📊 РЕЗЮМЕ БЕЗОПАСНОСТИ

Ваш проект использует качественные криптографические алгоритмы (AES-256-GCM, Curve25519) через библиотеку libsodium, но имеет несколько критических уязвимостей в операционной безопасности.

🔴 КРИТИЧЕСКИЕ УЯЗВИМОСТИ (требуют немедленного исправления)

УСТРАНЕНА ~~1. Plaintext Secret Key Export (key-exchange.c:364-372)~~

~~fprintf(file, "SECRET_KEY=%s\n", secret_hex); // Секретный ключ в открытом виде! Угроза: Секретные ключи сохраняются на диск без шифрования~~

- ~~Файл можно украсть и прочитать все переписки~~
- ~~Только комментарий "Keep this file secure!"~~
- ~~Как взломать: Получить доступ к файловой системе~~

~~Простое исправление: // Шифровать ключи паролем через crypto_pwhash (Argon2id) crypto_pwhash(derived_key, KEY_LEN, password, pwd_len, salt, crypto_pwhash_OPSLIMIT_INTERACTIVE, crypto_pwhash_MEMLIMIT_INTERACTIVE, crypto_pwhash_ALG_ARGON2ID13); // Затем шифровать secret_key с помощью derived_key через AES-256-GCM~~

УСТРАНЕНА 2. ~~Ключ в аргументах командной строки (audio_call.c)~~

~~./audio_call call 192.168.1.10 50000 a1b2c3d4e5f6... # Ключ виден в ps aux! Угроза: Ключ шифрования виден любому локальному пользователю~~

- ~~ps aux | grep audio_call покажет ключ~~
- ~~/proc/[pid]/cmdline содержит ключ~~
- ~~Как взломать: Локальный доступ к системе, выполнить ps~~

~~Простое исправление: // Читать ключ из stdin, а не из argv fprintf(stderr, "Enter encryption key (hex): "); if (fgets(hexkey, sizeof(hexkey), stdin) == NULL) exit(1); // Очистить argv после чтения memset(argv[3], 0, strlen(argv[3]));~~

3. Отсутствие защиты от replay-атак

Угроза: Злоумышленник может записать и переслать старые сообщения

- Nonce случайные, но не монотонные
- Sequence numbers в аудио НЕ проверяются на возрастание
- Как взломать: a. Перехватить зашифрованное сообщение b. Отправить его повторно серверу c. Сервер расшифрует и примет как новое

Код уязвимости (client.c:335-378): // Принимается любое сообщение с валидным AEAD тегом // НЕТ проверки timestamp или sequence number

Простое исправление: // Добавить timestamp в дополнительные данные AEAD uint32_t timestamp = (uint32_t)time(NULL); memcpy(ad + ad_len, &timestamp, sizeof(timestamp)); ad_len += sizeof(timestamp);

// При расшифровке проверять временную метку uint32_t msg_time = *(uint32_t*)(ad + old_ad_len); uint32_t now = (uint32_t)time(NULL); if (abs(now - msg_time) > 30) { // ±30 секунд допуск fprintf(stderr, "Replay

attack detected\n"); return -1; }

4. CRC-32 вместо криптографического хеша (client.c:57-66)

uint32_t crc32(const uint8_t *data, size_t len) { // CRC НЕ криптографический! Угроза: Файлы можно подменить, обновив CRC

- CRC-32 легко подобрать для любых данных
- Как взломать: a. Перехватить файл b. Изменить содержимое c. Пересчитать CRC-32 (миллисекунды) d. Отправить модифицированный файл

Простое исправление: // Заменить CRC-32 на HMAC-SHA256 crypto_auth_hmacsha256(file_mac, file_data, file_size, room_key); // Передавать file_mac вместо crc // При приеме проверять: if (crypto_auth_hmacsha256_verify(received_mac, file_data, file_size, room_key) != 0) { fprintf(stderr, "File integrity check failed!\n"); return -1; }

🟠 ВЫСОКИЙ РИСК (требуют исправления в ближайшее время)

5. Отсутствие Perfect Forward Secrecy

Угроза: Если ключ комнаты скомпрометирован, ВСЯ история переписки расшифровывается

- Один статический ключ на всю сессию
- Как взломать: Получить room key → расшифровать все сохраненные сообщения

Сложное исправление (требует переработки протокола): // Вариант 1: Rotation ключей по времени uint32_t hour = time(NULL) / 3600; uint8_t session_key[32]; crypto_kdf_derive_from_key(session_key, 32, hour, "FEARKEY1", room_key);

// Вариант 2: Per-message key derivation crypto_kdf_derive_from_key(msg_key, 32, msg_counter, "FEARMSG1", room_key);

6. Неполная очистка памяти

Угроза: Расшифрованные данные остаются в памяти

- Cold boot атаки
- Дампы памяти
- Найденные уязвимости:
  - client.c:335 - decrypted message не обнуляется
  - client.c:446 - plain buffer не очищается
  - Audio PCM буферы никогда не очищаются

Простое исправление: // После использования расшифрованных данных: sodium_memzero(decrypted, message_len); free(decrypted);

// Для файлов: sodium_memzero(file_data, file_size); free(file_data);

7. Нет взаимной аутентификации

Угроза: Невозможно проверить, что сервер/клиент - это именно тот, за кого себя выдает

- Как взломать: Man-in-the-middle атака a. Злоумышленник перехватывает подключение b. Притворяется сервером для клиента c. Притворяется клиентом для сервера d. Расшифровывает и читает все сообщения

Сложное исправление: // Добавить цифровые подписи Ed25519 crypto_sign_keypair(server_pk, server_sk); // Каждое сообщение подписывать: crypto_sign_detached(signature, &sig_len, message, msg_len, server_sk); // При приеме проверять: crypto_sign_verify_detached(signature, message, msg_len, peer_pk);

8. Sequence numbers в аудио не валидируются (audio_call.c:271-317)

// Последовательность извлекается, но НЕ проверяется на возрастание uint64_t seq = ntohll_u64(*(uint64_t*)*(enc + 1)); // НЕТ: if (seq <= last_seq) reject(); Угроза: Аудио фреймы можно дублировать или воспроизводить в неправильном порядке

Простое исправление: static uint64_t last_seq = 0; if (seq <= last_seq) { fprintf(stderr, "Duplicate or old audio frame rejected\n"); return -1; } last_seq = seq;

🟡 СРЕДНИЙ РИСК

9. Нет временных меток

- Невозможно определить возраст сообщения
- Temporal атаки возможны

10. Нет TLS/SSL

- Сырые TCP/UDP сокеты
- Теряется дополнительная защита транспортного уровня

11. Служебные сообщения с нулевым nonce (server.c:131-188)

memset(nonce, 0, sizeof(nonce)); // Для списка пользователей

- Смешивание зашифрованных и служебных сообщений

✅ ЧТО СДЕЛАНО ПРАВИЛЬНО

1. ✅ Отличные алгоритмы: - AES-256-GCM (NIST-approved AEAD) - Curve25519 (современная эллиптическая кривая) - XSalsa20-Poly1305 (проверенный stream cipher)
2. ✅ Правильный RNG: - libsodium randombytes_buf() - Использует /dev/urandom (Linux) и CryptGenRandom() (Windows)
3. ✅ Ключи НЕ передаются по сети: - Out-of-band обмен ключами - В протоколе только nonce префиксы
4. ✅ AEAD с дополнительными данными: - Room name + sender name аутентифицируются - Защита от подмены метаданных

📋 ПРИОРИТЕЗИРОВАННЫЙ ПЛАН ИСПРАВЛЕНИЙ

⚡ НЕМЕДЛЕННО (1-2 недели)

#	Проблема	Сложность	Время	Файлы
1	Replay protection	Низкая	1 день	client.c, server.c
2	Ключ в argv	Низкая	2 часа	audio_call.c
3	Шифрование экспортируемых ключей	Средняя	1 день	key-exchange.c
4	CRC→HMAC для файлов	Низкая	4 часа	client.c

🗓️ КРАТКОСРОЧНО (1-2 месяца)

#	Проблема	Сложность	Время	Файлы
5	Sequence validation аудио	Низкая	4 часа	audio_call.c
6	Полная очистка памяти	Средняя	3 дня	Все .c файлы
7	Timestamp validation	Низкая	1 день	client.c, server.c
8	Взаимная аутентификация	Высокая	1 неделя	Новый модуль

🗓️ СРЕДНЕСРОЧНО (3-6 месяцев)

#	Проблема	Сложность	Время
9	Perfect Forward Secrecy	Высокая	2 недели
10	TLS 1.3 для консоли	Высокая	2 недели
11	DTLS для аудио	Очень высокая	1 месяц
12	Внешний аудит безопасности	-	2-4 недели

🎯 ИТОГОВАЯ ОЦЕНКА

Аспект	Статус	Риск
Криптография	✅ Отлично	Низкий
Хранение ключей	❌ Плохо	КРИТИЧЕСКИЙ
Защита от replay	❌ Отсутствует	КРИТИЧЕСКИЙ
Целостность файлов	❌ Слабая	КРИТИЧЕСКИЙ
Очистка памяти	⚠️ Частично	Высокий
Forward Secrecy	❌ Отсутствует	Высокий
Аутентификация	⚠️ Частично	Высокий

Общая оценка: НЕ ГОТОВ К PRODUCTION без исправления критических уязвимостей

Рекомендация:

- ✅ Подходит для образовательных целей

- ✅ Подходит для тестирования в закрытой сети
- ❌ НЕ использовать для конфиденциальной переписки
- ❌ НЕ использовать в интернете без исправлений

Время до production-ready: ~2-3 месяца при активной разработке

💡 КОНКРЕТНЫЕ СЦЕНАРИИ ВЗЛОМА

Сценарий 1: "Украденный файл ключей"

Злоумышленник получает доступ к файловой системе

cat ~/.fear/my_keys.txt

Видит: SECRET_KEY=3627bd5f3f93722b...

Расшифровывает ВСЕ сохраненные сообщения

Сценарий 2: "Process snooping"

На той же машине другой пользователь:

ps aux | grep audio_call

Видит: audio_call call ... a1b2c3d4e5f6... (полный ключ!)

Сценарий 3: "Replay атака"

Злоумышленник перехватывает пакет:

captured = recv_from_network()

Через час отправляет его снова:

send_to_server(captured)

Сервер принимает как новое сообщение

Сценарий 4: "Подмена файла"

Перехватить файл + CRC

file_data, crc = intercept_file()

Изменить содержимое

malicious_data = inject_malware(file_data)

Пересчитать CRC-32 (миллисекунды)

new_crc = calculate_crc32(malicious_data)

Отправить

send_file(malicious_data, new_crc) # Успешно принят!

📚 ССЫЛКИ НА СТАНДАРТЫ

- NIST SP 800-38D: AES-GCM specification
- RFC 7539: ChaCha20-Poly1305 AEAD
- RFC 7748: Curve25519 and Curve448
- OWASP Cryptographic Storage Cheat Sheet
- libsodium Documentation: https://doc.libsodium.org/

Проект имеет хорошую криптографическую основу, но требует доработки операционной безопасности перед реальным использованием.