# FTPChat Project

Presented by:


## Ahmed Omar Saad

**Primary 6**

**Future Generation International School**

# Catalogue

# Personal Information

| | |
|---|---|
| Name: | Ahmed Omar Saad |
| Age: | 11 |
| Grade: | 6 |
| Governorate: | Sohag |
| Student code: | 493859824 |
| School: | Future Generation International School |
| Educational Department: | Akhmeem |

# Research Summary:

FTPChat is a Python-based messaging protocol that replaces traditional socket communication with FTP-based message relays. It repurposes FTP servers, such as those embedded in ZTE routers or hosted on platforms like SFTPCloud.io, into secure, decentralized messaging hubs.

Instead of maintaining persistent socket connections, FTPChat enables clients to exchange messages by downloading a shared file from the FTP server, decrypting its contents, appending their own message, re-encrypting the updated content using a 24-layer mono-alphabetic cipher, and uploading it back. This cycle allows each client to act as both a reader and a writer, maintaining a synchronized message flow across all participants.

The protocol supports both asynchronous and synchronous communication, making it ideal for environments where socket access is restricted, blocked by firewalls, or technically infeasible. Its layered encryption ensures that all messages remain secure during transit and storage, even on publicly accessible FTP servers.


### Short Abstract:

FTPChat is a Python-based messaging protocol that replaces sockets with FTP-based encrypted message relays.

It uses FTP servers as secure hubs, enabling asynchronous communication even in firewall-enabled PCs or legacy networks.

Messages are encrypted with a 24-layer cipher and exchanged via upload/download cycles.

Ideal for environments where sockets are blocked or unreliable.

# The Problem

In many restricted or legacy network environments, such as those behind firewalls, on home routers, or in educational settings, traditional socket-based communication is blocked or unsupported. This makes it difficult to build real-time messaging systems or peer-to-peer applications.

FTPChat solves this by replacing socket communication with FTP-based message relays. Instead of direct connections, clients use FTP servers to exchange encrypted messages by uploading and downloading a shared file. This allows secure, asynchronous or synchronous communication even in networks where sockets are unavailable, unreliable, or intentionally disabled.

# Research Hypotheses

1. It is possible to build a secure messaging system without relying on socket-based communication.

2. FTP servers can be repurposed as reliable intermediaries for encrypted message exchange.

3. A multi-layer mono-alphabetic encryption scheme can provide sufficient protection for messages stored on public or semi-public FTP servers.

Purpose of the Design:

The design aims to create a messaging protocol that functions in restricted or legacy network environments where socket communication is blocked or unsupported. By using FTP as the transport layer, the system enables asynchronous or synchronous encrypted messaging between clients. The goal is to offer a secure, lightweight, and accessible solution for communication across devices and platforms that lack direct connectivity.

# Results

FTPChat was designed to solve the problem of blocked or unsupported socket communication in restricted network environments. In many cases, such as school networks, legacy routers, or firewall enabled systems, real-time messaging via sockets is not possible due to technical limitations or security policies.

FTPChat replaces sockets with FTP-based message relays. By using FTP servers as intermediaries, clients can exchange encrypted messages asynchronously or synchronously without needing direct connections. This makes FTPChat ideal for environments where traditional networking protocols fail, while still maintaining secure and reliable communication.

# What has been done?

What I've Implemented:

- Converted a basic console-only messaging concept into a professional-grade protocol.
- Designed FTPChat to be fully terminal-based, with no GUI, making it ideal for Linux servers and environments that restrict graphical interfaces With an option for GUI
- Built the core system in Python, using FTP servers to relay encrypted messages between clients.
- Integrated a 24-layer mono-alphabetic encryption system to secure message content.

Experiments:

- Tested FTPChat on public FTP platforms like SFTPCloud.io and on routers with built-in FTP like ZTE ZXHN H188A.
- Verified compatibility with Linux server environments used by companies that enforce terminal-only access.
- Simulated multi-user communication to ensure synchronization, encryption integrity, and usability.

Designs:

- Created a modular architecture separating encryption, FTP handling, and message formatting.
- Focused on terminal clarity and accessibility without relying on screen size or graphical layout.

# Future Plans

1. Upgrade the encryption system by integrating advanced algorithms like AES or RSA, while maintaining lightweight performance for constrained environments.

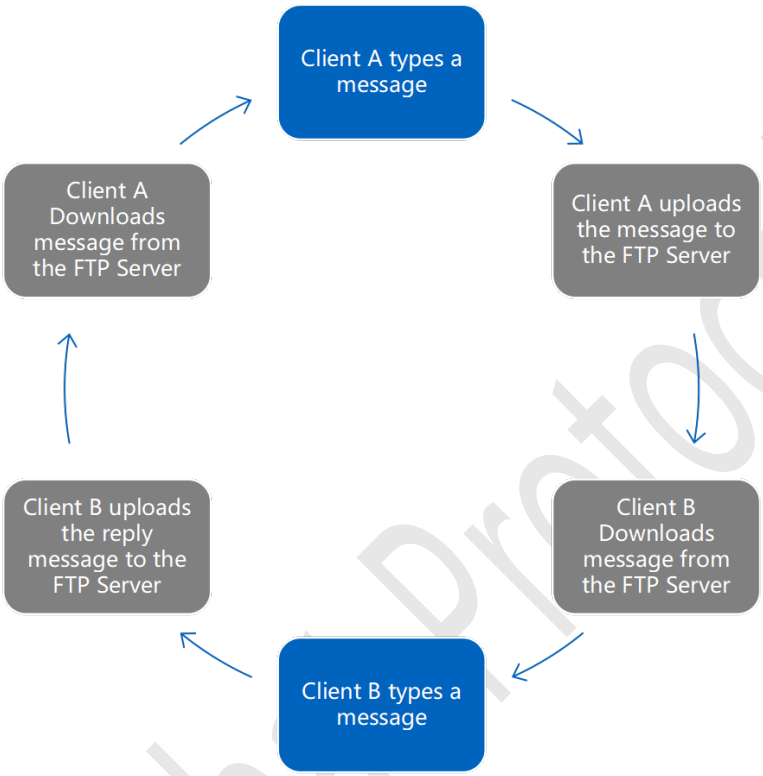
2. Implement session management features including user authentication, message tracking, and login control to support multi-user environments.

3. Introduce message compression to reduce file size and improve performance, especially on slow or limited networks.

# Project Work Steps

1. Identifying the problem (socket instability and cyberattacks)
2. Learning Python
4. Writing FTP relay logic
5. Implementing encryption
6. Testing the protocol
7. Hosting on routers and SFTPCloud.io

# Visualizing how it works



# Summary Results

- Built a socket-free encrypted messaging system
- Hosted on ZTE router and SFTPCloud.io
- Eliminated need for tunneling tools
- Enabled global access with modular encryption

# Compatibility

1. FTP (File Transfer Protocol)
- Port: 21 (TCP)
- Function: Transfers files between client and server using control and data channels.
- Behavior: Operates in active or passive mode depending on firewall and NAT configuration.
- Source: Cisco Networking Documentation

2. SFTP (SSH File Transfer Protocol)
- Port: 22 (TCP)
- Function: Transfers files securely over SSH.
- Behavior: Uses a single encrypted channel for authentication and data exchange.
- Source: Microsoft Learn – Secure File Exchange

3. Firewall Behavior and Port Accessibility

Port 21 is typically allowed by default in:
- School and university networks
- Legacy routers with built-in FTP (e.g., ZTE ZXHN H188A)
- Enterprise systems using FTP for firmware updates or configuration management

Port 22 is commonly permitted in:
- SSH-enabled routers (OpenWRT, DD-WRT)
- Cloud platforms such as Azure &, AWS
- Secure internal networks with administration access

# Source Availability

- This protocol was designed independently through personal experimentation, technical reasoning, and iterative testing.
- No external research papers or third-party code templates were used during development. All logic and encryption methods were custom-implemented.
- The project is currently closed-source. The source code is not publicly available at this stage.
- Future plans include potential open-source release after further testing, documentation, and security review.

# Source References


1. Cisco: "FTP uses TCP Port 21 for control and data transfer."
Link:
https://www.cisco.com/c/en/us/td/docs/ios/sw_upgrades/interlink/r2_0/user/ugftpc1.html

2. Microsoft Learn: "Port 22 is used for secure file exchange over SSH."
Link:
https://learn.microsoft.com/en-us/troubleshoot/azure/general/secure-file-exchange-transfer-files

3. IBM Docs: "Sockets require open ports and persistent connections, which may be blocked by firewalls."
Link:
https://www.ibm.com/docs/en/i/7.4.0?topic=programming-how-sockets-work