ISA Journal of Engineering and Technology (ISAJET)

Homepage: https://isapublisher.com/isajet/

Volume 3, Issue 6, Nov-Dec, 2026 ISSN: 3049-1843

Serialization and Transmission of CKKS Ciphertexts over Inter Integrated Circuit (I2C) Bus

Ame Funmilayo Udobong; Benjamin Okike; Olumide Owolabi & Hashim Bisallah

Independent researchers

Received: *10.09.2026* **|** ***Accepted:*** *03.11.2026* **|** ***Published:*** *07.11.2026*

***Corresponding Author:** Ame Funmilayo Udobong

DOI: *10.5281/zenodo.24030267*

Abstract **Original Research Article**

Data in use is primarily stored in volatile memory, such as random access memory (RAM), memory caches, and registers when they have been decrypted for processing. These data are vulnerable to exploitation or encryption keys being stolen from the devices. In most traditional architectures, the central processing unit (CPU) and main memory work together because they share a common communication bus, so the same data moves within their circuits. The internal bus channel connects all the internal components of a computer, such as the registers, CPU and memory, to the motherboard. Although encryption can protect data during transmission, most conventional cryptographic schemes will required that the ciphertext is decrypted before processing of data. Fully homomorphic encryption (FHE) provides an alternative by allowing computation on encrypted data. This study investigates the process of serializing and transmitting FHE ciphertexts through I2C channels on resource constrained microcontrollers.

Keywords: Fully Homomorphic Encryption, OpenFHE, CKKS, AES, I2C, STM32F411CEU6, Embedded systems, IoT Security.

Copyright © 2026 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

1. Introduction

Fully homomorphic encryption (FHE) is an encryption algorithm that evaluates ciphertexts, ensuring that data stays private and confidential. Because of this ability to compute on encrypted data, FHE is used for many privacy preserving applications where information must remain confidential throughout its lifecycle. CKKS is an FHE scheme that processes real numbers by the expansion of the scaling mode to convert these floating point numbers to integers. The CKKS scheme supports the addition and multiplication of ciphertexts, ciphertexts and plaintexts, or ciphertext and scalar values, as well as rotation operations. CKKS is a widely used FHE scheme built on the BGV scheme for FHE with integers. It uses both symmetric and asymmetric encryption getting its quantum resistant security from the ring learning with error (RLWE) problem which is a specialization of the learning with errors problem for polynomial rings over finite fields.

The challenge however that is because of the polynomial property of FHE, its ciphertext is significantly larger than traditional encryption

schemes such as AES. Consequently, integrating FHE in constrained embedded systems such as an STM32F411CEU6 microcontroller remains difficult. Notwithstanding, we would still explore practical ways of transmitting CKKS ciphertext over I2C communication bus on a microcontroller.

2. FHE Cryptography and Libraries

In 2009, Gentry constructed the first FHE scheme in his PhD dissertation "Fully homomorphic encryption using ideal lattices" (Gentry, 2009). Gentry introduced the bootstrapping process, which can evaluate an unlimited number of computations an unlimited number of times. In 2012 Brakerski et al. developed a similar scheme, but based security entirely on learning with errors (LWE), avoiding reliance on ideal lattices. A follow up paper in 2012 by Brakerski and Vaikuntanathan led to the introduction of a different type of homomorphic encryption called the leveled FHE (LFHE). This method described a new way of constructing leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), without Gentry's bootstrapping procedure (Brakerski et al, 2014). LFHE supports a limited number of operations (levels) on the ciphertext before the ciphertext becomes too noisy and too difficult to decrypt. Cheon, et al (2016) introduced a variant of the BGV scheme, an homomorphic encryption for arithmetic approximate numbers. This scheme uses a scaling mode to convert real numbers to integers before encryption. The Cheon Kim Kim Song (CKKS) HE scheme, which enables approximate arithmetic on real or complex numbers demonstrated its effectiveness in enabling the truncation of encrypted values, batch computation, and user defined settings. As a result, many studies have investigated the application of the CKKS HE scheme for empirical use cases (Lee et al., 2023).

The introduction of FHE libraries that incorporate a lot of FHE schemes has made it possible for noncryptographers to use FHE and encourage its applications in areas such as machine learning, Internet of things (IoT), and cloud computing. Examples of these libraries are PALISADE, HElib, Microsoft SEAL, Lattigo and OpenFHE.

3. Research Contributions

This paper provides a practical architecture for transporting CKKS serialized ciphertext from a Windows 10 computer through I2C to an STM32F411CEU6 microcontroller. Majority of existing researches focuses on transmission over other channels such as Wi-Fi, Bluetooth and RFID or using GPUs or Hardware acceleration to improve the latency of communication. OpenFHE I2C bus level transmission is fairly unexplored. We also reported our findings based on the speed and size of the packets as we compared to AES encryption moving over the same channel.

4. Research Objectives

The objectives of this study are

1. To configure the OpenFHE development environment that will generate the ciphertext and serialize it.
2. To develop a method of transmitting the CKKS ciphertext through the I2C communication bus.
3. To establish a communication path between the Windows host and the STM32F411CEU6 MCU using an FT232H bridge.

5. The Architecture

Our architecture consist of four major components

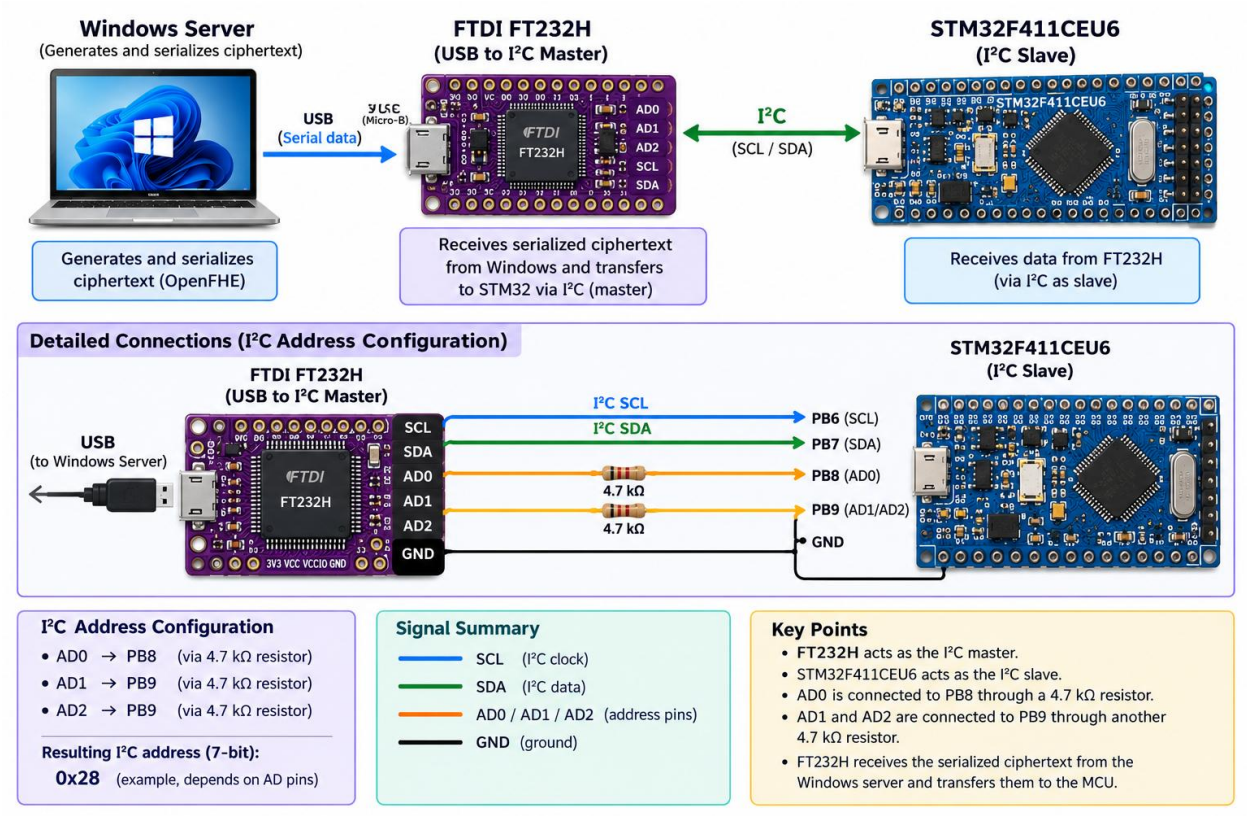


Figure 1: Data transmission path and device connection architecture

5.1 The Embedded Device

The STM32F411xC chip uses multiple Advanced High performance Bus (AHB) matrix that share traffic across master interfaces, such as the CPU, and slave peripherals. STM32F411CEU6 has 33 operational GPIO pins, three I2C ports, three USART ports, five SPI connections, and a full speed USB 2.0 controller (STMicroelectronics). The STM32 MCU would act as the I2C slave and will receive the ciphertext, perform the integrity verification on the received ciphers and store the ciphers in its memory.

5.2 I2C Communication Bus

I2C is a bidirectional bus used to handle data transmissions between peripherals, originally designed for short distance communications within a single device. The I2C protocol employs a simple bidirectional communication model in which only two wires are required to connect all devices on the bus (Valdez et al, 2015). The two wires are the Serial Data Line (SDA) and the Serial Clock Line (SCL), each carrying a different type of data; the SDA line is a bidirectional line that carries the data between the primary and secondary, and the SCL line is a single directional line that carries the clock signal from the Master device. The Master initiates a "start" by leaving SCL high and pulling SDA low and then sending a 7 bit target (slave) address, which is communicated to all targets connected to the BUS followed by a write or read bit. The I2C slave device receive data and stores it.

5.3 Communication Bridge

Because the Windows server does not provide native low level I2C access, we are using an FT232H USB to I2C as our communication bridge. The FT232H is

configured as a USB Serial converter to act as an I2C master device that communicates with the STM32F411CEU6 slave device. To connect the slave to the master, we connected the FT232H AD0 pin to STM32F411CEU6 GPIO PIN PB8 through a 4.7Ω resistor while AD1 and AD2 were connected to pin PB9 through another 4.7Ω resistor. FT232H receives the serialized ciphertext from the Windows server and transfers them to the MCU.

5.4 Host Server

The Windows 10 workstation acts as the edge server for these operations. It executes the OpenFHE library and performs the

- CKKS Parameter selection
    - Ring Dimension
    - Multiplicative Depth
    - ScalingMod
    - Batch Size
- Key generation
- Plaintext encoding
- Ciphertext generation
- Ciphertext serialization

OpenFHE is an open source cryptographic software library that provides post quantum tools for FHE (openfhe.org).

6. Related Works

Kumar et al (2025), aimed to implement a secure data transfer mechanism for the I2C protocol using Verilog. They demonstrated that the secure I2C protocol operates efficiently on the Zybo Z7 board that integrates an ARM Cortex A9 processor with an FPGA, with minimal impact on system performance.

Cheon et al (2026) addressed how CKKS encryption produces large ciphertexts and keys which degrade the performance of embedded clients. They analyzed techniques which minimized the keysize. They proposed key management systems called KG+ and BTS+, which reduce the client's FHE costs. The KG+ system significantly reduces the key size without compromising the efficiency of homomorphic computation.

Dhinakaran et al (2024), discussed challenges in privacy-preserving techniques for IoT-based cloud systems. They identified scalability issues, the dynamic nature of IoT environments, and privacy preserving AI integration as open challenges for IoT. Their research was limited to only the survey; they did not suggest which scheme would be more efficient and reliable based on their architecture.

7. Methodology

To set up the OpenFHE library project environment on a Windows 10 machine, we would need to follow the steps below

1. Download and install the most current version of MSYS2
2. Run the mingw-w64 toolchain library.
3. Download, install, and set up CMAKE and make
4. Download and install git. Clone the OpenFHE library from GitHub.
5. Set up the directory structure on Windows 10 and build for a test run.
6. Build with CMake by editing a CMakeLists.txt for OpenFHE

To send the FHE ciphertext, it first has to be serialized to transform it into a series of bits that can move through the bus. To serialize FHE you use the OpenFHE serial namespace that serializes to a binary format. We extracted the raw ciphertexts elements, serialized them into binary streams before converting them into byte vectors.

Listing 1.0: Method of serializing CKKS Ciphertexts

```
// Storing the ciphers
auto ciphertextElements = c1->GetElements();


 // Serializing ciphertext to a stream
    std::stringstream ss;
    Serial::Serialize(ciphertextElements, ss, SerType::BINARY);


    // Getting the raw bytes for transmission
    std::string serializedStr = ss.str();
    std::vector<uint8_t> serializedData(serializedStr.begin(), serializedStr.end());
```

The serialized ciphertext were divided into manageable byte sizes for transmission with each byte prefixed with a header field stating the maximum size of the data. At the receiving end, the size is calculated to check for the completeness of the data.

On the FT232H side we used the Multi-Protocol Synchronous Serial Engine (MPSSE) which allows FTDI chips to communicate with the I2C buses on microcontrollers using the D2XX USB drivers. The libMPSSE library provided an API for I2C connection. The FT232H receives the serialized ciphertext from the computer and in MPSSE mode, acts as an I2C master, sending I2C commands to the STM32F411CEU6. The STM32F411CEU6 will respond with an ACK for every data byte received.

Below is a subset of the collected data, including the parameters, time taken and the ciphertext size, while monitoring how well the I2C bus handles packet movement.

Table 1: Output of ciphertext transmission

SN	Parameters				Total Bytes	Time Elapsed (ms)
	Ring Dimension	Multiplicative Depth	Scaling Mod	Batch Size		
1	4	1	20	1	855	13254
2	16	3	30	1	1825	28339
3	32	5	30	1	4133	64532
4	64	5	50	2	7221	112749
5	128	5	59	4	13381	209263
6	512	1	30	2	17541	274970847

7	1024	1	30	2	33925	534980873
8	2048	1	30	2	66693	1051453113
9	4096	1	30	2	132229	2084551731
10	8192	1	30	2	263301	4138500776

From the data we can see that as the ring dimension increases in size, the ciphertext increases as well.

We recreated Kumar's AES 256 encryption algorithm and encrypted our existing sized data payload (that is payload sizes we had earlier gathered from our previous CKKS encryption). We executed the AES 256 program and serialized the bits to move from the Windows 10 computer across the physical bus interface through the native D2XX MPSSE drivers of the FT232H bridge. The serialized ciphers were then pushed on the I2C to the MCU's RAM.

We measured the time in milliseconds from the first I2C START condition to the final STOP condition for each transmission and documented the time taken for transmitting each sized data. Finally, we compared the ratio of the latency of the AES ciphertext to that of the CKKS ciphertexts. As we obviously expected the time was significantly longer for FHE transmission because of polynomial expansion of the CKKS ciphertext.

Table 2: Performance analysis to compare AES 256 with CKKS transmission.

Ring Dimension	**Time take**	**Kummar's Time taken (ms)**	**Ratio of Time taken between AES and CKKS**	**Actual Byte Size**	**Total I2C Packet size (IV + Ciphertext)**
512	274970847	1919.8733	143229:1	17541	17568
1024	534980873	3503.7512	152689:1	33925	33952
2048	1051453113	7289.8390	144236:1	66693	66720
4096	2084551731	14323.8759	145529:1	132229	132256
8192	4138500776	Transmission failed due to buffer constraint	-	263301	263328

At a dimension of N=4096, the homomorphic workflow requires 132,229 bytes of serialized data and consumes 35 minutes of aggregate processing time, completely exceeding the physical 128 KB SRAM size of the STM32F411CEU6 microcontrollers.

8. Discussion

The research demonstrates that though OpenFHE cannot execute directly on Cortex-M microcontrollers, practical integration with embedded systems remains possible through a host

based assisted architecture.

We observed that CKKS ciphertexts is significantly larger and slower than conventional encrypted messages. That while transmitting an equivalent volume of data encrypted using AES 256 symmetric cryptography across the physical 100 KHz I2C bus takes only 1,192.5 milliseconds (1.19 seconds) that of CKKS takes 274970847milliseconds. This presents a clear cryptographic design boundary showing that while the AES algorithms offers an optimization improvement of about 1,700 times faster throughput, it requires the target microcontroller to be completely trusted with private keys. However for the CKKS algorithm, despite its significant cost in transmission latency and memory usage, it completely removes the need for edge layer trust. We also saw that resource limitations prevented direct execution of homomorphic operations on STM32 devices and that serialization provides an effective mechanism for moving ciphertexts between the heterogeneous platforms. The architecture therefore shifts computationally intensive homomorphic operations to a more capable host platform while maintaining compatibility with embedded devices.

9. Limitations

The study is subject to several limitations:

1. OpenFHE cannot execute directly on Cortex-M microcontrollers.
2. CKKS ciphertexts remain relatively large and requires reasonable transmission time to move across a limited I2C channel.
3. The size of the ring dimension plays a major role in determining the size of the ciphertext hence the speed of the communication.

10. Conclusion

This research investigated the feasibility of transmitting Fully Homomorphic Encryption ciphertexts through an I2C communication channel in resource constrained embedded systems.

A CKKS based framework using OpenFHE was implemented on a Windows host platform and connected to an STM32F411CEU6 microcontroller through an FT232H USB to I2C bridge. The study demonstrated the successful generation, serialization, and transmission of ciphertext data while validating communication through protocol level monitoring.

Although direct execution of OpenFHE on Cortex-M microcontrollers remains impractical due to resource limitations, the proposed architecture provides a realistic pathway for integrating homomorphic encryption with embedded communication systems.

Future work will focus on reliable slave mode operation, optimized ciphertext transmission techniques, support for additional communication buses such as SPI and UART, and integration with monitoring devices to identify transmission weaknesses.

11. References

Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2012). (Leveled) fully homomorphic encryption without bootstrapping. ITCS '12: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. Pages 309 – 325. https://doi.org/10.1145/2090236.2090262

Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2014). (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), 6(3), 1-36. https://doi.org/10.1145/2632661

Cheon, J.H., Kim, A. , Kim, M. & Song, Y. (2016). Homomorphic Encryption for Arithmetic of Approximate Numbers. Cryptology {ePrint} Archive Paper. https://eprint.iacr.org/2016/421

Cheon, J. H., Kang, M., & Park, J. H. (2026). Towards lightweight CKKS: On client cost efficiency. In Proceedings of the 2026 ACM Asia Conference on Computer and Communications Security (ASIA CCS '26) (pp. 145–158). ACM.

https://doi.org/10.1145/3909896.3910012

Dhinakaran, D., Udhaya Sankar, S.M., Selvaraj, D., & Edwin Raja, S. (2024). Privacy-Preserving Data in IoT-based Cloud Systems: A Comprehensive Survey with AI Integration. https://doi.org/10.1007/s12652-024-04781-9

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. PhD thesis. https://doi.org/10.1145/1536414.1536440

Kumar , V. & Das K.S. (2025). Enhancing security in IIoT: RFID authentication protocol for edge computing and blockchain-enabled supply chain. Elsevier B.V. on behalf of KeAi Communications Co., Ltd. https://doi.org/10.1016/j.csa.2025.100087.
CC BY license (http://creativecommons.org/licenses/by/4.0/)
Lee, J., Duong, P. N., & Lee, H. (2023). Configurable Encryption and Decryption Architectures for CKKS-Based Homomorphic Encryption. *Sensors*, *23*(17), 7389. https://doi.org/10.3390/s23177389

OpenFHE. (n.d.) https://openfhe-development.readthedocs.io/en/latest/ (accessed 07/06/2025)

STMicroelectronics. (n.d.). STM32G0 Peripheral USART interface. https://www.st.com/resource/en/product_training/STM32G0-Peripheral-USART-interface-USART.pdf.(accessed 13/06/2025)