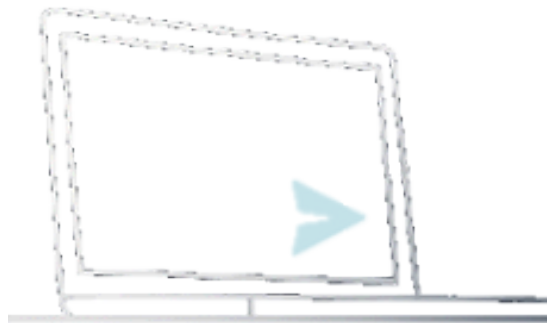


Chat Client

Technical Architecture Overview



Version 1.0 - January 2026

Created by Laurent Barraud.

Technologies Used

C#, .NET, WPF, async/await, TCP networking, custom binary packet protocol, RSA encryption, UTF-8 encoding.

1. System Overview

The solution is composed of three independent but tightly integrated modules:

Module	Role	Description
ChatClient	WPF Application	User interface, message handling, encryption, and network communication.
ChatProtocol	Shared Library	Defines the network protocol, packet types, serialization tools, and framing rules.
ChatServer	TCP Server	Manages connections, relays messages, maintains public keys, and orchestrates the protocol.

The architecture follows a classic client-server model with a simple, robust and extensible custom protocol.

2. Logical Architecture

2.1. ChatClient

Primary Responsibilities

- Reactive WPF interface (light MVVM)
- Asynchronous TCP connection management
- Sending and receiving framed packets
- RSA encryption (public key for encryption, private key for decryption)
- Protocol handling (handshake, messages, errors)
- Real-time message display
- Dynamic UI features (theme switching, language switching, sliders, toggles)

Key Components

- ClientConnection.cs – Opens and maintains the TCP connection
- PacketWriter.cs / PacketReader.cs – Binary serialization and framed packet parsing
- EncryptionManager.cs – Generates and stores RSA keypairs
- PublicKeyMonitor.cs – Real-time synchronization of public keys
- MessageDispatcher.cs – Routes packets to the appropriate handlers
- MainViewModel.cs – Core application logic and UI binding

Additional Client Features

- Emoji support through UTF-8 encoding
- Instant language switching (UI updates immediately when the ComboBox changes)

- WPF-friendly settings toggles (properly bound, no code-behind hacks)
- Customizable UI sliders for user preferences
- Dynamic dark theme that can be toggled at any moment
- System tray integration using the *Hardcodet.Wpf.TaskbarNotification* library (the only reliable option for WPF)

2.2. ChatProtocol

A shared, standalone library used by both client and server.

Contains

- PacketType (opcode enumeration)
- PacketReader.cs / PacketWriter.cs

Packet Format

Code

```
[1 byte]  PacketType
[4 bytes] PayloadLength
[n bytes] Payload
```

This design is:

- streaming-friendly
- easy to parse
- fully extensible

Framing Guarantees

- Every packet is length-prefixed, so the reader never desynchronizes
- Unknown opcodes cannot occur unless the stream is corrupted
- Multiple packets in a single TCP read are handled safely

2.3. ChatServer

Responsibilities

- Accept TCP connections
- Maintain the list of connected clients
- Store and distribute public keys
- Relay encrypted messages
- Handle protocol errors
- Implement the handshake and packet routing

Key Components

- Program.cs – Main entry point and listener
- ServerConnectionHandler.cs – Manages an individual client connection

3. Network Flow

3.1. Initial Handshake

1. The client connects.
2. If encryption is enabled, the client sends its public key.
3. The server updates its registry.
4. The server sends back the full list of known public keys.
5. The client updates its PublicKeyMonitor.
6. The lock icon updates to visually confirm that encryption is ready for all connected peers.

3.2. Sending a Message

1. The user types a message.
2. The client encrypts it using the recipient's public key.
3. The client sends a MessagePacket with opcode EncryptedMessage.
4. The server receives and relays the packet.
5. The recipient decrypts it using their private key.
6. The message appears instantly in the UI.

3.3. Public Key Updates

1. A new client connects.
2. The server updates its registry.
3. All connected clients receive a notification.
4. If encryption is enabled, the server distributes the new public key.
5. Each client updates its key store.

4. Technical Highlights

- Modular architecture with clear separation of responsibilities
- Custom network protocol, simple and extensible
- RSA encryption pipeline
- Fully asynchronous networking (no UI blocking)
- Robust packet system with framed packets to prevent stream desynchronization
- Public key synchronization that is resilient and repairable via the monitor
- UTF-8 support, enabling full emoji compatibility
- Dynamic UI features (sliders, toggles, theme switching, instant language switching)
- System tray support via Hardcodet (the only reliable WPF option)
- Responsive, ergonomic WPF interface

5. Encryption Pipeline

- Each client generates its own RSA keypair at startup
- Public keys are distributed through the server and reflected in real time
- All outgoing messages are encrypted using the recipient's public key

- Clear-text messages are only sent when encryption is explicitly disabled

5.1 Source of Truth

The Public Key Monitor is the authoritative source of truth.
It reflects the exact content of KnownPublicKeys at all times.

Whenever a key is added or updated:

- the ObservableCollection raises change notifications
- the UI refreshes automatically
- the lock icon updates to reflect global encryption validity

Lock Icon Semantics

- All keys valid: colored lock (safe to encrypt for all peers)
- At least one key missing/invalid: gray lock (encryption cannot be guaranteed)

5.2 Mixed readiness states

When the lock is gray, the client will still attempt to send the message, but it will be transmitted in clear text.

If an encrypted client sends a message to a peer who has encryption disabled, the recipient will be prompted to enable encryption before they can continue reading.

6. Animations and UX Polish

The client includes subtle WPF storyboards and animations to enhance the user experience without distracting from functionality.

7. Installer

The installer is built using Inno Setup, chosen for its simplicity and reliability. Users can choose to install the client, the server, or both.