IND-CPA-D of Relaxed Functional Bootstrapping: A New Attack, A General Fix, and A Stronger Model

Zeyu Liu
zeyu.liu@yale.edu
Yale University

Yunhao Wang
yunhao.wang@yale.edu
Yale University

Ben Fisch*
ben.fisch@yale.edu
Yale University

April 19, 2026

Abstract

Fully homomorphic encryption (FHE) is a powerful and widely used primitive in lots of real-world applications, with IND-CPA as its standard security guarantee. Recently, Li and Micciancio [Eurocrypt'21] introduced IND-CPA-D security, which strengthens the standard IND-CPA security by allowing the attacker to access a decryption oracle for honestly generated ciphertexts (generated via either an encryption oracle or an honest homomorphic circuit evaluation process).

Recently, Cheon et al. [CCS'24] and Checri et al. [Crypto'24] have shown that even exact FHE schemes like FHEW/TFHE/BGV/BFV may still not be IND-CPA-D secure, by exploiting the bootstrapping failure. However, such existing attacks can be mitigated by setting negligible bootstrapping failure probability.

On the other hand, Liu and Wang [Asiacrypt'24] proposed relaxed functional bootstrapping, which has orders of magnitude performance improvement and furthermore allows a free function evaluation during bootstrapping. These efficiency advantages make it a competitive choice in many applications but its "relaxed" nature also opens new directions of IND-CPA-D attack. In this work, we show that the underlying secret key could be recovered within 10 minutes against *all* existing relaxed functional bootstrapping constructions, and even within 1 minute for some of them. Moreover, our attack works even with a negligible bootstrapping failure probability, making it immune to existing mitigation methods.

Additionally, we propose a general fix that mitigates *all* the existing modulus-switching-error-based attacks, including ours, in the IND-CPA-D model. This is achieved by constructing a new modulus switching procedure with essentially no overhead. Lastly, we show that IND-CPA-D may not be sufficient for some applications, even in the passive adversary model. Thus, we extend this model to IND-CPA-D with randomness (IND-CPA-DR).

*Authors are listed alphabetically among students, followed by the advisor.

Contents

1 Introduction **3**
1.1 Technical Overview 4
1.2 Related Works 6

2 Preliminaries **6**
2.1 IND-CPA-D Security 7
2.2 BFV Homomorphic Encryption 8
2.3 Relaxed Functional Bootstrapping 9

3 Our Attacks **10**
3.1 Basic Attack 10
3.2 Extending the Attack for Less Structured Valid Input Set and Arbitrary Function 12
3.3 Extending the Attack for Arbitrary Valid Input Set 13
3.4 Extending the Attack for Valid Input Set Containing Ranges 15

4 Evaluation **16**
4.1 Parameter Analysis 16
4.2 Benchmarks on Actual Attacks 17

5 A General Fix with Zero Overhead **18**

6 IND-CPA-DR Model **20**

7 Concluding Remarks **21**

Acknowledgements **21**

References **22**

A Additional Discussion on Prior Works **26**

B Additional Preliminary **26**

C Additional Evaluation **26**
C.1 Larger N 26
C.2 How σ Affects 27

1 Introduction

Fully Homomorphic Encryption (FHE) allows one to compute over data encrypted under its owner's secret key without the knowledge of the key or interaction with the owner, thus resulting in a very strong primitive. FHE was first realized by Gentry in the groundbreaking work [30]. Since then, there have been lots of works trying to improve the efficiency of FHE [11, 10, 8, 24, 9, 31, 23, 19, 18].

A major application of FHE is to allow computation outsourcing, including but not limited to secure machine learning [44, 38, 45, 39, 22], secure data processing [29], private information retrieval [3, 58, 25], oblivious message retrieval [51, 52, 53, 46, 50], secret single leader election [7, 26, 62]. The main security guarantee of FHE is the so-called IND-CPA security: In particular, the server should not learn anything about the client's data encrypted under FHE, even with the public keys of the FHE scheme (including the evaluation key). On the contrary, the server can make an arbitrary number of encryptions and evaluate arbitrary circuits over the client's encrypted data without gaining any information about the encrypted data.

However, whether such a security guarantee is sufficient for real-world applications is unclear. Consider the scenario in which the server honestly evaluates the data, and the client leaks no information (including the exact data and the success of decryption) after receiving the evaluation results from the server. This aligns with the definition of IND-CPA, and the IND-CPA security guarantee suffices. Unfortunately, this may not be the case in practice.

Consider a client who uses FHE-encrypted lab results to ask a remote LLM (e.g., ChatGPT) to analyze them without revealing the data. The server evaluates the model homomorphically and returns an encrypted diagnosis. Suppose further that with this FHE scheme, the client has a small probability of obtaining an unexpected result, with probability correlated to the client's randomly sampled private key. Suppose that the model predicts that the client has the disease but the client is unlucky and obtains the opposite answer due to this unpredictability in FHE evaluation. The client then posts publicly on social media that they do not have the disease. Later, the original lab results leak. A third party, seeing both the lab result and the incorrect claim, can infer that the client obtained the wrong result and may gain information about their secret key. In the worst case, this could lead to full key recovery. This scenario highlights a gap in IND-CPA security: it does not account for what can be learned by observing both the encrypted inputs and decrypted outputs even if the evaluation process is honestly done, which is a much broader issue than this toy example.

To address this, Li and Micciancio [48] extends IND-CPA to IND-CPA-D security: The adversary is given a decryption oracle that can decrypt all honestly generated ciphertexts (either by the encryption process or after evaluating the circuit).[1] At a glance, it seems that such a decryption oracle is not helpful, since intuitively, the adversary should immediately know the decryption result for any honestly generated ciphertext. When first introduced, [48] applied this concept to analyze the bootstrapping security of CKKS, which is used for approximate homomorphic computations, and demonstrated that the decryption oracle indeed reveals the noise used in the encryption procedure. Using such revealed noise, the adversary is able to recover the secret key, thereby recovering all the encrypted data, including all future use. It was widely believed that the exact-FHE schemes like BGV/BFV [9, 8, 24] or FHEW/TFHE [23, 19] are IND-CPA-D secure naturally. However, two very recent works [13, 16] showed that this is not the case. [13, 16] both show an attack that iteratively adds one ciphertext and finds the threshold that the noise exceeds the bound, and [16] directly exploits the additional noise introduced by the bootstrapping procedure as well as its non-negligible failure probability. Both works assert that even for an exact FHE scheme, IND-CPA-D security can be a concern. However, it seems that all the existing attacks on exact FHE can be mitigated by either performing bootstrapping more often (to prevent the noise threshold being found as in [13, 16]) or making the bootstrapping failure probability negligible (e.g., 2^{-128} to prevent the bootstrapping-based attacks in [16]). [2]

[1] Note that this is different from IND-CCA since in IND-CPA-D, the decryption oracle only allows the decryption of honestly generated ciphertexts, while IND-CCA allows arbitrary decryption.

[2] In [16], the authors did not directly attack all the BGV/BFV bootstrapping constructions, but only targeted at CKKS-bootstrapping-based BGV/BFV bootstrapping [41]. However, intuitively, a similar attack applies to other regular BGV/BFV bootstrapping as well. Nevertheless, this is independent of the contribution in this paper. We discuss this in more detail in

An immediate question is whether these mitigations are enough for BFV/BGV. In this paper, we claim that this is not sufficient in some cases by analyzing a newly introduced concept, the so-called relaxed functional bootstrapping [54]. Relaxed functional bootstrapping is proposed to replace regular bootstrapping for better efficiency. As shown in [54], compared to the prior works on regular bootstrapping, relaxed functional bootstrapping (1) improves the efficiency (time per bit bootstrapped) by a factor of > 100x; (2) allows the bootstrapping to evaluate a function in a predetermined function family for free. Those two advantages make it a competitive building block in many FHE applications (e.g., [25, 37]).

However, despite its improved efficiency, it is unclear whether such a new bootstrapping paradigm shares the same security guarantee as regular bootstrapping. Prior work [16] showed that when regular bootstrapping is parametrized to guarantee negligible failure probability (e.g., 2^{-128}), the proposed attacks for IND-CPA-D do not work. However, in this work, we observe that this is not the case for relaxed functional bootstrapping, which only guarantees bootstrapping correctness on a subset of the input space.

Threat model. We follow the standard model used by all the prior works for IND-CPA-D security: the computationally bounded adversary is allowed to query an encryption and evaluation oracle to encrypt any message in the plaintext space and evaluate an arbitrary circuit over these queried ciphertexts (either output from the encryption or the evaluation procedure); The adversary is further allowed to ask for the decryption of any such well-formed ciphertexts. The goal of the adversary is to recover the secret key of the underlying encryption scheme. In the standard IND-CPA-D security definition, only a distinguishing attack is required but not a key recovery attack. In this paper, we mainly focus on the key recovery attack since it is strictly stronger.

Our contribution. The contribution of this work is as follows.

- We present IND-CPA-D attacks to BGV/BFV relaxed functional bootstrapping [54]. To be more specific, we show concrete, non-adaptive, and efficient key-recovery attacks for *all* the existing constructions of the relaxed functional bootstrapping procedure of BGV/BFV [54]. Our attacks cannot be mitigated by existing mitigating methods on previous attacks on bootstrapping of exact FHE schemes.

  We benchmark and provide extensive analysis on how the parameters are chosen for the attacks and the attacks on the existing relaxed functional bootstrapping library using SEAL library [60]. We believe such study can serve as detailed reference for future works on both attacks and defenses.

  Our evaluation demonstrates that the attacks can *fully* recover the keys in < 1 minute (2 bootstrapping calls) even if the bootstrapping procedure provides a $\geq 1 - 2^{128}$ correctness guarantee. Indeed, the runtime of our attack is *independent* of the failure probability.

- We show a general way to mitigate *any* modulus-switching-error-based attacks, including our attack and previous attacks on exact FHE bootstrappings [16]. In particular, we introduce a randomized modulus switching procedure that is provably secure under the IND-CPA-D model. Furthermore, this procedure has essentially no overhead.

- We discuss the insufficiency of IND-CPA-D model: it does not cover the case where the semi-honest server can observe the randomness used during the homomorphic evaluation procedure. To cover such cases, we modify the IND-CPA-D model to the IND-CPA-DR (IND-CPA-D with randomness) model, We show that in this stronger model, the mitigation above does not work.

1.1 Technical Overview

Our attack. The main reason that our attacks are not affected by the existing ways of mitigation is that we exploit a special property, the relaxed correctness of relaxed functional bootstrapping. More specifically, relaxed functional bootstrapping *only* guarantees correctness for a certain set of input, denoted as the *valid input set*, which is a subset of the whole plaintext space. If the input before bootstrapping is not a valid input, i.e., not within the valid input set (which might happen even if the circuit is honestly computed),

Appendix A.

the correctness of relaxed functional bootstrapping is not guaranteed. This is indeed why relaxed functional bootstrapping outperforms regular bootstrapping in terms of both speed and functionality. However, this leaves room for our attacks.

To launch the attack, we first encrypt some messages that are *not* in the valid input set. Then, we observe the decryption behavior after relaxed functional bootstrapping. According to the final output, we reproduce the modulus switching error (or drifting error [6]) induced inside the bootstrapping process, which can be used to generate the so-called *inequality hints*, allowing us to recover the secret key. The process is similar to the attacks on FHEW/TFHE in [16].

While this intuition is straightforward, one main challenge is that we need to explore the structure of the bootstrapping procedure. Therefore, we cannot simply view the procedure as a black-box and feed arbitrary values not in the valid input set, since the procedure treats invalid inputs differently for different valid input sets and different function families. It is especially the case that with an arbitrary function evaluation, the input not in the valid input set may be mapped to an arbitrary point in the plaintext space during output. Hence, for different constructions in [54] with respect to different sets and function families, our attacks leverage the underlying structure in distinct means. We discuss these details in Section 3.

The real-world scenarios to which our attack applies. One may wonder the real-world applications of our attack. In particular, we require the input before bootstrapping to be *not in* the vaid input set (VIS). Some may believe that this means that the server needs to be malicious (i.e., an active attacker). However, this is not correct.

As discussed in [54, Section 7.2], in some applications, it is hard to guarantee that the input is within the VIS, e.g., secure machine learning. In more detail, consider the concrete example above: one can view the large language model evaluation over the lab results above as a circuit C, which can be decomposed into $C := (C_1, \mathsf{Boot}, C_2)$, i.e., C consists of a deep circuit C_1 followed by a (relaxed functional) bootstrapping and then followed by another circuit C_2. Then, while the input of C_1 can be guaranteed to be in VIS, after C_1, a large language model that is itself not explainable, the input to Boot cannot be guarantee to be in VIS anymore. Furthermore, when using BGV/BFV for secure machine learning (e.g., [14]), the input field is used for fixed-point arithmetic, and the results are all approximate. It is even harder to guarantee that the input to Boot is within the VIS. Other examples may face a similar issue as discussed in [54].

Additionally, in some applications, the circuits are *private*. For example, for Private Set Intersection with computation [36], bootstrapping is necessary while the circuit needs to be private to protect the privacy of the server. In general, any 2PC using FHE [63] exhibits such a property. In these cases, it is also hard to guarantee that the input are in the VIS.

Even if the inputs are *not* in the VIS, the *closeness* property introduced in [54] still guarantees the correctness of the protocol in a restricted but meaningful way (see Section 2.3 for details), which can be leveraged by many applications as discussed in [54]. Our attacks are applicable to such applications.

Our mitigation. We observe that our attack and prior attacks over exact FHE bootstrapping all depend on the fact that the modulus switching error seen by the adversary calling the decryption oracle is (deterministically) *dependent* on the secret key.

This is true in practice since, to our knowledge, all the major libraries implementing FHE uses deterministic modulus switching [60, 5, 61]. However, this also leaves room for mitigation. In particular, we propose an alternative randomized modulus switching procedure. At a high level, before running the regular modulus switching procedure, the evaluation process first mask it with a fresh ciphertext encrypting zero. Then, we show that with such a mask, the adversary cannot learn any useful information about the secret key. Furthermore, such mitigation has essentially no overhead as a fresh encryption of zero is almost for free compared to the rest of the bootstrapping process.

The strengthened IND-CPA-DR model. As mentioned, IND-CPA-D model does not allow the adversary to obtain the randomness used during circuit evaluation. This is reasonable when the adversary is a third-party who can only observe the input and the output of the homomorphic evaluation, but not the entire process. In this case, our mitigation above provides a provably secure mitigation to all modulus-switching-based attacks. On the other hand, in practice, if the server itself is the passive attacker (e.g., the evaluation is enforced by SNARKs), the attacker can indeed see the intermediate randomness. To capture

this case, we strengthen IND-CPA-D to IND-CPA-DR (IND-CPA-D with randomness) and formally define this new model.

1.2 Related Works

IND-CPA-D security was first introduced by Li and Micciancio [48], and they provide key-recovery attacks against the approximate homomorphic computation, CKKS, by exploiting the error output of the decryption procedure. After this milestone work, there are some attempts to mitigate such attacks for CKKS [17, 49] with proven security for IND-CPA-D. One follow-up proposes an interesting attack against CKKS along this train of thought [32]: It exploits the vulnerability of heuristic noise analysis.

Recently, [13, 16] provides attacks against exact FHE schemes. They both propose the attack of repeated addition to test the threshold of the noise bound, which can be used to attack *any* Regev-like encryption system. [16] proposes key recovery attacks for FHEW/TFHE and CKKS (which can also be used for BGV/BFV bootstrapping)[3]. This attack in [16] exploits the fact that the failure probability of such bootstrapping procedures are usually a lot larger than 2^{-128}. Some of them can even have 2^{-30} or more failure probability. The attack efficiency is related to such failure probability instead of the standard computational security guarantee from RLWE. While both works propose very powerful and interesting attacks, they can be mitigated. The repeated addition attack can be mitigated by requiring bootstrapping after a certain amount of operations. The bootstrapping attack, on the other hand, can be mitigated by reducing the failure probability.

Another recent work [2] proposes an alternative security notion termed application-aware IND-CPA-D security that is slightly weaker. It essentially requires that the decryption oracle only allow encryptions for a certain set of plaintexts and circuits corresponding to the target applications. The formal definition requires that the parameters (e.g., ring dimension, ciphertext modulus, and error size) to be set according to the input and circuit desired by the underlying application. This mitigates some attacks like [32]. Unfortunately, it does not mitigate our attacks since the parameters we set are according to the application and circuit. We only require the parameters to be the parameters that are needed for a single relaxed functional bootstrapping, and thus for any applications that use relaxed functional bootstrapping as a building block, the parameters required by our attacks match the requirement in [2]. However, the intuition provided in [2] that FHE circuit designs should be aware of the underlying application is relevant. We discuss the potential mitigation of our attack in Section 7 in more detail.

The other two recent works [55, 12] define two alternative security models: both models allow the adversary to access a restricted decryption oracle, but also allow the adversary to tamper with the ciphertext in some malicious but limited way. While the constraints in these two models differ, these models allow for (restricted) active adversaries. Since our attack is passive, we believe our attack is stronger and can be applied to these models as well. Achieving even more efficient attacks under these models may be an interesting future direction.

One additional recent work [6] extensively studies the modulus switching error (drifting error). In their work, they introduced a way to estimate the drifting error (to ensure correctness with better parameters). They have also introduced a stronger IND-CPA-D model. We discuss these in more detail in Section 2.1 and Section 5.

2 Preliminaries

Notations. We introduce the notions needed to recall the relaxed functional bootstrapping procedure in [54].

Let $[u, v, r]$ denote the range from u to v with step value r (i.e., $[u, v, r] := \{u, u + r, u + 2r, \ldots, v\}$ for r dividing $u - v$). Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ denote the $2N$-th cyclotomic ring where N is a power-of-two, and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for some $q \in \mathbb{Z}$. The ring $\mathcal{R}_q$ is used for the BGV/BFV scheme. Let $[n]$ denote the set

[3]See Appendix A for a more detailed discussion on attacks over BGV/BFV bootstrapping.

$\{1, \ldots, n\}$, $[a, b)$ denote the set $\{a, a+1, \ldots, b-1\}$. Let $\vec{a}$ denote a vector and $\vec{a}[i]$ denote the i-th element of $\vec{a}$.

Unless otherwise specified, the key is taken implicitly and correctly for functions (e.g., $\mathsf{Dec}(\mathsf{ct})$ where ct is the BFV ciphertext and Dec is the decryption procedure of BFV).

All the divisions (i.e., a/b or $\frac{a}{b}$) are performed in real numbers and rounding (i.e., $\lceil \cdot \rfloor, \lceil \cdot \rceil, \lfloor \cdot \rfloor$) are from real numbers to integers.

2.1 IND-CPA-D Security

At a high level, the IND-CPA-D game introduced in [47] is equivalent to the IND-CPA game except that the adversary is additionally given a decryption oracle. However, unlike the IND-CCA game, the decryption oracle only takes *honestly generated* ciphertexts instead of arbitrary ones. In the context of FHE, these honestly generated ciphertexts are either freshly generated from Enc procedure or results of homomorphic evaluations based on those freshly generated ciphertexts. The formal definition is as follows.

Definition 2.1 (IND-CPA-D Security)**.** For security parameter λ and an FHE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, Π is IND-CPA-D secure if for any PPT adversary $\mathcal{A}$, it holds that, for any plaintexts m_0, m_1,

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen} \\ b \leftarrow_{\$} \{0,1\} \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Eval}}, \mathcal{O}_{\mathsf{Dec}}}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_b)) \end{array} \right] \leq 1/2 + \mathsf{negl}(\lambda)$$

The oracles $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Eval}}, \mathcal{O}_{\mathsf{Dec}}$ are defined in Algorithm 1.

Algorithm 1 Oracles for IND-CPA-D Security

procedure $\mathcal{O}_{\mathsf{Enc}}(m'_0, m'_1; \mathsf{pk}, b, i)$ ▷ For all these oracles, the adversary only controls inputs before ";".
  $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m'_b)$
  $S[i] \leftarrow (m'_0, m'_1, \mathsf{ct})$
  $i \leftarrow i + 1$
  Return ct, i
procedure $\mathcal{O}_{\mathsf{Eval}}(C, i_1, \ldots, i_k; S)$
  $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{pk}, C, (S[i_j].\mathsf{ct})_{j \in [k]})$
  $r_0 := C((S[i_j].m_0)_{j \in [k]})$
  $r_1 := C((S[i_j].m_1)_{j \in [k]})$
  $S[i] \leftarrow (r_0, r_1, \mathsf{ct})$
  $i \leftarrow i + 1$
  Return ct, i
procedure $\mathcal{O}_{\mathsf{Dec}}(j; \mathsf{sk}, S)$
  if $S[j].m_0 = S[j].m_1$ **then**
    Return $\mathsf{Dec}(\mathsf{sk}, S[j].\mathsf{ct})$
  else
    Return $\perp$

Strong IND-CPA-D model. In [6], the authors introduced the strong IND-CPA-D model, under which the user chooses the randomness in encryption. Specifically, the attacker has access to an additional encryption oracle which, on input message m and randomness r, adds (m, r, ct) to the state S (see Algorithm 1) and returns ct to the adversary, where ct is the ciphertext honestly generated via Enc with respect to m using randomnes r. Since this model is less relevant to our attack, we refer the readers to [6, Sec. 3] for a more formal definition.

2.2 BFV Homomorphic Encryption

The BFV homomorphic encryption scheme was first introduced in [8] using the standard LWE assumption, and later adapted to Ring LWE assumption by [24].

Given a (plaintext) polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$, the BFV scheme encrypts it into a ciphertext consisting of two polynomials $(a, b) \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$ for some larger ring with $q > t$. We refer t as the plaintext modulus, q as the ciphertext modulus, and N as the ring dimension. For the case of relaxed functional bootstrapping constructions in [54], it is always the case that N is a power of two and t satisfies that $t \equiv 1 \mod 2N$. Note that the relationship between t and N does not need to be satisfied in general (see [33, 34] for the general encoding). However, throughout our paper, we choose parameters in this way to maximize the concrete efficiency.

Plaintext encoding. In practice, instead of having a polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$ to encrypt, applications usually have a vector of messages $\vec{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$. To encrypt such input messages, BFV first encodes it by constructing another polynomial $y(X) = \sum_{i\in[N]} y_i X^{i-1}$ where $m_i = y(\zeta_j)$, at some points ζ_j for $j \in [N]$. Such encoding can be done using an Inverse Number Theoretic Transformation (INTT), which is a linear transformation represented as vector-matrix multiplication. Since it is irrelevant to our attacks, we omit the details.

Encryption and decryption. The BFV ciphertext encrypting $\vec{m}$ under $\mathsf{sk} \leftarrow \mathcal{D}$ has the following format: $\mathsf{ct} = (a, b) \in \mathcal{R}_q^2$, which satisfies $b - a \cdot \mathsf{sk} = \lfloor q/t \rfloor \cdot y + e$ where $\lfloor q/t \rfloor \cdot y \in \mathcal{R}_q$ and y is the polynomial encoded in the way above, and e is a small error term sampled from a Gaussian distribution over $\mathcal{R}_q$ with some constant standard deviation.

Symmetric key encryption is done by sampling a random a and constructing b using sk via the described way above. Public key encryption can also be achieved easily but it is not relevant to our paper so we omit the details (see, e.g., [8, 24, 40] for details).

Decryption calculates $y' \leftarrow \lceil (t/q) \cdot (b - a \cdot \mathsf{sk}) \rfloor \in \mathcal{R}_t$, i.e., $(b - a \cdot \mathsf{sk})$ is done over $\mathcal{R}_q$ and rounded to $\mathcal{R}_t$. One decodes it by applying a procedure to reverse the encoding process (which is also a linear transformation).

For simplicity, we assume $\mathsf{BFV.Dec}$ also embeds the decoding procedure and thus outputs plaintext $y' \in \mathbb{Z}_t^N$ in the decoded form directly (instead of a polynomial $y \in \mathcal{R}_t$). Similarly, we assume that $\mathsf{BFV.Enc}$ includes the encoding process, taking a plaintext $y' \in \mathbb{Z}_t^N$ as input instead of a polynomial. We want to point out that this *does not* affect our attack. In addition, define $\mathsf{PartialDec}(\mathsf{sk}, \mathsf{ct} = (a, b) \in \mathcal{R}_q^2) := b - a \cdot \mathsf{sk} \in \mathcal{R}_q$ (i.e., decryption without performing the rounding to $\mathcal{R}_t$).

BFV operations. BFV supports addition, multiplication, rotation, and polynomial function evaluation defined as follows. Note that we assume that all operations performed over the ciphertexts are evaluated homomorphically (i.e., one should compile those operations into an arithmetic circuit C and evaluate it via the Eval procedure). For readability, simple operations like "+" and "×" are directly used.

- (Addition) $\mathsf{BFV.Dec}(\mathsf{ct}_1 + \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) + \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Multiplication) $\mathsf{BFV.Dec}(\mathsf{ct}_1 \times \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) \times \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Rotation) $\mathsf{BFV.Dec}(\mathsf{BFV.Rotate}(\mathsf{ct}, j))[i] = \mathsf{BFV.Dec}(\mathsf{ct})[i + j \pmod N], \forall i, j \in [N]$
- (Polynomial evaluation) $\mathsf{BFV.Dec}(\mathsf{BFV.Eval}(\mathsf{ct}, f)) = f(\mathsf{BFV.Dec}(\mathsf{ct}))$, where $f : \mathbb{Z}_t \to \mathbb{Z}_t$ is a polynomial function. Note that this is implied by addition and multiplication.
- (Vector-matrix multiplication) $\mathsf{BFV.Dec}(\mathsf{ct} \times A) = \mathsf{BFV.Dec}(\mathsf{ct}) \times A$, where $A \in \mathbb{Z}_t^{N \times D}$, $0 < D \leq N$.

Given a BFV ciphertext ct and its corresponding secret key sk, we also assume that its noise budget can be derived via interface $\mathcal{B}(\mathsf{sk}, \mathsf{ct})$. Informally, the noise budget is the gap between the plaintext encrypted under ct and the noise inside ct, which serves as a buffer to tolerate the additional noises incurred by performing operations (e.g., multiplications) over the ciphertexts. We refer $\mathcal{B}$ with subscripts to hardcoded noise budget bounds. For example, $\mathcal{B}_{\mathsf{in}}$ represents the noise budget requirement of the input. This is used for relaxed functional bootstrapping definition in Definition 2.2.

All operations are performed over the entire plaintext vector $\vec{m} \in \mathbb{Z}_t^N$ (element-wise). Thus, all elements are evaluated via the same polynomial f by default. This is also known as the Single Instruction Multiple Data (SIMD) property. In this work, we assume all the operations work over $\vec{m}$ to be element-wise for all $\vec{m}[i], i \in [N]$.

Bootstrapping. First introduced by Gentry [30], bootstrapping is a way to reduce the noise of an FHE ciphertext. As aforementioned, each operation increases the noise in the (output) ciphertext. After a certain number of operations, the noise becomes too large and must be reduced to allow further operations. Such noise reduction is achieved by bootstrapping. There has been a lot of prior work on BFV/BGV bootstrapping [35, 15, 28, 27, 59, 41, 56, 57, 43]. However, these schemes are not relevant to our attacks since they work over the entire plaintext space.

2.3 Relaxed Functional Bootstrapping

In [54], the authors proposed "relaxed functional bootstrapping" of BFV/BGV.

The "relaxation". Roughly speaking, "relaxed" means that the correctness of bootstrapping only holds for the input within a subset of the entire plaintext space. This subset is denoted as the valid input set (VIS).

Functional bootstrapping. In addition to this, the authors proposed "functional bootstrapping": For $x \in \mathcal{X}$, bootstrapping outputs $f(x)$ for some $f : \mathcal{X} \to \mathbb{Z}_t$ instead of x itself. This function evaluation comes for free and thus provides additional efficiency to FHE. We will discuss what happens to $x \notin \mathcal{X}$ later.

With these, we recall the formal definition of functional bootstrapping as follows (mostly taken directly from [54]).

- $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}, \mathsf{pp}_{\mathsf{aux}}), \mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{Setup}(1^\lambda)$: takes a security parameter λ, and outputs a secret key sk, a bootstrapping key btk, and the public parameter pp including ring dimension N, plaintext space t, input noise budget $\mathcal{B}_{\mathsf{in}}$, output noise budget $\mathcal{B}_{\mathsf{out}}$, a function family $\mathcal{F}$, and auxiliary public parameters $\mathsf{pp}_{\mathsf{aux}}$.
- $\mathsf{ct}' \leftarrow \mathsf{Boot}(\mathsf{pp}, \mathsf{btk}, f, \mathsf{ct})$: takes the public parameter pp, a bootstrapping key btk, a function $f \in \mathcal{F}$, a ciphertext ct and outputs a ciphertext ct'.

Definition 2.2 (Correctness)**.** The bootstrapping procedure is correct, if it satisfies the following: let $(\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}, \mathsf{pp}_{\mathsf{aux}}), \mathsf{sk}, \mathsf{btk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for any function $f : \mathcal{X} \to \mathcal{Y} \in \mathcal{F}$ (where $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{Z}_t$ and $|\mathcal{X}| \geq |\mathcal{Y}| \geq 2$, $\mathcal{X}$ being the valid input set),[4] any honest input ciphertext ct with $\mathcal{B}(\mathsf{sk}, \mathsf{ct}) \geq \mathcal{B}_{\mathsf{in}}$, let $\mathsf{ct}' \leftarrow \mathsf{Boot}(\mathsf{pp}, \mathsf{btk}, f, \mathsf{ct})$, $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \in \mathbb{Z}_t^N$, $\vec{m}' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \in \mathbb{Z}_t^N$, it holds that:

$$\Pr\left[\begin{array}{ll} & \forall\, i \in [N],\ \text{if } \vec{m}[i] \in \mathcal{X}, f(\vec{m}[i]) = \vec{m}'[i] \\ \wedge & \mathcal{B}(\mathsf{sk}, \mathsf{ct}') \geq \mathcal{B}_{\mathsf{out}} > \mathcal{B}_{\mathsf{in}} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

When $x \notin \mathcal{X}$. In this case, the correctness definition *does not* specify what happens to the bootstrapped result. However, this is still a valid procedure and can be hard to prevent in some applications (see Section 1.1 for more details). This is why the construction of relaxed bootstrapping in [54] can be more efficient than regular bootstrapping procedures [35, 15, 28, 27, 59]. In some of the constructions in prior works, if $x \notin \mathcal{X}$, the output can be any $y \in \mathbb{Z}_t$ that is not related to x itself. In other constructions, the output can be x' (or $f(x')$ for the functional ones) where $x' \in \mathcal{X}$ is one of the ℓ-closest values of x (i.e., for all the valid input points, $|x' - x|$ is one of the ℓ-smallest), and [54] refers to this as "ℓ-closeness". We formally recall this definition in Appendix B. We discuss why this property does not affect our attack in Remark 3.3.

Constructions of relaxed functional bootstrapping. It is easy to see that a relaxed functional bootstrapping construction is parameterized by the VIS $\mathcal{X}$ and function f. In [54], the authors gave multiple relaxed functional bootstrapping constructions. For better readability, we use $\mathsf{RelaxBoot}_{\mathcal{X},f}$ to denote the relaxed functional bootstrapping with VIS $\mathcal{X}$ and function f.

A short summary for the main constructions in [54] is as follows.

[4] For $|\mathcal{Y}| = |[y]| = 1$, a trivial yet valid bootstrapping is to directly output a BFV ciphertext with all slots encrypting y.

- The first construction ([54, Algorithm 1]) has $\mathcal{X}_1 := [0, t-1-r, r]$ (where t is the plaintext modulus, and r is the modulus switching error to-be-discussed in Section 3.1) and $f(x) = x$ (i.e., f is the identity function I). It is denoted as $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$.
- The second construction ([54, Algorithm 2]) has $\mathcal{X}_2 := [u, v, r']$ (where $u < r \in \mathbb{Z}_t$ and $r' \in \mathbb{Z}_t$) and $f_2 : \mathcal{X}_2 \to \mathbb{Z}_t$ can be an arbitrary function. It is denoted as $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$.
- The third construction ([54, Algorithm 4]) has $\mathcal{X}_3 \subset \mathbb{Z}_t$ being an arbitrary set of points and $f_3 : \mathcal{X}_3 \to \mathbb{Z}_t$ can be an arbitrary function. It is denoted as $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$.
- The fourth construction [5] ([54, Algorithm 5-7]) has $\mathcal{X}_4 : \{[u_1, v_1], \ldots, [u_k, v_k]\}$ being a set of k different ranges that are well separated (i.e., $|v_j - u_i| \geq r$ for some r for any $i \neq j$) and $f_4 : \mathcal{X}_4 \to \mathbb{Z}_t$ can be an arbitrary function that maps one entire range to a single point. It is denoted as $\mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$.

3 Our Attacks

In this section, we show our attacks that break the IND-CPA-D security. In essence, we invoke the relaxed functional bootstrapping procedure and recover the secrets by calling the decryption oracle over the honestly generated ciphertexts (output from a relaxed functional bootstrapping).

We start by attacking the first relaxed functional bootstrapping construction $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$, and then extend our attack to other constructions $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}, \mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}, \mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$.

3.1 Basic Attack

We recall the construction of $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$ from [54, Algorithm 1] in sufficient detail to demonstrate our basic attack.

Recall $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$. Given a ciphertext $\mathsf{ct} := (a, b) \in \mathcal{R}_q^2$ (with ring dimension N) encrypting a plaintext vector $\vec{m} \in \mathbb{Z}_t^N$, the algorithm first modulus switches it to $\mathsf{ct}' := (a', b') \in \mathcal{R}_t^2$ (i.e., $(a', b') \leftarrow \lceil t(a, b)/q \rfloor$). Let $\mathcal{X}_1 := [0, t-1-r, r]$ be the valid plaintext space where r satisfies that the modulus switching error exceeds $r/2$ with probability $\mathsf{negl}(\lambda)$ for some security parameter λ. For simplicity, let r divide $t-1$).[6]

It holds that, if $\vec{m}[i] \in \mathcal{X}_1$, $(b' - a'\mathsf{sk})[i] = \lceil t(b - a\mathsf{sk})/q \rfloor \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2)$.[7] The bootstrapping procedure homomorphically evaluates $\mathsf{PartialDec}(\mathsf{ct}') = b' - a'\mathsf{sk}$ followed by a post-processing function f_{post} mapping $(x - r/2, x + r/2] \to x$ for all $x \in \mathcal{X}_1$. This guarantees correctness: if $\vec{m}[i] \in \mathcal{X}_1$, we have $f_{\mathsf{post}}((b' - a'\mathsf{sk})[i]) = \vec{m}[i]$.

Our attack for $x \notin \mathcal{X}_1$. This construction leaves room for attack in the IND-CPA-D model if $x \notin \mathcal{X}_1$ (see Section 1.1 for the applications where such a constraint is reasonable) . Again, consider $(a, b) \in \mathcal{R}_q^2$ encrypting $\vec{m} \in \mathbb{Z}_t^N$. After modulus switching and obtaining $(a', b') \in \mathcal{R}_t^2$, if $m[i] \notin \mathcal{X}_1$, let $\vec{m}' \leftarrow (b' - a'\mathsf{sk}) \in \mathbb{Z}_t^N$ being the decrypted plaintext vector. It still holds that $\vec{m}'[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2]$. Now, let $x_1, x_2 \in \mathcal{X}_1$ satisfying $x_1 \leq \vec{m}[i] \leq x_2$ and $x_2 = x_1 + r$. It is easy to see that:

$$\begin{aligned}(\vec{m}[i] - r/2, \vec{m}[i] + r/2] \subset & (x_1 - r/2, x_2 + r/2] \\ = & (x_1 - r/2, x_1 + r/2] \cup (x_2 - r/2, x_2 + r/2]\end{aligned}$$

Therefore, after bootstrapping, the result is either rounded to x_1 or x_2. Depending on whether the result is x_1 or x_2, we would deduce the modulus switching error. For example, let $\vec{m}[i] = x_1 + r/4 = x_2 - 3r/4$. If

[5] Note that in [54], there are three different algorithms for $\mathcal{X}_4, f_4$ defined in the paper. However, the differences in these constructions do not affect our attack, and thus we view them as a single construction.

[6] If not, simply round r up to the nearest value that divides $t-1$. Alternatively, reduce $t-1$ to $t-k$ for the smallest integer k such that r divides $t-k$. Note that this does not affect the choice of t but only affects the VIS size.

[7] Again, as mentioned in Section 2.3, we assume such decryption process performs decoding at the same time: i.e., $b' - a'\mathsf{sk}$ outputs a vector in $\mathbb{Z}_t^N$ instead of a polynomial in $\mathcal{R}_t$. This is for simplicity and is not relevant to our attack. For details on how it is realized for relaxed functional bootstrapping=, please see [54], discussion on "SlotToCoeff".

the bootstrapping result is x_1 instead of x_2, we know that the modulus switching error is $< r/4$. This gives us a "hint" to derive the secret key elements.

Formalize the hint. We now specify what we mean by "hint". The modulus switching error comes from the rounding procedure. For easier demonstration, consider a ciphertext $a(X) := \sum_i \vec{a}[i]X^i, b(X) := \sum_i \vec{b}[i]X^i$ with the underlying secret key $\mathsf{sk} := \sum_i \vec{s}[i]X^i$, we focus on the first plaintext of the decrypted plaintext vector, i.e., $(b - a\mathsf{sk})[0] = \vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle \in \mathbb{Z}_q$, where $\vec{s}'[0] = \vec{s}[0]$ and $\vec{s}'[i] = -\vec{s}[N-i]$ for $i \in \{1, \ldots, N-1\}$.

It satisfies that that $\vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle = \lceil q \cdot \vec{m}[0]/t \rfloor + e \in \mathbb{Z}_q$, where e is some error and $\vec{m}[0] \notin \mathcal{X}$ is some message not in the valid input set $\mathcal{X}$. By calculating the modulus switching before rounding, we have $t \cdot (b[0] - \langle \vec{a}, \vec{s} \rangle)/q = \vec{m}[0] + e' \in \mathbb{R}$, for some small $e' < 1 \in \mathbb{R}$. To finish the modulus switching procedure, we round the ciphertexts to their nearest integers: $(\vec{b}'[0], \vec{a}') \in \mathbb{Z}_t^{N+1}$ instead of $\mathbb{R}^{N+1}$. Therefore, $\vec{m}'[0] := \vec{b}'[0] - \langle \vec{a}', \vec{s}' \rangle \in \{x_1, x_2\}$ after bootstrapping, where $x_1 < \vec{m}[0] < x_2$, $x_2 = x_1 + r$, $x_1, x_2 \in \mathcal{X}$. In other words, x_1, x_2 are two elements in the VIS $\mathcal{X}$ closest to $\vec{m}[0]$.

W.l.o.g., we choose $\vec{m}[0]$ such that $|x_1 - \vec{m}[0]| < |x_2 - \vec{m}[0]|$ and consider the case of $\vec{m}'[0] = x_2$ (i.e., $\vec{m}'[0]$ is "rounded" to the farther point instead of the closer point in the relaxed functional bootstrapping procedure). This gives us:

$$\underbrace{(\vec{b}'[0] - \langle \vec{a}', \vec{s}' \rangle)}_{\in \mathbb{Z}_t, \text{ lifted to } \mathbb{R}} - \underbrace{t(\vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle)/q}_{\in \mathbb{R}} \geq T - e'$$

where $T := r/2 - (\vec{m}[0] - x_1) > 0$ is the *threshold*, and e' is the small decimal error when we perform the modulus switching without rounding. Recall that $\vec{m}[0] - x_1 > 0$ and thus $T \in [r/2]$.[8] Let $\vec{c} := -t\vec{a}/q + \vec{a}' \in [-0.5, 0.5]^N$ be a vector, we formalize the *hint* as: $\langle \vec{c}, \vec{s}' \rangle - e' > T$.[9] Such a linear relationship is called "inequality hint" introduced in [21], used by [16] for attacks on exact FHE schemes. We formally recall its definition below and then discuss why this hint is useful in our case.

Definition 3.1. For $\sigma > 0$ and a threshold $T > 0$, a (σ, T)-approximate-inequality hint on the secret vector $\vec{s} \in \{0, 1, -1\}^N$ is a vector $\vec{c} \in \mathbb{Z}_q^N$ such that $\langle \vec{c}, \vec{s} \rangle - e > T$, where e is a Gaussian error with standard deviation σ.

Observe that the probability of $\langle \vec{c}, \vec{s}' \rangle - e' > T$ (randomness taken over both the generation of $\vec{c}$ and $\vec{s}'$) is greater if $\vec{c}$ and $\vec{s}$ have the same signs. Moreover, if the inequality holds, the probability of $\vec{c}$ and $\vec{s}'$ sharing the same sign increases as T and $|\vec{c}[i]|$ increase. A formal argument regarding the binary secrets is presented in [16, Lemma 4.2]. We extend it to ternary secrets as follows.

Lemma 3.1. Let $\sigma > 0, T > 0$ and a secret vector $\vec{s} \in \{-1, 0, 1\}^N$. Let $\vec{c} \leftarrow_\$ [-0.5, 0.5]^N$ and $e \leftarrow \chi_\sigma$. Let $i \in [N]$ and $Y_i := \langle \vec{c}, \vec{s} \rangle - \vec{c}[i]\vec{s}[i]$. For all $x_i \in [-0.5, 0.5]$, the conditional probability density function f of $\vec{c}[i]$ conditioned on the event that $\langle \vec{c}, \vec{s} \rangle - e > T$ satisfies that:

$$f(x_i) = \begin{cases} \frac{\Pr[x_i + Y_i + e > T]}{\Pr[\vec{c}[i] + Y_i + e > T]} & \text{if } \vec{s}[i] = 1 \\ 1 & \text{if } \vec{s}[i] = 0 \\ \frac{\Pr[-x_i + Y_i + e > T]}{\Pr[-\vec{c}[i] + Y_i + e > T]} & \text{if } \vec{s}[i] = -1 \end{cases}$$

Proof. The proof is straightforward, extended from the case of a binary secret in [16]. By Bayes' law, $f(x_i) = \frac{\Pr[(\langle \vec{c}, \vec{s} \rangle + e > T) | \vec{c}[i] = x_i]}{\Pr[\langle \vec{c}, \vec{s} \rangle + e > T]}$. By definition of Y_i, $\vec{c}[i]\vec{s}[i] + Y_i = \langle \vec{c}, \vec{s} \rangle$ and by substituting $\vec{s}[i]$ with 1, 0, or -1, we get the lemma statement. □

Intuitively, the lemma says that the distribution of the coefficients in $\vec{c}[i]$ is very different depending on the value of $\vec{s}[i]$. The distribution would be uniform for $\vec{s}[i] = 0$, and has an increasing density function for $\vec{s}[i] = 1$ (for sufficiently large T) yet a decreasing density function for $\vec{s}[i] = -1$.

[8] Otherwise, since the modulus switching error is bounded by $r/2$ except with negligible probability, our attack does not work in time polynomial in λ.

[9] To be more detailed, we have $\langle \vec{c}, \vec{s}' \rangle - e' \geq T' + t(\vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle)/q$. However, $t(\vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle)/q$ is a known number independent of the secret key. For simplicity and w.l.o.g., throughout the paper, we use $T := T' + t(\vec{b}[0] - \langle \vec{a}, \vec{s}' \rangle)/q$ to absorb it.

We additionally conduct experiments illustrating this trend in Section 4.1, which derives the parameters we need for our attack.

Our attack. With these, we are now ready to discuss our attack in detail. We set $\vec{m}[i] = x_1 + r/2 - T < x_1 + r/2$. After bootstrapping, we collect all the samples where the decryption result of $\vec{m}'[i] = x_2 = x_1 + r$ instead of x_1. Guided by those samples, we use the original ciphertext to perform modulus switching and generate the inequality hints as described above. With enough samples, we calculate the average coefficients of $\vec{c}[i]$. If it is greater than some value α_1, we set $\vec{s}[i] = 1$, and if it is smaller than some $\alpha_2 < \alpha_1$, we set $\vec{s}[i] = -1$. Otherwise, $\vec{s}[i] = 0$. The values T, α_1, α_2 are chosen according to the analysis presented in Section 4.1. For simplicity, we choose $x_1 = 0, x_2 = r$. We formalize our attack in Algorithm 2.

Algorithm 2 Key recovery attack on basic relaxed bootstrapping

procedure $\mathsf{GenHint}(\mathsf{ct} = (a(X) = \sum_{i\in[N]} \vec{a}[i]X^{i-1}, b(X) = \sum_{i\in[N]} \vec{b}[i]X^{i-1}), j)$
  $\vec{c} \leftarrow (0, \ldots, 0) \in \mathbb{Z}_q^{N+1}$
  for $i \in [0, N-1]$ **do**
    $\vec{c}[i] \leftarrow \vec{a}[j - i + 1]$ if $i \le j$
    $\vec{c}[i] \leftarrow -\vec{a}[j - i + 1]$ if $i > j$
  ▷ Up to here, same as SampleExtract in [19].
  $\vec{c}_j \leftarrow t\vec{c}/q - \lceil t\vec{c}/q \rfloor \in [-0.5, 0.5]^N$
  Return $\vec{c}_j$
procedure $\mathsf{OurAttack}(C, T, \alpha_1, \alpha_2)$
  for $i \in [C]$ **do**
    $\vec{m} \leftarrow [r/2 - T, r/2 - T, ..., r/2 - T]$ ▷ A vector of length N with each element being $r/2 - T$
    $\mathsf{ct}_i \leftarrow \mathsf{BFV.Enc}(\vec{m})$
    $\mathsf{ct}'_i \leftarrow \mathsf{BFV.RelaxBoot}_{\mathcal{X}_1,I}(\mathsf{ct}_i)$
    $\vec{m}' \leftarrow \mathsf{BFV.Dec}(\mathsf{ct}'_i)$
    for $j \in [N]$ **do**
      if $\vec{m}'[j] = r$ **then** ▷ Meaning that the modulus switching error is at least T
        $\vec{c}_{i,j} \leftarrow \mathsf{GenHint}(\mathsf{ct}_i, j)$
  Compute $h_k \leftarrow \mathsf{Avg}(c_{i,j}[k])$ for $k \in [N]$ and i, j above
  If $h_k > \alpha_1$, set $\vec{s}[k] = 1$; if $h_k < \alpha_2$, $\vec{s}[k] = -1$; o.w., $\vec{s}[k] = 0$.
  Return $\vec{s}$

3.2 Extending the Attack for Less Structured Valid Input Set and Arbitrary Function

While the attack above effectively recovers the secrets, it only works for $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$ since it explores the special structure of the VIS $\mathcal{X}_1$ with step size r and the fact that the function evaluated is an identity function.

In this section, we show how to extend our attack to other constructions. In particular, we show (1) what happens when the valid input set $\mathcal{X}$ is less structured than $\mathcal{X}_1 := [0, t - r - 1, r]$, and (2) what happens when $f \neq I$ (i.e., not an identity function).

Recall $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$. We start by recalling the second construction $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$ ([54, Algorithm 2]). Recall that $\mathcal{X}_2 := [u, v, r']$, where $u < v \in \mathbb{Z}_p$ and $r' \in \mathbb{Z}_p$. The algorithm first computes a preprocessing function $f_{\mathsf{pre}}(x) := (x - u) \cdot r/r'$ over the input ciphertext, which maps every point in $\mathcal{X}_2$ to $\mathcal{X}_1$. Similar to $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$, the algorithm modulus switches the preprocessed ciphertext to $\mathsf{ct}' := (a', b')$. Then, the algorithm homomorphically partial-decrypts the preprocessed ciphertext ct' by evaluating $b' - a'\mathsf{sk}$ and computes a new post-processing function $f_{\mathsf{post},2}(x) := f_2(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$, which essentially achieves three tasks: 1) maps the modulus switched result to $\mathcal{X}_1$ (i.e., the "$r \cdot \lceil x/r \rfloor$" part), 2) maps $\mathcal{X}_1$ back to $\mathcal{X}_2$ (i.e., the "f_{pre}^{-1}" part) and 3) evaluates the intended arbitrary function f_2.

Extended attack. In the original attack, we leverage the fact that for any $m \notin \mathcal{X}$, the modulus switching process "rounds" it to one of the closest two points $x_1, x_2 \in \mathcal{X}$ (w.l.o.g, consider x_1 to be closer to m than x_2). However, in a more general case, m can be mapped to an arbitrary point $\in \mathbb{Z}_t$ that has nothing to do with x_1, x_2.

Thus, instead of viewing the bootstrapping construction as a black box, we need to exploit the algebraic structure inside. Note that $f_{\mathsf{pre}}(x) := (x-u) \cdot r/r'$, which has m mapped to $m'' \leftarrow f_{\mathsf{pre}}(m)$ before modulus switching, for some $m \notin \mathcal{X}$. The modulus switching procedure would map m'' to $x_1'', x_2'' \in \mathcal{X}_1$ being one of the two closest points to m''. Lastly, f_{post} maps the result to $f_2(f_{\mathsf{pre}}^{-1}(x_1''))$ or $f_2(f_{\mathsf{pre}}^{-1}(x_2''))$.

We first assume that $f_2(f_{\mathsf{pre}}^{-1}(0)) \neq f_2(f_{\mathsf{pre}}^{-1}(r))$. Again, we could simply set $x_1'' = 0, x_2'' = r$. We choose m such that $m'' \leftarrow f_{\mathsf{pre}}(m) = r/2 - T$, instead of $m \leftarrow r/2 - T$ for some predetermined threshold T as in Algorithm 2. Similar to the previous attack, we claim that the modulus switching error is larger than T if m is mapped to $m' = f_2(f_{\mathsf{pre}}^{-1}(r))$ instead of $f_2(f_{\mathsf{pre}}^{-1}(0))$, and collect the inequality hints as before. This completes our attack to $\mathsf{RelaxBoot}_{\mathcal{X}_2, f_2}$.

Notice that we allow f_2 to be an arbitrary function and thus it is possible to have $f_2(f_{\mathsf{pre}}^{-1}(0)) = f_2(f_{\mathsf{pre}}^{-1}(r))$. In this scenario, we could just choose another $x_1'', x_2'' = x_1'' + r$ such that $f_2(f_{\mathsf{pre}}^{-1}(x_1'')) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2''))$. Below we prove that such a pair (x_1'', x_2'') always exists.

Claim 3.2. Given the algorithm $\mathsf{RelaxBoot}_{\mathcal{X}_2, f_2}$ ([54, Algorithm 2]), and for any $f_2 : \mathcal{X}_2 \to \mathbb{Z}_t$, there always exists $x_1, x_2 \in \mathcal{X}_1$ such that: $f_2(f_{\mathsf{pre}}^{-1}(x_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2))$, and $x_2 = x_1 + r$. Otherwise, $\mathsf{RelaxBoot}_{\mathcal{X}_2, f_2}$ becomes trivial (i.e., there exists another algorithm that accomplishes relaxed functional bootstrapping (Definition 2.2) by directly outputting a freshly encrypted ciphertext).

Proof. To start with, it is easy to observe that since f_{pre} is a bijective mapping, if there does not exist $x_1, x_2 \in \mathcal{X}_1$ such that $f_2(f_{\mathsf{pre}}^{-1}(x_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2))$, there does not exist $x_1', x_2' \in \mathcal{X}_2$ such that $f_2(x_1') \neq f_2(x_2')$, which means that $f_2(x)$ is essentially just a constant function.

Then, we show by contrapositive that if there exists any pair $y_1, y_2 \in \mathcal{X}_1$ such that $f_2(f_{\mathsf{pre}}^{-1}(y_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(y_2))$, there exists $x_1, x_2 \in \mathcal{X}_1$ such that $f_2(f_{\mathsf{pre}}^{-1}(x_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2))$ where $x_2 = x_1 + r$. Recall that $\mathcal{X}_1 := [0, t-1-r, r]$. If for all $x_1 \in \mathcal{X}_1$, it satisfies that $f_2(f_{\mathsf{pre}}^{-1}(x_1 + r)) = f_2(f_{\mathsf{pre}}^{-1}(x_1))$, we trivially have $f_2(f_{\mathsf{pre}}^{-1}(x_1 + k \cdot r)) = f_2(f_{\mathsf{pre}}^{-1}(x_1))$ for any $k \in \mathbb{Z}$, and there would not exists any pair y_1, y_2 satisfying $f_2(f_{\mathsf{pre}}^{-1}(y_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(y_2))$.

With these, we conclude that if there does not exist $x_1, x_2 \in \mathcal{X}_1$ such that (1) $f_2(f_{\mathsf{pre}}^{-1}(x_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2))$ and (2) $x_2 = x_1 + r$, $f_2(x) = c$ must be a constant function for some constant c. For such f_2, a boostrapping algorithm could be trivially realized by outputting $\mathsf{Enc}(c)$, which has a noise budget larger than the input noise budget and also satisfies the correctness requirement of bootstrapping (Definition 2.2). □

With this claim, we are guaranteed to find $x_1'', x_2'' := x_1'' + r \in \mathcal{X}_1$, satisfying $f_2(f_{\mathsf{pre}}^{-1}(x_1'')) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2''))$, in bounded time (at most $O(|\mathcal{X}_1| + |f_2|)$ time). In the attack, instead of choosing m such that $m'' \leftarrow f_{\mathsf{pre}}(m) = r/2 - t$, we choose m such that $m'' \leftarrow f_{\mathsf{pre}}(m) = x_1'' + r/2 - t$. We formalize the general attack in Algorithm 3 (differences from Algorithm 2 are marked in blue).

Remark 3.3. For $\mathsf{RelaxBoot}_{\mathcal{X}_1, I}$, closeness (Definition B.1) naturally holds for $\ell = 2$, while for $\mathsf{RelaxBoot}_{\mathcal{X}_2, f_2}$ it is not the case (see [54, page 14] for details). However, one can see that it does not affect the efficiency of our attack at all. The only thing that changes is how we find the points to-be-bootstrapped. The cost of finding such points is negligible compared to the bootstrapping call as it is done in plaintext. Similarly, for the other relaxed functional bootstrapping constructions, some of them have closeness and some do not. Whether a construction has such a property depends on its structure, and the structure can affect how our attack is constructed. However, the closeness property itself does not affect our attack.

3.3 Extending the Attack for Arbitrary Valid Input Set

In [54], there is an even more general kind of VIS $\mathcal{X}_3 \subset \mathbb{Z}_t$ such that $|\mathcal{X}_3| \leq \frac{t-1}{r}$. This is achieved by $\mathsf{RelaxBoot}_{\mathcal{X}_3, f_3}$ ([54, Algorithm 3]) supporting arbitrary function $f_3 : \mathcal{X}_3 \to \mathbb{Z}_t$. The construction becomes more involved and in this section, we discuss how to modify our attack for $\mathsf{RelaxBoot}_{\mathcal{X}_3, f_3}$.

Algorithm 3 Key recovery attack on $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$

procedure $\mathsf{OurAttack}(C, T, \alpha_1, \alpha_2)$
  for $i \in [C]$ **do**
    $\vec{m} \leftarrow [m, ..., m]$ such that $f_{\mathsf{pre}}(m) = x_1 + r/2 - T$ ▷ A vector of length N with each element being m', for x_1 define in Section 3.2
    $\mathsf{ct}_i \leftarrow \mathsf{BFV.Enc}(\vec{m})$
    $\mathsf{ct}'_i \leftarrow \mathsf{BFV.RelaxBoot}_{\mathcal{X}_2,f_2}(\mathsf{ct}_i)$
    $\vec{m}' \leftarrow \mathsf{BFV.Dec}(\mathsf{ct}'_i)$
    for $j \in [N]$ **do**
      if $\vec{m}'[j] = f_2(f_{\mathsf{pre},2}(x_1 + r))$ **then** ▷ Meaning that the modulus switching error is at least T
        $\vec{c}_{i,j} \leftarrow \mathsf{GenHint}(\mathsf{ct}_i, j)$
  Compute $h_k \leftarrow \mathsf{Avg}(c_{i,j}[k])$ for $k \in [N]$ and i, j above
  If $h_k > \alpha_1$, set $\vec{s}[k] = 1$; if $h_k < \alpha_2$, $\vec{s}[k] = -1$; o.w., $\vec{s}[k] = 0$.
  Return $\vec{s}$

Recall $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$ for arbitrary $\mathcal{X}_3$. We briefly recall some details on $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$: First, a function $f_{\mathsf{pre},3}$ maps the message $m \in \mathcal{X}_3$ (encrypted in ciphertext (a, b)) $\mathcal{X}'_3$ where $\forall x \neq y \in \mathcal{X}'_3$, it holds that $|x - y| \geq r$.[10] The rest is exactly the same as $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$ by modulus switching ct to $(a', b') \in \mathcal{R}_t^2$, computing $c \leftarrow b' - a's$, and computing $f_{\mathsf{post},3}(c) := f_3(f_{\mathsf{pre},3}^{-1}(r \cdot \lceil c/r \rfloor))$.

When extended attack fails. If we could still find $m'' \in \mathcal{X}_3$ and $x_1 \in \mathcal{X}'_3$ such that $f_{\mathsf{pre},3}(m) = x_1 - r/2 - T$ for some threshold $T \in [r/2]$, our attack still works essentially in the same way as it did before. However, it is possible that such m does not even exist. A trivial example is that let $\mathcal{X}_3 = \{1, 2\}$ and $\mathcal{X}'_3 = \{0, r\}$. Then, $f_{\mathsf{pre},3}$ is a function such that $f_{\mathsf{pre},3}(1) = 0, f_{\mathsf{pre},3}(2) = r$ and $f_{\mathsf{pre},3}(x) \notin [-r, r]$ for all $x \in \mathbb{Z}_t \setminus \{1, 2\}$. In this case, there does not exist $m \notin \mathcal{X}_3$ such that $f_{\mathsf{pre},3}(m) = x_1 - r/2 - T$ for any $x_1 \in \mathcal{X}'_3$. Therefore, we employ a more general attack strategy for $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$ as follows.

General attack. Let $S_1 := [m'' - r/2, m'' + T]$ and $S_2 := (m'' + T, m'' + r/2)$. A more general attack is to find some m such that $m'' \leftarrow f_{\mathsf{pre},3}(m)$ satisfies that: $f_{\mathsf{post},3}(S_1) \cap f_{\mathsf{post},3}(S_2) = \emptyset$, where $f(S)$ for function f is the output set $\{y | y = f(s), \forall s \in S\}$. The rest remains almost the same as the previous attack: if the resulting plaintext message is in $f_{\mathsf{post},3}(S_2)$, we obtain the inequality hint.

It is straightforward to see that the previous attack is a special case of this general attack. If $f_2(f_{\mathsf{pre}}^{-1}(x_1)) \neq f_2(f_{\mathsf{pre}}^{-1}(x_2))$, $x_2 = x_1 + r$, it holds that $f_{\mathsf{post},2}(S'_1) \cap f_{\mathsf{post},2}(S'_2) = \emptyset$ where $S'_1 := [x_1 - r/2, x_1 + r/2)$ and $S'_2 := [x_2 - r/2, x_2 + r/2)$. Furthermore, let $S_1 := [m'' - r/2, m'' + T)$ and $S_2 := [m'' + T, m'' + r/2)$ where $m'' = f_{\mathsf{pre},2}(m) = x_1 + r/2 - T$. It is easy to see that $S_1 \subset S'_1, S_2 \subset S'_2$.

Efficiency-wise, the general attack is captured by $\mathsf{poly}(t)$, where the plaintext space is t being $\mathsf{poly}(\lambda)$ for security parameter λ. Furthermore, the brute-force of finding m takes very little time compared to bootstrapping, since everything is done in plaintext. According to our benchmark in Section 4, the worst case takes < 6 seconds while bootstrapping takes tens to hundreds of seconds. For most functions (e.g., the ones in [54]), it takes even much less than this worst case. We provide more details in Section 4.2.

When the general attack fails. There are some extreme cases where even the general attack fails. Specifically, if there is no value of m such that $f_{\mathsf{post},3}(S_1) \cap f_{\mathsf{post},3}(S_2) = \emptyset$. For example, consider $\mathcal{X}_3 = \{0, 1\}$ and $\mathcal{X}'_3 = \{0, r\}$. If we have $f_{\mathsf{pre},3}(x) := x^{t-1} \cdot r$, $f_{\mathsf{pre},3}(x) = r$ for all $x \neq 0 \in \mathbb{Z}_t$ and our general attack cannot find an m satisfying the aforementioned criteria. Fortunately, such a construction might not make sense heuristically. This requires $f_{\mathsf{pre},3}$ to map all points in $\mathbb{Z}_t$ to some fixed set $\mathcal{X}'_3$, which results in a degree $O(t)$ function. This to some extent destroys the purpose of the "bootstrapping", as it requires the ciphertext before bootstrapping to have a noise budget for $\log(t)$ levels of multiplication, which is almost the same as the depth of a regular bootstrapping circuit. In our above toy example, the function has degree $t - 1$ and it can simply be replaced by a degree-one function $f_{\mathsf{pre},3}(x) = rx$ with the same correctness guarantee and

[10] Note that it does not need to be r or a multiple of it. Being $\geq r$ allows us to separate the two inputs after modulus switching.

much better performance.

In practice, any construction with a reasonably efficient $f_{\mathsf{pre},3}$ (i.e., with the low degree to guarantee efficiency) could be attacked by our general attack approach. This is indeed the case for all the constructions benchmarked in [54]. We formalize this more general attack in Algorithm 4 (differences from Algorithm 3 are marked in blue).

Algorithm 4 Key recovery attack on $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$

procedure $\mathsf{OurAttack}(C, T, \alpha_1, \alpha_2)$
  Find m such that $f_{\mathsf{post},3}(S_1) \cap f_{\mathsf{post},3}(S_2) = \emptyset$. ▷ In the way described in Algorithm 4
  for $i \in [C]$ **do**
    $\vec{m} \leftarrow [m, ..., m]$
    $\mathsf{ct}_i \leftarrow \mathsf{BFV.Enc}(\vec{m})$
    $\mathsf{ct}'_i \leftarrow \mathsf{BFV.RelaxBoot}_{\mathcal{X}_3,f_3}(\mathsf{ct}_i)$
    $\vec{m}' \leftarrow \mathsf{BFV.Dec}(\mathsf{ct}'_i)$
    for $j \in [N]$ **do**
      if $\vec{m}'[j] \in f_{\mathsf{post},3}(S_2)$ **then** ▷ Meaning that the modulus switching error is at least T
        $\vec{c}_{i,j} \leftarrow \mathsf{GenHint}(\mathsf{ct}_i, j)$
  Compute $h_k \leftarrow \mathsf{Avg}(c_{i,j}[k])$ for $k \in [N]$ and i, j above
  If $h_k > \alpha_1$, set $\vec{s}[k] = 1$; if $h_k < \alpha_2$, $\vec{s}[k] = -1$; o.w., $\vec{s}[k] = 0$.
  Return $\vec{s}$

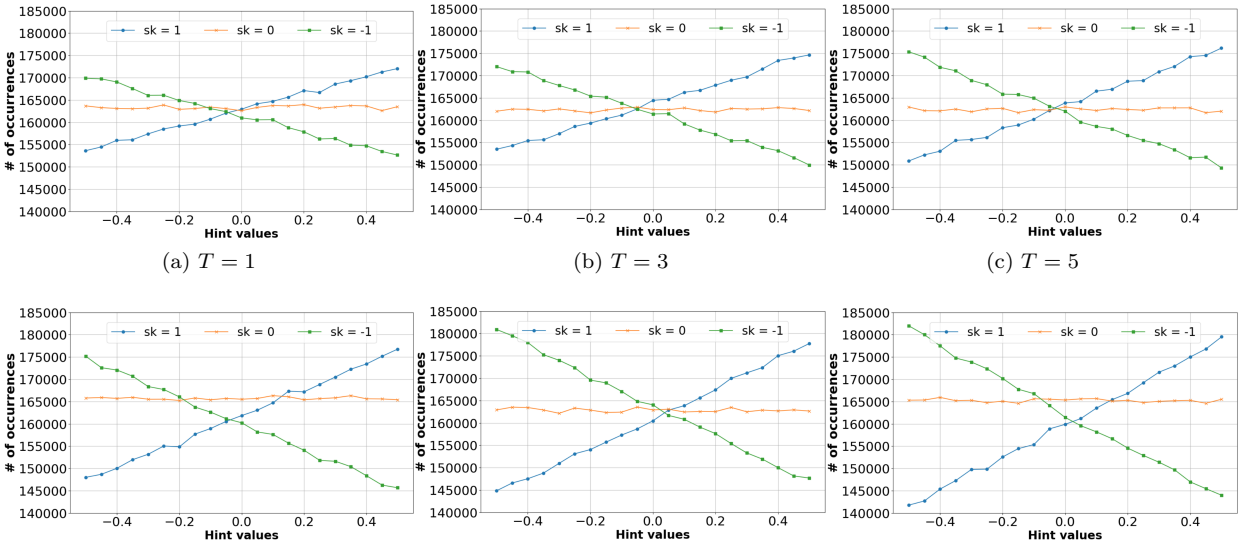
3.4 Extending the Attack for Valid Input Set Containing Ranges

Algorithm 5 Key recovery attack on $\mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$

procedure $\mathsf{OurAttack}(C, T, \alpha_1, \alpha_2)$
  Find m such that $f_{\mathsf{pre},4}(S_1) \cap f_{\mathsf{pre},4}(S_2) = \phi$. ▷ In the way described in Sections 3.3 and 3.4
  for $i \in [C]$ **do**
    $\vec{m} \leftarrow [m, ..., m]$
    $\mathsf{ct}_i \leftarrow \mathsf{BFV.Enc}(\vec{m})$
    $\mathsf{ct}'_i \leftarrow \mathsf{BFV.RelaxBoot}_{\mathcal{X}_4,f_4}(\mathsf{ct}_i)$
    $\vec{m}' \leftarrow \mathsf{BFV.Dec}(\mathsf{ct}'_i)$
    for $j \in [N]$ **do**
      if $\vec{m}'[j] \in f_{\mathsf{post},4}(S_2)$ **then** ▷ Meaning that the modulus switching error is at least T
        $\vec{c}_{i,j} \leftarrow \mathsf{GenHint}(\mathsf{ct}_i, j)$
  Compute $h_k \leftarrow \mathsf{Avg}(c_{i,j}[k])$ for $k \in [N]$ and i, j above
  If $h_k > \alpha_1$, set $\vec{s}[k] = 1$; if $h_k < \alpha_2$, $\vec{s}[k] = -1$; o.w., $\vec{s}[k] = 0$.
  Return $\vec{s}$

In [54], there are three additional constructions where the VIS $\mathcal{X}$ is a collection of k well-separated ranges instead of individual points in the previous sections. More formally, the input sets of these constructions are all of form: $\mathcal{X}_4 := \{[u_1, v_1], \ldots, [u_k, v_k]\}$ where $|u_i - v_j| \geq r$ for any $i \neq j$. In this case, $|\mathcal{X}_4|$ can be much larger than $\frac{t-1}{r}$, the cap in the constructions discussed in prior sections. On the other hand, the function f maps each range into a single point, i.e., $y_j = f(x_j)$ for all $x_j \in [u_j, v_j], j \in [k]$.

While these are different constructions, the attack is similar to our attack for $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$. The crux is to find m such that $f(S_1) \cap f(S_2) = \emptyset$ where $S_1 := [m - r/2, m + T)$ and $S_2 := [m + T, m + r/2)$. The analysis remains identical, and we omit the details. The attack is formalized in Algorithm 5 (differences from Algorithm 4 are marked in blue).



(a) $T = 1$ (b) $T = 3$ (c) $T = 5$

(d) $T = 7$ (e) $T = 9$ (f) $T = 11$

Figure 1: Figures show the number of occurrences of $\vec{c}[i] = [-0.5, -0.45, -0.4, \ldots, 0.4, 0.45, 0.5]$ values given $\mathsf{sk}[i] = 1, 0, -1$ respectively. Each figure has a different threshold T. Note that for $T = 7, 11$, the lines distribute slightly differently from the other figures. This is because the secret is uniform ternary, and would have more 0's than 1 or -1's, causing the lines representing $\mathsf{sk}[i] = 0$ to have more data points. This does not affect our conclusion.

4 Evaluation

In this section, we implement our attacks against all the constructions in [54] based on their public library which depends on SEAL [60], and report the corresponding results.

We choose the parameters being the same as the parameters in [54]: $N = 32768, t = 65537, \sigma = 3.2$, and use a uniform ternary secret key with length $N' = 1024$ to perform ring switching.[11] The ciphertext modulus q is chosen according to each construction as in [54, Table 2]. These parameters guarantee > 128-bit security by LWE estimator [1].[13] We first choose $r = 128$ for the modulus switching error as in [54], which guarantees $< 2^{-40}$ failure probability. We then increase r to 1024, which guarantees $< 2^{-128}$ failure probability, and show that our attack is as efficient and effective.

4.1 Parameter Analysis

How $\vec{c}$ is related to $\vec{s}$. We show how the hints reflect the secrets. Based on Lemma 3.1, the conditional probability density function is different for secrets $\in \{1, 0, -1\}$ given a certain threshold $T > 0$. We perform the following tests to empirically validate the statement. At the beginning, we sample a uniform ternary key sk of length $N' = 1024$ and a hint vector $\vec{c} \in [-0.5, 0.5, 0.05]^{N'}$ (i.e., $\{-0.5, -0.45, \ldots, 0.45, 0.5\}^{N'}$). For each $T \in [1, 3, 5, 7, 9, 11]$, we obtain 10000 hints $\vec{c}$ such that $\langle \vec{c}, \mathsf{sk} \rangle - e > T$, where e is sampled from discrete Gaussian with standard deviation 0.25,[14]. We count the number of occurrences of each possible value of $\vec{c}[i]$

[11] This is done to reduce the runtime of PartialDec and to increase the size of the valid input set. Note that in [54], the paper states that they use a fixed hamming weight of 512.[12] However, this difference is very minor and has essentially no effect on our attack.

[13] For one of the three constructions for $\mathsf{RelaxBoot}_{\mathcal{X}_4, f_4}$, the security guarantee is 106-bit security. However, as mentioned, our attack works exactly the same way for all the constructions in $\mathsf{RelaxBoot}_{\mathcal{X}_4, f_4}$ and thus this security difference is irrelevant to our attack.

[14] Note that we choose 0.25 since in bootstrapping, the error before modulus switching satisfies $E \ll q/t$ to guarantee basic correctness. As long as the correctness of BFV holds, $e \ll 1$. Thus, we choose a small standard deviation to reflect this.

given $\mathsf{sk}[i] \in \{1, 0, -1\}$.

As shown in Fig. 1, for $\mathsf{sk}[i] = 1$, it is more likely that $\vec{c}[i]$ is positive, and for $\mathsf{sk}[i] = -1$, it is more likely that $\vec{c}[i]$ is negative. For $\mathsf{sk}[i] = 0$, it is essentially a flat line. Furthermore, it is easy to see that the absolute value of the slopes for lines $\mathsf{sk}[i] = 1$ and $\mathsf{sk}[i] = -1$ are larger with a larger threshold value. Thus, we claim that the higher T is, the more significant the correlation between sk and $\vec{c}$.

	Threshold T	1	3	5	7	9	11
	Hint Generation Rate	∼1/2.37	∼1/3.09	∼1/4.24	∼1/6.19	∼1/9.47	∼1/15.41
Average value of $\vec{c}[i]$	$\mathsf{sk}[i] = 1$	0.009 92	0.011 88	0.013 99	0.016 15	0.018 46	0.020 85
	$\mathsf{sk}[i] = 0$	0.000 03	0.000 02	0.000 02	−0.000 20	−0.000 20	−0.000 18
	$\mathsf{sk}[i] = -1$	−0.009 99	−0.011 97	−0.014 05	−0.016 59	−0.018 88	−0.021 28

Table 1: Hint generation rate and average $\vec{c}[i]$ with respect to different $\mathsf{sk}[i]$ given different threshold T. The hint generation rate is the probability that given the randomly sampled uniform ternary sk (with length $N' = 1024$), we have $\langle \vec{c}, \mathsf{sk} \rangle + e > T$, where $\vec{c} \leftarrow_{\$} [-0.5, 0.5]^{N'}$ and $e \leftarrow_{\$} \chi_\sigma$ for $\sigma = 0.25$.

Choosing T, α_1, α_2. Recall that in Section 3.1, we set α_1, α_2 to be the threshold values of the recorded hints $\vec{c}$, s.t., if $\vec{c}[I] > \alpha_1$, $\vec{s}[I] = 1$; if $\vec{c}[I] < \alpha_2$, $\vec{s}[i] = -1$, and 0 otherwise. We now determine the values of α_1, α_2 by following the first several steps of relaxed functional bootstrapping. We generate a BFV ciphertext with ciphertext modulus q and plaintext modulus t, encrypting message $m \in \mathbb{Z}_t$. We ring-switch it to $N' = 1024$ and perform a modulus switch from q to t. Next, we use the switched key to decrypt the ciphertext to $m' \in \mathbb{Z}_t$. If $m' - m > T$, we record the hints. We repeat the process for 100 ciphertexts, each of which has 32768 slots, thereby $3,276,800$ potential samples. With all the samples recorded, we calculate 1) the hint generation rate (i.e., the number of successfully recorded hints over the total $3,276,800$ samples), and 2) the average value of $\vec{c}[i]$ given $\mathsf{sk}[i]$ being $1, 0, -1$ respectively. We summarize all these values in Table 1.

As both shown in Fig. 1 and Table 1, the larger T is, the better hints reflect our secret. However, a larger T also means fewer qualified samples collected from a certain number of iterations, as demonstrated by the hint generation rate in Table 1. Thus, we choose $T = 5$ for our attack that balances both metrics.

Based on the collected hints $\vec{c}$, we calculate the average $\vec{c}[i]$ for $\mathsf{sk}[i] = 1$ and $\mathsf{sk}[i] = 0$ for $T = 5$ in Table 1 to be $\alpha_1 = 0.007$. Similarly, we choose $\alpha_2 = -0.007$. If $\vec{c}[i] > \alpha_1$, we set our guessed secret key $\vec{s}[i] = 1$; if $\vec{c}[i] < \alpha_2$, we set $\vec{s}[i] = -1$; otherwise, we set $\vec{s}[i] = 0$.

4.2 Benchmarks on Actual Attacks

As shown in Table 2, we perform our attacks on $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$, $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$, $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$, $\mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$. We performed 10 attacks on each of the constructions and reported the average, minimum, and maximum recovery rate for all these attacks. It is clear that all the attacks achieve a quite high recovery rate. A lot of the attack trials achieve 100% recovery rate. With simply 2 bootstrapping calls, the average number of elements recovered is about 1023-out-of-1024 elements, and with 3 bootstrapping calls, the average number is almost 1024-out-of-1024. Note that there are some small fluctuations for different $\mathsf{RelaxBoot}$ constructions, which are normal and independent of the construction. Therefore, we claim that our attacks demonstrate great efficacy against all relaxed functional bootstrapping procedures.

Representative full key recovery time. We measure the total runtime used to recover the entire secret key, including correcting the wrongly guessed elements. Notice that with 2 or 3 bootstrapping calls, it is possible that there might be few mismatched elements between the recovered secret key and the actual one. To correct them, we test all the possible keys. For example, given that one element is wrong, there would be $3 \cdot 1024$ possible keys (since the element is ternary) based on our guess. We could test all of them by

Changing this to other small values like $0, 0.5, 0.75$ does not affect our conclusion. See Appendix C.2 for an empirical validation.

		$\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$			$\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}$			$\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$			$\mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$		
	Number of bootstrapping calls	1	2	3	1	2	3	1	2	3	1	2	3
	Representative full key recovery time (sec)	N/A	290	425	N/A	290	425	N/A	52	75	N/A	60	85
Number of	Average	1000.2	1022.8	1023.8	990.6	1023.2	1023.6	1004.6	1023.0	1023.8	1009.4	1023.2	1023.8
sk **elements**	Min	951	1022	1023	948	1022	1023	995	1020	1023	1006	1022	1023
Recovered	Max	1018	1024	1024	1015	1024	1024	1016	1024	1024	1015	1024	1024

Table 2: Summary of our attack results for each RelaxBoot construction with respect to different numbers of bootstrapping calls. Note that for $\mathsf{RelaxBoot}_{\mathcal{X}_2,f_2}, \mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}, \mathsf{RelaxBoot}_{\mathcal{X}_4,f_4}$, the functions we use are the ones benchmarked by [54]. The representative full key recovery time is the expected time needed to recover all the keys given the average number of sk elements recovered (rounded to the nearest integer) using Google Cloud Instance `N2` with 16GB RAM.

using the them to decrypt an honestly generated ciphertext. If the decryption is correct, we know that the rectified key is correct. It takes about 0.05 ms per decryption, and thus correcting 1 wrong element is about 1.5 seconds. In general, correcting k wrong elements in this way takes about $3^k \cdot \binom{N}{k}$ decryptions. This becomes impractical as soon as $k > 3$. Hence, we report the practical representative full key recovery time with 2 or 3 bootstrapping calls. There might be better ways other than brute force for error correction, which is orthogonal to our contribution, and we want to emphasize that our attack already makes the full-key recovery fast.

Other extended attacks. As mentioned in Section 3.2 and Section 3.3, we might need to sightly extend the simpler attack, or even a more general one to work with. For $\mathsf{RelaxBoot}_{\mathcal{X}_3,f_3}$ benchmarked in [54], we applied both ways and observed that the runtime are approximately the same. This is because the evaluation time of a degree-65536 function in plain is only 0.08ms. Thus, the time to find the inputs we need for the general attack is small compared to the bootstrapping calls.

The impact of failure probability. As mentioned, the attacks against BGV/BFV bootstrapping in [16] can be mitigated by improving the failure probability. The parameters in [54] guarantees 2^{-40} failure probability. If we improve it to $< 2^{-128}$, we an simply set $r = 1024$ instead of 128. The performance of our attacks would remain *exactly* the same as shown in Table 2.

5 A General Fix with Zero Overhead

As shown above, our attack can easily reconstruct the entire secret key. Interestingly, like existing attacks on exact FHE bootstrapping [16], one core aspect of our attack is that it relies on the fact that the modulus switching error is *dependent* on the secret key. Therefore, the fix we propose is to introduce an alternative modulus switching procedure that avoids such an issue, which introduces essentially no extra cost to the overall bootstrapping procedure.

Our alternative modulus switching procedure. We observe that in the current IND-CPA-D model (Section 2.1), the attacker does *not* obtain the randomness used for the Eval procedure when calling the evaluation oracle $\mathcal{O}_{\mathsf{Eval}}$. This is reasonable since to our knowledge, all the implemented homomomorphic operations in widely-used libraries are indeed *deterministic* [60, 5, 61]. And most of the widely-used FHE schemes are also deterministic [9, 19]. However, this then leaves room for a simple and effective fix.

Standard modulus switching procedure takes $(a, b) \in \mathcal{R}_q$ and obtains $(a', b') \leftarrow \lceil q'(a, b)/q \rfloor \in \mathcal{R}_{q'}$ for some $q' < q$. Instead, we propose the following alternative procedure:

$$\mathsf{ModSwitch}'(\mathsf{pk}, (a, b); r) := \lceil q' \cdot ((a, b) + (c, d))/q \rfloor \in \mathcal{R}_{q'}^2$$

where $(c, d) \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$.

In terms of correctness, this procedure is essentially identical to regular modulus switching, since adding an encryption of zero does not change the encrypted plaintext.

In terms of security, intuitively, this procedure hides the "hint" we generate since c is indistinguishable from a uniformly randomly drawn $\mathcal{R}_q$ element (by RLWE), and it acts a one-time-pad making a' indistinguishable from a uniform element over $\mathcal{R}_q$. Therefore, when computing $\lceil q'(a', b')/q \rfloor$, the rounding error is hidden from the adversary, which then prevents the adversary from generating the inequality hints based on the homomorphic decryption of $\lceil q'(a', b')/q \rfloor$.

Moreover, consider the following extreme scenario: if the secret sk is all-ones, the modulus switching error is likely to be large. On the other hand, if sk contains only a single one, the modulus switching error is close to 0. Specifically, the error has a standard deviation of $O(\sqrt{h})$ where h is the hamming weight of the secret key. Intuitively, the hints leaks the "hamming weight" of the secret key. To avoid this, one can simply use secret keys with fixed hamming weights as in many works (e.g., [35, 42, 20]) instead of uniform binary or ternary keys.

We formally capture our fix against any attacks based on modulus switching errors in the following lemma.

Lemma 5.1. For any ring dimension $N > 0$, encryption error distribution χ, modulus $q > q' > 0$ (satisfying $|\chi| \cdot q'/q = \mathsf{negl}(N)$ and $q/q' \gg t$, where t is the plaintext modulus, such that modulus switching does not affect correctness), hamming weight $h > 0$ and $k = \mathsf{poly}(N)$. Let $\mathcal{T}_h$ (resp. $\mathcal{B}_h$) denote ternary (resp. binary) distribution with hamming weight h. For $i \in [0, 1]$, let $\mathsf{sk}_i \leftarrow_\$ \mathcal{T}_h$ (resp. $\mathcal{B}_h$), $\mathsf{pk}_i \leftarrow \mathsf{KeyGen}(\mathsf{sk}_i)$[15]; for $j \in [k]$, and any message m_j, let $\mathsf{ct}_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, m_j)$, $\mathsf{ct}'_{i,j} := (a'_{i,j}, b'_{i,j}) \leftarrow \mathsf{ModSwitch}'(\mathsf{pk}_i, \mathsf{ct}_{i,j})$, and $m'_{1,j} \leftarrow b'_{i,j} - a'_{i,j}\mathsf{sk}$. For any PPT adversary, it holds that:

$$(\mathsf{pk}_1, (\mathsf{ct}_{1,j}, \mathsf{ct}'_{1,j}, m'_{1,j})_{j\in[k]}) \approx_c (\mathsf{pk}_2, (\mathsf{ct}_{2,j}, \mathsf{ct}'_{2,j}, m'_{2,j})_{j\in[k]})$$

Proof sketch. The proof is straightforward. First, we have $m'_{1,j} = \lceil q \cdot m_j/t \rfloor + e_{1,j} + E_{1,j}$, where $e_{1,j}$ is the encryption error sampled by χ and scaled by q'/q, and $E_{1,j}$ is the modulus switching error generated by $\langle c'_{1,j}, \mathsf{sk}_1 \rangle$ where $c'_{1,j} \leftarrow_\$ [-0.5, 0.5)^N$. Similarly, we have $m'_{2,j} = \lceil q \cdot m_j/t \rfloor + e_{2,j} + E_{2,j}$. Trivially, the distributions of $E_{1,j}, E_{2,j}$ are identical, since $\mathsf{sk}_1, \mathsf{sk}_2$ have the same hamming weight. Furthermore, with $|\chi| \cdot q'/q = \mathsf{negl}(N)$, $|e_{i,j}|/|E_{i,j}| = \mathsf{negl}(\lambda)$. Based on the smudging lemma [4], the distribution of $e_{i,j} + E_{i,j}$ is statistically indistinguishable from the distribution of $E_{i,j}$. Hence, we have $m'_{1,j} \approx_c m'_{2,j}$ even given $\mathsf{pk}_i, \mathsf{ct}_{i,j}, \mathsf{ct}'_{i,j}$. □

Efficiency analysis. As aforementioned, our fix incurs essentially no overhead by simply generating an encryption of zeros for each modulus switching operation. This cost is as small as a plaintext multiplication. Compared to the mitigation methods provided in prior works, which requires to reduce the error probability from 2^{-40} to 2^{-128}, our fix has a negligible cost and can be applied to *any* such kind of attacks.

Empirical Analysis. We have also implemented our fix, and our evaluation shows that with this fix, our proposed attacks do not work anymore. And this similarly extends to the attacks on exact FHE bootstrapping procedures introduced in [16].

Comparison to [6]. A recent work [6] also proposes an alternative way to perform modulus switching via adding an encryption of zero. However, they propose to use the added encryption of zero to analyze the drifted noise, which is different from our purpose: disallowing the adversary to learn the hints after modulus switching. Similar to our discussion in Section 6, [6] also proposes an alternative strengthened IND-CPA-D model, which allows the adversary to call an encryption oracle to generate ciphertexts honestly but with the capability of reject sampling. This is different from our model defined in Section 6, which is to allow the adversary to learn the randomness during circuit evaluation.

[15] This denotes the public key, including evaluation key, generation of the targeted FHE scheme given secret key.

6 IND-CPA-DR Model

In this section, we examine the assumption we hold in the previous section and also in the prior works [55, 13, 16], that in the IND-CPA-D model, the attacker does *not* obtain the randomness used for the evaluation procedure. This is indeed reasonable in some applications, when the adversary is a third party only observing the output from the client and the server, while the honest server would use a fresh randomness for each evaluation, hidden from the adversary. On the other hand, if the passive attacker is the server itself (or any other attackers who can see the internal states of the server), the fix breaks immediately. The server can see the randomness itself, recover c, d, and extract the inequality hints.

Hence, we propose a more general model to capture this stronger model that allows the adversary to obtain the randomness. We call it IND-CPA-DR (IND-CPA-D with Randomness), formally as follows.

Definition 6.1 (IND-CPA-DR Security)**.** For an FHE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, Π is IND-CPA-DR secure if for any PPT adversary $\mathcal{A}$, for any plaintexts m_0, m_1, it holds that:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen} \\ b \leftarrow_{\$} \{0,1\} \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Eval}}, \mathcal{O}_{\mathsf{Dec}}}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_b)) \end{array}\right]$$

The oracles $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Eval}}, \mathcal{O}_{\mathsf{Dec}}$ are defined in Algorithm 6, with difference from the ones in Algorithm 1 marked in blue.

Algorithm 6 Oracles for IND-CPA-D Security

procedure $\mathcal{O}_{\mathsf{Enc}}(m'_0, m'_1; \mathsf{pk}, i)$
    Same as Algorithm 1
procedure $\mathcal{O}_{\mathsf{Eval}}(C, i_1, \ldots, i_k; b, S)$
    Sample randomness r.
    $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{pk}, C, (S[i_j].\mathsf{ct})_{j\in[k]}; r)$
    $r_0 := C((S[i_j].m_0)_{j\in[k]})$
    $r_1 := C((S[i_j].m_1)_{j\in[k]})$
    $S[i] \leftarrow (r_0, r_1, \mathsf{ct})$
    $i \leftarrow i + 1$
    Return ct, i, r
procedure $\mathcal{O}_{\mathsf{Dec}}(j; \mathsf{sk}, S)$
    Same as Algorithm 1

Our attack and our fix in the new model. Under the model of IND-CPA-DR, our fix does not stand and the attack again works the same way as presented in Section 3: With the randomness, our attack simply generates the inequality hints as before. In other words, our proposed mitigation above relies on the assumption that the ciphertext output from the $\mathsf{ModSwitch}'$ procedure is hidden from the adversary. In this new model, this assumption no longer holds, thus making the mitigation moot. To conclude, for stronger adversaries that *can* observe the randomness used by the server, our attack remains effective; for weaker adversaries who *cannot* obtain the randomness, our mitigation fixes *all* the existing modulus-switching-error-based attacks, including ours. **An even stronger model.** As discussed in Section 2.1, in [6], a strong IND-CPA-D model is defined. This model can be combined with our IND-CPA-DR model to produce an even stronger model (strong IND-CPA-DR model). This stronger model can capture stronger adversaries, which could be helpful in other applications. We leave the study of this stronger model for future work to explore.

7 Concluding Remarks

In this paper, we introduce IND-CPA-D attacks on relaxed functional bootstrapping constructions. Our attacks show that all the existing relaxed functional bootstrapping constructions suffer from the IND-CPA-D security issue: The secret key can be *fully recovered* by using 2-3 bootstrapping calls. Even with a single bootstrapping call, on average, $> 98\%$ of the secret key elements can be recovered and thus the security is greatly reduced. Furthermore, our attack is independent of the valid input set and the function used for bootstrapping, making it powerful and widely applicable. Moreover, as mentioned, our attack is immune to the existing mitigation of IND-CPA-D attacks to BGV/BFV bootstrapping.

We additionally show a randomized modulus switching method with no overhead to the bootstrapping procedure, serving as a general mitigation to all modulus-switching-based attacks, including ours and the ones in [16], in the IND-CPA-D model. Lastly, we extend the IND-CPA-D model to IND-CPA-DR model to capture the real-world cases where the adversaries are able to observe the randomness used for homomorphic evaluations.

Potential mitigation. Lastly, we briefly discuss potential mitigation in the IND-CPA-DR model. As mentioned in Section 3.3, it is possible to prevent our attacks by setting f_{pre} to map the plaintexts that are $\notin \mathcal{X}$ into certain points in $\mathbb{Z}_t$. However, this requires f_{pre} to have degree $O(t)$, making the existing relaxed bootstrapping constructions no longer competitive in terms of efficiency. It remains an open question to design an efficient relaxed bootstrapping that is secure against such attacks in the IND-CPA-DR model.

In short, we claim that in relaxed functional bootstrapping, it is important to have all the inputs remain in $\mathcal{X}$ to avoid our attacks. With this, the existing relaxed functional bootstrapping procedures in [54] maintains the same IND-CPA-D security as regular bootstrapping procedures [35, 15, 28, 27, 59, 41, 56, 57, 43]. Alternatively, as in Appendix C.1, one could enlarge N', i.e., the length of the secret key. This greatly harms the efficiency of the attack by requiring a lot more bootstrapping calls, which becomes impractical in practice. See Appendix C.1 for more details.

Acknowledgements

We are grateful to all the anonymous reviewers for all of the insightful suggestions. We are also grateful to Marc Joye for providing feedback on the manuscript of this work and helpful suggestions on the discussion of [6]. Zeyu Liu is generously supported by Yale CADMY.

References

[1] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[2] A. Alexandru, A. A. Badawi, D. Micciancio, and Y. Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. Cryptology ePrint Archive, Paper 2024/203, 2024.

[3] S. Angel, H. Chen, K. Laine, and S. T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[4] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501, Cambridge, UK, Apr. 15–19, 2012. Springer, Heidelberg, Germany.

[5] A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. `https://eprint.iacr.org/2022/915`, commit: 122f470e0dbf94688051ab852131ccc5d26be934.

[6] O. Bernard, M. Joye, N. P. Smart, and M. Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. Eurocrypt'25, 2024.

[7] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020.

[8] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*, page 868–886. Springer-Verlag, 2012.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[10] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 22–25, 2011.

[11] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, Germany, Aug. 14–18, 2011.

[12] B. Chaturvedi, A. Chakraborty, N. Mishra, A. Chatterjee, and D. Mukhopadhyay. IND-CPA^c: A new security notion for conditional decryption in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2025/045, 2025.

[13] M. Checri, R. Sirdey, A. Boudguiga, and J.-P. Bultel. On the practical cpad security of "exact" and threshold fhe schemes and libraries. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part III*, page 3–33, Berlin, Heidelberg, 2024. Springer-Verlag.

[14] H. Chen, R. Gilad-Bachrach, H. Kyoohyung, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Medical Genomics*, 11, 10 2018.

[15] H. Chen and K. Han. Homomorphic lower digits removal and improved FHE bootstrapping. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018.

[16] J. H. Cheon, H. Choe, A. Passelègue, D. Stehlé, and E. Suvanto. Attacks against the IND-CPA-d security of exact FHE schemes. CCS 2024.

[17] J. H. Cheon, S. Hong, and D. Kim. Remark on the security of CKKS scheme in practice. Cryptology ePrint Archive, Paper 2020/1581, 2020.

[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[20] A. Costache, B. R. Curtis, E. Hales, S. Murphy, T. Ogilvie, and R. Player. On the precision loss in approximate homomorphic encryption. In *Selected Areas in Cryptography – SAC 2023: 30th International Conference, Fredericton, Canada, August 14–18, 2023, Revised Selected Papers*, page 325–345, Berlin, Heidelberg, 2023. Springer-Verlag.

[21] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi. LWE with side information: Attacks and concrete security estimation. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Heidelberg, Germany.

[22] A. Dalvi, A. Jain, S. Moradiya, R. Nirmal, J. Sanghavi, and I. Siddavatam. Securing neural networks using homomorphic encryption. In *2021 International Conference on Intelligent Technologies (CONIT)*, pages 1–7, 2021.

[23] L. Ducas and D. Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer.

[24] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[25] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou. Thorpir: Single server pir via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1448–1462, New York, NY, USA, 2024. Association for Computing Machinery.

[26] L. Freitas, A. Tonkikh, A.-A. Bendoukha, S. Tucci-Piergiovanni, R. Sirdey, O. Stan, and P. Kuznetsov. Homomorphic sortition – single secret leader election for PoS blockchains. Cryptology ePrint Archive, Paper 2023/113, 2023.

[27] R. Geelen, I. Iliashenko, J. Kang, and F. Vercauteren. On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. Eurocrypt 2023. `https://eprint.iacr.org/2022/1364`.

[28] R. Geelen and F. Vercauteren. Bootstrapping for bgv and bfv revisited. *J. Cryptol.*, 36(2), mar 2023.

[29] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Heidelberg, Germany.

[30] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[31] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, Germany, Aug. 18–22, 2013.

[32] Q. Guo, D. Nabokov, E. Suvanto, and T. Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2025. USENIX Association.

[33] S. Halevi and V. Shoup. HElib, 2014. `https://github.com/homenc/HElib`.

[34] S. Halevi and V. Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. `https://eprint.iacr.org/2020/1481`.

[35] S. Halevi and V. Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, Jan. 2021.

[36] J. HU, J. Chen, W. Dai, and H. Wang. Fully homomorphic encryption-based protocols for enhanced private set intersection functionalities. Cryptology ePrint Archive, Paper 2023/1407, 2023. `https://eprint.iacr.org/2023/1407`.

[37] I. Hwang, H. Lee, J. Seo, and Y. Song. Practical zero-knowledge PIOP for public key and ciphertext generation in (multi-group) homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1879, 2024.

[38] W. jie Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 24–27, 2021.

[39] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In W. Enck and A. P. Felt, editors, *USENIX Security 2018*, pages 1651–1669. USENIX Association, Aug. 15–17, 2018.

[40] A. Kim, Y. Polyakov, and V. Zucca. Revisiting homomorphic encryption schemes for finite fields. In *ASIACRYPT 2021*, page 608–639. Springer, 2021.

[41] J. Kim, J. Seo, and Y. Song. Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from ckks. CCS'24, 2024. `https://eprint.iacr.org/2024/109`.

[42] S. Kim, M. Park, J. Kim, T. Kim, and C. Min. Evalround algorithm in ckks bootstrapping. Asiacrypt 2022, 2022. `https://eprint.iacr.org/2022/1256`.

[43] D. Lee, S. Min, and Y. Song. Functional bootstrapping for packed ciphertexts via homomorphic LUT evaluation, 2024.

[44] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[45] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In J. Guo and R. Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 36–68, Singapore, 2023. Springer Nature Singapore.

[46] K. Lee and Y. Yeo. SophOMR: Improved oblivious message retrieval from SIMD-aware homomorphic compression. Cryptology ePrint Archive, Paper 2024/1814, 2024.

[47] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. Eurocrypt 2021.

[48] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Heidelberg, Germany, Oct. 17–21, 2021.

[49] B. Li, D. Micciancio, M. Schultz-Wu, and J. Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*, page 560–589, Berlin, Heidelberg, 2022. Springer-Verlag.

[50] Z. Liu, K. Sotiraki, E. Tromer, and Y. Wang. Snake-eye resistant PKE from LWE for oblivious message retrieval and robust encryption. In *EUROCRYPT 2025, Part III*, LNCS, pages 126–156, June 2025.

[51] Z. Liu and E. Tromer. Oblivious message retrieval. In Y. Dodis and T. Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 753–783, Santa Barbara, CA, USA, Aug. 15–18, 2022. Springer, Heidelberg, Germany.

[52] Z. Liu, E. Tromer, and Y. Wang. Group oblivious message retrieval. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4367–4385, 2024.

[53] Z. Liu, E. Tromer, and Y. Wang. PerfOMR: Oblivious message retrieval with reduced communication and computation. In *USENIX Security 2024*. USENIX Association, Aug. 2024.

[54] Z. Liu and Y. Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. Asiacrypt'23, 2024. All the references in our paper are with respect to the full version on eprint.

[55] G. Ma and H. Li. On the security of homomorphic encryption schemes with restricted decryption oracles. *Journal of Systems Science and Complexity*, 37:2240–2261, 08 2024.

[56] S. Ma, T. Huang, A. Wang, and X. Wang. Accelerating bgv bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. Cryptology ePrint Archive, Paper 2024/115, 2024. `https://eprint.iacr.org/2024/115`.

[57] S. Ma, T. Huang, A. Wang, and X. Wang. Faster BGV bootstrapping for power-of-two cyclotomics through homomorphic NTT. Asiacrypt'24, 2024.

[58] S. J. Menon and D. J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

[59] H. Okada, R. Player, and S. Pohmann. Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. Asiacrypt 2023. `https://eprint.iacr.org/2023/1304`.

[60] Microsoft SEAL, 2020. `https://github.com/Microsoft/SEAL`.

[61] Zama-ai, tfhe-rs, 2023. `https://github.com/zama-ai/tfhe-rs`, commit: 509bf3e2846bc98dd42d0e8eeb7f27852e5b632a.

[62] Y. Wang and F. Zhang. Qelect: Lattice-based single secret leader election made practical. Usenix Security 2025, 2025.

[63] X. Zhang, R. Wang, Z. Liu, B. Xiang, Y. Deng, and X. Lu. FHE-SNARK vs. SNARK-FHE: From analysis to practical verifiable computation. Cryptology ePrint Archive, Paper 2025/302, 2025.

A Additional Discussion on Prior Works

As mentioned, in Footnote 2, [16] did not directly attack all BFV/BGV bootstrapping procedures. They have attacked FHEW/TFHE, which are both exact FHE schemes as well, and have attacked CKKS-bootstrapping-based BGV/BFV bootstrapping. However, intuitively, for other regular BGV/BFV bootstrapping procedures [35, 15, 28, 27, 59, 41, 56, 57, 43], a similar attack applies: this is because for plaintext space p^γ, regular BGV/BFV bootstrapping need to choose some $\gamma' > \gamma$ and this γ' is to accommodate the noise dependent on the secret key sk. Therefore, intuitively, the attacks in [16] extend to such a setting as well. However, even if this extension is correct, it can again be mitigated by having a negligible failure probability (by setting γ' to be large enough). In either scenario, it does not affect any of the contributions in this work.

B Additional Preliminary

Formal definition of ℓ-closeness. We formally recall ℓ-closeness here.

Definition B.1 (ℓ-closeness)**.** The bootstrapping procedure is ℓ-close, if it satisfies the following: For the same quantifiers as correctness; for all $x \in \mathbb{Z}_t \setminus \mathcal{X}$, let $y_{x,1}, \ldots, y_{x,|\mathcal{Y}|}$ denote all the points in $\mathcal{Y}$ satisfying $|f_x^{-1}(y_{x,1}) - x| \leq |f_x^{-1}(y_{x,2}) - x| \leq \cdots \leq |f_x^{-1}(y_{x,|\mathcal{Y}|}) - x|$,[16][17] and $\mathcal{S}_x := \{y_{x,1}, \ldots, y_{x,\ell}\}$; it holds that for all $i \in [N]$, if $\vec{m}[i] \notin \mathcal{X}$:

$$\Pr\left[f(\vec{m}[i]) \in \mathcal{S}_{\vec{m}[i]}\right] > 1 - \mathsf{negl}(\lambda).$$

C Additional Evaluation

		$\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$		
	Number of bootstrapping calls	50	75	100
	Representative full key recovery time (hrs)	N/A	N/A	4.1
Number of	Average	32576.6	32760.8	32766.6
sk **elements**	Min	32319	32754	36764
Recovered	Max	32727	32763	32768

Table 3: Summary of our attack results against each $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$ with $N' = 32768$ instead of 1024 with $T = 30$.

C.1 Larger N

One additional consideration is the size of N'. [54] chooses $N' = 1024$ for better performance since r is proportional to N': the larger N' is, the larger r needs to be, and thus the smaller valid input set the procedure can support. If they choose $N' = 32768$, the performance (i.e., time per bit bootstrapped) will be reduced by about 10x.

However, it also seems intuitive that the larger N' is, the harder the attack gets. One may ask whether a larger N' helps mitigate the attack. We thus launched our attack in a similar procedure as above, with larger thresholds $T := [6, 18, 30, 42, 54, 66]$. We choose $T = [6, 18, 30, 42, 54, 66]$ since $\sqrt{32768/1024} \approx 6$ when rounded to the nearest integer and we used $T = [1, 3, 5, 7, 9, 11]$ for $N' = 1024$ Note that it is square-root since

[16] Let $f_x^{-1}(y)$ denote a point z where $f(z) = y$ and $z - x = \min_{z' \in \mathcal{X}, f(z')=y}(z' - x)$. In other words, $f_x^{-1}(y)$ outputs a point that is (1) a valid input in $\mathcal{X}$; and (2) the closest point to x among all possible points z' satisfying $f(z') = y$.

[17] If $|f_x^{-1}(y_{x,i}) - x| = |f_x^{-1}(y_{x,j}) - x|$ for $i \neq j$, then any order is accepted.

	Threshold T	6	18	30	42	54	66
	Hint Generation Rate	$\sim 1/2.27$	$\sim 1/3.01$	$\sim 1/3.72$	$\sim 1/6.27$	$\sim 1/9.94$	$\sim 1/16.86$
Average value of $\tilde{c}[i]$	$\mathsf{sk}[i] = 1$	0.001 791	0.002 174	0.002 443	0.003 010	0.003 456	0.003 914
	$\mathsf{sk}[i] = 0$	0.000 039	0.000 040	0.000 039	−0.000 041	−0.000 043	−0.000 043
	$\mathsf{sk}[i] = -1$	−0.001 713	−0.002 094	−0.002 364	−0.002 932	−0.003 382	−0.003 850

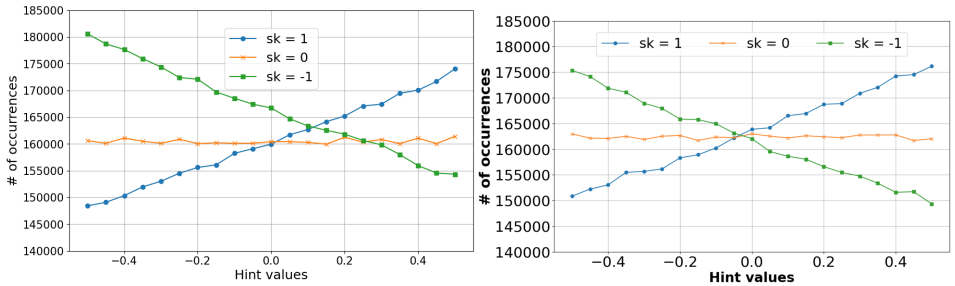
Table 4: Hint generation rate and average $\tilde{c}[i]$ with respect to different $\mathsf{sk}[i]$ given different threshold T. Here $N' = 32768$.

the standard deviation of the modulus switching error is $O(\sqrt{N'})$. Since the results we obtain are similar with $N' = 1024$, for better readability, we defer the figure (Fig. 3) and table (Table 4) to be after reporting our actual attack below. With these, we choose $T = 30$, $\alpha_1 = 0.0012$ and $\alpha_2 = -0.0012$ as parameters for our actual attack.

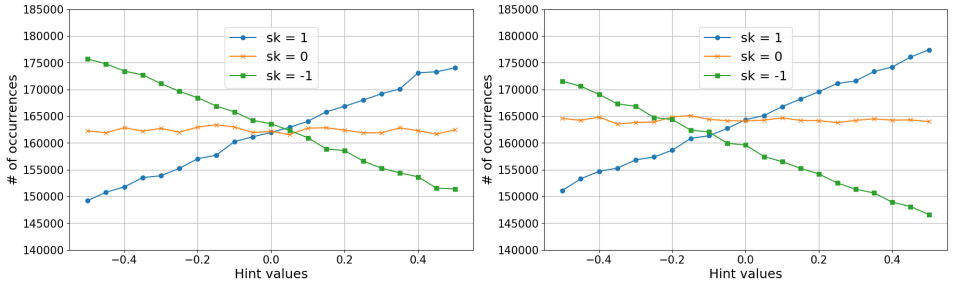
With parameters chosen, we perform our attack against $\mathsf{RelaxBoot}_{\mathcal{X}_1,I}$ with $N' = 32768$ and $r = 768$ for five trials. As in Table 3, our attack shows a high key recovery rate, though it now requires more bootstrapping calls and thus more time. However, it only takes hours to fully recover the key, thus remaining a strong attack. Note that since our attack extends to other RelaxBoot constructions and to larger r as for the case of $N' = 1024$, we omit the details. **Mitigation with such large N'.** As shown above, for large N', it takes ≥ 50 bootstrapping calls to recover most secret key elements, with about 200 elements guessed wrongly. Furthermore, our tests showed that with only 25 calls, there are > 1024 elements guessed wrongly on average (and at least 781-out-of-32768 elements were guessed wrongly) for the 5 trials we ran. Since N' is only used when there is one multiplicative level left (thereby the ciphertext modulus for the ring-switching key is small), with only 1024 secret elements, there is $>$ 128-bit security. Thus, our attack with 25 calls barely affects security. Therefore, in applications where it is impossible for the adversary to obtain > 25 bootstrapping results to further derive the hints, enlarging the ring-switching secret-key length can be used to achieve a "bounded form" of IND-CPA-D security. However, as we pointed out before in 4.2, this mitigation reduces the efficiency by roughly 10x. If one enlarge N' to a smaller number, e.g. 8192, the bootstrapping efficiency cost is smaller while the attack is still significantly harder (may take ¿ 10 calls to recover most of the secrets). And this may be sufficient for practical uses. Further mitigation with better efficiency can be an interesting future direction.

C.2 How σ Affects

As mentioned in Footnote 14, we use $\sigma = 0.25$ since e is normally an error $\ll 1$. As long as $\sigma < 1$, it has very little effect on the simulation as shown in Fig. 2.

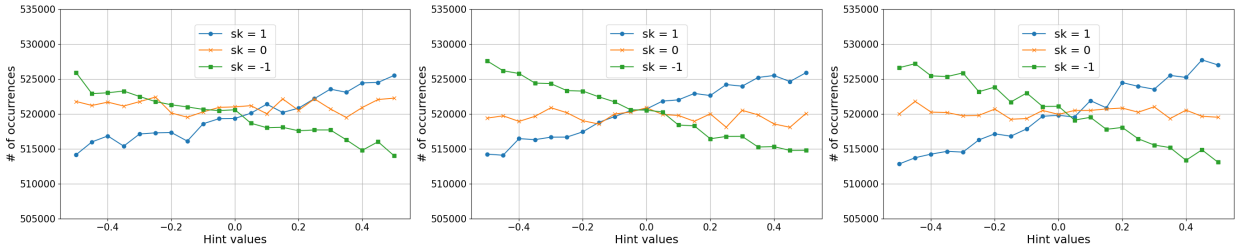


(a) $\sigma = 0$ (b) $\sigma = 0.25$

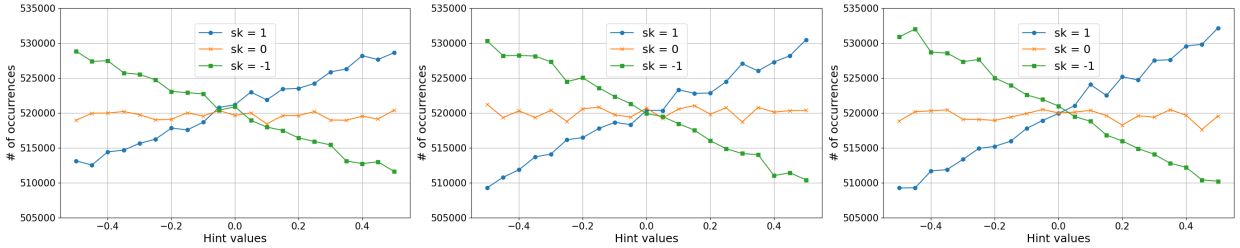


(c) $\sigma = 0.5$ (d) $\sigma = 0.75$

Figure 2: Figures show the number of occurrences of $\vec{c}[i] = [-0.5, -0.45, -0.4, \ldots, 0.4, 0.45, 0.5]$ values given $\mathsf{sk}[i] = 1, 0, -1$ respectively. Each figure has a the same threshold $T = 5$ but different σ.



(a) $T = 6$ (b) $T = 18$ (c) $T = 30$



(d) $T = 42$ (e) $T = 54$ (f) $T = 66$

Figure 3: Figures show the number of occurrences of $\vec{c}[i] = [-0.5, -0.45, -0.4, \ldots, 0.4, 0.45, 0.5]$ values given $\mathsf{sk}[i] = 1, 0, -1$ respectively. Each figure has a different threshold T. Here $N' = 32768$.