Міністерство освіти і науки України

Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерної інженерії та охоронних систем

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СИСТЕМА ШИФРУВАННЯ ВІДЕОПОТОКІВ І МЕТАДАНИХ ДЛЯ ХМАРНОЇ СИСТЕМИ МОНІТОРИНГУ БЕЗПЕКИ ОБ'ЄКТА

VIDEO STREAM AND METADATA ENCRYPTION SYSTEM FOR CLOUD-BASED FACILITY SECURITY MONITORING SYSTEM

спеціальність 126 Інформаційні системи та технології
(шифр і назва спеціальності)

освітня програма «Інформаційні системи та технології охорони і безпеки»
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи ІСТО-41
ПАВЛЕНКО Євгеній Ігорович
_____
(підпис)

Керівник:
к.т.н., доцент
КОСТЮЧКО Сергій Миколайович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
к.т.н., доцент
ТЕРЛЕЦЬКИЙ Тарас Володимирович
_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет: *комп'ютерних та інформаційних технологій*
Кафедра: *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *126 Інформаційні системи та технології*
Освітня програма: «*Інформаційні системи та технології охорони і безпеки*»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІБ
к.т.н., доцент Терлецький Т. В.
_____
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

ПАВЛЕНКУ Євгенію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Система шифрування відеопотоків і метаданих для хмарної системи моніторингу безпеки об'єкта*

Керівник роботи: *к.т.н., доц. Костючко Сергій Миколайович*

затверджені наказом закладу вищої освіти від «16» грудня 2026 р. № 529/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: «30» травня 2026 р.

3. Вихідні дані до роботи: *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Анотація. Перелік умовних позначень Вступ. Розділ 1. Аналітичний огляд стану предметної області (базові принципи формування відеоданих та характеристики відеопотоків; методи компресії відео та стандарти кодування; мережеві протоколи передавання мультимедіа у реальному часі; фрагментація та інкапсуляція відеоданих на клієнті; архітектура конвеєрів обробки відео; постановка мети та завдань роботи). Розділ 2. Обґрунтування вибору засобів та методів реалізації(уразливості систем відеоспостереження та загрози перехоплення вибір та обґрунтування криптографічних алгоритмів; механізми узгодження ключів сеансу за протоколом ECDH; математичні основи еліптичних кривих; алгоритм узгодження сеансового ключа ECDHE; функція виведення ключа HKDF). Розділ 3. Програмна реалізація та тестування (архітектурне рішення та структура системи; аналіз можливостей застосування принципу нульового копіювання; ограмна реалізація серверної частини; програмна реалізація клієнтської частини; повна архітектура системи; тестування та оцінка результатів;вимірювання наскрізної затримки відеопотоку). Загальні висновки та рекомендації. Список використаних джерел. Додатки.

5. Перелік графічного (ілюстративного) матеріалу: *Презентація на 10 слайдах. Спрощена схема роботи кодувальника. Схема інкапсуляції H.264 у fMP4 для відтворення через MSE. Схема лінійного відеоконвеєра як DAG із послідовних етапів Архітектурна схема взаємодії компонентів. Визначення затримки системи під час потокової передачі відео.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Розділ 1 Аналітичний огляд стану предметної області	*Костючко С.М.*		
Розділ 2 Обґрунтування вибору засобів та методів реалізації	*Костючко С.М.*		
Розділ 3 Практична реалізація	*Костючко С.М.*		
Загальні висновки та рекомендації	*Костючко С.М.*		
Нормоконтроль	*Кайдик О. Л.*		
Гарант ОП	*Терлецький Т. В.*		
Показник запозичень тексту			
Академічна доброчесність	*Кайдик О. Л.*		

7. Дата видачі завдання: «16» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1.	Обґрунтування теми	До 12.12.2025 р.	
2.	Огляд літератури із досліджуваної проблеми	До 12.12.2025 р.	
3.	Розділ 1 Аналітичний огляд стану предметної області	До 28.02.2026 р.	
4.	Розділ 2 Обґрунтування вибору засобів та методів реалізації	До 31.03.2026 р.	
5	Розділ 3 Практична реалізація	До 30.04.2026 р.	
6.	Загальні висновки та рекомендації	До 16.05.2026 р.	
7.	Формування списку використаних джерел	До 20.05.2026 р.	
8.	Формування додатків.	До 20.05.2026 р.	
9.	Формування презентації за темою кваліфікаційної роботи	До 20.05.2026 р.	
10.	Нормоконтроль	До 21.05.2026 р.	
11.	Інструментальна перевірка на академічний плагіат	До 22.05.2026 р.	
12.	Представлення кваліфікаційної роботи бакалавра до захисту	До 02.06.2026 р.	

Здобувач вищої освіти _____ (Павленко Є.І.)
(підпис)

Керівник кваліфікаційної роботи _____ (Костючко С.М.)
(підпис)

АНОТАЦІЯ

Павленко Є. І. Система шифрування відеопотоків і метаданих для хмарної системи моніторингу безпеки об’єкта. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інформаційні системи та технології охорони і безпеки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, загальних висновків та рекомендацій, списку використаних джерел та додатків.

У роботі досліджено теоретичні основи відеопотоків і методи їх компресії, зокрема розглянуто ключові параметри відеосигналу, принципи роботи сучасних кодеків та протоколи передавання мультимедійних даних.

На основі проведеного аналізу визначено основні загрози безпеці при передачі відеопотоків у мережі та обґрунтовано вибір криптографічних алгоритмів AES-GCM, ECDH і HKDF як засобів забезпечення конфіденційності та цілісності даних. Розроблено програмну систему захищеної передачі відеоданих, що охоплює серверну та клієнтську складові, а також проведено тестування, яке підтвердило коректність і ефективність запропонованого рішення.

Ключові слова: відеодані, відеопотік, кодування, контейнер, потокове передавання, криптографічний захист, WebSocket, AES-GCM, ECDH.

ANNOTATION

Pavlenko E. Video stream and metadata encryption system for cloud-based facility security monitoring system. Manuscript.

Bachelor’s qualification work EP «Security and safety information system and technologies». Lutsk National Technical University. Lutsk, 2026.

This bachelor’s thesis comprises an introduction, three sections, general conclusions and recommendations, a list of references, and appendices.

This paper examines the theoretical foundations of video streams and methods for their compression, specifically addressing key parameters of the video signal, the operating principles of modern codecs, and protocols for transmitting multimedia data. Based on the analysis, the main security threats during the transmission of video streams over a network were identified, and the selection of the AES-GCM, ECDH, and HKDF cryptographic algorithms as means of ensuring data confidentiality and integrity was justified. A software system for secure video data transmission has been developed, encompassing server and client components, and testing has been conducted, confirming the correctness and effectiveness of the proposed solution.

Keywords: video data, video stream, coding, container, streaming, cryptographic protection, WebSocket, AES-GCM, ECDH.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AAD – додаткові автентифіковані дані (Additional Authenticated Data), що не шифруються, але захищаються від модифікації.

AEAD – автентифіковане шифрування з додатковими автенти-фікованими даними (Authenticated Encryption with Associated Data).

AES – симетричний стандарт шифрування (Advanced Encryption Standard) для захисту даних.

AES-GCM – режим автентифікованого шифрування на основі AES у режимі GCM.

AES-NI – набір апаратних інструкцій процесора для прискорення операцій AES.

API – програмний інтерфейс застосунку (Application Programmi-ng Interface).

AVC – вдосконалене кодування відео (Advanced Video Coding), інша назва стандарту H.264.

CBR – постійний бітрейт (Constant Bit Rate).

CPU – центральний процесор (Central Processing Unit).

DCT – дискретне косинусне перетворення (Discrete Cosine Transform), що використовується під час стиснення сигналів.

ECDH – алгоритм узгодження спільного ключа на основі еліптичних кривих (Elliptic Curve Diffie–Hellman).

fMP4 – фрагментований формат MP4 (fragmented MP4), придатний для потокового передавання.

FPS – кількість кадрів за секунду (Frames Per Second).

GCM – режим роботи блочного шифру (Galois/Counter Mode), що поєднує шифрування та перевірку цілісності.

GMAC – механізм автентифікації повідомлень на основі режиму GCM без шифрування даних (Galois Message Authentication Code).

GOP – група кадрів, що визначає структуру ключових і прогнозних кадрів.

H.264/AVC – стандарт стиснення відео, широко застосовуваний у потокових мультимедійних системах.

HKDF – функція розгортання ключів на основі HMAC (HMAC-based Key Derivation Function).

HMAC – код автентифікації повідомлення на основі криптографічної хеш-функції (Hash-based Message Authentication Code).

HTML5 – сучасна версія мови розмітки HTML, що підтримує мультимедійні можливості браузера.

HTTP – протокол передавання гіпертексту (HyperText Transfer Protocol).

IV/Nonce – унікальне значення ініціалізації, що використовується один раз для коректної та безпечної роботи режимів шифрування.

MAC – кодавтентифікаціїповідомлення(MessageAuthentication Code), який підтверджує цілісність і автентичність даних.

MSE – розширення джерела медіа (Media Source Extensions) для буферизації та відтворення потокового відео в браузері.

NAL – рівень абстракції мережі (Network Abstraction Layer) у стандарті H.264/AVC.

RTP – протокол передавання даних реального часу (Real-time Transport Protocol).

TCP – протокол керування передаванням (Transmission Control Protocol).

TLS – протокол захисту транспортного рівня (Transport Layer Security).

VBR – змінний бітрейт (Variable Bit Rate).

Web Crypto API – браузерний програмний інтерфейс для виконання криптографічних операцій.

WebRTC – набір вебтехнологій для передавання аудіо, відео та даних у реальному часі між вузлами мережі.

WebSocket – протокол повнодуплексного обміну даними між клієнтом і сервером через одне постійне з’єднання.

YUV – колірна модель подання відеосигналу, де яскравість і колірні компоненти зберігаються окремо.

ВСТУП

Актуальність теми поширенням хмарних систем відеоспостереження, дистанційного моніторингу та аналітики безпеки, у яких відеопотоки й супровідні метадані передаються через публічні або корпоративні мережі. У таких умовах зростає ризик перехоплення, модифікації, підміни або несанкціонованого використання мультимедійної інформації, що може призвести до втрати конфіденційності, порушення цілісності даних і зниження довіри до системи безпеки загалом. Особливої ваги набуває потреба в проектних рішеннях, які поєднують низьку затримку передавання, надійний криптографічний захист, цілісність службових метаданих та можливість інтеграції з веборієнтованими засобами перегляду й адміністрування.

До основних вимог проектування належать забезпечення безперервного передавання відеопотоку в реальному часі, захист конфіденційності та цілісності відеоданих і метаданих, підтримка автентифікованого обміну ключовою інформацією.

Метою роботи є обґрунтування та проектування системи шифрування відео потоків і метаданих для хмарної системи моніторингу безпеки об’єкта, яка забезпечує конфіденційність, цілісність, автентичність даних і низьку затримку їх передавання у веборієнтованому середовищі.

Об’єктом проектування є хмарна система моніторингу безпеки об’єкта, у межах якої здійснюється формування, передавання, приймання та відтворення відеопотоків і супровідних метаданих у реальному часі.

Предметом проектування є архітектура, методи та програмні засоби шифрування відеопотоків і метаданих у веборієнтованій клієнт-серверній системі хмарного моніторингу безпеки об’єкта.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Базові принципи формування відеоданих

Відео – це цифрове подання рухомого зображення у часі, яке складається з послідовності окремих кадрів. Для людини воно сприймається як безперервний рух, але для комп’ютера це дискретні дані, що відтворюються з певною частотою. У науковому контексті відео розглядають як сигнал у часі, який можна аналізувати, стискати, передавати та відновлювати. Цей сигнал має просторову структуру (пікселі в кадрі) та часову структуру (зміни між кадрами).

Відео часто поєднане із звуком, тому на практиці воно є мультимедійним об’єктом. Такий об’єкт містить не лише зображення, а й синхронізовані аудіодоріжки та службові дані. Важливо розрізняти самі дані відео і спосіб їх зберігання. Відеодані кодуються кодеком, а зберігаються у контейнері, який визначає структуру файла. З позиції інформатики відео можна трактувати як потік байтів, що описує зміну пікселів у часі. Цей потік підлягає алгоритмічному обробленню декодуванню, фільтрації, стабілізації, аналізу сцен. Таким чином, відео структурований цифровий об’єкт із параметрами, форматами та правилами відтворення [1].

З технічної точки зору відеофайл є впорядкованою послідовністю байтів, що відповідає специфікації контейнера. Ця послідовність містить заголовки, таблиці індексації та дані потоків. Узагальнену логічну структуру відеофайла наведено в таблиці 1.1.

Під час читання файла програма аналізує заголовки, щоб зрозуміти, де знаходиться відео, де аудіо, і як їх декодувати [2].

Для комп’ютера, відео – це послідовність байтів, зчитуваних із носія або мережі. Ці байти мають чітку структуру, яка визначається форматом контейнера, наприклад ISO Base Media File Format. Контейнер зберігає кілька потоків відео, аудіо, субтитри, метадані. Кожен потік має власну внутрішню структуру, часові мітки та параметри відтворення. Відеопотік зберігається у стиснутому вигляді,

бо «сирі» кадри займають надто багато пам’яті. Стиснення досягається завдяки кодуванню повторюваних та прогнозованих ділянок між кадрами. Комп’ютер під час відтворення читає заголовки контейнера, знаходить потрібні потоки та визначає, як їх декодувати. Далі кодек відновлює кадри з стиснених даних і відправляє їх у графічну підсистему для показу. Важливу роль відіграють часові мітки (timestamps), які задають порядок і момент показу кожного кадру. Без них синхронізація відео зі звуком була б неможливою.

Таблиця 1.1 – Приклад логічної структури відеофайла [2]

Компонент	Приклад вмісту
Заголовки	Тип контейнера, версія, метадані
Таблиці індексації	Часові мітки, офсети кадрів
Дані потоків	Відео та аудіопакети

Базовим елементом відео є кадр, тобто окреме зображення, що має певну роздільну здатність. Кадри відтворюються послідовно і створюють ілюзію руху. Окрім кадрів, відео містить часову структуру, яка визначає, коли кожен кадр має бути показаний. Це задається частотою кадрів та часовими мітками всередині потоку. Важливою складовою є кодек алгоритм стиснення і декодування. Кодек визначає, як саме перетворити візуальну інформацію у компактний потік байтів і як її відновити [1].

Більшість сучасних кодеків використовують міжкадрове стиснення, де зберігається не повне зображення кожного кадру, а лише різниця між сусідніми кадрами дозволяє суттєво зменшити розмір файлу без помітної втрати якості.

Контейнер формує «упаковку» відео він містить відео та аудіо потоки, а також метадані про тривалість, роздільну здатність, автора, мову і інше. Сам контейнер не змінює зображення, але задає структуру файла. Аудіо є окремою складовою, яка має власні параметри частоту дискретизації, кількість каналів, бітрейт. Важливо розрізняти контейнер і кодек, наприклад, файл із розширенням `.mp4` це контейнер, який може містити відео, стиснене різними кодеками, зокрема H.264 або H.265. ефективність стиснення, а контейнер забезпечує коректне об'єднання всіх потоків в єдиний файл. Та кількістю бітів, що передаються за секунду, цей показник визначає баланс між якістю зображення.

1.2 Характеристики та параметри відеопотоків

Бітрейт визначає, скільки даних використовується для кодування відео за одиницю часу. Це один із ключових параметрів, що впливають на якість і розмір файла.

Вищий бітрейт дозволяє зберегти більше деталей, менше артефактів і плавнішу передачу руху. Збільшення бітрейту призводить до зростання обсягу даних та вимог до пропускної здатності.

Одиниця бітрейту – це кількість бітів за секунду (bit/s). Префікс показує масштаб одиниці (наприклад, k для 10^3 або Ki для 2^{10}), а суфікс «/s» означає «за секунду». Таким чином, kbit/s – це тисяча бітів за секунду, а Kibit/s – 1024 біти за секунду.

Одиниці та префікси для бітрейту з десятковими (SI) і двійковими (IEC) множниками наведено в таблиці 1.2 [3].

Таблиця 1.2 – Одиниці бітрейту та їхні множники

Назва	Позначення	Множник (біт/с)
біт за секунду	bit/s	1
кілобіт за секунду (SI)	kbit/s	10^3
мегабіт за секунду (SI)	Mbit/s	10^6
гігабіт за секунду (SI)	Gbit/s	10^9
терабіт за секунду (SI)	Tbit/s	1012
кібі-біт за секунду (IEC)	Kibit/s	210
Назва	Позначення	Множник (біт/с)
біт за секунду	bit/s	1
кілобіт за секунду (SI)	kbit/s	10^3

Для обчислення бітрейту у найпростішому випадку застосовують формулу (1.1):

$$R = N_{bits}/t \tag{1.1}$$

де R – бітрейт у bit/s;

N_{bits} – кількість переданих бітів;

t – час у секундах.

Оскільки 1 байт дорівнює 8 бітам, то зв’язок між швидкістю у байтах за секунду та бітрейтом описується виразом (1.2):

$$R_{bit/s} = 8\ R_{B/s}. \quad (1.2)$$

де $R_{bit/s}$ – швидкість передавання вbit/s;

$R_{B/s}$ – швидкість передавання в B/s.

Існують режими постійного бітрейту (CBR), де швидкість потоку майже не змінюється, і змінного бітрейту (VBR), де більше даних виділяється на складні сцени. VBR дає кращу якість при однаковому середньому бітрейті. Під час трансляцій і потокового відео важливу роль відіграє адаптивний бітрейт. Система може змінювати якість у реальному часі залежно від швидкості інтернету користувача.

Роздільна здатність визначає, скільки пікселів містить кадр по горизонталі та вертикалі. Наприклад, 1920×1080 означає ширину 1920 пікселів і висоту 1080 пікселів. Вища роздільна здатність забезпечує детальніше зображення, це також означає більший обсяг даних для зберігання та обробки. Тому одна й та сама сцена у 4K вимагатиме значно вищого бітрейту, ніж у 720 р.

До інших базових параметрів належить частота кадрів (FPS), яка визначає, скільки кадрів відтворюється за одну секунду. Колірна модель описує спосіб кодування кольору (найпоширенішими є RGB та YUV). Глибина кольору визначає, скільки бітів виділяється на піксель. Групи кадрів (GOP) визначають структуру, де частина кадрів зберігається повністю, а інші як різниця [1, 2].

1.3 Методи компресії відео та стандарти кодування

Кодування відео – це процес перетворення візуальної інформації у компактний потік байтів відповідно до правил конкретного кодека. Мета кодування полягає у зменшенні обсягу даних без критичної втрати якості. Основним принципом сучасного кодування є усунення надлишковості в просторі

та часі. Просторова надлишковість зменшується через перетворення та квантування в межах кадру, а часова – через прогнозування між кадрами. Це дозволяє замінювати повні кадри на опис змін, що істотно знижує бітрейт при збереженні візуальної цілісності.

Поширеним перетворенням є блокове DCT, після якого дані квантуються й підлягають ентропійному кодуванню; порядок сканування коефіцієнтів і кодування нульових серій впливають на ефективність стиску. Кодеки працюють із різними типами кадрів повні (ключові) та прогнозні. У контексті H.264 (рис. 1.1) ці типи зазвичай описують як I-frames, P-frames і B-frames. I-кадр (Intra) кодується незалежно від інших кадрів та містить повну просторову інформацію про сцену.

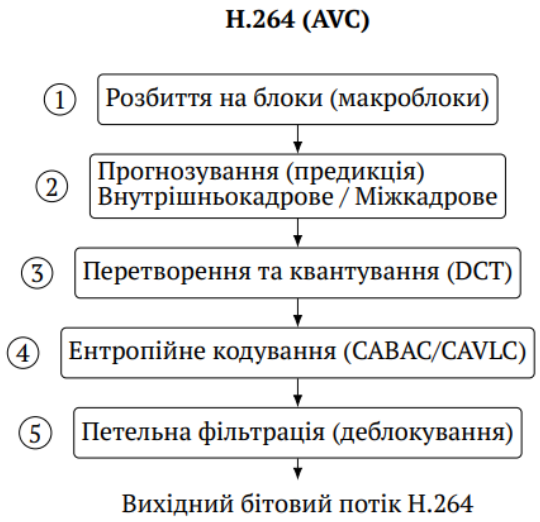

Рисунок 1.1 – Спрощена схема роботи кодувальника H.264/AVC

P-кадр (Predicted) використовує міжкадрове прогнозування від попередніх опорних кадрів і зберігає переважно вектори руху та залишкову похибку. B-кадр (Bi-predicted) застосовує двонапрямне прогнозування одночасно з минулих і майбутніх опорних кадрів.

Це дає найкращу ефективність стискання, однак вимагає очікування «майбутніх» опор, що збільшує алгоритмічну затримку (reordering delay), що є критичним для zero-latency систем. Тому обирають конфігурації на основі

Baseline Profile, короткий GOP і відсутність B-кадрів. Обґрунтування такого вибору полягає у зменшенні структурної затримки кодування, хоча це зазвичай потребує вищого бітрейту [1].

1.4 Мережеві протоколи передавання мультимедіа у реальному часі

Live-stream відрізняється від завантаження файлу тим, що відтворення починається ще підчас надходження даних, а затримка між захопленням І показом має бути мінімальною. Для цього потрібні короткі сегменти, буферизація на кілька секунд і механізми швидкої адаптації якості. Низька затримка зазвичай зменшує запас буфера й підвищує чутливість до коливань мережі, тому архітектури оптимізують шлях доставки та час обробки. Часто обирають ключові кадри з меншою періодичністю, щоб зменшити бітрейт, але це збільшує залежність від прогнозних кадрів тому схеми відновлення після втрат є критичними [5].

Класичний HTTP орієнтований на запит-відповідь не гарантує мілісекундні затримки через накладні витрати встановлення з’єднань, буферизацію та поведінку кешів. Для двонаправленої передачі малих порцій даних у реальному часі застосовують WebSockets, які підтримують постійне з’єднання поверх TCP дають змогу передавати сирі бінарні дані з низькими накладними витратами [6]. Відтворення потокового відео в браузері часто реалізують через Media Source Extensions (MSE), де додаток самостійно керує сегментацією, чергами даних та буфером відтворення.

HTTP-полінг базується на циклічних запитах клієнта й має зайві затримки. WebSockets забезпечують повнодуплексний канал зв’язку, що важливо для інтерактивних сценаріїв.

На відміну від повторних HTTP-запитів, постійне з’єднання зменшує накладні витрати та затримку на встановлення нових сесій. WebRTC орієнтований на інтерактивні медіасеанси з мінімальною затримкою, але його інтеграція складніша. Узагальнену схему відмінностей наведено в таблиці 1.3.

Таблиця 1.3 – Схема відмінностей між HTTP-полінгом, WebSockets і WebRTC

Характеристика	HTTP-полінг	WebSockets	WebRTC
Модель обміну	Запит відповідь	Постійне з'єднання	Сеанс P2P
Затримка	Висока/ змінна	Низька	Дуже низька
Накладні витрати	Високі	Низькі	Високі на старті
Бінарні дані	Обмежено	Так	Так
Складність інтеграції	Низька	Середня	Висока

Джерело: [6, 7]

1.5 Фрагментація та інкапсуляція відеоданих на клієнті

Неперервний потік H.264 розбивається на NAL-одиниці, які потім пакуються для передавання мережею. У транспортних протоколах визначено способи інкапсуляції NAL-одиниць, їх фрагментації та маркування, що дозволяє приймачу відновити порядок і синхронізацію кадрів. Фрагментація потрібна, коли розмір NAL-одиниці перевищує допустимий розмір мережевого пакета. У таких випадках дані діляться на послідовні фрагменти з маркерами початку та кінця [8].

Звичайний формат MP4 містить індексну таблицю moov, яка описує всю структуру файла. У live-стрімінгу файл ще не завершено, тому повна таблиця moov невідома, що унеможливлює стандартне відтворення.

Формат Fragmented MP4 (fMP4) вирішує цю проблему, розбиваючи відео на незалежні фрагменти. Кожен фрагмент містить заголовок `moof` (movie fragment), який описує саме цей шматок, та відповідні медіадані `mdat`. У сценаріях WebSockets передаються сирі NAL-одиниці H.264 (рис. 1.2).

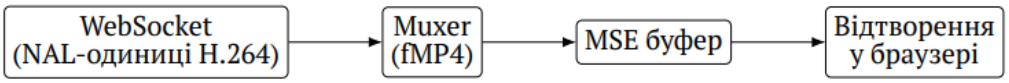

Рисунок 1.2 – Схема інкапсуляції H.264 у fMP4 для відтворення через MSE

Щоб такі дані відтворювались у браузері, на клієнті працює мультиплексор (muxer), який «на льоту» формує fMP4-фрагменти з `moof+mdat` та додає необхідні часові мітки [7].

Додатково архітектурно доцільним є використання Server-Sent Events (SSE). Це механізм односпрямованої доставки повідомлень від сервера до клієнта поверх довгоживучого HTTP-з'єднання з типом `text/eventstream`. SSE зручно використовувати для легких текстових повідомлень heartbeat, станів, попереджень і операційних метрик [9]. Такий підхід ізолює критичний медіатрафік (WebSocket) від контрольних повідомлень (SSE), зменшує взаємний вплив черг і робить систему простішою для масштабування.

1.6 Архітектура конвеєрів обробки (Video Pipelines)

Поняття pipeline походить від «трубопроводу» – це впорядкована система, у якій матеріал рухається послідовно крізь окремі ділянки. У комп'ютерних системах pipeline трактують як ланцюг елементів обробки, де вихід одного етапу є входом наступного, а між етапами часто присутні буфери [10, 11].

У відео pipeline – це структурована послідовність компонентів, які перетворюють контент від джерела до глядача, забезпечуючи оптимізацію для передачі, зберігання та відтворення. До типових елементів належать джерела, транскодування, пакування та доставка через CDN [11].

Flussonic описує відеоконвеєр як повний шлях «від об'єктива до очей», де присутні захоплення сирого відео, компресія, пакування в контейнер, доставка, розпакування, декодування і відтворення. Такий ланцюг може бути простим у випадку IP-камери або складним у сервісах із багатьма етапами обробки [10].

Конвеєр моделюють як спрямований ациклічний граф (DAG , де кожен вузол виконує одну спеціалізовану дію Source (захоплення), конвертацію кольору, кодування, маршрутизацію або Sink (рис. 1.3) [10, 11].

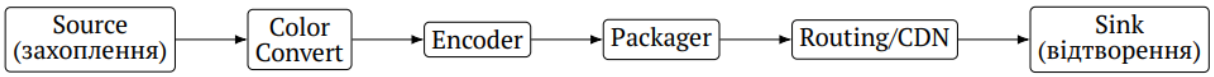

Рисунок 1.3 –Схема лінійного відеоконвеєра як DAG із послідовних етапів

Керування буфером реалізується через потокобезпечні черги між вузлами. Наприклад, у GStreamer елемент queue створює окремий потік, відокремлює етапи та акумулює буфери до заданих лімітів, щоб повільні операції не блокували захоплення кадрів [10].

1.7 Постановка завдань на кваліфікаційну роботу бакалавра

Значимість вирішуваного завдання полягає в тому, що система шифрування відеопотоків і метаданих є критично важливою складовою сучасної інфраструктури охорони об’єктів. Від якості її проектування залежать своєчасність реагування на події, захищеність архівів і потоків реального часу, коректність ідентифікації джерел даних, а також можливість безпечного використання хмарних сервісів у сфері фізичної безпеки. Тому розроблення архітектури захищеного передавання відеоданих для хмарної системи моніторингу є актуальним і практично значущим завданням.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

– розглянути базові властивості відеоданих, параметри відеопотоків, принципи контейнеризації та особливості їх потокового передавання;

– проаналізувати сучасні методи компресії, мережеві протоколи та засоби доставки мультимедійних даних у клієнт-серверних системах;

– визначити основні загрози безпеці під час передавання відеопотоків і супровідних метаданих;

– обґрунтувати вибір криптографічних алгоритмів і механізмів узгодження ключів для захисту відеоданих у реальному часі;

– опрацювати архітектуру взаємодії серверної та клієнтської частин системи;

– встановити основні функціональні та нефункціональні вимоги до об’єкта проектування;

– подати структуру програмної реалізації та оцінити результати тестування спроектованої системи.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ ТА МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Уразливості систем відеоспостереження та загрози перехоплення

Відеопотік є безперервною послідовністю пакетів і кадрів, де кожна мілісекунда важлива для якості сприйняття. Тому захист має одночасно забезпечувати конфіденційність, цілісність і автентичність без істотної деградації продуктивності.

Потокове відео вразливе до мережевих атак, оскільки передавання відбувається через багато проміжних вузлів. У випадку WebSocket важливо розрізняти `ws://` та `wss://`. За RFC 6455, без захищеного транспортного шару дані можуть передаватися у відкритому вигляді [6]. Пасивні атаки включають перехоплення трафіку (eavesdropping) та аналіз метаданих. Активні атаки є небезпечнішими, бо передбачають підміну або видалення пакетів, що призводить до десинхронізації або ін’єкції фальшивих кадрів [12]. Класичний сценарій Man-in-the-Middle (MitM) створює дилему між швидкістю та безпекою, проте використання сучасних AEAD-режимів дозволяє ефективно вирішити цю проблему.

2.2 Вибір та обґрунтування криптографічних алгоритмів

Вибір алгоритму визначається трьома критеріями швидкодія, рівень безпеки та простота інтеграції. Рекомендації NIST розділяють ролі симетричних і асиметричних примітивів асиметрія для встановлення ключів, симетрія для шифрування великих обсягів даних [13].

AES-GCM є режимом автентифікованого шифрування (AEAD), який одночасно забезпечує конфіденційність і перевірку цілісності. У NIST SP 800-38D GCM описано як поєднання лічильникового режиму (CTR) та поля Галуа для обчислення тега автентичності. Математично це описується операцією XOR для формування шифротексту.

1. Вхідні параметри (K, Nonce). Ключ K є симетричним криптографічним ключем алгоритму AES, що використовується для генерації ключового потоку та формування автентифікаційного тега. Nonce (number used once) унікальне значення для кожної операції шифрування, яке гарантує відсутність повторного використання послідовності лічильників Ctr_i.

2. Plaintext $P_1, P_2, ..., P_n$. Вхідне повідомлення подається як бітова послідовність, що складається з n блоків. Це може бути корисне навантаження відеокадру або його фрагмент.

3. Розбиття на 128-бітні блоки. Алгоритм AES оперує блоками довжиною 128 біт, вхідне повідомлення розбивається на послідовність блоків (2.1):

$$P = (P_1, P_2, ..., P_n), \tag{2.1}$$

де P – послідовність блоків відкритого тексту.

Якщо довжина повідомлення не кратна 128 бітам, останній блок обробляється у потоковому режимі CTR без необхідності доповнення (padding).

4. Генерація ключового потоку (CTR-потік). Ключовий потік формується шляхом шифрування послідовності лічильників (2.2):

$$S_i = E_K(CTR_i), \tag{2.2}$$

де S_i – і-й блок ключового потоку;

E_K – функція шифрування алгоритму AES під ключем K;

CTR_i – і-й лічильник кожне значення CTR_i формується на основі nonce і збільшується для кожного наступного блоку.

5. Операція XOR та формування шифротексту. Конфіденційність даних досягається накладанням ключового потоку на блоки відкритого тексту (2.3):

$$C_i = P_i \oplus S_i, \quad (2.3)$$

де C_i – і-й блок шифротексту;

P_i – і-й блок відкритого тексту;

S_i – і-й блок ключового потоку.

У результаті формується послідовність блоків шифротексту (2.4):

$$C = (C_1, C_2, ..., C_n), \quad (2.4)$$

де C – послідовність блоків шифротексту.

Дешифрування виконується аналогічною операцією (2.5):

$$P_i = C_i \oplus S_i. \quad (2.5)$$

де P_i – відновлений і-й блок відкритого тексту.

6. Додаткові автентифіковані дані (AAD). AAD (Additional Authenticated Data) – це службова інформація, яка не підлягає шифруванню, але включається до процесу автентифікації.

7. Для забезпечення цілісності та автентичності використовується універсальна хеш-функція GHASH, яка виконує поліноміальну акумуляцію блоків AAD і шифротексту у скінченному полі (2.6):

$$GF(2^{128}), \quad (2.6)$$

де $GF(2^{128})$ – скінченне поле, у якому виконується множення GHASH.

Ітераційне обчислення здійснюється за виразом (2.7):

$$X_i = (X_{i-1} \oplus Y_i) \cdot H, \quad (2.7)$$

де X_i – проміжний стан GHASH після і-го кроку;

X_{i-1} – попередній стан GHASH;

Y_i – черговий блок даних (AAD або ciphertext);

$H = E_K(0^{128})$ – хеш-підключ GHASH.

8. Формування автентифікаційного тега. Фінальний тег автентифікації обчислюється із виразу (2.8):

$$T = E_K(J_0) \oplus GHASH. \quad (2.8)$$

де T – фінальний тег автентифікації;

J_0 – початкове значення лічильника, сформоване на основі nonce.

Отриманий тег T передається разом із шифротекстом і використовується отримувачем для перевірки цілісності та автентичності повідомлення.

Для відеопотоку це означає, що кожен фрагмент/кадр може шифруватися незалежно з формуванням власного authentication tag. На стороні клієнта невірний тег одразу сигналізує про пошкодження або підміну даних у мережі.

Важливо, що GCM не вимагає паддінгу блоків у класичному сенсі, що спрощує роботу з довільними розмірами пакетів.

2.3 Механізми узгодження ключів сеансу за протоколом ECDH

Безпека симетричного шифрування AES-GCM принципово залежить від конфіденційності сеансового ключа та унікальності вектора ініціалізації. Основна криптографічна проблема, що виникає при побудові захищеного каналу між клієнтом і сервером, полягає у тому, що сторони змушені погодити спільний секрет через відкритий (потенційно перехоплюваний) канал зв’язку. Вирішення

цієї проблеми реалізується засобами протоколу ECDH (Elliptic Curve Diffie-Hellman) асиметричного криптографічного алгоритму, що дозволяє двом сторонам незалежно обчислити однаковий спільний секрет без його явної передачі.

Застосування ECDH у цій архітектурі зумовлено кількома чинниками значно меншим розміром ключів порівняно з RSA при еквівалентному рівні захисту, низькими обчислювальними витратами на стороні клієнта (що є критичним для мобільних та вбудованих пристроїв) та підтримкою властивості Perfect Forward Secrecy (PFS) у варіанті ECDHE. Таким чином, ECDH виступає криптографічним фундаментом, на якому будується весь механізм захищеного обміну даними між вузлами [12].

2.4 Математичні основи еліптичних кривих

На відміну від класичного алгоритму Діффі-Геллмана (DH), стійкість якого базується на складності задачі дискретного логарифмування у групі цілих чисел за простим модулем, ECDH використовує алгебраїчну структуру еліптичних кривих над скінченними полями.

Еліптична крива над полем F_p визначається рівнянням (2.9):

$$y^2 \equiv x^3 + ax + b \ \ |p|, \quad \text{де } 4a^3 + 27b^2 \neq 0 \ |p| \tag{2.9}$$

Умова $4a^3 + 27b^2 \neq 0$ виключає наявність сингулярних точок, що є необхідним для коректного визначення групової операції. Множина точок кривої разом з операцією додавання утворює абелеву групу, закриту щодо цієї операції. Скалярне множення точки G на ціле число k визначається як k-разове послідовне додавання точки до самої себе і позначається як $P = k \cdot G$.

Криптографічна стійкість ECDH ґрунтується на обчислювальній складності задачі дискретного логарифму на еліптичній кривій (ECDLP) маючи

точки G (публічний генератор групи) та $P = d \cdot G$, знайти скаляр d за поліноміальний час обчислювально неможливо.

Найкращі відомі алгоритми розв'язання ECDLP (Pohlig–Hellman, Pollard's rho) мають субекспоненційну складність $O(\sqrt{n})$, де n – порядок групи, тоді як для класичного DH існують субекспоненційні атаки (алгоритм числового поля), що знижують фактичний рівень безпеки.

Для реалізації обрано стандартизовану криву NIST P-256 (secp256r1) з такими параметрами:

– простий модуль: $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$;

– порядок групи: $n \approx 1{,}158 \times 10^{77}$;

– довжина ключа: 256 біт.

За оцінками NIST та ECRYPT, крива P-256 забезпечує рівень безпеки 128 біт, що еквівалентно стійкості 3072-бітного RSA або 256-бітного симетричного шифру AES. Такий рівень достатній для захисту конфіденційних даних на горизонті понад 20 років при поточному стані обчислювальних технологій. Разом із тим довжина публічного ключа складає лише 64 байти (дві координати по 32 байти; у стиснутому представленні – 33 байти), що мінімізує мережевий трафік при рукостисканні та знижує затримку встановлення з'єднання [13].

2.5 Алгоритм узгодження сеансового ключа (ECDHE)

Реалізовано ефемерну версію протоколу – ECDHE (Elliptic Curve Diffie-Hellman Ephemeral), що відрізняється від базового ECDH генерацією нових випадкових скалярів для кожного сеансу. Відмінність полягає у тому, що ефемерні закриті ключі не зберігаються після завершення сеансу, що гарантує властивість Perfect Forward Secrecy (PFS) компрометація довгострокових ключів не дозволяє розшифрувати раніше записаний трафік, оскільки сеансові ключі для кожного з'єднання були унікальними і вже знищені. Процедура ECDHE рукостискання виконується у такій послідовності:

– клієнт генерує криптографічно стійке випадкове число $d_A \in [1, n - 1]$ та обчислює публічну точку $Q_A = d_A \times G$;

– сервер генерує криптографічно стійке випадкове число $d_B \in [1, n - 1]$ та обчислює публічну точку $Q_B = d_B \times G$;

– сторони обмінюються публічними точками Q_A та Q_B по відкритому каналу;

– клієнт обчислює спільну точку: $S = d_A \times Q_B = d_A \times (d_B \times G)$;

– сервер обчислює спільну точку: $S = d_B \times Q_A = d_B \times (d_A \times G)$.

Завдяки комутативності скалярного множення на еліптичній кривій виконується тотожність (2.10):

$$d_A \times (d_B \times G) = d_B \times (d_A \times G) = S. \quad (2.10)$$

Обидві сторони отримують ідентичний результат, не передаючи жодного секретного значення каналом зв’язку. Потенційний зловмисник, перехопивши Q_A та Q_B, не може обчислити S без вирішення задачі ECDLP, обчислювальна складність якої для кривої P-256 унеможливлює атаку навіть при використанні сучасних кластерних обчислень.

Для забезпечення коректної роботи протоколу публічні точки перевіряються на належність до кривої та на те, що вони не є точкою у нескінченності (нейтральний елемент групи), що захищає від атаки малою підгрупою (small subgroup attack).

2.6 Функція виведення ключа (HKDF)

Координата x спільної точки S не може бути безпосередньо використана як ключ для AES-GCM з кількох причин.

По-перше, вона може мати нерівномірний розподіл ентропії внаслідок математичної структури кривої.

По-друге, один і той самий спільний секрет може бути помилково використаний у різних криптографічних контекстах, що створює ризик крос-протокольних атак.

По-третє AES-256-GCM потребує рівно 256 біт ключа з максимально рівномірним розподілом, якого «сира» координата точки не гарантує.

Для перетворення спільного секрету у криптографічно придатний ключ застосовується алгоритм HKDF (HMAC-based Key Derivation Function), стандартизований у RFC 5869. HKDF реалізує двоетапний процес [12].

Етап Extract – конденсація вхідного матеріалу у псевдовипадковий ключ (PRK) формується з (2.11):

$$PRK = HMAC\text{-}SHA256(salt, IKM). \quad (2.11)$$

де IKM (Input Keying Material) – координата x точки S;

salt – публічно відомий випадковий байтовий рядок, що перешкоджає попередньо обчисленим атакам.

Операція HMAC розмазує ентропію вхідного матеріалу рівномірно по вихідному 256-бітному рядку.

Етап Expand – розширення PRK до цільового ключа з прив’язкою до контексту (2.12):

$$OKM = T_1 \parallel T_2 \parallel \ldots, \quad T_i = HMAC\text{-}SHA256(PRK, T_{i-1} \parallel info \parallel i). \quad (2.12)$$

де info – рядок контекстної інформації, що прив’язує отриманий ключ до конкретного призначення (ідентифікатор протоколу, версія, напрям передачі).

Завдяки полю info виключається можливість ненавмисного використання ключа в іншому протоколі або напрямку сеансу, навіть якщо вхідний IKM збігається.

Результатом є 256-бітний ключ K з рівномірним розподілом ентропії, придатний для режиму AES-256-GCM. Застосування HKDF також гарантує незалежність виведених ключів компрометація одного з них не дозволяє відновити інші ключі, виведені з того ж PRK з різними значеннями info.

2.7 Алгоритм взаємодії клієнта та сервера

Повний цикл захищеного з'єднання реалізується у п'яти послідовних етапах. На кроці 1 обидві сторони незалежно генерують ефемерні пари ключів, використовуючи криптографічно стійкий генератор псевдовипадкових чисел (CSPRNG).

На кроці 2 відбувається відкритий обмін публічними точками Q_A та Q_B; при цьому обмін може відбуватися без шифрування, оскільки перехоплення публічних точок не дозволяє відтворити спільний секрет без вирішення ECDLP.

На кроці 3 кожна зі сторін незалежно обчислює спільну точку S завдяки математичній тотожності $d_A \times Q_B = d_B \times Q_A$ обидва обчислення дають однаковий результат.

На кроці 4 координата x точки S перетворюється на симетричний ключ через HKDF, що забезпечує рівномірний розподіл ентропії та прив'язку ключа до контексту протоколу.

На кроці 5 встановлюється зашифрований канал на основі AES-256-GCM, при цьому кожне повідомлення супроводжується тегом автентифікації GCM (128 біт), що захищає від модифікації переданих даних.

Після завершення сеансу ефемерні закриті ключі d_A та d_B безповоротно знищуються з пам'яті, що реалізує властивість PFS навіть якщо зловмисник у майбутньому отримає доступ до довгострокових облікових даних будь-якої зі сторін, він не зможе відновити сеансовий ключ K та розшифрувати раніше перехоплений трафік. Загальна криптографічна стійкість описаного механізму оцінюється на рівні 128 біт безпеки, що відповідає галузевим стандартам для захисту конфіденційних даних згідно з рекомендаціями NIST SP 800-57 [13].

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Архітектурне рішення та структура

Розроблена багаторівнева клієнт-серверна архітектура забезпечує захоплення, криптографічний захист та передачу відеоданих у режимі реального часу із застосуванням наскрізного шифрування (End-to-End Encryption). Програмне рішення базується на інтеграції мультимедійного фреймворку GStreamer із мовою програмування Rust, що дозволяє досягти детермінованого контролю над розподілом пам’яті та мінімізувати накладні витрати середовища виконання [10].

Структурна організація розробки передбачає взаємодію двох автономних складових:

– `stream-server` (Backend): виконує захоплення відеопотоку через GStreamer-пайплайн, ініціює криптографічну обробку даних та здійснює оркестрацію WebSocket-сесій;

– `stream-client` (Frontend): приймає зашифрований бінарний потік, дешифрує його засобами Web Crypto API та відтворює за допомогою H.264-декодера.

Внутрішньосерверна маршрутизація потоку побудована на базі асинхронного broadcast-каналу (бібліотека Tokio). Цей механізм гарантує неблокуючу ретрансляцію кадрів від єдиного джерела (GStreamer AppSink) до довільної кількості WebSocket-підключень.

Методика захисту інформації ґрунтується на пофреймовому шифруванні за стандартом AES-256-GCM. Для кожного кадру детерміновано генерується унікальний вектор ініціалізації (nonce) на основі його порядкового індексу. Після криптографічних перетворень оброблені дані інкапсулюються у бінарні WebSocket-фрейми для подальшої маршрутизації до кінцевих споживачів.

3.2 Аналіз можливостей застосування принципу нульового копіювання

Принцип нульового копіювання (zero-copy) передбачає передачу даних між підсистемами без проміжного копіювання вмісту буферів, що дозволяє суттєво знизити навантаження на процесор та підсистему пам’яті.

Застосування даного принципу в контексті автентифікованого шифрування з додатковими даними (AEAD) наштовхується на структурні обмеження, зумовлені семантикою самого алгоритму.

Алгоритм AES-256-GCM, що належить до класу AEAD-схем, за своєю природою є розширюючим перетворенням вихідний буфер завжди перевищує вхідний на фіксовану величину тегу автентифікації Poly1305, що становить 16 байт. Це означає, що результат шифрування не може бути розміщений у тій самій області пам’яті без попереднього резервування додаткового простору.

Виділення пам’яті GStreamer (`GstAllocator`) надає кодеку `x264enc` буфер, розмір якого відповідає рівно розміру стисненого NAL-юніту H.264 без будь-якого резерву.

Відсутність зарезервованого простору унеможливлює in-place шифрування і вимагає копіювання даних у новий буфер, у якому такий резерв передбачений заздалегідь.

Теоретично усунення єдиної копії можливе двома шляхами:

– перший передбачає реалізацію власного `GstAllocator`, який при кожному запиті від кодека виділяє N + 28 байт (N + 12 байт під nonce та 16 байт під тег), що дозволило б виконувати шифрування безпосередньо у GStreamer-буфері методом `seal_in_place_append_tag` без жодних проміжних копій;

– другий підхід ґрунтується на застосуванні механізму розсіяного введення-виведення (scatter-gather I/O). Його ключова ідея полягає у тому, що AES-GCM дозволяє отримати тег автентифікації як окремий артефакт, незалежний від буфера шифротексту через виклик

`encrypt_in_place_detached`. Це усуває необхідність суміжного розміщення шифротексту і тегу в єдиній неперервній ділянці пам’яті.

Відтак WebSocket-заголовок, nonce, шифротекст у GStreamer-буфері та тег описуються як масив незалежних дескрипторів `IoSlice` і передаються у мережевий стек єдиним системним викликом `writev`, що на рівні ядра реалізується через `sendmsg` зі scatter-gather DMA. Злиття фрагментів у проміжний буфер у просторі користувача не відбувається, а коректність структури фрейму забезпечується протокольним інваріантом одержувач апріорно знає, що останні 16 байт корисного навантаження є тегом.

Проте практична реалізація цього підходу наштовхується на архітектурне обмеження публічний API бібліотеки tokio-tungstenite не надає доступу до `vectored write` на рівні WebSocket-фреймування, оскільки бібліотека інкапсулює запис у власному буферизованому шарі, що знімає можливість передачі масиву `IoSlice` безпосередньо до системного виклику. Обхід цього обмеження вимагає ручної реалізації WebSocket-протоколу поверх сирого TCP-з’єднання, що тягне за собою самостійну підтримку стану машини з’єднання, обробку керуючих фреймів (ping, pong, close) та забезпечення відповідності специфікації RFC 6455 суттєво збільшуючи складність і поверхню помилок.

Також підхід власного `GstAllocator` має критичний недолік апаратні та оптимізовані програмні енкодери вимагають використання власних апаратно залежних пулів пам’яті з жорстким вирівнюванням. Інтеграція стороннього алокатора призводить до помилок узгодження форматів (negotiation failures) на рівні пайплайну або до примусового копіювання даних драйвером.

3.3 Програмна реалізація серверної частини

Для роботи з мультимедіа використано бібліотеку `gstreamer-rs`. Конвеєр налаштований на мінімальну затримку за допомогою параметрів кодування `zerolatency` та `speed-preset=ultrafast` (лістинг 3.1).

Лістинг 3.1 – Ініціалізація GStreamer конвеєра в Rust

```
let pipeline_str = format!(
        "{} ! queue max-size-buffers=1 leaky=downstream !
            videoconvert !
            x264enc tune=zerolatency bitrate=2000 speed-preset=ultrafast
key-int-max=30 !
            video/x-h264, profile=baseline, stream-format=byte-stream !
            queue max-size-buffers=1 leaky=downstream !
            appsink name=sink emit-signals=true max-buffers=1 drop=true",
        pipeline_source
    );
let pipeline = gstreamer::parse::launch(&pipeline_str)
        .expect("Failed to create pipeline")
        .dynamic_cast::<gstreamer::Pipeline>()
        .expect("Could not cast to Pipeline");
```

кінець лістингу 3.1

Після ініціалізації конвеєра необхідно отримати з нього закодовані відеокадри для подальшого використання чи трансляції. Для цього застосовується елемент `appsink`, який дозволяє застосунку перехоплювати медіадані безпосередньо з графа GStreamer.

Програма знаходить елемент appsink за його іменем та реєструє функцію зворотного виклику (callback) на подію `new_sample`.

У цій функції кожен новий кадр витягується з конвеєра, після чого його бінарне представлення загортається у структуру `GstPacket` та асинхронно відправляється іншим компонентам системи за допомогою каналу `tokio::sync::broadcast::Sender` (лістинг 3.2).

Для забезпечення стабільної роботи та коректної реакції на позаштатні ситуації реалізовано моніторинг стану через шину повідомлень GStreamer (bus).

Головний потік функції блокується в циклі, який очікує на повідомлення від шини. Якщо надходить сигнал про помилку (Error) або (Eos – End of Stream),

цикл переривається, і конвеєр коректно зупиняється шляхом переведення його у стан `Null`, що звільняє виділені під нього ресурси (лістинг 3.3).

Лістинг 3.2 – Налаштування зворотного виклику для отримання кадрів

```
let sink = pipeline
        .by_name("sink")
        .expect("Sink not found")
        .dynamic_cast::<gst_app::AppSink>()
        .expect("Sink is not an AppSink!");
    sink.set_callbacks(
        gst_app::AppSinkCallbacks::builder()
            .new_sample(move |sink| {
                let sample = sink.pull_sample().map_err(|_|
gstreamer::FlowError::Eos)?;
                if let Some(buffer) = sample.buffer() {
                    println!("Video Frame: {} bytes", buffer.size());
                    let _ = tx.send(GstPacket {
                        buffer: buffer.to_owned(),
                    });
                } Ok(gstreamer::FlowSuccess::Ok)})build(),
    );
```

кінець лістингу 3.2

Лістинг 3.3 – Запуск конвеєра та обробка повідомлень шини

```
if let Err(err) = pipeline.set_state(gstreamer::State::Playing) {
        tracing::error!("Failed to start pipeline: {}", err);
        return;}
    let bus = pipeline.bus().unwrap();
    for msg in bus.iter_timed(gstreamer::ClockTime::NONE) {
        use gstreamer::MessageView;
        match msg.view() {
            MessageView::Eos(..) => break,
            MessageView::Error(err) => {
```

```
                eprintln!("GST Error: {}", err.error());
                break;
            }
            _ => (),
        }
    }
    pipeline.set_state(gstreamer::State::Null).unwrap();
```

кінець лістингу 3.3

Криптографічна логіка базується на бібліотеці ring. Частиною є процес узгодження ключів за протоколом ECDH з подальшим виведенням сесійного ключа через HKDF (лістинг 3.4).

Лістинг 3.4 – Узгодження ключа за допомогою бібліотеки ring

```
let shared_secret =
            agreement::agree_ephemeral(server_private_key,
&peer_public_key, |key_material| {
                let salt = hkdf::Salt::new(hkdf::HKDF_SHA256, &[]);
                let prk = salt.extract(key_material);
                let info = [b"video-key-exchange".as_slice()];
                let okm = prk.expand(&info, hkdf::HKDF_SHA256).unwrap();
                let mut key = [0u8; 32];
                okm.fill(&mut key).unwrap();
                key
            })
            .map_err(|_| (StatusCode::BAD_REQUEST, "Invalid client public
key".into()))?;
```

кінець лістингу 3.4

Після успішного отримання спільного секрету, він використовується як ключ для симетричного алгоритму AES-256-GCM. За допомогою цього ключа сервер шифрує основний симетричний ключ відеопотоку (`key_bytes`),

генеруючи для цього криптографічно стійкий вектор ініціалізації (IV). Зашифрований ключ відеопотоку разом із відкритим ключем сервера та вектором ініціалізації повертаються клієнту у форматі `Base64` (лістинг 3.5).

Лістинг 3.5 – Шифрування ключа відеопотоку за допомогою AES-256-GCM

```
let cipher = Aes256Gcm::new_from_slice(&shared_secret).map_err(|_| {(
     StatusCode::INTERNAL_SERVER_ERROR,
     "Cipher initialization failed".into(),)})?;
let mut iv_bytes = [0u8; 12];
    rng.fill(&mut iv_bytes).map_err(|_| {
        (StatusCode::INTERNAL_SERVER_ERROR,"Failed to generate
IV".into(),
    )})?;
let nonce = Nonce::from_slice(&iv_bytes);
let encrypted_video_key = cipher.encrypt(nonce, key_bytes).map_err(|_| {
        (StatusCode::INTERNAL_SERVER_ERROR,
            "Encryption failed".into(),)
    })?;
```

кінець лістингу 3.5

Для криптографічного захисту медіаданих реалізовано асинхронну функцію `cypher_video_data`, яка отримує кадри GStreamer через канал типу `broadcast::Receiver<GstPacket>` і публікує зашифровані бінарні пакети через `broadcast::Sender<Bytes>`. Вибір асинхронної моделі виконання зумовлений необхідністю уникнення блокування потоку при очікуванні нового кадру.

На етапі ініціалізації з переданого масиву байт `key_bytes` конструюється екземпляр `LessSafeKey` бібліотеки ring з алгоритмом AES256 GCM.

Розмір тегу автентифікації `tag_len` отримується програмно через `aead::AES_256_GCM.tag_len()` і становить 16 байт – ця величина

використовується для точного попереднього розрахунку розміру вихідного буфера, що виключає реалокацію під час шифрування.

Цикл обробки кадрів реалізований з явною обробкою трьох станів каналу. При отриманні `RecvError::Lagged(n)` функція фіксує кількість пропущених кадрів засобами tracing і продовжує роботу без переривання потоку – така поведінка є коректною для систем реального часу, де актуальність даних важливіша за повноту. При отриманні `RecvError::Closed` цикл завершується, що відповідає штатному завершенню роботи GStreamer-пайплайну.

Для кожного отриманого пакету виконується відображення GStreamer-буфера у режимі читання через `map_readable()`, що повертає захищений дескриптор без копіювання даних. Вектор ініціалізації (nonce) формується детерміновано з 64 бітного лічильника `sequence_number` функцією `create_nonce_bytes`, яка розміщує беззнакове ціле у старших 8 байтах 12-байтного масиву. Така схема гарантує унікальність nonce в межах одного сеансу, оскільки при частоті 30 кадрів/с простір лічильника `u64` вичерпується приблизно через 19,5 млрд років, що виключає загрозу повторного використання nonce.

Вихідний буфер формується з розрахованою ємністю 12 + |frame| + 16 байт через `BytesMut::with_capacity`, що забезпечує єдину алокацію без подальших реалокацій. Послідовно записуються nonce (12 байт) та необроблені дані кадру остання операція є єдиним копіюванням у межах функції, зумовленим семантичною несумісністю між GStreamer-буфером і вимогами AEAD-шифрування щодо суміжного розміщення відкритого тексту та резерву під тег. Після цього через `split_off(12)` відокремлюється частина буфера, що містить відкритий текст, без копіювання операція лише переміщує внутрішній курсор `BytesMut`.

Метод `seal_in_place_append_tag` виконує шифрування безпосередньо у виділеному буфері та дописує 16-байтний тег автентифікації Poly1305 в заздалегідь зарезервований простір. Повторне об’єднання частин буфера через `unsplit` також не спричиняє копіювання. Фінальний виклик `split().freeze()` передає право власності на алокацію структурі `Bytes` через

атомарний лічильник посилань (Arc) без копіювання, після чого пакет публікується у канал.

Структура вихідного пакету має вигляд [nonce: 12][ciphertext: N][tag: 16], де розмір N відповідає розміру оригінального кадру (лістинг 3.6).

Лістинг 3.6 – Потокове шифрування відеоданих з використанням лічильника

```
pub async fn cypher_video_data(
    key_bytes: &[u8],
    mut rx: broadcast::Receiver<GstPacket>,
    tx: broadcast::Sender<Bytes>,
) {
    let cipher_key = LessSafeKey::new(UnboundKey::new(&aead::AES_256_GCM,
key_bytes).unwrap());
    let tag_len = aead::AES_256_GCM.tag_len(); // 16 byte
    let mut sequence_number: u64 = 0;
    loop {
        let packet = match rx.recv().await {
            Ok(p) => p,
            Err(RecvError::Lagged(n)) => {
                tracing::warn!("Cipher dropped {} frames", n);
                continue;
            } Err(RecvError::Closed) => break,};
        let map = packet.buffer.map_readable().expect("Failed to map
buffer");
        let raw_bytes = map.as_slice();
        let nonce_bytes = create_nonce_bytes(sequence_number);
        let nonce =
Nonce::try_assume_unique_for_key(&nonce_bytes).unwrap();
        let mut buf = BytesMut::with_capacity(12 + raw_bytes.len() +
tag_len);
        buf.put_slice(&nonce_bytes);
        buf.put_slice(raw_bytes);
```

```
        let mut payload = buf.split_off(12);
        cipher_key
            .seal_in_place_append_tag(nonce, Aad::empty(), &mut payload)
            .unwrap();
        buf.unsplit(payload);
        tx.send(buf.split().freeze()).ok();sequence_number += 1;}
}
```

кінець лістингу 3.6

3.4 Програмна реалізація клієнтської частини

Клієнтська частина використовує Web Crypto API для виконання криптографічних операцій на рівні браузера. Це забезпечує високу швидкість обробки завдяки апаратному прискоренню, і захист криптографічного матеріалу.

Узгодження ключів та отримання доступу. Для того, щоб отримати основний ключ дешифрування відео, клієнт виконує асиметричне узгодження за протоколом ECDH. У функції `getCryptoKey` генерується тимчасова пара ключів клієнта на еліптичній кривій P-256, після чого відкритий ключ кодується у формат `Base64` та передається на сервер (лістинг 3.7).

Лістинг 3.7 – Генерація клієнтських ключів ECDH та запит до сервера

```
const keyPair = await window.crypto.subtle.generateKey(
{ name: "ECDH", namedCurve: "P-256" }, true,["deriveKey", "deriveBits"]);
const clientPubKey = await window.crypto.subtle.exportKey("raw",
keyPair.publicKey);
const clientPubBase64 = btoa(String.fromCharCode(...new
Uint8Array(clientPubKey)));
const response = await fetch("http://127.0.0.1:8080/api/exchangekey", {
method: "POST", headers: { "Content-Type": "application/json" }, body:
JSON.stringify({ client_public_key_base64: clientPubBase64 }),
});
```

кінець лістингу 3.7

Отримавши у відповідь відкритий ключ сервера, клієнт обчислює спільний секрет. Аналогічно до серверної логіки, цей секрет не використовується напряму, а пропускається через функцію формування ключа HKDF для виведення ключа шифрування ключів (KEK). За допомогою KEK та вектора ініціалізації клієнт розшифровує закритий симетричний ключ відеопотоку (лістинг 3.8).

Лістинг 3.8 – Виведення KEK через HKDF та розшифрування відеоключа

```
const sharedSecretBits = await window.crypto.subtle.deriveBits(
{ name: "ECDH", public: serverPubKey }, keyPair.privateKey,256);
const hkdfBaseKey = await window.crypto.subtle.importKey("raw",
sharedSecretBits, { name: "HKDF" }, false, ["deriveKey "]);
const kek = await window.crypto.subtle.deriveKey( { name: "HKDF", hash:
"SHA-256", salt: new Uint8Array(0), info: new
TextEncoder().encode("video-key-exchange")}, hkdfBaseKey, { name: "AES-
GCM", length: 256 }, false, ["decrypt"]);
const videoAesKeyRaw = await window.crypto.subtle.decrypt({ name: "AES-
GCM", iv: ivBytes }, kek, encryptedKeyBytes
);
```

кінець лістингу 3.8

Для захисту від XSS-атак розшифрований ключ потоку імпортується в середовище браузера з параметром `extractable: false`. Це гарантує, що навіть у разі виконання шкідливого скрипту, сирі байти ключа не зможуть бути викрадені з пам’яті а кешування об’єкта CryptoKey у змінній для уникнення повторних мережевих запитів та криптографічних обчислень при подальшій роботі з потоком (лістинг 3.9).

Лістинг 3.9 – Безпечний імпорт сесійного ключа у Web Crypto API

```
cryptoKeyCache = await window.crypto.subtle.importKey(
"raw", videoAesKeyRaw, "AES-GCM", false, ["decrypt"]);
```

кінець лістингу 3.9

Отримані через WebSocket бінарні дані проходять дешифрування за допомогою кешованого об’єкта `cryptoKeyCache` та передаються в бібліотеку jmuxer для подальшої інкапсуляції у fMP4 контейнер. Формування готового медіапотоку дозволяє передати розшифровані кадри безпосередньо у HTML5-елемент `<video>` з мінімальною затримкою.

3.5 Повна архітектура

Архітектура поєднує серверний, транспортний, криптографічний та клієнтський рівні в єдиний конвеєр обробки відеоданих.

На серверному рівні виконується захоплення відеопотоку, його попередня підготовка, кодування та фрагментація на окремі пакети, придатні для передавання мережею. Далі кожен фрагмент проходить криптографічну обробку, після чого інкапсулюється у формат повідомлень, який підтримує швидке доставлення до клієнта.

Транспортна та безпекова підсистеми функціонують узгоджено.Після встановлення з’єднання сторони виконують узгодження параметрів сеансу і формують спільний криптографічний контекст. Для захисту медіаданих використовується симетричне шифрування з перевіркою цілісності, а службові повідомлення застосовуються для передавання відкритих параметрів, керувальних команд і даних, необхідних для синхронізації клієнта із сервером.

Клієнтська частина завершує архітектурний цикл, виконуючи приймання, дешифрування, буферизацію та відтворення відео у браузері.

Після одержання зашифрованих фрагментів клієнт перевіряє їх цілісність, відновлює корисні дані та передає їх до підсистеми відтворення, яка формує безперервний медіапотік для користувача (рис. 3.1). Архітектура передбачає мінімізацію затримок між етапами обробки, ізоляцію криптографічних ключів у клієнтському середовищі та можливість заміни окремих модулів без зміни загальної логіки.

Рисунок 3.1 – Архітектурна схема взаємодії компонентів

3.6 Тестування та оцінка результатів

Для оцінки проведено функціональне тестування та вимірювання споживання ресурсів у реальних умовах роботи. Тестове середовище розгорнуто у вигляді двох ізольованих Docker-контейнерів stream-client та stream-server. Обидва контейнери функціонують у межах спільного ліміту оперативної пам’яті 2,763 ГіБ.

Апаратно-програмне середовище тестування CPU – AMD Ryzen 5 5500U, RAM – 8 ГіБ, ОС – WSL Ubuntu 24.04.3 LTS, Docker Engine 26.x.

Джерелом під час проведення тестів слугувала інтегрована камера ноутбука моніторинг ресурсів здійснювався засобами команди `docker stats`,

яка забезпечує отримання метрик у реальному часі без втручання у роботу процесів. Отримані показники наведено у таблиці 3.1.

Таблиця 3.1 – Показники споживання ресурсів контейнерів під час роботи

Контейнер	CPU %	Використання RAM	MEM %	NET I/O	BLOCK I/O
stream-client	0,05%	37,03 МіБ / 2,763 ГіБ	1,31%	4,59 кБ / 1,94 кБ	0 Б / 0 Б
stream-server	8,25%	18,53 МіБ / 2,763 ГіБ	0,66%	31,4 кБ / 2,38 МБ	2,91 МБ / 1,32 МБ

Навантаження на процесор з боку серверного контейнера становить 8.25%, Споживання оперативної пам’яті серверним контейнером становить лише 18,53 МіБ (0,66% від ліміту). Клієнтський контейнер демонструє мінімальне навантаження на процесор (0,05%), оскільки криптографічні операції дешифрування делеговані Web Crypto API браузера з апаратним прискоренням, а вся логіка відтворення зводиться до передачі розшифрованих байт у бібліотеку jmuxer та подальшої інкапсуляції у fMP4-контейнер. Вищий об’єм пам’яті клієнтського контейнера (37,03 МіБ) пояснюється накладними витратами середовища обслуговування статичних ресурсів.

Показники мережевого введення-виведення серверного контейнера 31,4 кБ вхідного та 2,38 МБ вихідного трафіку відповідають очікуваній асиметрії медіасистем, де вихідний потік формується безперервним відеопотоком у форматі [nonce: 12][ciphertext: N][tag: 16].

3.7 Вимірювання наскрізної затримки (End-to-End Latency) відеопотоку

Для комплексної оцінки важливим показником є наскрізна затримка (end - to - end latency) час, необхідний для захоплення кадру, його кодування, шифрування, передачі мережею, дешифрування та остаточного відтворення на стороні клієнта.

Для точного вимірювання затримки було застосовано метод візуальної фіксації еталонного часу (метод екранного секундоміра).

Методика тестування полягала у наступному:

– на екрані хост-машини запускався цифровий секундомір з мілісекундною роздільною здатністю;

– камера направлялася на екран для захоплення поточного часу секундоміра;

– здійснювалася трансляція через розроблений сервер на клієнтський додаток;

– на одному екрані одночасно відображалися секундомір та вікно клієнта з отриманою трансляцією;

– шляхом створення знімка екрана фіксувалася різниця у часі.

Наскрізна затримка розраховується за формулою:

$$\Delta t = t_{src} - t_{dst} \quad (3.1)$$

де t_{src} – час, зафіксований на секундомірі;

t_{dst} – час, відображений на відеокадрі, що пройшов через камеру та був відтворений у браузері.

Під час проведення експерименту як це видно з рисунку 3.2 було зафіксовано такі показники час становив – 53,23 с, час на трансляції – 52,98 с.

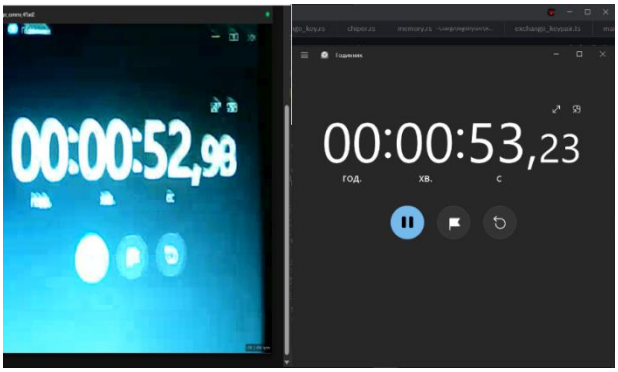

Рисунок 3.2 – Визначення затримки системи під час потокової передачі відео

Відповідно, наскрізна затримка становить:

$$\Delta t = 53{,}23 – 52{,}98 = 0{,}25 \text{ (с)} \quad (3.2)$$

Отримана затримка на 250 мілісекунд є цілком прийнятною для систем відеотрансляції у реальному часі. Враховуючи, що у цей час закладено накладні витрати на AES-256-GCM шифрування кожного пакету, перепакування у fMP4 та буферизацію у плеєрі, результат підтверджує високу швидкодію обраного стека технологій та правильність налаштувань кодека x264.

ЗАГАЛЬНІ ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

У кваліфікаційній роботі досягнуто поставленої мети, яка полягала в обґрунтуванні та проєктуванні системи шифрування відеопотоків і метаданих для хмарної системи моніторингу безпеки об’єкта.

У ході виконання роботи розглянуто базові властивості відеоданих, параметри відеопотоків, принципи контейнеризації та особливості потокового передавання. Встановлено, що для систем реального часу важливими параметрами є бітрейт, роздільна здатність, частота кадрів, структура GOP, формат контейнера та спосіб фрагментації мультимедійних даних.

Проаналізовано сучасні методи компресії, мережеві протоколи та засоби доставки мультимедійних даних у клієнт-серверних системах. Обґрунтовано доцільність використання H.264/AVC для кодування відео, WebSocket для передавання бінарних фрагментів у реальному часі та Media Source Extensions для відтворення потоку на стороні браузера.

Визначено основні загрози безпеці під час передавання відеопотоків і супровідних метаданих. До них належать перехоплення трафіку, модифікація пакетів, підміна відеофрагментів, несанкціонований доступ до потоку, повторне використання службових даних і порушення цілісності метаданих.

Обґрунтовано вибір криптографічних алгоритмів і механізмів узгодження ключів для захисту відеоданих у реальному часі. Для шифрування та контролю цілісності використано AES-GCM, для узгодження спільного секрету – E DH, а для формування сеансових ключів – H DF. Такий підхід забезпечує конфіденційність, цілісність та автентичність переданих даних.

Опрацьовано архітектуру взаємодії серверної та клієнтської частин системи. Серверна частина відповідає за захоплення, кодування, шифрування та передавання відеоданих, а клієнтська – за отримання ключів, дешифрування фрагментів, буферизацію та відтворення відеопотоку у браузері.

Встановлено основні функціональні та нефункціональні вимоги до об’єкта проєктування. Система має забезпечувати безперервне передавання відеопотоку,

низьку затримку, криптографічний захист відеоданих і метаданих, сумісність із вебтехнологіями, масштабованість і стійкість до типових мережевих загроз.

Подано структуру програмної реалізації та виконано оцінювання результатів тестування спроєктованої системи. Результати підтвердили працездатність запропонованої архітектури, можливість захищеного передавання відеофрагментів і доцільність використання розроблених рішень у хмарних системах відеоспостереження та дистанційного моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bross B., Chen J., Ohm J.-R., Sullivan G. J., Wiegand T. Overview of the Versatile Video Coding (VVC) Standard and its Applications. IEEE Transactions on Circuits and Systems for Video Technology. URL: https://shorturl.at/Hd24T (дата звернення: 18.05.2026).

2. ISO/IEC 14496-12:2022. Information technology. Coding of audio-visual objects. Part 12: ISO base media file format URL: https://shorturl.at/NTZWE (дата звернення: 18.05.2026).

3. Janusz K. та ін. Assessment of the Quality of Video Sequences Performed by Viewers at Home and in the Laboratory URL: https://www.mdpi.com/2076-3417/13/8/5025 (дата звернення: 18.05.2026).

4. Netflix Technology Blog. Bringing AV1 to more Netflix members. URL: https://netflixtechblog.com/ (дата звернення: 18.05.2026).

5. Twitch Engineering Blog. How Twitch Delivers Low-Latency Video. URL: https://blog.twitch.tv/en/tags/engineering/ (дата звернення: 18.05.2026).

6. WHATWG. WebSockets Living Standard. URL: https://websockets.spec.whatwg.org/ (дата звернення: 18.05.2026).

7. W3C ISOBMFF Byte Stream. URL: https://w3c.github.io/media-source/isobmff-byte-stream-format.html (дата звернення: 18.05.2026).

8. [MS-H264PF]: RTP Payload Format for H.264 Video Streams Extensions 2021 URL: https://shorturl.at/vVrmj (дата звернення: 18.05.2026).

9. W3C. WebTransport. Working Draft. URL: https://tinyurl.com/myxxwcp9 (дата звернення: 18.05.2026).

10. GStreamer Development Team. GStreamer Design Documentation: Pipeline Construction. URL: https://gstreamer.freedesktop.org/documentation/design (дата звернення: 18.05.2026).

11. FFmpeg Developers. FFmpeg Filtering Guide and Video Pipeline Documentation. URL: https://ffmpeg.org/documentation.html (дата звернення: 18.05.2026).

12. BSI TR-02102-1 Cryptographic Mechanisms: Recommendations and Key Lengths URL: https://shorturl.at/SMKxP (дата звернення: 18.05.2026).

13. NIST SP 800-57 Recommendation for Key Management Rev.6 2025 URL: https://csrc.nist.gov/pubs/sp/800/57/pt1/r6/ipd (дата звернення: 18.05.2026)..

14. W3C. WebRTC: Real-Time Communication in Browsers. W3C Recommendation. URL: https://www.w3.org/TR/webrtc/ (дата звернення: 18.05.2026).

15. Iyengar J., Thomson M. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. IETF, 2021 URL: https://datatracker.ietf.org/doc/rfc9000/ (дата звернення: 18.05.2026).

16. Chen Y. та ін. An Overview of Core Coding Tools in the AV1 Video Codec. IEEE Transactions on Broadcasting. 2020. URL: https://shorturl.at/pnO4o (дата звернення: 18.05.2026).

17. Sun L., Zong T., Wang S., Liu Y., Wang Y. Towards Optimal Low-Latency Live Video Streaming. IEEE/ACM Transactions on Networking. URL: https://tinyurl.com/y935upaw (дата звернення: 18.05.2026).

18. Терлецький Т. В., Кайдик О. Л. Кваліфікаційна робота : методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Інформаційні системи та технології охорони і безпеки» галузі знань 12 Інформаційні технології спеціальності 126 Інформаційні системи та технології денної та заочної форм навчання. Луцьк: ЛНТУ, 2025. 53 с.