App Information
Encryption Overview
🔐 Encryption Implementation Details
Cryptographic Libraries Used
Libraries:
Encryption Algorithms
Key Management Architecture
User Key Pair Generation
Group Key Generation
Encryption Processes
Message Encryption (Group Messages)
Algorithm: XSalsa20 stream cipher with Poly1305 MAC
Process:
Key Exchange (New Member Onboarding)
Algorithm: Curve25519 ECDH + XSalsa20 + Poly1305
Process:
Security Features
Forward Secrecy
Authentication
Key Storage Security
📱 Platform Integration
iOS Implementation
Android Implementation
🌍 Compliance & Standards
International Standards Compliance
Export Control Classification
🔧 Implementation Verification
Code References
Algorithm Verification
📋 Technical Summary
🎯 Conclusion

App Information

- App Name: GroopTroop
- Bundle ID: com.nick.grooptroop
- Version: 1.0.0

Encryption Overview

GroopTroop implements end-to-end encryption for group messaging and multimedia content using standard, internationally accepted cryptographic algorithms. The app does not use any proprietary encryption methods.

🔐 Encryption Implementation Details

Cryptographic Libraries Used

// Primary encryption library
import nacl from 'tweetnacl'; // NaCl (Networking and Cryptography Library)
import util from 'tweetnacl-util';
import * as Crypto from 'expo-crypto'; // React Native crypto wrapper for iOS/Android APIs
import * as FileSystem from 'expo-file-system'; // File system operations for media encryption

Libraries:

TweetNaCl: Industry-standard NaCl implementation (Daniel J. Bernstein's Networking and Cryptography Library)
Expo Crypto: React Native wrapper for platform-native cryptographic APIs
Expo FileSystem: Secure file operations for encrypted media storage
Platform APIs: iOS CryptoKit and Android cryptographic services

Encryption Algorithms

Component	Algorithm	Key Size	Standard

Symmetric Encryption	XSalsa20 + Poly1305 (via NaCl secretbox)	256-bit	RFC 7539, NaCl standard
Asymmetric Encryption	Curve25519 + XSalsa20 + Poly1305 (via NaCl box)	256-bit	RFC 7748, NaCl standard
Media Encryption	XSalsa20 + Poly1305 (via NaCl secretbox)	256-bit	RFC 7539, NaCl standard
Key Generation	Platform CSPRNG (iOS: SecRandomCopyBytes, Android: SecureRandom)	256-bit	FIPS 140-2
Hashing	SHA-256	256-bit	FIPS 180-4

Key Management Architecture

User Key Pair Generation

```
// Each user gets an asymmetric key pair for secure key exchange
static async generateAndStoreUserKeys(userId: string): Promise<UserKeys | null> {
  const keyPair = nacl.box.keyPair(); // Curve25519 key pair
  // Keys stored securely using platform keychain
}
```

Group Key Generation

```
// Each group gets a symmetric key for message encryption
static async generateGroopKey(groopId: string): Promise<string | null> {
  const key = await Crypto.getRandomBytesAsync(nacl.secretbox.keyLength); // 256-bit
  // Stored securely using iOS Keychain/Android Keystore
}
```

Media Key Synchronization

```
// Group keys are synchronized between text and media encryption services
await MediaCryptoService.storeMasterKey(groopId, keyBase64);
await EncryptionService.syncKeyToMediaService(groopId);
```

Encryption Processes

Message Encryption (Group Messages)

Algorithm: XSalsa20 stream cipher with Poly1305 MAC

Process:

- Generate random 192-bit nonce using platform CSPRNG
- Encrypt message with group symmetric key
- Authenticate with Poly1305 MAC
- Format: nonce (24 bytes) + encrypted_data + mac (16 bytes)

Image/Media Encryption

- Algorithm: XSalsa20 stream cipher with Poly1305 MAC (same as messages)
- Process:
    1. Generate unique image ID for each uploaded media file
    2. Read image file as binary data using secure file system APIs
    3. Generate random 192-bit nonce using platform CSPRNG
    4. Encrypt image data with same group symmetric key as messages
    5. Authenticate with Poly1305 MAC
    6. Store encrypted file in device cache with consistent naming
    7. Upload encrypted blob to Firebase Storage
    8. Format: nonce (24 bytes) + encrypted_image_data + mac (16 bytes)

Media Decryption & Caching

// Efficient caching system for decrypted media

static async decryptToCache(encryptedUrl: string, groopId: string, imageId: string): Promise<string | null> {

// Check cache first - instant loading for previously viewed images

// Download encrypted file if not cached

// Decrypt using group key with NaCl secretbox

// Store decrypted file in persistent cache

// Return local file path for display

}

Key Exchange (New Member Onboarding)

Algorithm: Curve25519 ECDH + XSalsa20 + Poly1305

Process:

- Admin encrypts group key using new member's public key
- Uses sender's private key and recipient's public key
- Creates shared secret via ECDH
- Encrypts group key with shared secret

Security Features

Forward Secrecy

- New random nonce for each message
- Group keys can be rotated independently
- No key reuse across messages

Authentication

- All encrypted data includes Poly1305 MAC for integrity
- Public key cryptography ensures sender authenticity
- Platform-level secure storage protects private keys

Media Security Features

- Cache Security: Decrypted images stored in app-private cache directory
- Unique Identifiers: Each encrypted image has unique ID preventing replay attacks
- Consistent Encryption: Same key and algorithm for all group content
- Automatic Cleanup: Cache can be cleared without affecting message access

Key Storage Security

Primary Key Storage

// Keys stored using platform secure storage

- - iOS: Keychain Services (kSecAttrAccessibleWhenUnlockedThisDeviceOnly)
- - Android: Android Keystore (AES encryption with TEE/HSM when available)

Media Key Storage

// Media encryption uses same keys as message encryption

// Synchronized between EncryptionService and MediaCryptoService
// No separate key storage - unified security model

Cache Storage

- Encrypted media: Stored in Firebase Storage (encrypted blobs)
- Decrypted cache: Stored in app-private directory on device
- Cache access: Protected by iOS/Android app sandboxing
- Cache persistence: Survives app restarts for performance

📱 Platform Integration

iOS Implementation

- Uses iOS CryptoKit and Security Framework
- Keys stored in iOS Keychain with device-only access
- CSPRNG via SecRandomCopyBytes
- Media files: Stored in iOS app cache directory
- Respects App Transport Security requirements

Android Implementation

- Uses Android Keystore and Crypto APIs
- Hardware-backed security when available (TEE/HSM)
- CSPRNG via SecureRandom
- Media files: Stored in Android app cache directory
- Follows Android security best practices

🌍 Compliance & Standards

International Standards Compliance

- ✅ FIPS 140-2: Key generation using certified CSPRNGs
- ✅ RFC 7748: Curve25519 elliptic curve cryptography
- ✅ RFC 7539: ChaCha20-Poly1305 authenticated encryption
- ✅ NaCl Standard: Daniel J. Bernstein's NaCl specification
- ✅ IETF Standards: All algorithms approved by IETF✅
- Media Encryption: Uses same standards as message encryption

Export Control Classification

- ECCN: 5D002 (standard cryptographic software)
- Classification: Mass market cryptographic software
- Export License: Not required (standard algorithms, publicly available)
- Media Coverage: Image/media encryption covered under same classification

🔧 Implementation Verification

Code References

- Encryption Service: EncryptionService.tsx
- Key Exchange Service: KeyExchangeService.tsx
- Authentication Service: AuthService.tsx
- Media Crypto Service: MediaCryptoService.tsx
- Image Upload Service: imageUploadService.ts
- Info.plist Setting: ITSAppUsesNonExemptEncryption = false

Algorithm Verification

Verify TweetNaCl version and authenticity
- npm list tweetnacl

Current: tweetnacl@1.0.3 (official implementation)

Verify Expo Crypto integration
- npm list expo-crypto

Uses platform-native cryptographic APIs
Verify file system security
npm list expo-file-system
Secure file operations for media encryption

📋 Technical Summary

GroopTroop uses only standard, internationally accepted cryptographic algorithms for all content types:

- No proprietary encryption - All algorithms are IEEE/IETF/NIST approved
- Standard implementations - TweetNaCl is the reference NaCl implementation
- Platform integration - Uses iOS/Android native crypto APIs via React Native
- Unified security model - Same encryption for messages and media

- Export compliance - Qualifies for mass market software exemption
- Open source libraries - All cryptographic code is publicly auditable
- Performance optimized - Efficient caching for encrypted media

Media Encryption Highlights

- Same Algorithm: Images encrypted with identical XSalsa20+Poly1305 as messages
- Same Keys: Uses group symmetric key for all content types
- Same Security: Equal protection for text and multimedia contentEfficient Caching:
- Decrypted images cached locally for performance
- Standard Compliance: All media encryption uses IETF-approved algorithms

🎯 Conclusion

GroopTroop's encryption implementation uses only standard, well-vetted cryptographic algorithms and libraries for both messaging and multimedia content. The app does not implement any custom cryptographic protocols or use any proprietary encryption methods. All encryption is performed using internationally accepted standards through established, open-source libraries.

Media encryption uses the same security standards as message encryption, using identical algorithms and keys to ensure consistent protection across all content types.

This implementation qualifies for the standard software exemption under U.S. export control regulations and does not require additional export documentation.

Document Prepared: June 2025
App Version: 1.0.0
Contact: Nick.sanders.a@gmail.com