HE is all you need: Compressing FHE Ciphertexts using Additive HE

Rasoul Akhavan Mahdavi, Abdulrahman Diaa, and Florian Kerschbaum

University of Waterloo, Waterloo, Ontario, Canada

Abstract. Fully Homomorphic Encryption (FHE) permits the evaluation of an arbitrary function on encrypted data. However, FHE ciphertexts, particularly those based on lattice assumptions such as LWE/RLWE are very large compared to the underlying plaintext. Large ciphertexts are hard to communicate over the network and this is an obstacle to the adoption of FHE, particularly for clients with limited bandwidth.
In this work, we propose the first technique to compress ciphertexts sent from the server to the client using an additive encryption scheme with smaller ciphertexts. Using the additive scheme, the client sends auxiliary information to the server which is used to compress the ciphertext. Our evaluation shows up to 95% percent and 97% compression for LWE and RLWE ciphertexts, respectively.

Keywords: Homomorphic Encryption · LWE · RLWE · Compression

1 Background

1.1 Homomorphic Encryption

Homomorphic Encryption is a form of public-key cryptography which permits computation on messages while in encrypted form, without the need to access the secret key. Similar to other public-key cryptosystems, homomorphic ciphertexts are larger than the underlying plaintext. The ratio between the ciphertext and plaintext is denoted as the *expansion factor*.

In a typical protocol using homomorphic encryption, a client encrypts its private input using a homomorphic cryptosystem and sends the resulting ciphertexts to a server. This constitutes the client's *request*. The server computes the desired function over the client's encrypted input. The result is then transmitted back to the client as the *response*. While there are some techniques to reduce the request size, there is no work addressing response size. This work focuses on reducing the response size.

1.2 LWE ciphertexts

For the purpose of this work, we describe a simple version of an encryption system based on the Learning With Errors (LWE) [Reg09] problem which we will denote by $\mathcal{E}_{\text{LWE}}$. Existing schemes such as FHEW [DM15] and CGGI(TFHE) [CGGI20] have ciphertexts of a similar format.

$\mathcal{E}_{\text{LWE}}$ uses the following parameters: dimension n, ciphertext modulus q, plaintext modulus p, $\Delta = \lfloor q/p \rfloor$, a discrete error distribution over $\mathbb{Z}_q$ called χ. We sample the secret key, $\texttt{sk}$, from $\mathbb{Z}_q^n$. The encryption and decryption algorithm is shown below. $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

Algorithm 1 LWEEncrypt$_{\texttt{sk}}$

Input: $\mu \in \mathbb{Z}_p$

1: Sample $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$ and $e \leftarrow \chi$

2: $b = \sum_{i=1}^{n} \mathbf{a}[i] \cdot \texttt{sk}[i] + \Delta \cdot \mu + e \mod q$

Output: $c = (\mathbf{a}, b)$

Algorithm 2 LWEDecrypt$_{\texttt{sk}}$

Input: $c = (\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$

1: $\mu^* = (b - \sum_{i=1}^{n} \mathbf{a}[i] \cdot \texttt{sk}[i]) \mod q$

2: $\mu' = \lfloor \mu^* / \Delta \rceil$

Output: μ'

Compressing Fresh Ciphertexts. Fresh ciphertexts can be compressed to reduce network costs. Since $\mathbf{a}$ is sampled at random, we can send the seed used to generate $\mathbf{a}$ instead of the vector itself. Concretely, instead of sending $ct = (\mathbf{a}, b)$, the client can produce $\bar{ct} = (\theta, b)$ where $\theta \leftarrow \{0,1\}^\lambda$ is the seed of a cryptographically secure PRG used to generate $\mathbf{a}$, i.e., $\mathbf{a} \leftarrow \texttt{PRG}(\theta)$. With this technique, ciphertexts are only $\lambda + \log_2 q$ bits instead of $n \log_2 q$.

1.3 RLWE ciphertexts

Similar to LWE, we can also construct an encryption scheme based on the Ring Learning with Errors Problem (RLWE) [LPR13], which we will denote as $\mathcal{E}_{\text{RLWE}}$. Cryptosystems such as BGV [BGV12], BFV [FV12], and CKKS [CKKS17] have ciphertexts of a similar format. RLWE ciphertexts are useful since they can encrypt a polynomial, i.e. a vector of numbers, instead of just one scalar. For RLWE encryption, we select a dimension N, ciphertext modulus q, plaintext modulus p, and $\Delta = \lfloor q/p \rfloor$. Define $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ and R_p similarly. Moreover, define a discrete error distribution χ over R_q. For key generation, sample $S(X)$ uniformly from R_q.

Algorithm 3 RLWEEncrypt$_{S(X)}$

Input: $\mu(X) \in R_p$

1: Sample $A(X) \xleftarrow{\$} R_q$ and $E(X) \leftarrow \chi$

2: $B(X) = A(X) \cdot S(X) + \Delta \cdot \mu(X) + E(X) \mod R_q$

Output: $C = (A(X), B(X))$

Algorithm 4 RLWEDecrypt$_{S(X)}$

Input: $C = (A(X), B(X)) \in R_q \times R_q$

1: $\mu^*(X) = (B(X) - A(X) \cdot S(X)) \mod R_q$

2: $\mu'(X) = \lfloor \mu^*(X) / \Delta \rceil$

Output: $\mu'(X)$

Compressing Fresh RLWE ciphertexts. Similar to LWE, we can also compress fresh RLWE ciphertexts by sending the seed used to generate $A(X)$ [ACLS18].

Using this technique, the size of a ciphertext can be reduced from $2N \log_2 q$ bits to $\lambda + N \log_2 q$.

2 Additive HE for FHE: Reducing Response Size

Ciphertexts that have been processed by the server can not be compressed using the technique mentioned in Section 1. We propose a technique to compress LWE/RLWE ciphertexts using auxiliary information provided by the client. These techniques apply to cryptosystems that use LWE/RLWE ciphertexts such as TFHE [CGGI20] and BFV [FV12], and BGV [BGV12].

The main insight behind our solution is that the first step of LWE/RLWE decryption is linear in the secret key. Hence, if the client sends encryptions of the bits of the secret key to the server, encrypted under an additive encryption scheme, it can compute the first step of decryption homomorphically and send back only an encrypted scalar to the client. If the additive encryption has small ciphertexts, there is an overall size reduction.

The Additive Encryption Scheme. For the compression protocol, we require an additive encryption scheme which we denote $\mathcal{E}_A$ such that the plaintext space is $\mathbb{Z}_m$, for some m. Specifically, $\mathcal{E}_A$ supports addition and plaintext multiplications. We denote addition and plaintext multiplication with $\oplus$ and $\otimes$, respectively. Moreover, denote the secret key generated by $\mathcal{E}_A$ as s_A and the corresponding encryption and decryption algorithms as $\texttt{AEnc}_{s_A}$ and $\texttt{ADec}_{s_A}$.

Paillier [Pai99] and ElGamal [ElG85] are examples of cryptosystems that can be used for this purpose.

2.1 Compressing LWE Ciphertexts

The ciphertext compression algorithm for LWE and the corresponding modified decryption algorithm is given below.

Algorithm 5 LWECompress

Input:
$\bar{\texttt{sk}}[i] = \texttt{AEnc}_{s_A}(\texttt{sk}[i])$
$c = (\boldsymbol{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}$

1: $x = b \oplus \sum_{i=1}^{n} (q - \mathbf{a}[i]) \otimes \bar{\texttt{sk}}[i]$

Output: x

Algorithm 6 ModifiedLWEDecrypt$_s$

Input: Compressed Ciphertext x

1: $y = \texttt{ADec}_s(x)$
2: $\mu' = \lfloor \frac{y \mod q}{\Delta} \rceil$

Output: $\mu' \in \mathbb{Z}_p$

Theorem 1 (Correctness). *If $m > q + nq^2$, Algorithm 5 produces a compressed ciphertext which decrypts to the correct message using Algorithm 6. More formally,*

$$\text{ModifiedLWEDecrypt}_s(\text{LWECompress}(\bar{\texttt{sk}}, c)) = \text{LWEDecrypt}_{\texttt{sk}}(c) \qquad (1)$$

Proof. In line 1 of Algorithm 5, we calculate $b + \sum_{i=1}^{n}(q - a[i]) \cdot \mathtt{sk}[i]$, encrypted under additive encryption, which is achievable due to linear properties. We know that $\mathtt{sk}[i], a[i]$ and b are elements in $\mathbb{Z}_q$ so $0 \leq \mathtt{sk}[i], a[i], b < q$ and

$$b + \sum_{i=1}^{n}(q - a[i]) \cdot \mathtt{sk}[i] \leq q + \sum_{i=1}^{n} q \cdot q = q + nq^2 < m. \tag{2}$$

So there is no overflow in the plaintext space. In the Modified Decryption (Algorithm 6), we have

$$y \mod q = \mathtt{ADec}_s(x) \mod q \tag{3}$$

$$= \left((b + \sum_{i=1}^{n}(q - a[i]) \cdot \mathtt{sk}[i]) \mod m \right) \mod q \tag{4}$$

$$= \left(b + \sum_{i=1}^{n}(q - a[i]) \cdot \mathtt{sk}[i] \right) \mod q \tag{5}$$

$$= b - \sum_{i=1}^{n} a[i] \cdot \mathtt{sk}[i] \mod q = \mu^* \tag{6}$$

This is identical to μ^* in line 1 of Algorithm 2, hence, since the subsequent steps of Algorithm 2 and Algorithm 6 are similar, they produce the same response, and the theorem is proven.

Security. In Gentry's original construction of a bounded depth encryption scheme, he proposed the idea of using a chain of semantically secure cryptosystems, such that each cryptosystem encrypts the secret key of the next [Gen09]. Gentry proved that if the secret key of each cryptosystem is sampled independently, the composed scheme is also semantically secure.

Let $\mathcal{E}'$ denote the cryptosystem which is the combination of $\mathcal{E}_{\text{LWE}}$ and $\mathcal{E}_A$. The encryption and decryption procedure of $\mathcal{E}'$ is Algorithm 1 and Algorithm 6, respectively. The secret key of $\mathcal{E}'$ is the combination of the secret keys of $\mathcal{E}_{\text{LWE}}$ and $\mathcal{E}_A$. The same holds for the public key as well. Moreover, we also release encryptions of the bits of the secret key of $\mathcal{E}_{\text{LWE}}$ under the secret key of $\mathcal{E}_A$.

Proposition 1 (Security). *If $\mathcal{E}_{LWE}$ and $\mathcal{E}_A$ are semantically secure, then $\mathcal{E}'$ is also semantically secure.*

2.2 Compressing RLWE Ciphertexts

The expansion factor of RLWE ciphertexts does not depend on N and is much smaller than the expansion factor of LWE ciphertexts. Hence, the same technique used in the previous section does not yield an improvement. However, our approach is beneficial if the user is only interested in some coefficients of the plaintext polynomial and not all of them.

The main observation is that each coefficient of $\mu'(X)$ in Algorithm 4 can be calculated separately. Specifically, for $0 \le k \le N-1$

$$\mu'[k] = \lfloor \frac{\mu^*[k]}{\Delta} \rceil = \left\lfloor \frac{B[k] - \sum_{i=0}^{k} A[k-i] \cdot S[i] + \sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i]}{\Delta} \right\rceil \tag{7}$$

Note that the operations in the numerator are happening modulo q. The numerator of Equation (7) is a linear combination of the coefficients of the secret key, hence it can be computed given the encrypted coefficients of the secret key.

The complete procedure to compress the k^{th} coefficient of an RLWE ciphertext is shown in Algorithm 7. The corresponding decryption function is shown in Algorithm 8.

Algorithm 7 RLWECompress

Input:

$\bar{\mathtt{sk}}[i] = \mathtt{AEnc}_{s_A}(S[i])$

$C = (A(X), B(X)) \in R_q \times R_q$

$k \in \{0, 1, \cdots, N-1\}$

1: $x = B[k] \oplus \left(\sum_{i=0}^{k} (q - A[k-i]) \otimes \bar{\mathtt{sk}}[i]\right) \oplus \left(\sum_{i=k+1}^{N-1} A[N+k-i] \otimes \bar{\mathtt{sk}}[i]\right)$

Output: x

Algorithm 8 ModifiedRLWEDecrypt$_s$

Input: Compressed Ciphertext x

1: $y = \mathtt{ADec}_s(x)$

2: $\mu'_k = \lfloor \frac{y \mod q}{\Delta} \rceil$

Output: $\mu'_k \in \mathbb{Z}_p$

Theorem 2 (Correctness). *If $m > q + Nq^2$, Algorithm 7 produces a compressed ciphertext which decrypts to the k^{th} coefficient of the plaintext using Algorithm 8. More formally,*

$$\text{ModifiedRLWEDecrypt}_s(\text{RLWECompress}(\bar{\mathtt{sk}}, c, k)) \tag{8}$$

is equal to the k^{th} coefficient of

$$\mu'(X) = \text{RLWEDecrypt}_{\mathtt{sk}}(c) \tag{9}$$

Proof. Line 1 of Algorithm 7 computes

$$B[k] + \sum_{i=0}^{k} (q - A[k-i]) \cdot S[i] + \sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i]$$

encrypted under additive encryption, which is possible due to the linear properties. We know that all coefficients of $A(X)$, $B(X)$, and $S(X)$ are elements in $\mathbb{Z}_q$, hence

$$B[k] + \left(\sum_{i=0}^{k}(q - A[k-i]) \cdot S[i]\right) + \left(\sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i]\right) \tag{10}$$

$$\leq q + \left(\sum_{i=0}^{k} q \cdot q\right) + \left(\sum_{i=k+1}^{N-1} q \cdot q\right) = q + Nq^2 \leq m \tag{11}$$

so there is no overflow in the plaintext space of the additive cryptosystem.

$$\begin{aligned}
&y \mod q \;=\; \mathtt{ADec}_s(x) \mod q \\
&= \left(\left(B[k] + \sum_{i=0}^{k}(q - A[k-i]) \cdot S[i] + \sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i]\right) \mod m\right) \mod q \\
&= \left(B[k] + \sum_{i=0}^{k}(q - A[k-i]) \cdot S[i] + \sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i]\right) \mod q \\
&= B[k] - \sum_{i=0}^{k} A[k-i] \cdot S[i] + \sum_{i=k+1}^{N-1} A[N+k-i] \cdot S[i] \mod q
\end{aligned}$$

which is equivalent to the k^{th} coefficient of

$$\mu^*(X) = B(X) - A(X) \cdot S(X) \mod R_q$$

which can be seen by expanding the equation. Given that line 2 of Algorithm 4 performs rounding coefficient-wise, it produces the same result as line 1 of Algorithm 8.

Security. A similar argument can be made about the security of compression over RLWE. Let $\mathcal{E}''$ denote the cryptosystem which is the combination of $\mathcal{E}_{RLWE}$ and $\mathcal{E}_A$. The following proposition holds regarding security.

Proposition 2 (Security). *If $\mathcal{E}_{RLWE}$ and $\mathcal{E}_A$ are semantically secure, then $\mathcal{E}''$ is also semantically secure.*

2.3 Batched Compression

If the plaintext space of the additive encryption is large, multiple LWE ciphertexts (encrypted using the same secret key) can be compressed within the same additive ciphertext. Each LWE ciphertext takes up $\log_2(q + nq^2)$ bits of the total bitwidth of the plaintext space. So, if m is the modulus of the plaintext space, then $\lfloor \log_2 m / \log_2(q + nq^2) \rfloor$ LWE ciphertexts can be compressed into one ciphertext from the additive cryptosystem.

Similarly, many coefficients of an RLWE ciphertext can be extracted and compressed into one additive ciphertext. Specifically, since each RLWE coefficient takes up to $\log_2(q + Nq^2)$ of the bitwidth, then $\lfloor \log_2 m / \log_2(q + Nq^2) \rfloor$ coefficients can be extracted simultaneously.

3 Evaluation

To evaluate our technique, we implement the compression and modified decryption algorithm using the Paillier cryptosystem [Pai99] as the additive encryption system. Paillier is an additive homomorphic cryptosystem with semantic security so it satisfies all the requirements for correctness and security. We demonstrate the compression protocol over a prototype implementation and an existing library, OpenFHE.

Prototype Implementation. We implement the two cryptosystems proposed in Section 1.2 and 1.3 in Python and implement compression using the Python-Paillier[1] library [Dat13]. We use a 3072-bit modulus for Paillier which is the recommended modulus size for 128-bit security. We use two parameter sets for LWE and four parameter sets for RLWE. All parameter sets are chosen to satisfy 128-bit security. In the evaluation of LWE compression, one ciphertext is compressed. In the evaluation of RLWE, one coefficient of the RLWE ciphertext is extracted and compressed. The results for LWE and RLWE are shown in Table 1 and Table 2, respectively.

OpenFHE integration. We also integrate this technique into the OpenFHE framework [2] which implements multiple homomorphic cryptosystems such as BGV, CKKS, and CGGI. We apply compression to CGGI ciphertexts which have ciphertext similar to the format described in Section 1.2. Currently, OpenFHE only offers one parameter set for the CGGI cryptosystem which offers 128-bit security. We use the Intel Paillier Cryptosystem Library [3] as the additive cryptosystem, which implement Paillier encryption. The plaintext modulus of this implementation is 2048 bits.

Results. The evaluation shows significant size reduction for all parameter sets. There is at least 86% reduction for the evaluated parameter sets for LWE and up to 95% for CGGI in OpenFHE. Similarly, for RLWE, there is up to 97% size reduction using the compression technique.

Encryption of the LWE/RLWE secret key using Paillier, which is more expensive, is only performed once. Moreover, the encrypted secret key is still small compared to other keys used in FHE, e.g. public keys, key-switching keys and bootstrapping keys, which can be as large as many megabytes.

---

[1] https://github.com/data61/python-paillier
[2] https://github.com/openfheorg/openfhe-development
[3] https://github.com/intel/pailliercryptolib

	LWE (Prototype) $n = 630$ $\log_2 q = 64$	LWE (Prototype) $n = 750$ $\log_2 q = 64$	CGGI (OpenFHE) $n = 1305$ $\log_2 q = 11$
Encrypt Secret Key (Time)	28 s	33 s	0.15 s
Encrypted Secret Key	483 KB	575 KB	669 KB
Ciphertext Compression Time	0.67 s	0.79 s	0.31 s
Compressed Ciphertext Size	768 B	768 B	525 B
Uncompressed Ciphertext Size	5 KB	6 KB	10.25 KB
Size Reduction	**86 %**	**87.2 %**	**95.0%**

Table 1. Evaluation of the ciphertext compression technique for a single LWE ciphertext. Two sample parameter sets are chosen for prototype LWE. We use the STD128 configuration for CGGI in OpenFHE.

	RLWE (Prototype) $N = 1024$ $\log_2 q = 27$	$N = 2048$ $\log_2 q = 54$	$N = 4096$ $\log_2 q = 36$	$N = 8192$ $\log_2 q = 43$
Encrypt Secret Key (Time)	45 s	90 s	182 s	369 s
Encrypted Secret Key	786 KB	1572 KB	3145 KB	6290 KB
Ciphertext Compression Time	0.50 s	1.77 s	2.52 s	5.97 s
Compressed Ciphertext Size	767 B	767 B	767 B	767 B
Uncompressed Ciphertext Size	2.5 KB	5.6 KB	12.3 KB	26.6 KB
Size Reduction	**70.0%**	**86.36%**	**93.75%**	**97.11%**

Table 2. Evaluation of the ciphertext compression technique for a single RLWE coefficient. Four sample parameter sets are chosen for the prototype RLWE implementation.

4 Discussion

4.1 Related work

Our work is the first to address large ciphertext sizes with this particular technique. However, there exist relevant concepts in the literature. Below, we describe these concepts and how they can relate to this work.

Scheme Switching. Scheme switching has previously been proposed in the literature but was not used as an approach to reduce the size of ciphertexts.

Gentry et al. use scheme switching as an alternative to squashing the decryption circuit in the bootstrapping process [GH11]. They switch the scheme to ElGamal which is a multiplicative encryption scheme.

Boura et al. propose scheme switching as a method to benefit from the features of many cryptosystems and not be confined to one [BGGJ20]. They provide procedures to switch between many lattice-based schemes such as BFV, TFHE, and CKKS. All of the schemes that they switch between are based on the LWE and RLWE hardness assumptions.

Modulus Switching. Modulus switching is a technique used for many purposes. First, it can be used as a method to limit noise growth in lattice-based homomorphic schemes. This technique was first proposed by Brakerski and Vaikuntanathan [BV14] and subsequently used by Brakerski et al. to construct the BGV cryptosystem [BGV12].

In later work, this technique was frequently used as a method to reduce the size of ciphertexts before they are communicated back to the client. Using our notation from Section 1.3, this technique results in a smaller q. However, this technique is limited by the fact that the size of the ciphertext is still linear in N, which significantly impacts the size of the ciphertext.

RLWE Coefficient Extraction. TFHE [CGGI20] supports coefficient extraction over RLWE ciphertexts. This allows the server to extract one coefficient of the underlying plaintext of an RLWE ciphertext as an LWE ciphertext. This technique is useful for the purpose of bootstrapping. It has not been proposed as a method for size reduction.

Coefficient extraction can be used in combination with our techniques. For example, the desired coefficient in an RLWE ciphertext can be extracted into an LWE ciphertext, and then LWE compression can be used. However, using compression over RLWE ciphertexts, we can directly compress the RLWE coefficient without the need to initially perform coefficient extraction.

4.2 Applications.

Large ciphertext sizes are an obstacle in the practical deployment of many applications using FHE. Here we identify two specific use cases of FHE where our compression techniques can be applied.

Filters over Encrypted Images. The compression of LWE ciphertexts is suitable for applications where the output is large, such as applying a filter on an encrypted image. For example, Signal offers a tool to blur faces for the privacy and safety of individuals in images [sig]. Currently, this feature runs locally on the client's device. However, to reduce overhead on small devices, the client can send the encrypted image to the server to process and receive the response. This situation is a suitable fit for FHE since the client device can establish cryptographic keys with the server and reuse them many times.

Applying filters over encrypted images has also been implemented using the Concrete library [con]. They encode the image as an array of LWE ciphertexts, which makes it ideal for compression using our proposed algorithm.

Private Data Analysis. Circuit-PSI and Private Decision Tree Evaluation are two examples of data analysis over private data.

In Circuit-PSI, the server computes a function of the intersection of the client encrypted set with its own private set. In the protocol proposed by Kacsmar et al. [KKL+20], the client's set is encrypted as an RLWE ciphertext. The response, however, is small and only occupies one coefficient slot. This is an ideal application of the compression technique over RLWE ciphertexts.

In Private Decision Tree Evaluation, the server evaluates a decision tree over the client's private, encrypted input. Cong et al. [CDPP22] propose a protocol which encrypts the client's input as multiple LWE ciphertexts and returns the result of the classification as an LWE ciphertext. Compression can be applied to this application as well.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*, pages 962–979. IEEE, 2018.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[CDPP22] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V.L. Pereira. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 563–577, New York, NY, USA, 2022. Association for Computing Machinery.

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

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[con] Encrypted Image Filtering Using Homomorphic Encryption. `https://www.zama.ai/post/encrypted-image-filtering-using-homomorphic-encryption`. Accessed: 2023-03-14.

[Dat13] CSIRO's Data61. Python Paillier Library. `https://github.com/data61/python-paillier`, 2013.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications* *of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pages 617–640. Springer, 2015.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[GH11] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 107–109, 2011.

[KKL+20] Bailey Kacsmar, Basit Khurram, Nils Lukas, Alexander Norton, Masoumeh Shafieinejad, Zhiwei Shang, Yaser Baseri, Maryam Sepehri, Simon Oya, and Florian Kerschbaum. Differentially Private Two-Party Set Operations. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 390–404, 2020.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Reg09] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM*, 56(6), sep 2009.

[sig] Blur tools for Signal. `https://signal.org/blog/blur-tools/`. Accessed: 2023-03-14.