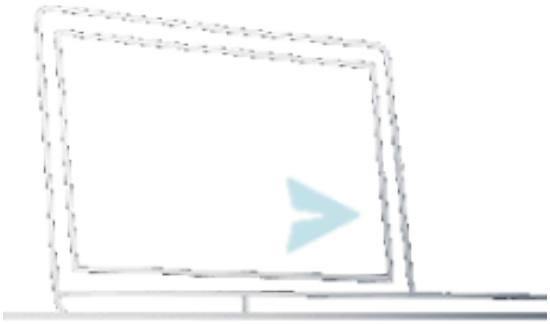
Chat Client

Aperçu de l’architecture technique



Version 1.0 – Janvier 2026

Créé par Laurent Barraud.

Technologies utilisées

C#, . NET, WPF, async/await, réseau TCP, protocole de paquets binaires personnalisé, chiffrement RSA, encodage UTF-8.

1. Vue d'ensemble du système

La solution est composée de trois modules indépendants mais étroitement intégrés:

Module	Role	Description
ChatClient	Application WPF	Interface utilisateur, gestion des messages, chiffrement et communication réseau.
ChatProtocol	Librairie partagée	Définit le protocole réseau, les types de paquets, les outils de sérialisation et les règles de framing.
ChatServer	Serveur TCP	Manages connections, relays messages, maintains public keys, and orchestrates the protocol.Gère les connexions, relaie les messages, maintient les clés publiques et orchestre le protocole.

L'architecture suit un modèle client-serveur classique, avec un protocole personnalisé simple, robuste et extensible.

2. Architecture logique

2.1. ChatClient

Responsabilités principales

• Interface WPF réactive (MVVM léger)
• Gestion asynchrone de la connexion TCP
• Envoi et réception de paquets framés
• Chiffrement RSA (clé publique pour chiffrer, clé privée pour déchiffrer)
• Gestion du protocole (handshake, messages, erreurs)
• Affichage des messages en temps réel
• Fonctionnalités UI dynamiques (changement de thème, changement de langue, sliders, toggles)

Composants clés

Helpers

• EncryptionPipeline.cs – Implémente l'ensemble du workflow RSA : stocke la paire de clés du client, chiffre les données sensibles sortantes avec la clé publique du serveur, et déchiffre les charges utiles entrantes (en anglais le « payload »).

• EncryptionHelper.cs – Fournit les utilitaires RSA bas niveau : génération de clés, export, chiffrement/déchiffrement brut.

• Framing.cs – Définit les règles de framing des paquets (préfixe de longueur, validation, limites) pour garantir une reconstruction fiable des messages sur TCP.

• ClientLogger.cs – Utilitaire de journalisation léger pour les événements de connexion, les traces de paquets et les diagnostics d'erreurs.

• ConsoleManager.cs – Attache une console optionnelle pendant le développement pour afficher les logs en temps réel.

• LocalizationManager.cs – Gère le changement de langue dynamique en chargeant et appliquant les ResourceDictionaries.

• ThemeManager.cs – Applique instantanément les thèmes clair ou sombre et garantit une cohérence visuelle sur toutes les fenêtres.

• StartupConfigurator.cs – Charge les paramètres sauvegardés au démarrage et applique le thème, la langue et la configuration initiale avant l'apparition de l'UI.

• RelayCommand.cs – Implémentation standard MVVM permettant de lier les actions UI à la logique du ViewModel.

Models

• LanguageOptions.cs – Représente les langues UI disponibles et celle actuellement sélectionnée.

• PublicKeyEntry.cs – Stocke une clé publique reçue du serveur, incluant identifiant, timestamp et octets bruts.

• UserModel.cs – Représente un utilisateur connecté et alimente la liste en temps réel.

Views

• MainWindow.cs – Interface principale contenant la liste des messages, la liste des utilisateurs, le champ de saisie ajustable, les toggles de thème et de langue, les contrôles de connexion et l'intégration dans la zone de notification.

• MonitorWindow.cs – Fenêtre de diagnostic dédiée à la synchronisation des clés publiques, affichant toutes les clés connues avec timestamps et proposant une action de synchronisation manuelle.

• SettingsWindow.cs – Fenêtre de configuration permettant d'ajuster le port, la langue, le thème, le mode local et d'autres préférences, appliquées instantanément et sauvegardées.

ViewModel

• MainViewModel.cs – ViewModel central reliant l'UI à la couche réseau : expose l'état de connexion, la liste des utilisateurs et les messages, traite les paquets entrants et envoie les données via ClientConnection.

Net

• ClientConnection.cs – Gère tout le cycle de vie TCP côté client : ouverture de la connexion, handshake RSA, boucle de réception asynchrone, envoi non bloquant, et transfert des données parsées au ViewModel.

Fonctionnalités supplémentaires du client

• Support des emojis via UTF-8
• Changement de langue instantané (mise à jour immédiate de l'UI)
• Toggles WPF correctement liés (sans code-behind)
• Sliders UI personnalisables
• Thème sombre dynamique activable à tout moment
• Intégration dans la zone de notification via Hardcodet.Wpf.TaskbarNotification (la seule option fiable pour WPF)

2.2. ChatProtocol

Librairie partagée utilisée par le client et le serveur.

Contient

• PacketBuilder.cs – Construction des paquets framés utilisés par les deux côtés.
• PacketReader.cs – Lecture et parsing des paquets framés de manière cohérente sur client et serveur.
• Protocol.cs – Définit l'énumération des opcodes du protocole : handshake, échange de clés publiques, mises à jour de la liste des utilisateurs, messages, erreurs.
• Framing.cs – Règles de framing définissant la structure binaire de chaque paquet échangé.

Format des paquets

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

Ce design est :
• adapté au streaming
• simple à parser
• entièrement extensible

Garanties du framing

• Chaque paquet est préfixé par sa longueur, empêchant toute désynchronisation
• Les opcodes inconnus ne peuvent apparaître que si le flux est corrompu
• Plusieurs paquets dans une seule lecture TCP sont gérés correctement

2.3. ChatServer

Responsabilités

• Accepter les connexions TCP
• Maintenir la liste des clients connectés
• Stocker et distribuer les clés publiques
• Relayer les messages chiffrés
• Gérer les erreurs du protocole
• Implémenter le handshake et le routage des paquets

Composants clés

• Program.cs – Point d'entrée principal et listener ; accepte les clients TCP, gère leur cycle de vie et diffuse les messages à tous les utilisateurs connectés.
• ServerConnectionHandler.cs – Gère la communication avec un client individuel : réception des paquets, traitement des opcodes, transfert des messages au serveur pour distribution.

3. Flux réseau

3.1. Handshake initial

1. Le client se connecte.
2. Si le chiffrement est activé, le client envoie sa clé publique.
3. Le serveur met à jour son registre interne.
4. Le serveur renvoie la liste complète des clés publiques connues.
5. Le client met à jour son PublicKeyMonitor.
6. L'icône de cadenas se met à jour pour confirmer visuellement que le chiffrement est prêt pour tous les pairs connectés.

3.2. Envoi d’un message

1. L’utilisateur saisit un message.
2. Le client le chiffre avec la clé publique du destinataire.
3. Le client envoie un MessagePacket avec l’opcode EncryptedMessage.
4. Le serveur reçoit et relaie le paquet.
5. Le destinataire le déchiffre avec sa clé privée.
6. Le message apparaît instantanément dans l’interface.

3.3. Mises à jour des clés publiques

1. Un nouveau client se connecte.
2. Le serveur met à jour son registre.
3. Tous les clients connectés reçoivent une notification.
4. Si le chiffrement est activé, le serveur distribue la nouvelle clé publique.
5. Chaque client met à jour son magasin de clés.

4. Points techniques clés

- Architecture modulaire avec séparation claire des responsabilités
- Protocole réseau personnalisé, simple et extensible
- Pipeline de chiffrement RSA
- Réseau entièrement asynchrone (aucun blocage de l’UI)
- Système de paquets robuste avec framing pour éviter toute désynchronisation du flux
- Synchronisation des clés publiques résiliente et réparable via le moniteur
- Support UTF-8, permettant la compatibilité complète avec les emojis
- Fonctionnalités UI dynamiques (sliders, toggles, changement de thème, changement de langue instantané)
- Support de la zone de notification via Hardcodet (la seule option fiable pour WPF)
- Interface WPF réactive et ergonomique

5. Pipeline de chiffrement

• Chaque client génère sa propre paire de clés RSA au démarrage

• Les clés publiques sont distribuées via le serveur et reflétées en temps réel

• Tous les messages sortants sont chiffrés avec la clé publique du destinataire

• Les messages en clair ne sont envoyés que si le chiffrement est explicitement désactivé

5.1. Source de vérité

Le moniteur de clés publiques est la source de vérité.

Il reflète en permanence le contenu exact de KnownPublicKeys.

À chaque ajout ou mise à jour d’une clé :

• l’ObservableCollection déclenche des notifications de changement

• l’UI se met automatiquement à jour

• l’icône de cadenas reflète l’état global du chiffrement

Sémantique de l’icône de cadenas

• Toutes les clés valides : cadenas coloré (chiffrement garanti pour tous les pairs)

• Au moins une clé manquante/invalide : cadenas gris (chiffrement non garanti)

5.2. États de préparation mixtes

Quand le cadenas est gris, le client tente tout de même d'envoyer le message, mais il sera transmis en clair.

Si un client chiffré envoie un message à un pair ayant le chiffrement désactivé, le destinataire sera invité à activer le chiffrement avant de pouvoir continuer à lire.

6. Animations et finition UX

Le client inclut des storyboards WPF subtils et des animations légères pour améliorer l'expérience utilisateur sans distraire de la fonctionnalité principale.

7. Programme d'installation

L'installateur est construit avec Inno Setup, choisi pour sa simplicité et sa fiabilité.
Les utilisateurs peuvent choisir d'installer le client, le serveur, ou les deux.