Modern file transfer protocol using lossless compression, lattice-based encryption, and a data integrity hashing function

Vince Jerald A. Villamora¹, Marvin S. Lim², and Archival J. Sebial³
University of San Carlos, Nasipit, Talamban, Cebu City, Cebu, Philippines

¹vincevillamora@gmail.com
²mrvn_lm@yahoo.com
³ajsebial@usc.edu.ph

Abstract. With the recent advances in technology, the creation and access to voluminous amounts of data has become more and more common. From consumer computers and smartphones to big data, the transfer of data entailed a longer download and upload time as the size gets bigger, thus taking a significant amount of time for data to reach its destination. Moreover, data can be corrupted as it is being transferred over the network and most file transfer protocols do not have a file integrity check. Lastly, most file transfer protocols employ the RSA cryptosystem which can be broken through the use of a quantum algorithm known as Shor’s algorithm. The proponents have implemented a file transfer protocol that integrates file hashing, compression, and a quantum-resilient cryptosystem. The results show that the proposed file transfer protocol is faster than plain FTP and even faster against FTP over TLS.

1. Introduction
With the advent of the digital age, there has been an exponential growth in the number of internet users in the world. Along with this is the emergence of cloud technologies such as cloud storage, Software as a Service (SAAS), hosting, cloud-based databases, content delivery networks, and big data analytics. Data has become increasingly more common. Anyone with a smartphone or a computer can create data or content such as a photo, a video, or a message which can be transferred or shared over the internet. This transfer of data utilizes a file transfer protocol which are anything but fast, secure, and reliable.

Lossless compression is a type of data compression where a data is compressed without any loss or degradation from the original uncompressed data [1]. Compression is a possible solution for a faster file transfer protocol. One such compression algorithm is called the Zstandard Compression, also known as “Zstd”. It is a lossless compression algorithm that employs asymmetric numeral systems, an entropy coding method 30 times faster than other methods. Compressing the file prior to sending reduces transfer time by sending less bytes over the network. An appropriate lossless compression algorithm applied to an internet protocol must not saturate CPU time so it would not bottleneck the throughput.

A data’s integrity may be compromised as it is being sent over the network. One high-speed hashing algorithm being used is called xxHash. It is a hashing algorithm that works at speeds close to RAM limits. This hashing algorithm verifies the file as it is successfully transmitted over the network.
TCP/IP (Transmission Control Protocol / Internet Protocol) is a group of protocols that allows a collection of computers to communicate and exchange data within a network in general, and the Internet in particular [2]. A protocol is a set of rules that allows a computer to connect with one another, usually in the form of time, sequence during data transmission [3]. Computers are able to connect to the internet and communicate because of this protocol.

Encryption is the process of encoding a message or data in such a way that only authorized parties can access it [4]. Most file transfer protocols rely on three core cryptographic functionalities to protect data: public key encryption, digital signatures, and key exchange [5]. Peter Shor of Bell Laboratories showed that quantum computers can efficiently solve these problems, rendering most public key cryptosystems ineffective [5]. Quantum computers pose a great risk to modern cryptosystems: being able to break current cryptosystems in use, thus compromising data security and integrity. One quantum-resilient cryptosystem is a lattice-based encryption algorithm that is an alternative to RSA (Rivest-Shamir-Adleman) and ECC (Elliptic-curve cryptography) which relies on lattice-based constructions that cannot be solved efficiently even with the use of quantum computers. In time, quantum computers will supersede classical computers and as long as old protocols are not keeping up, data confidentiality is in great peril.

The proponents have confidence that the methods stated above are crucial features for modern file transfer protocols. Reliability, performance, and security have never been more important. The proponents hope to pave the way for the widespread adoption of lattice-based encryption, real-time compression, and hashing.

2. Related Works

2.1. Network Protocol
Packet loss and performance degradation are effects of packet corruption in data center networks. To try to mitigate packet corruption, a research observed hundreds of thousands of links across data centers [6]. The study concluded that there is a significant loss of corruption in packets in data center networks and that packet corruption can be mitigated through the simple act of cleaning the connectors or replacing the cables.

2.2. Compression
The study of File Management System for P2P Environment using Lossless Compression Algorithm aimed to reduce file transfer time by applying a lossless compression on the fly as a file is being sent through the network [7]. The research showed that even a zip compression is sufficient enough for an improved file transfer speed and that the newly-developed file management system provided better file accountability and management.

2.3. Encryption
Regardless of the exact time of the arrival of the quantum computing era, information security systems as early as now should be able to resist quantum computing [5]. A study was conducted on lattice-based cryptography using a four-degree polynomial over z [8]. The research concluded that without sacrificing security, more speed can still be achieved in the encryption and decryption process.

2.4. Hashing
Storage, server, networking, database, and more are the services delivered by cloud computing over the internet and provides virtualization scalability, and fine-grained data access control and performance benefit. Despite the benefits of cloud computing, there are various issues of security and data integrity for the cloud [9]. Adding a generic hash function to determine the integrity of the data being transferred over the network ensures the data transmitted is corruption-free because TCP alone already exhibits a high amount of error rate [10].
3. Methodology
The flow of implementation in Figure 1 shows the key exchange and dictionary initialization.

**Figure 1. Initial handshake**
A detailed description of Figure 1 from top to bottom:
1. The sending end generates a dictionary based on the data type which tunes the algorithm for the data being sent. This results to a faster compression and decompression, also increasing the compression ratio.
2. The server generates the lattice-based encryption public key and sends it to the client.
3. The client generates the AES (Advanced Encryption Standard) key and encrypts it with the server’s public key and sends that encrypted key back to the server.

**Figure 2. Key exchange and connection establishment.**
A detailed description of Figure 2 from top to bottom:
1. The connection tunnel is being encrypted by the AES key.
2. The AES encrypted payload is appended with an IV (initialization vector) encrypted with the server’s public key. This approach is a hybrid encryption of lattice-based encryption and AES since the payload needs the IV along with the AES key in order for it to be decrypted.
3. The generated dictionary and other files being sent are encrypted on the methods described above.

3.1. Encryption
Lattice-based cryptography is an encryption and decryption algorithm that involves lattices. This cryptography arose from the idea that factoring large numbers or finding the discrete logarithm is mathematically proven to be easily deciphered by quantum-based computers with the help of Shor’s algorithm [11].

1. Generate a random arbitrary matrix $A$ with size $m \times n$ satisfying the definition of a lattice.
2. Generate an arbitrary binary $\vec{x}$ with size $m$ to be used as public key.
3. To get the private key, matrix $A$ is multiplied to $\vec{x}$ to obtain vector $\vec{u}$.
4. Encrypting the data requires the sending end to encapsulate the data in $\overline{b}_1$ and $\overline{b}_2$ wherein $\overline{b}_1 = A \times \vec{s} + \vec{e}_1$ where $\vec{s}$ is an arbitrary with size $m$ and $\vec{e}_1$ with the same size but the magnitude of its numbers must be smaller than the numbers found in $\vec{s}$ and $\overline{b}_2 = \vec{s} \times \vec{u} +$
$e_2 + bit \times \frac{9}{2}$, where $bit$ is either 0 or 1, the data that’s going to be encrypted and $e_2$ is any arbitrary value less than $\frac{9}{4}$.

5. To decrypt the data, the following equation is used $b_2 = \overrightarrow{b_1} \times \vec{x}$, the resulting number would either be close to 0 or 1.

3.2. Hashing

xxHash is general purpose hashing function. It has high Memory bandwidth usage and works closely to the speed of RAM [12]. The following is the 32-bit implementation algorithm of the hashing function:

1. Initialize internal accumulator based on optional seed input.
2. Process stripes, in which a stripe is a contiguous segment of 16 bytes evenly divided into 4 lanes of 4 bytes each. Each lane read is associated to its 32-bit value using little-endian convention.
3. For each \{lane, accumulator\}, the bits are shuffled so that any bit from input lane impacts several bits in output accumulator. Input is consumed one full stripe at a time. Step 2 is looped as many times as necessary to consume the whole input, except the last remaining bytes which cannot form a stripe.
4. All 4 lane accumulators from the previous steps are merged to produce a single remaining accumulator of same width (32-bit).
5. Adding an input length so that it participates to the final mixing.
6. Remaining inputs are consumed, which ensures that all input bytes are present in the final mix.
7. Final mix. The final mix ensures that all input bits have a chance to impact any bit in the output digest, resulting in an unbiased distribution.
8. The hashing function produces an unsigned 32-bit value as output.

3.3. Compression

The Zstandard compression algorithm is a modified LZ77 lossless compression algorithm that is designed for fast compression and decompression speed while maintaining a high compression ratio by tuning the compression algorithm to a specific data set or type [13]. A LZ77 algorithm implements compression through storing references of repetitive bits of data in a dictionary [14]. The reference to that data is stored as two integers in memory occupying 4 bytes each, one for the number of jumps in the file to reach the referred data and the other as the length of the data that is being referenced to. The resulting output is a series of literals (original bytes) and length/distance pairs (references). Figure 3 demonstrates the algorithm explained above and its output.

![LZ77 lossless compression technique](image)

To test the performance of the proposed protocol, the researchers opted to test using the Windows operating system and write the protocol in the Java programming language and with the critical parts of the protocol written in Java Native Interface. There are ten scenarios with different file types and sizes.
Compression ratio is calculated by the formula $\text{compression ratio} = \frac{\text{size after compression}}{\text{size before compression}}$. The compression ratio in percent is the inverse of the compression ratio formula. The lower the percentage, the better.

Space savings is calculated by the formula $\text{space savings} = 1 - \frac{\text{compressed size}}{\text{uncompressed size}}$ [15].

4. Results and Discussion

The following tables show the results of the conducted tests. Table 1 shows the difference in bytes using the file transfer protocol.

| Scenarios | File Type | No. of Files | Actual Bytes Read (MB) | Bytes Transferred Over the Network (MB) | Difference (Bytes) | Difference (MB) | Space Savings (Percent) | Compression Ratio | Compression Ratio (Percent) |
|-----------|-----------|--------------|------------------------|-----------------------------------------|-------------------|-----------------|------------------------|----------------|-----------------------------|
| Video     | AVI       | 85           | 7,252.25               | 7,193.85                               | 58.40             | 0.07            | 97.92                  | 1.02           | 97.92                       |
| Audio     | WAV       | 800          | 13,828.615             | 10,382.88                              | 3,445.729         | 24.91           | 75.09                  | 1.33           | 75.09                       |
| Photos    | PNG, JPEG | 3,500        | 1,455.31               | 1,438.73                               | 16.58             | 1.14            | 98.86                  | 1.01           | 98.86                       |
| Text      | TXT       | 2,688        | 1,217.35               | 268.84                                 | 948.984           | 77.92           | 92.08                  | 4.53           | 92.08                       |
| Executables | EXE     | 760          | 2,811.26               | 2,709.06                               | 102.19            | 3.64            | 99.36                  | 1.04           | 99.36                       |
| Mixed     | AVI, WAV, PNG | 500   | 6,612.54               | 5,718.08                               | 874.46            | 13.22           | 86.78                  | 1.15           | 86.78                       |
| Mixed     | MP3, JPG  | 1,410        | 1,466.89               | 1,394.46                               | 72.43             | 4.94            | 95.06                  | 1.05           | 95.06                       |
| Mixed     | AVI, WAV, EXE | 4,101 | 20,833.00              | 16,760.92                              | 4,072.08          | 19.55           | 80.45                  | 1.24           | 80.45                       |
| Photo     | PNG       | 2,415        | 2219.60                | 2184.13                                | 35.469            | 1.60            | 98.40                  | 1.02           | 98.40                       |
| Image     | PNG       | 1,812        | 1.80                   | 12002.95                               | 5980.08           | 33.18           | 66.82                  | 1.50           | 66.82                       |

The data in Table 2 shows the difference in time with other FTP protocols in 100 MBPS and 1 GBPS connections respectively and figure 4 presents a graphical representation of this data.

| Scenarios | File Type | No. of Files | Plain FTP | FTP over TLS | Windows SMB v3 | Proposed Protocol |
|-----------|-----------|--------------|-----------|---------------|----------------|-------------------|
| Video     | AVI       | 85           | 0.016000   | 0.016000      | 0.016000       | 0.016000          |
| Audio     | WAV       | 800          | 0.016000   | 0.016000      | 0.016000       | 0.016000          |
| Photos    | PNG, JPEG | 3,500        | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Text      | TXT       | 2,688        | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Executables | EXE     | 760          | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Mixed     | AVI, WAV, PNG | 500   | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Mixed     | MP3, JPG  | 1,410        | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Mixed     | AVI, WAV, EXE | 4,101 | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Photo     | PNG       | 2,415        | 0.023000   | 0.023000      | 0.023000       | 0.023000          |
| Image     | PNG       | 1,812        | 0.023000   | 0.023000      | 0.023000       | 0.023000          |

Figure 4. Graphical representation derived from the time data.
5. Conclusion
In both 100MBPS and 1GBPS uplinks, results show significant space savings in terms of bytes transferred over the network but compressing uncompressible data can possibly lead to higher transfer time due to compression overhead but this overhead is reduced due to the usage of a real-time compression algorithm Zstd and its dictionary training function.

Lattice-based encryption did not add any noticeable slowdown in encryption and even performs faster. By combining AES and the lattice-based encryption, the data being transferred over the network is securely transmitted and is safe from quantum-based attacks using Shor’s algorithm.

The exchange of files through the network is a crucial role in the field of computing. Extensive researches and studies have been done regarding the uses and real-world applications of a quantum computer and there are serious long-term threats to modern cryptosystems. As hardware continues to evolve, the proponents believe that the software and protocols that work together with it must evolve too. The results shown in this research demonstrates significant improvements over mainstream protocols and paves the way for the adoption of such algorithms in a modern file transfer protocol.

References
[1] McAnlis C and Haecky A 2016 *Understanding Compression: Data Compression for Modern Developers 1st Edition* (Sebastopol, CA: O’Reilly Media)
[2] Rahim R et al 2018 *Int. J. Eng. Technol. 7* pp 44-7
[3] Nurdiyanto H, Rahim R, Ahmar A S, Syahril M, Dahria M, and Ahmad H 2018 *J. Phys. Conf. Ser. 1028* pp 1-11
[4] Barenghi A, Beretta M, Di Federico A and Pelosi G 2014 *IEEE Int. Conf. on High Performance Computing and Communications (Paris)* pp 763-70
[5] Chen L, Jordan S, Liu Y K, Moody D, Peralta R, Perlner R, Smith-Tone D 2016 *Report on Post-Quantum Cryptography* (Gaithersburg: NIST Pubs) pp 1-10
[6] Zhuo D, Ghobadi M, Mahajan R, Forster K, Krishnamurthy A and Anderson T 2017 *SIGCOMM ’17 Proc. of the Conf of the ACM Special Interest Group on Data Communication (Los Angeles)* (New York: ACM) pp 362-75
[7] Maderazo C, Ceniza A, Abangan B and Apor J 2016 *Proc. of the 14th National Conference on IT Education (NCITE 2016)* (Dipolog) (Philippines: NCITE) pp 295-300
[8] Baligasa C 2018 *Cryptography Using a Four Degree Polynomial Over Z* (Philippines: University of San Carlos)
[9] Singh S, Sharma P and Arora D 2017 *Int. J. of Advanced Research in Computer Science* 8 pp 1974-8
[10] Duarte R P 2008 Transport protocols for large bandwidth-delay product networks *Preprint arXiv:1705.08929v1*
[11] Bahaa-Eldin A, Mohsen A and Sobh M 2017 12th *Int. Conf. on Computer Engineering Systems (Cairo)* (New Jersey: IEEE) pp 462-7
[12] Thejaswi S and Westrup C 2016 *Approximate Computation of Frequency Moments* (Helsinki: Aalto University) pp 1-17
[13] Najmabadi S M, Tungal H S, Tran T and Simon S 2017 *Design and Architectures for Signal and Image Processing (DASIP)* (Dresden) (New Jersey: IEEE) pp 1-6
[14] Dong Q, Li B, Zhang L and Shang, Z 2014 *Electronics Letters (IET)* 50 pp 1522-4
[15] Sparsh M and Vetter J 2016 *IEEE Transactions on Parallel and Distributed Systems* 27 pp 1524-36