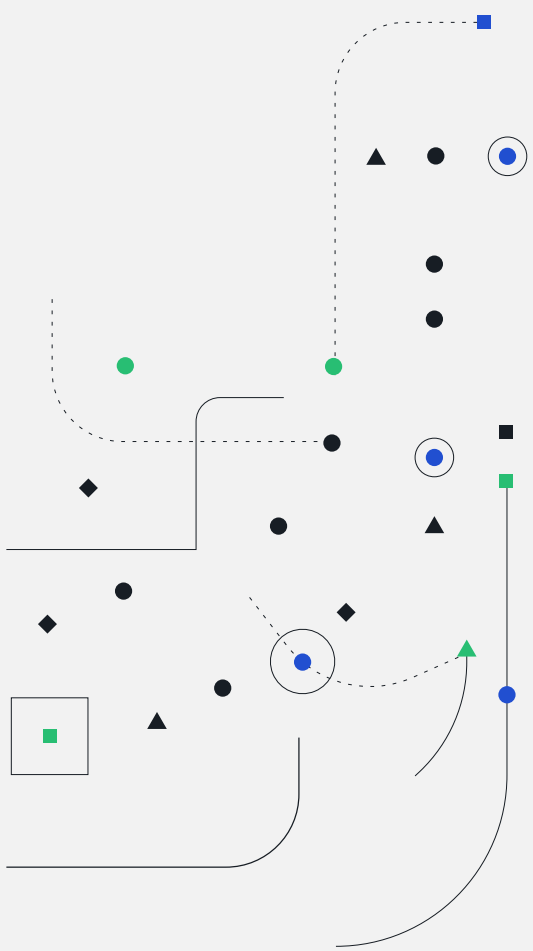


Duality

# Bootstrapping in FHEW-like Cryptosystems

Daniele Micciancio
Yuriy Polyakov

# Introduction

Bootstrapping is the core operation of fully homomorphic encryption (FHE)

- Bootstrapping: homomorphic evaluation of a decryption circuit using an encryption of the secret key
  - Refreshes the noise to support more computations
- There are two common bootstrapping approaches for exact homomorphic encryption schemes: BGV-like and FHEW-like
- BGV-like approach: Simultaneously refresh a vector of many (hundreds/thousands) encrypted numbers
  - The latency of BGV-like bootstrapping is relatively high: from 10 seconds up to thousands of seconds (single-threaded CPU)
  - Amortized time is small (on the order of 1 ms)
- FHEW-like approach: Refresh one small number, e.g., bit, at a time, also performing an arbitrary computation during bootstrapping
  - Latency is low: from 10 ms to hundreds of milliseconds

# FHEW-like approach: History

- Ducas and Micciancio [DM15] proposed and implemented FHEW as a scheme for Boolean operations. Each Boolean operation (except for negation) requires bootstrapping
  - They achieved a runtime of less than of one second for bootstrapping (0.69 seconds)
- The FHEW cryptosystem introduced two important technical innovations:
  - The observation that the evaluation of an arbitrary function can be split into the computation of a linear function followed by a table look-up, which can be easily performed during bootstrapping essentially at no additional cost.
  - A ring version of the GSW cryptosystem (based on the Ring LWE problem), and a method to use it to efficiently implement the cryptographic accumulator using a single (Ring) LWE ciphertext.
- Chillotti, Gama, Georgieva and Izabachene developed TFHE as an improved version of FHEW
  - In [CGGI16], they reduced the runtime to 52 milliseconds by using a different bootstrapping technique and also applying several algorithmic optimizations
  - In [CGGI17], the authors further reduced the bootstrapping time to 13 milliseconds by applying further (implementation-specific) optimizations

# Differences between FHEW and TFHE

The main differences can be summarized as follows

- FHEW uses a ring version of the bootstrapping procedure proposed by Alperin-Sherif and Peikert [AP14] based on a GSW cryptosystem
- TFHE employs a ring version of the bootstrapping procedure proposed by Gama, Izabachene, Nguyen and Xie [GINX16], also based on a GSW cryptosystem
- The TFHE (FHE over the Torus) approach replaces integer arithmetic modulo q with real arithmetic over the unit interval [0; 1)
- Binary secret distribution is used for Ring LWE instead of Gaussian distribution in FHEW
- The TFHE papers also proposed several optimizations
- There has not been a fair comparison of FHEW vs TFHE
    - It was not clear which improvements come from changing the arithmetic/secret distribution and which optimizations equally apply to both schemes

# Goals of our work

The purpose of our work is twofold:

- Producing a standard-compliant version of the FHEW/TFHE cryptosystem, within the PALISADE lattice cryptography library, to enable the comparison of FHEW with the other main FHE schemes (e.g., BGV and BFV) currently considered for standardization.
    - Current community standard published at HomomorphicEncryption.org only includes uniform ternary, Gaussian, and uniform secret key distributions
    - The binary secret key distribution is not properly implemented in the LWE estimator, main tool to estimate the work factor for (Ring) LWE parameters; hence it is hard to get an accurate estimate of bits of security
- Implement (Ring LWE versions of) both bootstrapping procedures [AP14, GINX16] within the same (integer-based) FHEW cryptosystem, in order to better understand the relative merits of the two bootstrapping methods, and the differences between FHEW and TFHE.

# Basic concepts

- $\mathrm{RLWE}_s(m) = (a, as + e + m)$
- $\mathrm{RLWE'}_s(m) = \mathrm{RLWE}_s(m), \mathrm{RLWE}_s(Bm), \mathrm{RLWE}_s(B^2 m), \ldots, \mathrm{RLWE}_s\left(B^{k-1} m\right)$
- $\mathrm{RGSW}_s(m) = \left(\mathrm{RLWE'}_s(-s \cdot m), \mathrm{RLWE'}_s(m)\right)$
- Two products that will be important for the accumulator
  - $\mathrm{RGSW} \times \mathrm{RGSW} \rightarrow \mathrm{RGSW}$
  - $\mathrm{RLWE} \times \mathrm{RGSW} \rightarrow \mathrm{RLWE}$

# High-level description of bootstrapping procedure

Bootstrapping is implemented by means of a cryptographic accumulator $\mathrm{ACC}$ holding values from $Z_q$ and supporting the following operations:

1. **Initialize**: $\mathrm{ACC} \leftarrow b$, setting the content of $\mathrm{ACC}$ to any known value $b \in Z_q$

2. **Update**: $\mathrm{ACC} \xleftarrow{+} c \cdot E(s)$ , modifying the content of the accumulator from $\mathrm{ACC}[v]$ to $\mathrm{ACC}[v + c \cdot s]$

3. **Extract**: $f(\mathrm{ACC})$, returning an encryption $E(f(v))$ of function $f$ applied to the current content of the accumulator $\mathrm{ACC}[v]$ .

Using this cryptographic data structure with

$\mathrm{ek} = E(\mathrm{s}) = \big(E(s_1), \cdots, E(s_n)\big)$

as a bootstrapping (also called “evaluation" or “refreshing") key, the bootstrapping procedure is easily implemented by the pseudo-code

$\mathrm{Bootstrap}(\mathrm{ek} = (E(s_i))_i, (\mathbf{a}, b))$:
  $\mathrm{ACC} \leftarrow b$
  **for** $i = 1, \ldots, n$ **do**
    $c_i = -a_i \pmod q$
    $\mathrm{ACC} \xleftarrow{+} c_i \cdot \mathrm{ek}_i$
  **return** $f(\mathrm{ACC})$

# AP bootstrapping procedure

- The AP bootstrapping procedure [AP14] supports basic updates $\mathrm{ACC} \xleftarrow{+} c \cdot E(s)$ for arbitrary $s \in R_q$.
- Then, $\mathrm{ACC} \xleftarrow{+} c \cdot E(s)$ is implemented by providing (in the bootstrapping key) encryptions $E(2^i s)$ of multiples of the secret key elements $s$, taking the binary expansion of $c = \sum_j 2^j c_j$, and then executing $\mathrm{ACC} \xleftarrow{+} c \cdot E(s)$ for all $j$ such that $c_i = 1$.
- $\mathbf{Z}_{j,c_j}$ are encryptions of j-th digit for the encryption of each $s_i$
- $B_r$ is the base for digits (choosing more than 2 improves the runtime)

$$
\begin{aligned}
&\mathrm{Update}(c, \{\mathbf{Z}_{j,v}\}_{j,v} = E(s)):\\
&\quad \textbf{for } j = 0, \ldots, \log_{B_\mathrm{r}} q - 1\\
&\qquad c_j = \lfloor c / {B_\mathrm{r}}^j \rfloor \bmod B\\
&\qquad \textbf{if } c_j > 0\\
&\qquad\quad \mathrm{ACC} \leftarrow \mathrm{ACC} \diamond \mathbf{Z}_{j,c_j}\\
&\quad \textbf{return } \mathrm{ACC}
\end{aligned}
$$

# GINX bootstrapping procedure (extended to arbitrary secrets)

- The GINX bootstrapping procedure [GINX16] supports basic updates $\mathrm{ACC} \xleftarrow{+} c \cdot E(s)$ where $c \in Z_q$ is arbitrary, but $s = \{0,1\}$ is a single bit.
- For an arbitrary secret $s = \sum_i 2^i c_i \in Z_q$, one can execute $\mathrm{ACC} \xleftarrow{+} (2^i c) \cdot E(s)$ for all $i$.
- Arbitrary elements of $Z_q$ can be expressed using $U = \{1,2,4,\ldots,2^{k-1}\}$. For ternary secrets one can use $U = \{1,-1\}$ or $U = \{1,-2\}$ to make the representation unambiguous.

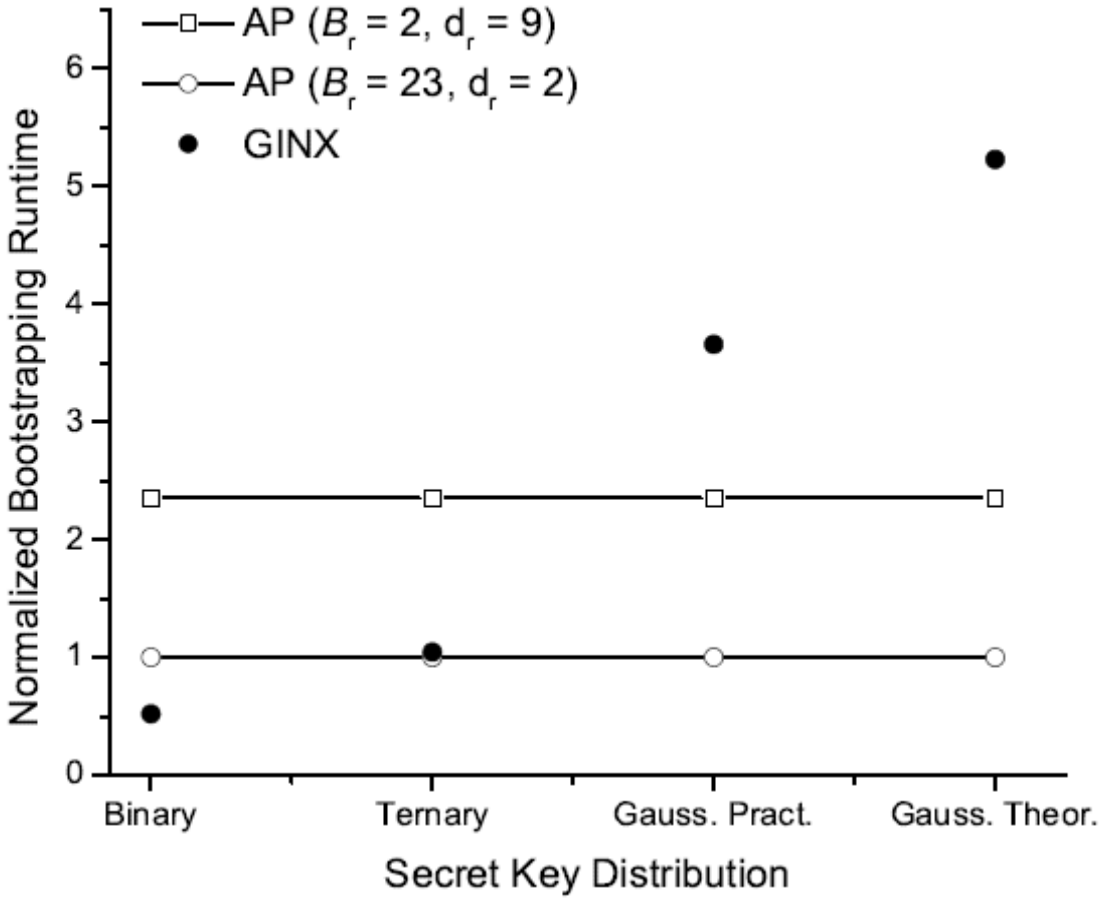
$$\begin{aligned}&\mathrm{Update}'(c,\{\mathbf{Z}_u\}_u = E'(t)):\\ &\quad \textbf{for } u \in U\\ &\qquad \mathrm{ACC} \leftarrow \mathrm{ACC} + (X^{u\cdot c} - 1)\,(\mathrm{ACC} \diamond \mathbf{Z}_u)\\ &\quad \textbf{return } \mathrm{ACC}\end{aligned}$$

# Optimizations applicable to both FHEW and TFHE

- The authors of [CGGI16] noticed that if multiplications are not needed, the product $\mathrm{RGSW} \times \mathrm{RGSW} \rightarrow \mathrm{RGSW}$ van be replaced with $\mathrm{RLWE} \times \mathrm{RGSW} \rightarrow \mathrm{RLWE}$
  - Speed-up of 6x as compared [DM15]
- The extraction function $f$ can be directly included in the Update step. This leads to a simpler and more efficient procedure.

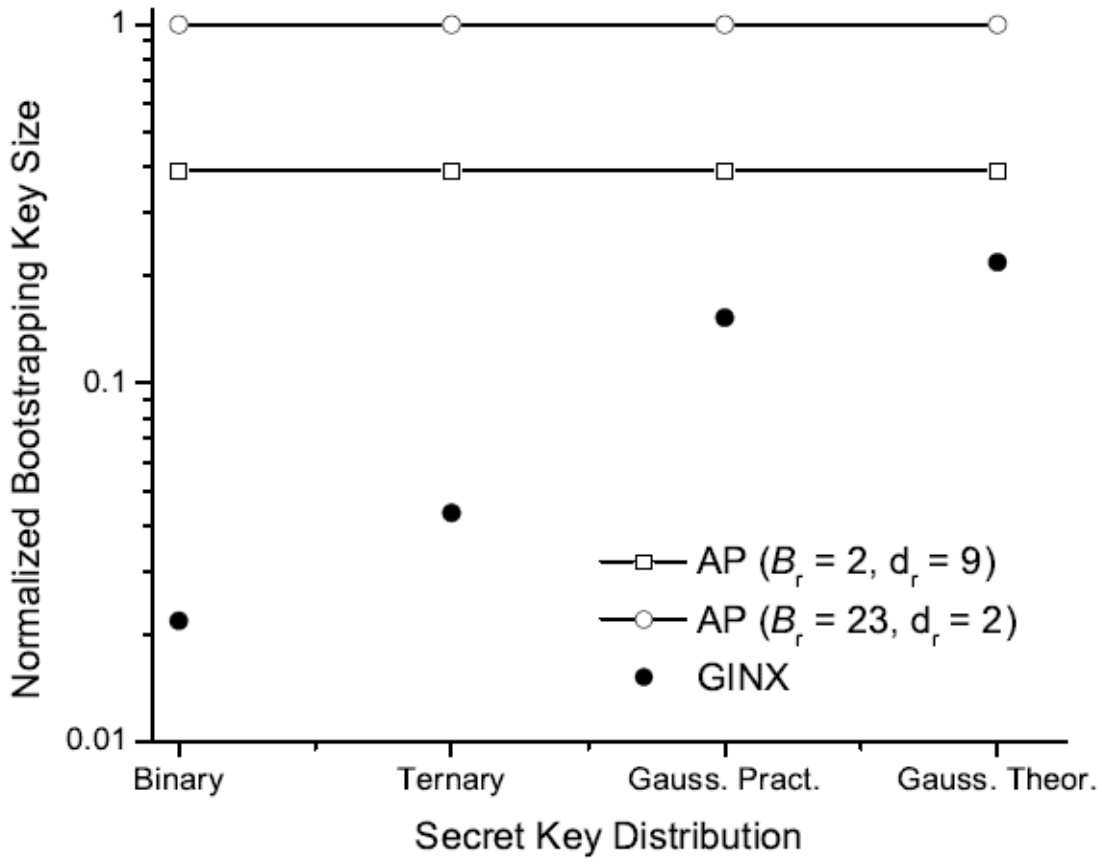
# Runtime complexity

- Comparison of bootstrapping computational complexity of TFHE/GINX and FHEW/AP methods.
- All estimates of computational complexity are normalized to the AP complexity for $B_r$ = 23; $d_r$ = 2; $q$ = 512.
- “Gauss Pract." corresponds to the Gaussian secret key distribution with standard deviation = 3:19.
- “Gauss. Theor." to the case of $\sigma = \sqrt{n}$; both secret key distributions are assumed to be bounded by $12\sigma$.



# Bootstrapping key size

- Comparison of bootstrapping key size for TFHE/GINX and FHEW/AP methods.
- All estimates of computational complexity are normalized to the AP complexity for $B_r$ = 23; $d_r$ = 2; $q$ = 512.
- “Gauss Pract." corresponds to the Gaussian secret key distribution with standard deviation = 3.19.
- “Gauss. Theor." to the case of $\sigma = \sqrt{n}$; both secret key distributions are assumed to be bounded by $12\sigma$.



# Parameter sets

| Parameter Set | $n$ | $q$ | $N$ | $\log_2 Q$ | $B_{\mathrm{ks}}$ | $B_{\mathrm{g}}$ | $B_{\mathrm{r}}$ | $P_{\mathrm{AP}}$ | $P_{\mathrm{GINX}}$ |
|---|---|---|---|---|---|---|---|---|---|
| MEDIUM | 256 | 512 | 1024 | 27 | 25 | $2^9$ | 23 | $2^{-64}$ | $2^{-43}$ |
| STD128 | 512 | 512 | 1024 | 27 | 25 | $2^7$ | 23 | $2^{-53}$ | $2^{-53}$ |
| STD128_AP | 512 | 512 | 1024 | 27 | 25 | $2^9$ | 23 | $2^{-37}$ | $2^{-25}$ |
| STD192 | 512 | 512 | 2048 | 37 | 25 | $2^{13}$ | 23 | $2^{-52}$ | $2^{-52}$ |
| STD256 | 1024 | 1024 | 2048 | 29 | 25 | $2^{10}$ | 32 | $2^{-57}$ | $2^{-33}$ |
| STD128Q | 512 | 512 | 2048 | 50 | 25 | $2^{25}$ | 23 | $2^{-57}$ | $2^{-52}$ |
| STD192Q | 1024 | 1024 | 2048 | 35 | 25 | $2^{12}$ | 32 | $2^{-100}$ | $2^{-100}$ |
| STD256Q | 1024 | 1024 | 2048 | 27 | 25 | $2^7$ | 32 | $2^{-79}$ | $2^{-75}$ |

- Parameter sets for ternary secret distribution; $\mathrm{P}_{AP}$ and $\mathrm{P}_{GINX}$ are estimated upper bounds for decryption failure probabilities of FHEW/AP and TFHE/GINX, respectively

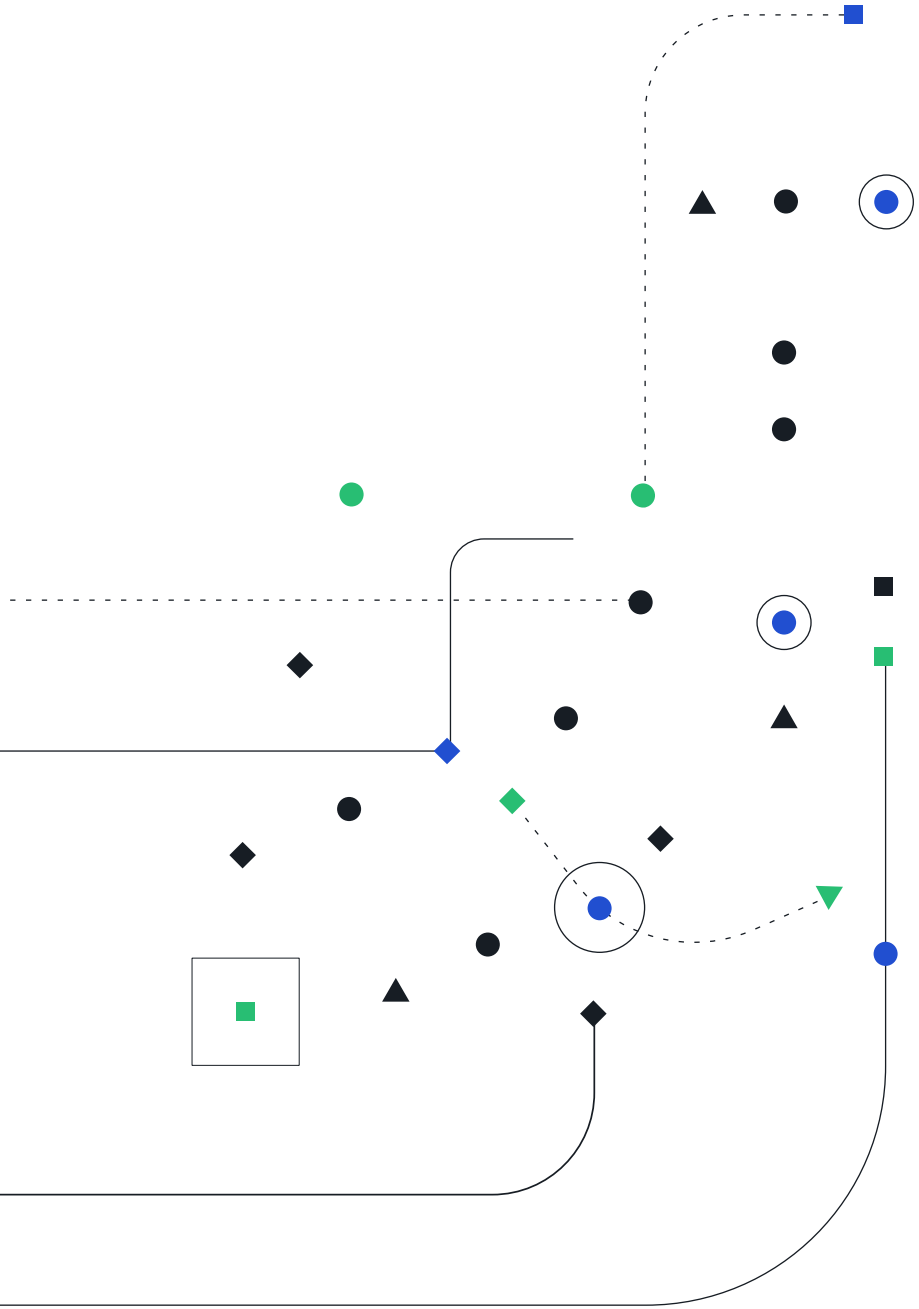
# Experimental results

| Parameter Set | AP [ms] | GINX [ms] | KeySwitch [ms] |
|---|---|---|---|
| MEDIUM | 54 | 57 | 0.98 |
| STD128 | 137 | 143 | 2.1 |
| STD128_AP | 107 | - | 2.1 |

- Implementation was done in PALISADE v1.10.3
- Single-threaded results without AVX acceleration

# Concluding remarks

- We presented a theoretical comparison of the FHEW/AP and TFHE/ GINX cryptosystems for common secret key distributions. Our analysis suggests that the TFHE/GINX cryptosystem is more efficient for binary and ternary secret key distributions while the AP bootstrapping provides better computational complexity for Gaussian secret key distributions. We also provide an opensource implementation of both cryptosystems in PALISADE.
- The implementation presented here did not use AVX acceleration. One of our future goals is to utilize Intel HEXL acceleration to improve the runtime. Intel HEXL is already integrated into PALISADE but has not been optimized for FHEW/TFHE.
- Our analysis suggests that the cost of going from binary to ternary uniform ternary secret distribution is about 3.3x. In other words, the prior best runtime in TFHE is expected to be increased from 13 ms to roughly 43 ms.



Duality

# Q&A

**Yuriy Polyakov**
ypolyakov@dualitytech.com

# References


[AP14] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In CRYPTO 2014, volume 8616 of Lecture Notes in Computer Science, pages 297-314, 2014.

[CGGI16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In ASIACRYPT (1), volume 10031 of Lecture Notes in Computer Science, pages 3-33, 2016.

[CGGI17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In ASIACRYPT 2017, volume 10624 of Lecture Notes in Computer Science, pages 377-408, 2017.

[DM15] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In EUROCRYPT (1), volume 9056 of Lecture Notes in Computer Science, pages 617-640. Springer, 2015.

[GINX16] N. Gama, M. Izabachene, P. Q. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In EUROCRYPT 2016, volume 9666 of Lecture Notes in Computer Science, pages 528-558, 2016.