct}_{\mathsf{cvs},i}$, it outputs $|S|$ or 1 depending on whether $i \in S$ or not. That is, the server can *homomorphically distinguish* whether $i \in S$ or not. Therefore, an adversary can construct a cheating circuit (Algorithm 3) that deterministically outputs $\mathsf{Enc}(J_{S^c})$. To design such a circuit, first we define three functions $\mathsf{RotSum}, \mathsf{Interpolate}$ and $\mathsf{Normalize}_k$ as follows:

$$\mathsf{RotSum}: (x_1, \ldots, x_n) \mapsto \left(\sum_{i=1}^{n} x_i, \ldots, \sum_{i=1}^{n} x_i \right)$$

$$\mathsf{Interpolate}: \begin{cases} 1 \mapsto 0 \\ n/2 \mapsto 1 \\ \text{otherwise anywhere.} \end{cases} \qquad \mathsf{Normalize}_k: \begin{cases} 0 \mapsto 0 \\ 1, \ldots, k \mapsto 1, \\ \text{otherwise anywhere} \end{cases}$$

Note that $\mathsf{Eval}_{\mathsf{RotSum}}$ can be evaluated using $\lceil \log n \rceil$ homomorphic rotations, while $\mathsf{Eval}_{\mathsf{Interpolate}}$ and $\mathsf{Eval}_{\mathsf{Normalize}_k}$ can be evaluated using $O(1)$ and $O(\log k)$ homomorphic multiplications, respectively, when the plaintext modulus t is prime.

Theorem 3. *If the rotation key with index 1 is given,* $\mathsf{Recover}_{S^c}(\mathsf{ct}, k)$ *in Algorithm 3 can be evaluated within a computational cost of* $O(k \log t)$*, outputting* $\mathsf{Enc}(J_{S^c})$ *with probability*

$$1 - \frac{\binom{n/2}{k}}{\binom{n}{k}}.$$

Algorithm 3 Deterministic Recovery of Encryption of S

Input: A ciphertext ct encrypted via REP and an index i.
Output: A ciphertext $\mathsf{ct}_{\mathsf{bool},i}$ which is an encryption of 0^N if $i \in S$ and 1^N otherwise.
procedure $\mathsf{is_in_set}_S(\mathsf{ct}, i)$
  $\mathsf{ct}_{\mathsf{cvs},i} \leftarrow \mathsf{CVS}(\mathsf{ct}, i)$
  $\mathsf{ct}_{\mathsf{rs}} \leftarrow \mathsf{RotSum}(\mathsf{ct}_{\mathsf{cvs},i})$ ▷ $\mathsf{ct}_{\mathsf{rs}} = \mathsf{Enc}(1, \ldots, 1)$ or $\mathsf{Enc}(n/2, \ldots, n/2)$
  $\mathsf{ct}_{\mathsf{bool},i} \leftarrow \mathsf{Eval}_{\mathsf{Interpolate}}(\mathsf{ct}_{\mathsf{rs}})$
  return $\mathsf{ct}_{\mathsf{bool},i}$ ▷ $\mathsf{ct}_{\mathsf{bool},i} = \mathsf{Enc}(0, \ldots, 0)$ or $\mathsf{Enc}(1, \ldots, 1)$
end procedure

Input: A ciphertext ct encrypted via REP and an index i, an integer k.
Output: A ciphertext ct possibly encrypting J_{S^c}
procedure $\mathsf{Recover}_{S^c}(\mathsf{ct}, k)$ ▷ Deterministically outputs $\mathsf{Enc}(J_{S^c})$ when $k = n/2 + 1$.
  $\mathsf{ct}_{\mathsf{cvs},i} \leftarrow \mathsf{CVS}(\mathsf{ct}, i)$
  $\mathsf{ct}_{\mathsf{bool},i} \leftarrow \mathsf{is_in_set}_S(\mathsf{ct}, i)$
  $\mathsf{ct}' \leftarrow \sum_{i=1}^{k} \mathsf{Mult}(\mathsf{ct}_{\mathsf{bool,i}}, \mathsf{ct}_{\mathsf{cvs,i}})$
  $\mathsf{ct}_{S^c} \leftarrow \mathsf{Eval}_{\mathsf{Normalize}_k}(\mathsf{ct}')$
  return ct_{S^c}
end procedure

Otherwise, it outputs an encryption of zero vector $\mathsf{Enc}(0, \ldots, 0)$.

Proof. Since the slots in $\mathsf{ct}_{\mathsf{cvs},i}$ contain $n/2$ ones if $i \in S^c$, and a single one if $i \in S$, applying $\mathsf{Eval}_{\mathsf{RotSum}}$ yields either $\mathsf{Enc}\left(\frac{n}{2}, \ldots, \frac{n}{2}\right)$ or $\mathsf{Enc}(1, \ldots, 1)$. Consequently, by interpolation, the output $\mathsf{ct}_{\mathrm{bool},i}$ is a vector of Boolean values, either $(1, \ldots, 1)$ or $(0, \ldots, 0)$. Thus, for $i = 1, \ldots, k$, the sum of the $\mathsf{ct}_{\mathsf{cvs},i}$ after homomorphic multiplication with $\mathsf{ct}_{\mathrm{bool},i}$ decrypts to the same value as $|S^c \cap \{1, \ldots, k\}| \cdot \mathsf{Enc}(J_{S^c})$. Since $S \subset \{1, \ldots, n\}$ is a random subset of size $n/2$, $|S^c \cap \{1, \ldots, k\}|$ is zero with probability $\frac{\binom{n/2}{k}}{\binom{n}{k}}$ and otherwise nonzero. Thus, after evaluating $\mathsf{Normalize}_k$, we obtain $\mathsf{Enc}(0, \ldots, 0)$ with probability $\frac{\binom{n/2}{k}}{\binom{n}{k}}$, or $\mathsf{Enc}(J_{S^c})$ with probability $1 - \frac{\binom{n/2}{k}}{\binom{n}{k}}$.

Furthermore, the procedure $\mathsf{Recover}_{S^c}(\mathsf{ct})$ calls CVS_i for $i = 1, \ldots, k$, which incurs a cost of $O(k \log t)$ homomorphic operations. The cost of the remaining operations does not exceed $O(k \log t)$. □

By taking $k = n/2 + 1$, this attack can achieve a 100% success rate[5]. Even when $k < n$, the failure probability of the attack is bounded by $\frac{\binom{n/2}{k}}{\binom{n}{k}} \leq 2^{-k}$, which decreases exponentially with k. Therefore, if the adversary is willing to trade off success probability for reduced circuit latency, they may choose a sufficiently small value of k. Moreover, since the circuit $\mathsf{Recover}_{S^c}(\mathsf{ct}, k)$ is par-

[5] With only negligible probability of failure, due to the properties of the underlying RLWE scheme; see Definitions 2 and 4.

allelizable, increasing k may not lead to increased latency, depending on the adversary's computational capabilities.

Even if the attack fails, the output remains $\mathsf{Enc}(0, \ldots, 0)$, and thus, by the definition of $\mathsf{ct}_{\mathsf{forged}}$ in 1, the attack remains undetected. Hence, the attack can still be carried out by a covert adversary, even when k is sufficiently small.

Corollary 3. *The scheme* REP *is not secure against the covert adversary* $\mathcal{A}_{\mathsf{cov}}^{\mathsf{rtk}_1}$.

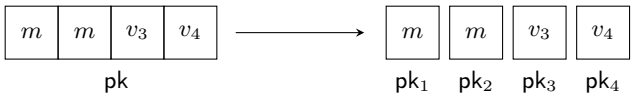
The following result follows directly from Corollaries 2 and 3.

Corollary 4. *If BFV bootstrapping requires the key* rtk_1*, then* REP *is either not bootstrappable or insecure against covert/malicious adversaries.*

4.3 Attack on Replication Encoding with Multiple Secret Keys

Patch on REP using Multiple Secret Keys. We introduce $\mathsf{REP}^{\mathsf{msk}}$, a countermeasure to the previous deterministic attack. The previous attack evaluates homomorphic rotations to compare values in different slots and recover $\mathsf{Enc}(J_{S^c})$. Notably, a rotation key with index 1 is essential for executing this attack. Meanwhile, in REP, a single value m is encoded into n slots. Therefore, to perform operations on two values m_1 and m_2 encoded via REP, only rotation keys whose index is a multiple of n will be required. Nevertheless, we note that a rotation key with index 1 remains essential for the bootstrapping of homomorphic encryption. Thus, the idea of preventing the above attack by not providing a rotation key with index 1 is not adequate.

Instead of restricting the rotation key with index 1, we consider the following possibility: Replicate the messages into n different ciphertexts encrypted with *different keys*. This would prevent interoperability between ciphertexts, making it difficult for the adversary to homomorphically determine in which slots the messages belong to. First, prepare a random subset $S \subset \{1, \ldots, n\}$ with $|S| = n/2$, a message $m \in \mathbb{Z}_t$, and verification values $v_i \leftarrow \mathbb{Z}_t$ for $i \in S$. Next, generate n different keys $\{\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{evk}_i\}$ for $i \in \{1, \ldots, n\}$. Now, instead of encrypting messages as a vector, we encrypt them elementwisely, i.e. $\mathsf{ct}_i = \mathsf{Enc}_{\mathsf{pk}_i}(m)$ for $i \in S$ and $\mathsf{ct}_i = \mathsf{Enc}_{\mathsf{pk}_i}(v_i)$ for $i \notin S$. Then, the server can compute those ciphertexts with different evk_i's. This patch achieves the same effect as the original REP while preventing the previous attack in Algorithm 3 by restricting the homomorphic comparison, while still providing the bootstrapping functionality.



However, we can also attack $\mathsf{REP}^{\mathsf{msk}}$ similarly to our attack on vADDG, namely, by implementing a pseudorandom characteristic function. It is possible since the evaluation does not rely on homomorphic rotation or comparison but only on slot-wise operations. We analyze how this attack applies to this variant.

Moreover, in vADDG, it was able to patch the attack by increasing the number of verification values by adjusting some parameters. We intend to investigate whether the same approach can be applied to patch $\mathsf{REP}^{\mathsf{msk}}$ or not.

Attack on $\mathsf{REP}^{\mathsf{msk}}$. First, let the adversary fix a random subset $A \subset \mathbb{Z}_t$ where the message m is expected to belong. Next, evaluate χ_A homomorphically on a ciphertext. After evaluating χ_A, it becomes possible to forge the values that exactly belong to A. If $\{m, v_i : i \in S\} \cap A = \{m\}$, then CMult between $\mathsf{ct}_S := \mathsf{Eval}_{\chi_A}(\mathsf{ct})$ and the encoding of the vector $(\overbrace{1 \dots 1}^{n} \| \overbrace{0 \dots 0}^{N-n})$ is an encryption of a vector $(J_{S^c} \| \overbrace{0 \dots 0}^{N-n})$.

Algorithm 4 Extended Attack on REP

Input: A ciphertext ct encrypted via REP and a subset $A \subset \mathbb{Z}_t$.
Output: A ciphertext possibly encrypting J_{S^c} in the first n slots.
procedure $\mathsf{Extended_Attack}(\mathsf{ct}, A)$
    $\mathsf{ct}_{\mathsf{bool}} \leftarrow \mathsf{Eval}_{\chi_A}(\mathsf{ct})$
    $\mathsf{pt}_{\mathsf{Masking}} \leftarrow (\overbrace{1, \dots, 1}^{n}, \overbrace{0, \dots, 0}^{N-n})$
    $\mathsf{ct}_{S^c} \leftarrow \mathsf{CMult}(\mathsf{ct}_{\mathsf{bool}}, \mathsf{pt}_{\mathsf{Masking}})$
    return ct_{S^c}
end procedure

Let us calculate the attack success probability with respect to the size of A. Since A is chosen randomly, the success probability of the attack depends only on the size of A. Let $p = |A|/t$ be the probability of each element m or v_i belonging to the set A. Set $q := 1 - p$ and $\mu := |\{m, v_i : i \in S\}| = 1 + \frac{n}{2}$.

Theorem 4. *Let A be a randomly chosen subset of $\mathbb{Z}_t$ with size $|A| = p \cdot t$. For $\mathsf{ct}_{S^c} \leftarrow \mathsf{Extended_Attack}(\mathsf{ct}, A)$ in Algorithm 4, the probability*

$$\Pr\left[\mathsf{Dec}(\mathsf{ct}_{S^c}) = (J_{S^c} \| \overbrace{0 \dots 0}^{N-n})\right] =: Q(p)$$

is maximized when $p = \mu^{-1} = \frac{1}{1+(n/2)}$. Also $Q(p) > e^{-1}p$ when restricted to $p = \mu^{-1}$.

Proof. Let's define $\mathcal{V} := \{m, v_i : i \in S\}$. Then, there are three possible cases:

- Case (1): $\mathcal{V} \cap A = \emptyset$. The probability is q^{μ}.
- Case (2): $\{v_i : i \in S\} \cap A \neq \emptyset$. The probability is $1 - q^{\mu-1}$.
- Case (3): $\mathcal{V} \cap A = \{m\}$. The probability is $q^{\mu-1} - q^{\mu}$.

Nothing happens in the case (1), and the adversary is caught in the case (2). The case (3) is the case of a successful attack. Now represent the probability of case (3) $Q(p)$ in terms of p: $Q(p) = q^{\mu-1} - q^{\mu} = (1-p)^{\mu-1} - (1-p)^{\mu}$.

By simple calculus, one can verify that $\frac{dQ}{dp}(\mu^{-1}) = 0$ and $Q(p)$ is maximized when $p = \mu^{-1}$: $Q(\mu^{-1}) = (1-\mu^{-1})^{\mu-1} - (1-\mu^{-1})^{\mu} = (1-p)^{(1/p)-1} - (1-p)^{(1/p)}$.

Now let $R(p) := (1-p)^{(1/p)-1} - (1-p)^{(1/p)}$. Then, $\lim_{p\to 0+} \frac{dR}{dp}(p) = e^{-1}$ and $\frac{d^2R}{dp^2} > 0$. Thus $R(p) > e^{-1}p$. □

Therefore, for any given parameter $\mu = 1 + (n/2)$, if the adversary randomly chooses a subset A so that $p = |A|/t$ is approximately μ^{-1}, and homomorphically evaluates χ_A, then the adversary's advantage is maximized with a probability at least $(e\mu)^{-1}$.

Thus, unlike in the vADDG attack where a countermeasure was possible by adjusting parameters, in this variant of REP a forgery attack can be mounted with non-negligible probability. However, under the assumption of a covert adversary model, such an attack is not feasible.

Pseudorandom Characteristic Function over $\mathbb{Z}_t$. The attack cost in Theorem 4 is determined by the evaluation cost of the characteristic function. In the attack on vADDG, the evaluation was straightforward since extracting encrypted bits from the TFHE ciphertext string was easy. However, homomorphically evaluating χ_A for any randomly given subset A with the full support $\mathbb{Z}_t$ generally requires $O(\sqrt{t})$ homomorphic operations, which results in an exponential cost with respect to $\log t$. Therefore, it is necessary to devise an efficient (polynomial time) method for evaluating a pseudorandom characteristic function over $\mathbb{Z}_t$.

Before constructing a pseudorandom characteristic function, note that the pseudorandom characteristic function used in this attack does not need to be cryptographically secure; rather, it is sufficient for it to be a function that heuristically produces an unbiased uniform distribution.

To construct an efficient pseudorandom characteristic function, we assume the pseudorandomness of the distribution of the roots of unity in $\mathbb{Z}_t$: For an integer t and $d \mid \phi(t)$, let $U_{t,d} := \{x \in \mathbb{Z}_t^{\times} : x^{\phi(t)/d} = 1\} \subset \mathbb{Z}_t^{\times}$. We assume that this subset $U_{t,d}$ of size $\phi(t)/d$ is an unbiased sample from a uniform distribution of $(\phi(t)/d)$-combinations from $\mathbb{Z}_t^{\times}$. Specifically, for a large odd prime p, $U_{p,2}$ is the set of quadratic residues modulo p, whose pseudorandomness is utilized to construct the Legendre PRF [Dam90]. Although the specific assumptions and parameters in here are different from those of Legendre PRF, we again note that it is sufficient for it to be a function that heuristically produces an unbiased uniform sampling.

Now we state the property of the characteristic function $\chi_{U_{t,d}} : \mathbb{Z}_t \to \{0,1\}$ for $t = p$. For $t = p^r, r > 1$, refer Appendix B.

Lemma 1. *For $t = p$ and $d \mid p - 1 = \phi(t)$, we have*

$$d \cdot \chi_{U_{t,d}}(x) = \sum_{i=0}^{d-1} \left(x^{\phi(t)/d}\right)^i.$$

Proof. 1. First, suppose that $x = 0$. Then the left-hand side (LHS) and the right-hand side (RHS) is 0.
2. Next, suppose that $x \in U_{t,d}$. Since $x^{\phi(t)/d} = 1$ by definition of $U_{t,d}$, both LHS and RHS are equal to d.
3. Lastly, suppose that $x \in \mathbb{Z}_t^\times \setminus U_{t,d}$. Then, $x^{\phi(t)/d} \neq 1$ by definition of $U_{t,d}$ Also, $(x^{\phi(t)/d})^d - 1 = (x^{\phi(t)/d} - 1)\left(\sum_{i=0}^{d-1} \left(x^{\phi(t)/d}\right)^i\right) = 0$. Thus, $\sum_{i=0}^{d-1} \left(x^{\phi(t)/d}\right)^i = 0$ since $\mathbb{Z}_t$ is an integral domain.

□

Note that this function can be homomorphically evaluated using $O(\log t)$ homomorphic multiplications with a multiplication depth of $\lceil \log \phi(t) \rceil$.

By homomorphically evaluating $\chi_{U_{t,d}}(x + a)$ for a random integer a and an appropriate divisor $d \mid (p - 1)$ with $d \approx \mu$, the adversary can construct a characteristic function χ_A whose support has size $|A| = \phi(t)/d \approx \lfloor t/\mu \rceil$, thereby achieving the maximal attack probability in Theorem 4. If necessary, the adversary may evaluate several characteristic functions to construct another size of characteristic function.[6]

Corollary 5. $\mathsf{REP}^{\mathsf{msk}}$ *is not secure against the presence of* $\mathcal{A}_{\mathsf{mal}}$.

Whether there exists a method to successfully execute a forgery attack with overwhelming probability remains an open question.

5 Cryptanalysis on Polynomial Encoding

5.1 Polynomial Encoding

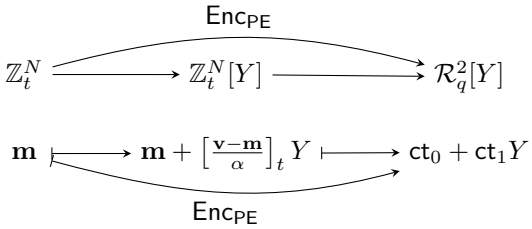
There is another suggested scheme for verifiable HE in [CKP+24], the Polynomial Encoding (PE). Its plaintext and ciphertext spaces are inherited from the original homomorphic encryption scheme, augmented with a new indeterminate Y. For example, in the BFV scheme, its encryption is

$$\mathbb{Z}_t^N[Y] \xrightarrow{\mathsf{Enc}_{\mathsf{PE}}} \mathcal{R}_q^2[Y],$$

where the $\mathsf{Enc}_{\mathsf{PE}}$ is the coefficient-wise encryption. To avoid confusion, we represent the elements of $\mathcal{R}_q^2[Y]$ using capitalized sans-serif font: $\mathsf{F}(Y) = \mathsf{ct}_0 + \mathsf{ct}_1 Y + \cdots + \mathsf{ct}_d Y^d$. We refer to $\mathsf{F}(Y)$ as a ciphertext polynomial and each of its coefficients as a ciphertext.

Encryption in PE proceeds as follows: For the number of slots N, a random verification value $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{Z}_t^N$ and a secret value $\alpha \in \mathbb{Z}_t^\times$ are chosen. Then, the message $\mathbf{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$ and the verification value $\mathbf{v}$ are interpolated at $Y = 0$ and $Y = \alpha$, respectively. After that, each message is encrypted coefficientwise:

[6] For example, one may utilize the formula $\chi_A(x) \cdot \chi_B(x) = \chi_{A \cap B}(x)$.



The server coefficientwise performs homomorphic operations like addition, constant multiplication, rotation, and even bootstrapping, except for multiplication. The multiplication is evaluated as a polynomial in Y: For example, the multiplication between two ciphertext polynomials $\mathsf{ct}_0 + \mathsf{ct}_1 Y$ and $\mathsf{ct}_2 + \mathsf{ct}_3 Y$ is

$$\mathsf{Mult}(\mathsf{ct}_0, \mathsf{ct}_2) + \mathsf{Mult}(\mathsf{ct}_1, \mathsf{ct}_2)Y + \mathsf{Mult}(\mathsf{ct}_0, \mathsf{ct}_3)Y + \mathsf{Mult}(\mathsf{ct}_1, \mathsf{ct}_3)Y^2.$$

If the server has performed the operations correctly, the result $\mathsf{F}(Y) \in \mathcal{R}_q^2[Y]$ satisfies $\mathsf{Dec}(\mathsf{F}(0)) = f(\mathbf{m})$ and $\mathsf{Dec}(\mathsf{F}(\alpha)) = f(\mathbf{v})$. In the verification step, the client checks whether $\mathsf{Dec}(\mathsf{F}(\alpha)) = f(\mathbf{v})$ and if it is correct, accepts $\mathsf{Dec}(\mathsf{F}(0))$ as $f(\mathbf{m})$.

Re-Quadratization Protocol. As the computation progresses, the degree of Y increases exponentially, leading to significant computational overhead. To mitigate this, [CKP+24] proposed a client-assisted computing protocol called the Re-Quadratization (ReQ) protocol. This protocol makes a quartic polynomial $\mathsf{Q}_4(Y) \in \mathcal{R}_q^2[Y]$ into a quadratic polynomial $\mathsf{Q}_2(Y) \in \mathcal{R}_q^2[Y]$ while ensuring that $\mathsf{Dec}(\mathsf{Q}_4(0)) = \mathsf{Dec}(\mathsf{Q}_2(0))$.

However, if $\mathsf{Dec}(\mathsf{Q}_4(\alpha)) = \mathbf{a}$, then $\mathsf{Dec}(\mathsf{Q}_2(\alpha)) = \mathbf{a} + \mathbf{r}$ for some uniform random blinding vector $\mathbf{r} \in \mathbb{Z}_t^N$. Because of this random blinding vector $\mathbf{r}$, the client must compute and handle the deviations of the circuit introduced by $\mathbf{r}$. For instance, when performing squaring $(\mathbf{a}+\mathbf{r})^2 = \mathbf{a}^2 + 2\mathbf{a}\mathbf{r} + \mathbf{r}^2$, the user must compute the deviation $2\mathbf{a}\mathbf{r}+\mathbf{r}^2$ and subtract it to recover $\mathbf{a}^2$ in the ReQ protocol. This self-correctness property through client-side computation forces the client to perform as much computation as if the client itself were running the delegated computation in plaintext, negating much of the intended benefit of HE.

Despite this drawback, the ReQ protocol reduces the computational overhead of homomorphic computation via PE. If the server multiplies two quadratic polynomials $\mathsf{Q}_2(Y)$, $\mathsf{Q}_2'(Y)$ and gets a quartic polynomial $\mathsf{Q}_4(Y)$, then the server can use ReQ to reduce $\mathsf{Q}_4(Y)$ back to a quadratic polynomial $\mathsf{Q}_2''(Y)$. Therefore the server can evaluate a deep circuit avoiding an exponential overhead. See V.D. and Appendix E in [CKP+24] for details. Here, we only use ReQ protocol as an oracle for avoiding overhead in deep circuit evaluation.

5.2 Attack on Polynomial Encoding

Now we present the attack on the Polynomial Encoding. We assume that the ReQ protocol serves as a subroutine in the computation.

Let $f(x_1, \ldots, x_k)$ be an honest polynomial circuit that operates on the ciphertext polynomial space $\mathcal{R}_q^2[Y]$. Given k inputs $\mathsf{L}_i(Y) = \mathsf{ct}_i^{(0)} + \mathsf{ct}_i^{(1)} Y$ for $i = 1, \ldots, k$, the polynomial circuit f processes these inputs and produces the output $f(\mathsf{L}_1(Y), \ldots, \mathsf{L}_k(Y)) = \mathsf{F}(Y) \in \mathcal{R}_q^2[Y]$. When evaluated at $Y = \alpha$, we obtain

$$\mathsf{F}(\alpha) = \mathsf{Eval}_f(\mathsf{ct}_1^{(0)} + \alpha \cdot \mathsf{ct}_1^{(1)}, \ldots, \mathsf{ct}_k^{(0)} + \alpha \cdot \mathsf{ct}_k^{(1)}) = \mathsf{Enc}(f(v_1, \ldots, v_k)).$$

Now let g be any malicious polynomial circuit that outputs $\mathsf{G}(Y)$. Similarly, when evaluated at $Y = 0$, it has

$$\mathsf{G}(0) = \mathsf{Eval}_g(\mathsf{ct}_0^{(0)}, \ldots, \mathsf{ct}_k^{(0)}) = \mathsf{Enc}(g(m_1, \ldots, m_k)).$$

Since only the result $\mathsf{G}(0)$ will be needed, the circuit g does not need to act on the entire ciphertext polynomial; it only needs to operate on its constant term, namely $\mathsf{ct}_0^{(0)}, \ldots, \mathsf{ct}_k^{(0)}$. Consequently, the evaluation of g does not require ReQ. To forge the result, we introduce a trick: define $\mathsf{L}^\star(Y) := \mathsf{Enc}(u_1, \ldots, u_N)Y$ where all $u_i \in \mathbb{Z}_t^\times$. For the polynomial circuit $p(y) := y^{\phi(t)}$, evaluating p at $\mathsf{L}^\star(Y)$ yields $\mathsf{P}(Y)$ which satisfies

$$\mathsf{P}(0) = p(\mathsf{L}^\star(0)) = \mathsf{Enc}(0, \ldots, 0), \quad \mathsf{P}(\alpha) = p(\mathsf{L}^\star(\alpha)) = \mathsf{Enc}(1, \ldots, 1).$$

Thus, if we evaluate the following polynomial circuit

$$h(y_1, \ldots, y_k, \ell^\star) := g(\tilde{y}_1, \ldots, \tilde{y}_k) + (f(y_1, \ldots, y_k) - g(\tilde{y}_1, \ldots, \tilde{y}_k))p(\ell^\star)$$

at $(y_1, \ldots, y_k, \ell^\star) \leftarrow (\mathsf{L}_1(Y), \ldots, \mathsf{L}_k(Y), \mathsf{L}^\star(Y))$ where $\tilde{y}_i = \mathsf{L}_i(0)$ for $y_i = \mathsf{L}_i(Y)$, the output $\mathsf{H}(Y)$ satisfies

$$\mathsf{Dec}(\mathsf{H}(0)) = \mathsf{Dec}(\mathsf{G}(0)), \quad \mathsf{Dec}(\mathsf{H}(\alpha)) = \mathsf{Dec}(\mathsf{F}(\alpha))$$

which successfully passes the verification and gives the forged results.

Algorithm 5 Attack against PE

Input: $\{\mathsf{L}_i(Y)\}_{1 \le i \le k}$ are input points and $\mathsf{F}(Y)$ is a honest result.
Output: $\mathsf{H}(Y)$, a forged ciphertext polynomial.
procedure $\mathsf{PE_Attack}(\{\mathsf{L}_i(0)\}, \mathsf{F}(Y))$
  $\mathsf{ct}_{\mathsf{forged}} \leftarrow \mathsf{Eval}_g(\mathsf{L}_1(0), \ldots, \mathsf{L}_k(0))$ ▷ g is any circuit different from f.
  $\mathsf{L}^\star(Y) \leftarrow \mathsf{Enc}(u_1, \ldots, u_N)Y$ ▷ Every u_i is a unit.
  $\mathsf{P}(Y) \leftarrow p(\mathsf{L}^\star(Y))$ ▷ $p(y) = y^{\phi(t)}$.
  $\mathsf{H}(Y) \leftarrow \mathsf{ct}_{\mathsf{forged}} + (\mathsf{F}(Y) - \mathsf{ct}_{\mathsf{forged}}) \cdot \mathsf{P}(Y)$
  return $\mathsf{H}(Y)$
end procedure

Theorem 5. *If the adversary can access the ReQ protocol, Algorithm 5 can be executed within the cost of $O(\log t)$ with deterministic output $\mathsf{H}(Y)$.*

Proof. The correctness of the attack circuit is given above. Take g to be a constant circuit different from f. Then the compuational cost is determined by evaluating $p(y) = y^{\phi(t)}$ at $y = \mathsf{L}^\star(Y)$, which requires $\lceil \log(\phi(t)) \rceil$ multiplications in $\mathcal{R}_q^2[Y]$ and ReQ protocol. Therefore the total computational complexity is $O(\log t)$. □

Corollary 6. *PE is not secure against the presence of* $\mathcal{A}_{\mathsf{cov}}^{\mathcal{O}_{\mathsf{ReQ}}}$.

However, this attack can be detected by counting the number of ReQ calls. The honest output is $\mathsf{F}(Y)$, and in order to compute $H(Y)$, one must additionally compute $\mathsf{P}(Y)$ and then multiply it by $\mathsf{F}(Y) - \mathsf{ct}_{\mathsf{forged}}$. Therefore, compared to the honest circuit, the number of ReQ calls increases by at least $\lceil \log \phi(t) \rceil$. Thus, in this case, the adversary needs to deceive the client about the number of ReQ calls required to evaluate the honest circuit.

6 Attack Implementation on REP and REP^{msk}

We implemented our attacks in the VERITAS library, which is based on the Lattigo BFV Implementation [MBTPH20]. Specifically, we implemented the attack on REP (Algorithm 3) as well as the attack on $\mathsf{REP}^{\mathsf{msk}}$ (Algorithm 4). Since the Lattigo library currently does not support BFV bootstrapping, we modified the value of t in the parameters provided by the example code in the VERITAS library to ensure that the forgery circuit operates without bootstrapping. However, when bootstrapping is supported, the attack remains feasible regardless of the value of t. [7]. Specific parameters are given in Table 1.

As for vADDG, since the verifiable variant is not implemented, we did not implement the attack. However, since the attack circuit is very shallow and simple, and in particular very similar to the attack on $\mathsf{REP}^{\mathsf{msk}}$, the feasibility of the attack on vADDG is straightforward even without a demonstration via implementation.

6.1 Attack Description

The attack on REP and $\mathsf{REP}^{\mathsf{msk}}$ aims to implement the following circuit:

$$\mathsf{Mult}(\mathsf{Eval}_f(\mathsf{ct}), \mathsf{Enc}(J_S)) + c \cdot \mathsf{Enc}(J_{S^c})$$

for some nonzero constant c. Since $\mathsf{Enc}(J_S) = (1, \ldots, 1) - \mathsf{Enc}(J_{S^c})$, the attack reduces to recovering a scalar multiple of $\mathsf{Enc}(J_{S^c})$. Therefore, the attack cost corresponds to the amount of resources required to recover a scalar multiple of $\mathsf{Enc}(J_{S^c})$.

[7] For PE, t must be of size roughly equal to the security parameter λ, since the secret α is chosen from $\mathbb{Z}_t^\times$. With this parameter setting, the implementation of the attack is not feasible without bootstrapping. Thus, we did not implement attack on PE due to the lack of bootstrapping functionality.

6.2 Attack Results

The experiments were performed on an Intel(R) Xeon(R) Silver 4114 CPU at 2.20GHz running Linux in a single-threaded environment. The attack results are summarized in Tables 2, 3. The time cost only measured evaluation time for $\mathsf{Enc}(J_{S^c})$.

Attack on REP. In experiment 1 to 4, we implement the attack on REP described in Algorithm 3 with various $k = 1, 2, 4, 33$. We generated keys whose rotation indices are divisors of n, assuming a scenario in the adversarial server needs to bootstrap with these rotation keys. We optimized the attack circuit described in Algorithm 3 as follows. Instead of evaluating $\mathsf{Eval}_{\mathsf{Interpolate}}$, which maps $\mathsf{Enc}(1, \ldots, 1)$ to $\mathsf{Enc}(0, \ldots, 0)$ and $\mathsf{Enc}\,(n/2, \ldots, n/2)$ to $\mathsf{Enc}(1, \ldots, 1)$, we directly performed the subtraction $\mathsf{ct} \mapsto \mathsf{ct} - \mathsf{Enc}(1, \ldots, 1)$. This operation maps $\mathsf{Enc}(1, \ldots, 1)$ to $\mathsf{Enc}(0, \ldots, 0)$ and $\mathsf{Enc}\,(n/2, \ldots, n/2)$ to $\mathsf{Enc}\,(n/2 - 1, \ldots, n/2 - 1)$. As a result, we obtain a ciphertext that corresponds to a constant multiple of J_{S^c}.

According to Theorem 3, the expected attack success probabilities given by $1 - \binom{32}{k}/\binom{64}{k}$ are 50%, 75.40%, 94.34%, and 100% for $k = 1$, 2, 4, and 33, respectively. This attack circuit can be implemented without bootstrapping under the parameters $(\log N = 15, \log PQ = 880, \log Q = 759, t = 2^{16} + 1)$. As shown in Table 2, the attack time increases linearly with k, but we note that Algorithm 3 is inherently parallelizable, as it performs $\mathsf{CVS}(\mathsf{ct}, i)$ and $\mathsf{is_in_set}(\mathsf{ct}, i)$ independently for each $i = 1, \ldots, k$. Thus, by parallelizing the execution of $\mathsf{CVS}(\mathsf{ct}, i)$, the overall attack time can be reduced to under one minute even with $k = 33$. In addition, leveraging GPU-based optimizations could potentially reduce the runtime to just a few seconds. This assumption is realistic, given that adversaries in the context of HE are typically assumed to have access to significant computational resources.

Attack on $\mathsf{REP}^{\mathsf{msk}}$. We implement the attack on $\mathsf{REP}^{\mathsf{msk}}$ described in Algorithm 4. Since this attack naturally applies to REP, without any modification to the REP scheme we implemented this attack: Specifically, we did not generate the rtk_1.

Since $\phi(t) = 2^{16}$, when constructing the characteristic function in Lemma 1, we can choose d to be a power of 2. In Experiment 5, where $n = 64$, we set $d = 32 \cong \mu = 33$ for the characteristic function. As a result, the output becomes a scalar multiple of $\mathsf{Enc}(J_{S^c})$ with non-negligible probability. According to Theorem 4, the corresponding expected attack success probability is $\left(1 - \frac{2^{11}}{2^{16}+1}\right)^{32} - \left(1 - \frac{2^{11}}{2^{16}+1}\right)^{33} \cong 1.13\%$. Also in Experiment 6, where $n = 32$, we set $d = 16 \cong \mu = 17$. The corresponding expected attack success probability is $\left(1 - \frac{2^{12}}{2^{16}+1}\right)^{16} - \left(1 - \frac{2^{12}}{2^{16}+1}\right)^{17} \cong 2.22\%$. The parameters and results are presented in Table 1,2 and 3.

Parameters		$\log N$	$\log PQ$	$\log Q$	t	$\lvert S\rvert = \lvert\{1,\ldots,n\}\rvert$	k
REP Attack	Exp1	15	880	759	$2^{16}+1$	64	1
	Exp2						2
	Exp3						4
	Exp4						33
$\mathsf{REP}^{\mathsf{msk}}$Attack	Exp5	15	880	759	$2^{16}+1$	64	-
	Exp6					32	-

Table 1: Parameters Selection

Evaluation Time (Sec/op)		Attack Time (Evaluating $\mathsf{Enc}(J_{S^c})$)	Remark
REP Attack	Exp1	8.37s	rtk_1 generated.
	Exp2	15.91s	
	Exp3	29.67s	
	Exp4	246.54s	
$\mathsf{REP}^{\mathsf{msk}}$Attack	Exp5	6.05s	-
	Exp6	6.36s	-

Table 2: Attack Circuit Evaluation Cost

Probability		Theoretical Attack Success Probability	Experimental Attack Success Probability	# of Iterations
REP Attack	Exp1	50.00%	51.8%	1000
	Exp2	75.40%	75.5%	1000
	Exp3	94.34%	93.4%	1000
	Exp4	100%	100%	1
$\mathsf{REP}^{\mathsf{msk}}$Attack	Exp5	1.13%	0.8%	1000
	Exp6	2.22%	1.8%	1000

Table 3: Attack success probability (1000 iterations unless otherwise noted)

We remark that the drop in success probability observed in Exp. 5 and Exp. 6 is within one standard deviation. Also, due to the lack of BFV bootstrapping functionality in the Lattigo library, we implemented the attacks on REP and $\mathsf{REP}^{\mathsf{msk}}$ only for specific parameter settings, and did not implement the attack on PE. However in a bootstrappable setting, not only is an attack on PE feasible, but the above attacks could also be carried out independently of the parameter settings.

7 Discussion

7.1 Homomorphic Cryptanalysis

To provide verifiability, the schemes considered in this paper utilize auxiliary secret information in addition to the secret key of the HE scheme: a set of indices S in REP (which also applies essentially to vADDG) and an element $\alpha \in \mathbb{Z}_t^\times$ in

PE. Attacks against the schemes described in this paper consist of two steps as follows;

1. Recover secret values S or α in encrypted state.
2. Modify a legitimate ciphertext using the encryption of secret values.

Step 1: Recovering Secret Information in an encrypted state. The first step is to recover secret values, S or α, in an encrypted state. For REP, we homomorphically find S in two other ways: one way is based on the homomorphic comparison which deterministically recovers $\mathsf{Enc}(S)$, and the other way is based on the random characteristic function evaluation, which probabilistically outputs $\mathsf{Enc}(S)$ with non-negligible probability.

For PE, if $\mathsf{Enc}(\alpha)$ is recovered we can modify a legitimate ciphertext easily. However, ReQ adopted a re-randomized process to make recovering $\mathsf{Enc}(\alpha)$ hard. Here the proposed attack does not learn α. This is not only because, as mentioned earlier, PE encoding can be performed without knowledge of α or $\mathsf{Enc}(\alpha)$, but it can also be explained from the following perspective: We can think of the ciphertext polynomial $\mathsf{F}(Y) \in \mathcal{R}_q^2[Y]$ as a ciphertext with two secret keys

$$(1, s) \otimes (1, \alpha, \ldots, \alpha^{\deg F}), \quad (1, s) \otimes (1, 0, \ldots, 0).$$

For example, we think of the ciphertext $\mathsf{ct}_0 + \mathsf{ct}_1 Y$ as an ciphertext with two secret keys $(1, s) \otimes (1, \alpha) = (1, s, \alpha, \alpha s)$ and $(1, s) \otimes (1, 0) = (1, s, 0, 0)$, in the meaning that $(\mathsf{ct}_0, \mathsf{ct}_1) = \big((b_0, a_0), (b_1, a_1)\big)$ would be decrypted by the secret key $(1, s, \alpha, \alpha s)$ or $(1, s, 0, 0)$. In this sense,

$$\big((b_0, a_0), (b_1, a_1)\big) = \big(\mathsf{Enc}(0, \ldots, 0), \mathsf{Enc}(1, \ldots 1)\big)$$

is an encryption of α under $(1, s, \alpha, \alpha s)$ and an encryption of 0 under $(1, s, 0, 0)$. Indeed, one of the possible choice of $\mathsf{L}^\star(Y)$ is $\mathsf{Enc}(0, \ldots, 0) + \mathsf{Enc}(1, \ldots, 1)Y$, which is as an encryption of α and 0 in this very sense.

Step 2: Modify a legitimate ciphertext The second step is to homomorphically generate a cheating circuit using the information on secret values α or S. In the plaintext state, the forgery of the secret encoding is just a composition of decoding and encoding, where the decoding and encoding are performed with the aid of the secret values S or α, namely $\mathsf{Dcd}_{\mathsf{sv}}$ and $\mathsf{Ecd}_{\mathsf{sv}}$ where sv stands for secret values S or α. Thus the attack is just a homomorphic evaluation of such cheating circuits, with the aid of the information of S or α in an encrypted state: See Fig. 3.

For REP, the secret value for Decoding and Encoding is the verification slot S. Accordingly, the homomorphic decoding is just masking a ciphertext by multiplying the encryption of J_S and J_{S^c}, and the homomorphic encoding is adding ciphertexts masked by J_S and J_{S^c}.

For PE, the decoding involves evaluating at $Y = \alpha$ and $Y = 0$, while the encoding process interpolates $\mathbf{m}, \mathbf{v} \in \mathbb{Z}_t^N$ at $Y = 0$ and $Y = \alpha$ to obtain

$$\mathbf{m} + (\mathbf{v} - \mathbf{m})\alpha^{-1} \cdot Y.$$

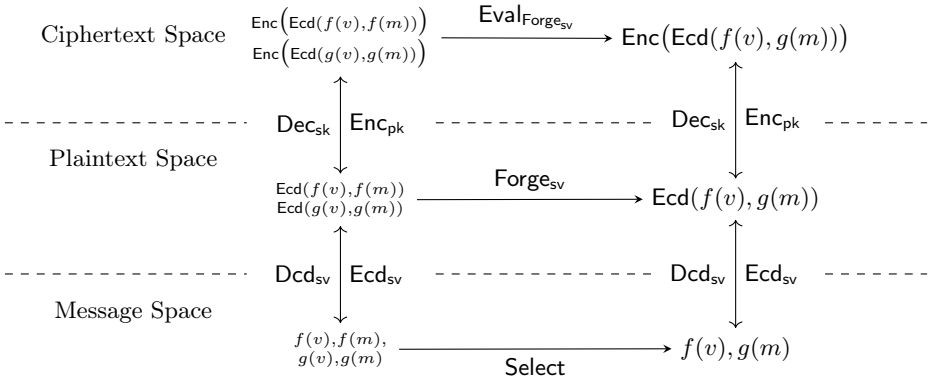


Fig. 3: An Overview of Homomorphic Forgery on Lightweight VHE. f and g denote the requested circuit and a malicious circuit, respectively.

According to the earlier perspective that $\mathsf{L}^\star := \mathsf{Enc}(0) + \mathsf{Enc}(1)Y$ can be seen as an encryption of 0 and α, interpolating the two ciphertext polynomials F and G yields

$$\begin{aligned}&\mathsf{G} + (\mathsf{F} - \mathsf{G}) \cdot (\mathsf{L}^\star)^{-1} Y \\ =&\mathsf{G} + (\mathsf{F} - \mathsf{G}) \cdot (\mathsf{L}^\star)^{\phi(t)-1} Y.\end{aligned}$$

This exhibits the structure of the previously discussed cheating circuit.

8 Conclusion

In this paper, we presented attacks against the verifiable homomorphic encryption schemes introduced in [CKP+24, ADDG24].

For the vADDG scheme of [ADDG24], we introduced a shallow pseudorandom characteristic function to overcome the limitation imposed by bootstrapping depth. The attack succeeds with probability $(1 - (1/2)^\alpha) \cdot (1/2)^\beta$ and has computational cost $O(\gamma)$. For REP in [CKP+24], we constructed a circuit that homomorphically computes the position vector of the common values. This attack has cost $O(k \log t)$ and succeeds with probability 1 when $k = n/2 + 1$. Our patching solution using multiple secret keys reduces the success probability to $O(n^{-1})$, which yields only very weak security, arguably sufficient only against a covert adversary. For PE, we exploit Euler's theorem to construct a deterministic forgery attack with computational cost $O(\log t)$.

One possible approach to decreasing the success probability of attacks is to restrict the circuit depth that can be homomorphically evaluated. However, as demonstrated by the attack on vADDG, a cheating circuit can still be constructed even at shallow depths. Moreover, an adversary may perform forgery despite risking computation failure due to noise robustness. Hence, the approach of limiting depth requires more thorough analysis.

Another approach is to employ multiple secrets, as discussed in Section 4.3. This restricts the class of HE circuits that can be evaluated. However, as shown in Section 4.3, simply assigning different secrets slotwise is insufficient to achieve malicious security. Thus, this approach of using multiple secrets also requires further and more precise analysis.

Acknowledgement

We would like to thank Damien Stehlé for his guidance and valuable comments on this research. We also thank Changmin Lee and the anonymous Asiacrypt2025 reviewers for their helpful feedback. This research was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP), grant funded by the Korea Government (MSIT) (RS-2021-II210727, A Study on Cryptographic Primitives for SNARK)

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.

ACGaSV24. 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 *Advances in Cryptology – ASIACRYPT 2024: 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9–13, 2024, Proceedings. Part V*, page 302–334, Berlin, Heidelberg, 2024. Springer-Verlag.

ADDG24. Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious PRFs from shallow PRFs and TFHE. In *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part VI*, page 447–476, Berlin, Heidelberg, 2024. Springer-Verlag.

AL07. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: efficient protocols for realistic adversaries. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, page 137–156, Berlin, Heidelberg, 2007. Springer-Verlag.

BCFK21. Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In *Public-Key Cryptography – PKC 2021: 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part II*, page 528–558, Berlin, Heidelberg, 2021. Springer-Verlag.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

CCRS24. Davide Carnemolla, Dario Catalano, Mario Di Raimondo, and Federico Savasta. Implementation and performance analysis of homomorphic signature schemes. Cryptology ePrint Archive, Paper 2024/655, 2024.

CF13. Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 336–352, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved fhe bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 315–337, Cham, 2018. Springer International Publishing.

CHLR18. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1223–1237, New York, NY, USA, 2018. Association for Computing Machinery.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

CKP+24. Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. VERITAS: Plaintext encoders for practical verifiable homomorphic encryption. CCS '24, New York, NY, USA, 2024. Association for Computing Machinery.

Dam90. Ivan Bjerre Damgård. On the randomness of Legendre and Jacobi sequences. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, pages 163–172, New York, NY, 1990. Springer New York.

FGP14. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 844–855, New York, NY, USA, 2014. Association for Computing Machinery.

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, *Public-Key Cryptography – PKC 2020*, pages 124–154, Cham, 2020. Springer International Publishing.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

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.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 465–482, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

GGW24. Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part X*, page 449–487, Berlin, Heidelberg, 2024. Springer-Verlag.

GNSV23. Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *Journal of Cryptology*, 36, 2023.

GV23. Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV revisited. *J. Cryptol.*, 36(2), March 2023.

GW13. Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 301–320, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Joy22. Marc Joye. Guide to fully homomorphic encryption over the [discretized] torus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):661–692, Aug. 2022.

KPZ21. Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 608–639, Cham, 2021. Springer International Publishing.

LHW+25. Junkai Liang, Daqi Hu, Pengfei Wu, Yunbo Yang, Qingni Shen, and Zhonghai Wu. SoK: Understanding zk-SNARKs: The gap between research and practice. Cryptology ePrint Archive, Paper 2025/172, 2025.

MBTPH20. Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Lattigo: A multiparty homomorphic encryption library in go. page 64–70, 2020.

SXLS24. Muhammad Husni Santriaji, Jiaqi Xue, Qian Lou, and Yan Solihin. Dataseal: Ensuring the verifiability of private computation on encrypted data. *CoRR*, abs/2410.15215, 2024.

VKH23. Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption, 2023.

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.

A BFV Scheme

We recall the details of the BFV homomorphic encryption scheme [Bra12,FV12]. For more comprehensive descriptions, we refer the reader to [GV23].

Structure of $\mathcal{R}_t$. Let M be a power of 2 and define the cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for $q \in \mathbb{N}$. Let p be a prime and let d be the multiplicative order of p in $\mathbb{Z}_M^\times$. For $t = p^r$, there exists an injective ring homomorphism $\mathbb{Z}_t^N \hookrightarrow \mathcal{R}_t$ where $N = M/d$. Moreover, the Galois automorphisms on $\mathcal{R}_t$ corresponds to some permutation actions, called a 'rotation', on $\mathbb{Z}_t^N$. Thus, operations on slots (i.e., on $\mathbb{Z}_t^N$) are realized via the ring operations in $\mathcal{R}_t$.

When encoding data from $\mathbb{Z}_t$ into $\mathcal{R}_t$, one may either use the injective ring homomorphism described above or directly encode values into the coefficients of a polynomial in $\mathcal{R}_t$. The former is referred to as *slot encoding*, while the latter is called *coefficient encoding*. Note that coefficient encoding is not ring-homomorphic, but it is occasionally used in special cases.

BFV Scheme. The BFV scheme is a homomorphic encryption scheme that supports both ring operations and Galois automorphisms over $\mathcal{R}_t$. Its ciphertext space is $\mathcal{R}_q^2$ for $q > t$. Let $\Delta = \lfloor q/t \rfloor$, and let χ_{key} and χ_{err} be probability distributions over $\mathcal{R}_q$.

- KeyGen: Sample a secret key $s \leftarrow \chi_{\mathsf{key}}$ and generate the associated evaluation keys. The relinearization key rlk is generated so that, for any $a \in \mathcal{R}_q$, one can (oblivious to s) find $\alpha, \beta \in \mathcal{R}_q$ such that $[as^2]_q \cong [\alpha s + \beta]_q$. Similarly, for a Galois automorphism σ, the rotation key rtk_σ is generated so that, for any $a \in \mathcal{R}_q$, one can (oblivious to s) find $\alpha, \beta \in \mathcal{R}_q$ such that $[a\sigma(s)]_q \cong [\alpha s + \beta]_q$.
- Enc: For a message $m \in \mathcal{R}_t$ and secret key s, sample $c_1 \leftarrow U(\mathcal{R}_q)$ and $e \leftarrow \chi_{\mathsf{err}}$, and output $(c_1, c_0) = (c_1,\ -c_1 s + \Delta m + e) \in \mathcal{R}_q^2$.
- Dec: For a ciphertext (c_1, c_0) and secret key s, output $\left\lfloor \frac{t}{q}[c_1 s + c_0]_q \right\rceil \in \mathcal{R}_t$.
- Eval: For simplicity, here we identify a plaintext m with an insecure ciphertext $(0, \Delta m)$, so that plaintext-ciphertext operations can be described as ciphertext–ciphertext operations.
  - Add: Given (c_1, c_0) and (c_1', c_0'), output $(c_1 + c_1', c_0 + c_0')$.
  - Mult: Given (c_1, c_0) and (c_1', c_0'), first compute $(d_2, d_1, d_0) = (c_1 c_1', c_1 c_0' + c_1 c_0', c_0 c_0')$. Scale each by t/q, round, and mod-out q, i.e., $(\lceil \lfloor \frac{t}{q} d_2 \rceil \rceil_q, \lceil \lfloor \frac{t}{q} d_1 \rceil \rceil_q, \lceil \lfloor \frac{t}{q} d_0 \rceil \rceil_q)$. Then, using rlk, convert $\lceil \lfloor \frac{t}{q} d_2 \rceil \rceil_q \in \mathcal{R}_q$ into (α, β) such that $\lceil \lfloor \frac{t}{q} d_2 \rceil s^2 \rceil_q \cong [\alpha s + \beta]_q$. Output $\left(\lceil \lfloor \frac{t}{q} d_1 \rceil + \alpha \rceil_q,\ \lceil \lfloor \frac{t}{q} d_0 \rceil + \beta \right\rceil_q)$.
  - Rot_σ: Given (c_1, c_0) and a Galois automorphism σ, compute $(\sigma(c_1), \sigma(c_0))$. Using rtk_σ, convert $\sigma(c_1)$ into (α, β) such that $[\sigma(c_1)\sigma(s)]_q \cong [\alpha s + \beta]_q$. Output $(\alpha,\ [\sigma(c_0) + \beta]_q)$.

Modulus Switching. One can change the ciphertext modulus q. For $(c_1, c_0) \in \mathcal{R}_q^2$, the pair $\left(\lfloor \frac{q'}{q} c_1 \rceil, \lfloor \frac{q'}{q} c_0 \rceil \right) \in \mathcal{R}_{q'}^2$ is an encryption of the same plaintext, and this procedure is called modulus switching.

Slot–Coefficient Transformation. When computations on the coefficients themselves are required, it is also possible to switch between slot encoding and coefficient encoding in the encrypted state. Such conversions essentially amount to $\mathbb{Z}_t$-linear algebra over $\mathcal{R}_t$ and can be performed using only rotation keys and the relinearization key.

B Attacks on REP and REP^{msk} for $t = p^r$

The attack circuit described in the main text is a function over $\mathbb{Z}_t$. When $t = p$ is prime, every function can be represented by a polynomial. However, when t is not prime, not every function $f : \mathbb{Z}_t \to \mathbb{Z}_t$ can be represented by a polynomial over $\mathbb{Z}_t$, since in this case p is not invertible in $\mathbb{Z}_t$. Depending on the underlying HE scheme, however, there exists a way to overcome this issue. This appendix provides additional technical details for extending our attacks to the case $t = p^r$ on the BFV scheme. For BGV, one can apply the standard BFV-BGV ciphertext conversion (cf. Appendix A in [KPZ21]).

B.1 Evaluating Non-polynomial Functions.

Divide-by-p and Its Inverse. One can evaluate the 'divide-by-p' function

$$p\mathbb{Z}_{p^r} \xrightarrow{p^{-1}} \mathbb{Z}_{p^{r-1}}$$

together with its inverse operation. Consider a BFV ciphertext $(c_0, c_1) \in \mathcal{R}_q^2$ satisfying $c_0 + c_1 s = \Delta pm + e + qI$, where pm is the plaintext of multiple of p and e is error. If we interpret the scaling factor as $\Delta' = \Delta p$, then the plaintext modulus is naturally p^{r-1}, and the ciphertext can be regarded as an encryption of m under this modulus.

Divide-by-p^{r-1} without Modulus Reduction. However, while the above operation is computationally free, it has the drawback of reducing the plaintext modulus, which in turn limits the available p-adic precision. We now show that it is possible to implement the following operation for a BFV ciphertext without reducing the plaintext modulus:

$$p^{r-1} \cdot \mathbb{Z}_{p^r} \xrightarrow{(p^{r-1})^{-1}} \mathbb{Z}_{p^r}.$$

First, we modulus-switch the BFV ciphertext so that q becomes a multiple of $t = p^r$ ($q = \Delta \cdot p^r$). Then, assuming via a slot–coefficient transformation that

the message is coefficient-encoded, the BFV ciphertext $(c_0, c_1) \in \mathcal{R}_q^2$ satisfies $c_0 + c_1 s = qI + \Delta p^{r-1} m + e$. Then, $\left(\left\lfloor \frac{c_0}{p^{r-1}} \right\rceil, \left\lfloor \frac{c_1}{p^{r-1}} \right\rceil\right) \in \mathcal{R}_{q/p^{r-1}}^2$ satisfies

$$\begin{aligned}\left\lfloor \tfrac{c_0}{p^{r-1}} \right\rceil + \left\lfloor \tfrac{c_1}{p^{r-1}} \right\rceil s &= \tfrac{q}{p^{r-1}} I + \Delta m + \left\lfloor \tfrac{e}{p^{r-1}} \right\rceil + (\delta_0 + \delta_1 s + \delta_2) \\ &= \Delta(pI + m) + \left\lfloor \tfrac{e}{p^{r-1}} \right\rceil + (\delta_0 + \delta_1 s + \delta_2)\end{aligned}$$

where the δ_i are small polynomials arising from the rounding, of size less than 1. Embedding this ciphertext back into $\mathcal{R}_q^2$, it becomes an encryption of $(pI + m) \bmod p^r$. By subsequently applying the digit-extraction polynomial [CH18] on the coefficients, one can recover an encryption of $m \bmod p^r$. This operation is computationally expensive, involving slot-coefficient trasformations, yet it can be evaluated in polynomial time.

B.2 Attacks on REP

We first modify the $\mathsf{Compare}$ function by evaluating its multiple $\phi(t) \cdot \mathsf{Compare}$. Note that $\phi(t) = \phi(p^r) = (p-1)p^{r-1}$.

Lemma 2. *$\mathsf{Eval}_{\phi(t)\mathsf{Compare}}$ can be evaluated with $O(\log t)$ homomorphic multiplications with plaintext modulus $t = p^r$ for $r > 1$.*

Proof. Let $e(x) = 1 - x^{\phi(t)}$, and define

$$\alpha(x) := e(x) \cdot \prod_{u \in \mathbb{Z}_t^\times \setminus \{1\}} (x + 1 - u).$$

Note that

$$\prod_{u \in \mathbb{Z}_t^\times \setminus \{1\}} (x + 1 - u) = \frac{(x+1)^{\phi(t)} - 1}{(x+1) - 1} = \sum_{i=0}^{\phi(t)-1} (x+1)^i,$$

so α can be evaluated with $O(\log t)$ homomorphic multiplications. Now consider the following cases:

1. If $x = 0$, then $\alpha(0) = e(0) \cdot \phi(t) = \phi(t)$.
2. If $x \in \mathbb{Z}_t^\times$, then $e(x) = 0$ and hence $\alpha(x) = 0$.
3. If x is nonzero and non-unit, then $x + 1 \in \mathbb{Z}_t^\times \setminus \{1\}$, so $\alpha(x) = e(x) \cdot \prod_u (x + 1 - u) = e(x) \cdot 0 = 0$.

Thus $\alpha(x - y) = \phi(t) \cdot \mathsf{Compare}(x, y)$. □

By replacing $\mathsf{Eval}_{\mathsf{Compare}}$ with $\mathsf{Eval}_{\phi(t)\cdot\mathsf{Compare}}$ in Algorithm 2, we obtain:

- For $i \in S^c$: $\mathsf{ct}_{\mathsf{CVS},i}$ is an encryption of $\phi(t) \cdot J_{S^c}$.
- For $i \in S$: $\mathsf{ct}_{\mathsf{CVS},i}$ is an encryption of $\phi(t) \cdot e_i$, where e_i denotes the ith standard vector.

Consequently, the resulting ciphertext $\mathsf{Eval}_f(\mathsf{ct}) + \mathsf{ct}_{\mathsf{CVS},i}$ constitutes a forgery with probability 1/2.

Increasing Attack Success Probability. We now modify Algorithm 3 to increase the attack success probability. Let $\mathsf{ct}_{\mathsf{bool},i}$ denote the output obtained by applying the function $x \mapsto x - \phi(t)$ coefficient-wise to $\mathsf{RotSum}(\mathsf{ct}_{\mathsf{CVS},i})$. Then:

- For $i \in S^c$: $\mathsf{ct}_{\mathsf{bool},i}$ is an encryption of $((\frac{n}{2}-1)\cdot\phi(t), \ldots, (\frac{n}{2}-1)\cdot\phi(t))$.
- For $i \in S$: $\mathsf{ct}_{\mathsf{bool},i}$ is an encryption of $(0, \ldots, 0)$.

If $(\frac{n}{2}-1) \mid p$, then both cases degenerate to the zero vector, which causes a problem. For such exceptional parameter choices, one can resolve the issue by multiplying $(1, \ldots, 1) - e_j$ to $\mathsf{ct}_{\mathsf{CVS},i}$ for some $j \neq i$, and then applying the above procedure. This yields $(\frac{n}{2}-2)\cdot\phi(t)$ for $i \in S^c$ instead, thereby avoiding the problem. In the following, we assume without loss of generality that $(\frac{n}{2}-1) \nmid p$.

When attempting to multiply the resulting ciphertext with $\mathsf{ct}_{\mathsf{CVS},i}$ as in Algorithm 3, an issue arises. Since $\mathsf{ct}_{\mathsf{CVS},i}$ encrypts multiples of $\phi(t)$ in its slots, multiplying it with the output $\mathsf{ct}_{\mathsf{bool},i}$ reduces the plaintext to 0. We can apply the previously described 'divide-by-p' technique here. Since both $\mathsf{ct}_{\mathsf{bool},i}$ and $\mathsf{ct}_{\mathsf{CVS},i}$ contain plaintexts that are multiples of p^{r-1}, we first divide them by p^{r-1} while setting the plaintext space to $\mathbb{Z}_p$.

As a result, let $\mathsf{ct}_{\mathsf{mult},i}$ denote the ciphertext obtained by this procedure:

- For $i \in S^c$: $\mathsf{ct}_{\mathsf{mult},i}$ is an encryption of $(\frac{n}{2}-1)(p-1)^2 \cdot J_{S^c} = (\frac{n}{2}-1)\cdot J_{S^c}$.
- For $i \in S$: $\mathsf{ct}_{\mathsf{mult},i}$ is an encryption of the zero vector.

We can further increase the forgery success probability by summing $\mathsf{ct}_{\mathsf{mult},i}$ over i as in Theorem 3.

However, if the p is small enough, an edge case arises when the number of indices $i \in S^c$ is a multiple of p. Since the summation is performed over $\mathbb{Z}_p$, this issue can be avoided when $p \neq 2$ by applying normalization after each addition. Concretely, one can apply a polynomial $q \in \mathbb{Z}_p[X]$ such that $q(0) = 0$ and $q(\frac{n}{2}-1) = q(2(\frac{n}{2}-1)) = \frac{n}{2}-1$ so that encryptions of $(\frac{n}{2}-1)J_{S^c}$ remains unchanged under repeated additions.

For the case $p = 2$, it is still possible to perform the 'divide-by-p^{r-1}' operation without reducing the plaintext modulus. As noted above, an encryption of $J_{S^c} \pmod 2$ can be regarded as an encryption of $2^{r-1} \cdot J_{S^c} \pmod{2^r}$. By dividing by 2^{r-1}, we obtain an encryption of $J_{S^c} \pmod{2^r}$. For typical parameters (e.g., $r \geq 16$ and $n \leq 128$), the summation of $J_{S^c} \pmod{2^r}$ does not exceed 2^r.

B.3 Attacks on $\mathsf{REP}^{\mathsf{msk}}$

The only difference from the main text lies in the evaluation of the pseudorandom characteristic function in Lemma 1. As in the main text, let $U_{t,d} = \{x : x^{\phi(t)/d} = 1\}$.

Lemma 3. *For $t = p^r$ and $d \mid (p-1)$, we have*

$$d \cdot \chi_{U_{t,d}}(x) = x^{\phi(t)} \cdot \sum_{i=0}^{d-1} \left(x^{\phi(t)/d}\right)^i.$$

Before the proof, note that $a \in \mathbb{Z}_{p^r}$ is a zero divisor if and only if $a \equiv 0 \pmod{p}$.

Proof. 1. Suppose x is a non-unit. Then the left-hand side (LHS) is zero. Since x is a multiple of p, the right-hand side (RHS) is also zero, as it becomes a multiple of $p^{\phi(t)} \equiv 0 \pmod{p^r}$ $(r \leq (p-1)p^{r-1} = \phi(t))$
2. Suppose $x \in U_{t,d}$. Since x is a unit, $x^{\phi(t)} = 1$. Also $x^{\phi(t)/d} = 1$ by definition of $U_{t,d}$. Thus both LHS and RHS equal d.
3. Suppose $x \in \mathbb{Z}_t^\times \setminus U_{t,d}$. Since x is a unit, $x^{\phi(t)} = 1$. Moreover $x^{\phi(t)/d} \neq 1$ by definition, while

$$(x^{\phi(t)/d})^d - 1 = (x^{\phi(t)/d} - 1)\left(\sum_{i=0}^{d-1}(x^{\phi(t)/d})^i\right) = 0.$$

Hence $\sum_{i=0}^{d-1}(x^{\phi(t)/d})^i = 0$, unless both $x^{\phi(t)/d} - 1$ and the sum are zero divisors. To rule out this second case, let $y = x^{\phi(t)/d}$. If $y - 1$ is a zero divisor, then $y = pm + 1$ for some integer m, which implies $\sum_{i=0}^{d-1} y^i \equiv d \not\equiv 0 \pmod{p}$. Thus it is not a zero divisor, a contradiction. Hence RHS $= 0$.

□

This completes the extension of Lemma 1 to the case $t = p^r$. As noted in the main text, one can further combine such characteristic functions to efficiently construct another characteristic function whose support size approximates a desired value. In the case $p = 2$, however, the lemma does not yield a meaningful result. We leave the study of efficient pseudorandom characteristic function evaluation for this case as future work.

We note that the cryptanalysis on PE in Section 5 naturally extends to the case $t = p^r$, and thus requires no further explanation.