УДК 004.056.5 **DOI:10.30837/rt.2025.2.221.09**

А.А. ТЕЛЬНОВА, Д.С. БАЛАГУРА, канд. техн. наук, В.О. ФРОЛЕНКО,
В.М. СУХОТЕПЛИЙ, С.В. ФЛОРОВ, канд. техн. наук

АНАЛІЗ ВИКОРИСТАННЯ КРИПТОПРОВАЙДЕРІВ У ПРОТОКОЛІ TLS

Вступ

Щодня кожна людина надсилає десятки повідомлень і завантажує гігабайти трафіку. При цьому достатньо велика кількість даних проходить відкритими мережами. Це накладає додаткову відповідальність на end-to-end протоколи захисту даних. Одним з наймасовіших протоколів захисту даних у мережі Інтернет є протокол HTTPS (HyperText Transfer Protocol Secure). Згідно зі звітами Google Transparency Report [1], доля використання цього протоколу суттєво збільшилась: від 40 % у 2015 р., до більш ніж 90 % відсотків у 2022–2024 рр. Загальний тренд зі збільшення частки HTTPS трафіку наведено на рис. 1. Графік демонструє узагальнену інформацію щодо доступу до сайтів через протокол HTTPS у браузері Chrome, але загальна тенденція зрозуміла.

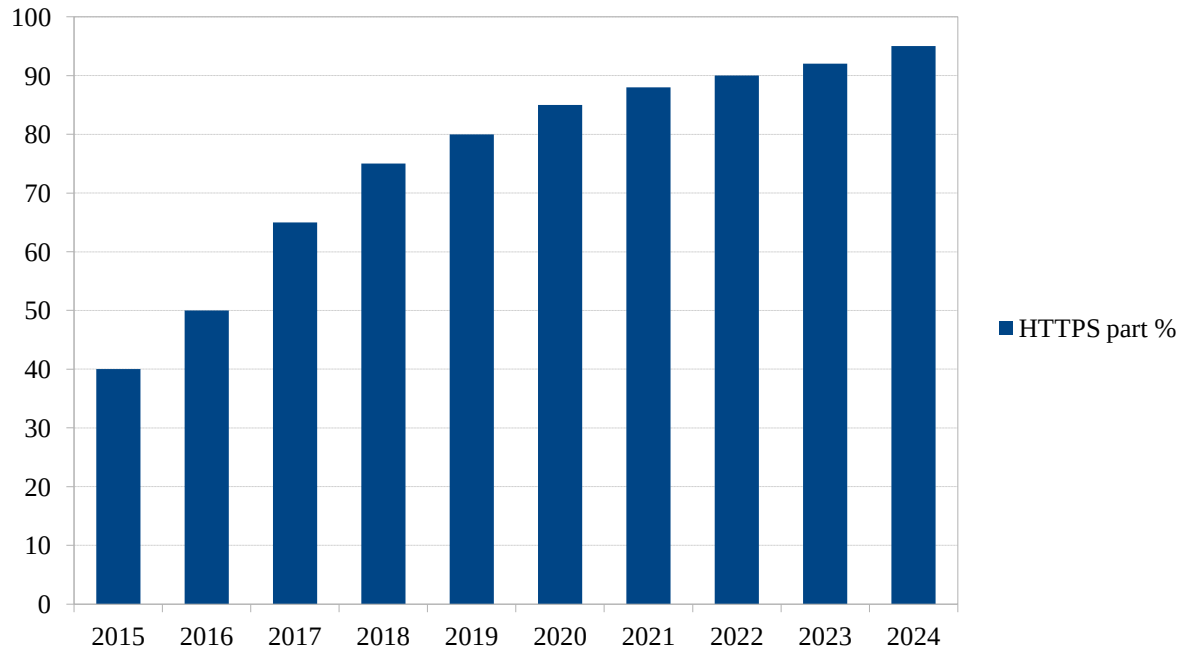

Рис. 1. Відсоток використання HTTPS протоколу

Схожа тенденція спостерігається і з шифруванням електронних повідомлень: станом на 2014 р. лише 26 – 28 % усіх вхідних і вихідних повідомлень були зашифровані відповідно. Але з того часу ситуація значно покращилася, і зараз цей показник становить майже 100 %, ми повинні постійно дбати про безпеку наших даних.

Як відомо, протокол комунікації TLS є базовим елементом протоколу HTTPS. Можна сказати, HTTPS – це HTTP over TLS. Крім того, TLS може використовуватися для забезпечення безпеки даних під час роботи й великої кількості інших протоколів, в тому числі для потреб електронної пошти, протоколів обміну файлами та інше. По суті TLS є фундаментальним елементом сучасних механізмів забезпечення безпечної передачі даних в Інтернеті.

Актуальність

Важливість TLS для захисту даних під час передачі у відкритих мережах важко переоцінити. Разом з тим, TLS, як протокол захисту даних, не може бути ефективно реалізований

без криптографічних постачальників. Вони діють як основний інструмент, який забезпечує обчислювальну підтримку для всіх криптографічних операцій.

Постачальник послуг криптографії (CSP) – це незалежний модуль, який дозволяє виконувати криптографічні операції. Яскравим прикладом постачальника послуг криптографії є CryptoAPI та CNG в операційних системах Microsoft. Також існують «незалежні» CSP, які не є елементами жодної операційної системи, наприклад OpenSSL, BoringSSL. Вони можуть бути задіяні незалежно від операційної системи. Вибір, розробка та вдосконалення криптопровайдерів безпосередньо впливає на продуктивність, безпеку та поширення протоколу TLS у мережах.

TLS вимагає шифрування для захисту даних, а постачальники криптографії пропонують реалізацію таких алгоритмів:

- симетричні алгоритми шифрування (AES, ChaCha20) – для захисту основного каналу передачі даних;
- асиметричні алгоритми шифрування (RSA, ECC) – для обміну ключами між клієнтом і сервером;
- ґеш-функції (SHA-256, SHA-3) – для забезпечення цілісності даних.

Крім того, криптографічні постачальники відповідають за створення, зберігання та керування криптографічними ключами, які використовуються в протоколі TLS. Вони обробляють сертифікати X.509, які використовуються TLS для автентифікації сервера та оптимізації обчислювальних операцій для зменшення затримки, що є критично важливим для TLS.

Метою даної роботи є аналіз ефективності роботи криптопровайдерів у протоколі TLS, а також визначення особливостей криптопровайдерів, що впливають на їх вибір та використання в TLS протоколі з урахуванням різних аспектів функціонування протоколу.

Для досягнення поставленої мети висуваються такі завдання:

- аналіз архітектури протоколу TLS;
- визначення вразливостей протоколу;
- вивчення ролі криптографічних провайдерів;
- порівняння сучасних рішень;
- формулювання рекомендацій.

Проведемо аналіз наукових робіт, які також досліджували це питання.

Аналіз наукових робіт

У сучасних дослідженнях, що стосуються використання криптопровайдерів у протоколі TLS, основна увага приділяється питанням криптостійкості, продуктивності та зручності впровадження на різних платформах. Різні криптографічні бібліотеки й API, зокрема Microsoft CryptoAPI, CNG, OpenSSL і BoringSSL, аналізуються у контексті їх здатності ефективно забезпечувати захищене з’єднання.

У роботі [2] розглядається реалізація TLS через бібліотеку CNG. Детально розглянуто внутрішню побудову та ефективність використання CNG у TLS-з’єднаннях. У роботі показано, що CNG пропонує сучасний API для криптографічних операцій, однак має обмежену гнучкість порівняно з відкритими реалізаціями, а також потенційно нижчу продуктивність через сильну інтеграцію із системним сховищем ключів.

Дослідження [3] акцентує увагу на проблемах управління секретами в мобільних реалізаціях TLS. Зокрема, продемонстровано, що навіть сучасні провайдери, такі як BoringSSL, які широко використовуються в Android, можуть зберігати ключову інформацію у пам’яті після завершення TLS-сесії. Це створює потенційну вразливість, яка може бути використана через аналіз дампів пам’яті або сайд-канальні атаки. Цей аспект підкреслює важливість не лише крипостійкості, а й коректного управління життєвим циклом ключів.

У контексті високопродуктивних обчислювальних систем заслуговує на увагу стаття [4], яка досліджує ефективність роботи бібліотек OpenSSL, BoringSSL, Libsodium і Crypto++

у середовищах із паралельними обчисленнями. Зокрема, порівнюються латентність, пропускна здатність та масштабованість TLS-сесій у середовищі MPI-комунікацій. Результати демонструють перевагу BoringSSL у масштабованості, що робить його привабливим для серверних рішень і хмарних інфраструктур.

Окрему увагу в літературі приділено Microsoft CNG API як заміні традиційної CryptoAPI. У статті [5] надається практичний огляд можливостей нового API. CNG пропонує більш модульний і гнучкий підхід до криптографії в Windows, підтримує сучасні алгоритми, інтеграцію з апаратними модулями та централізоване управління ключами через NCrypt API. Однак, використання CNG лишається обмеженим Windows-платформою та має менше прикладів кросплатформеного впровадження у TLS.

Крім того, в роботі [6] було запропоновано реалізацію постквантового алгоритму обміну ключами на базі NTRU у TLS 1.3 з використанням OpenSSL. Результати показують, що навіть за умов додаткових обчислювальних витрат, продуктивність TLS-сесій із постквантовим обміном ключів залишалась високою, що відкриває перспективи впровадження OpenSSL у майбутні стандарти стійких до квантових атак систем.

Кожна робота висвітлює окремі питання застосування криптопровайдерів з точки зору швидкодії або безпеки використання, в тому числі у протоколі TLS. Разом з тим, ці роботи не надають загальної картини щодо використання криптопровайдерів у протоколі TLS.

Архітектура протоколу TLS

Протокол TLS є фундаментальним елементом безпечного обміну даними в Інтернеті, який забезпечує шифрування, автентифікацію та цілісність інформації. Його ефективність ґрунтується на поєднанні механізмів взаємодії та різних криптографічних механізмів, які гарантують конфіденційність та захист від несанкціонованого доступу. Серед ключових компонентів TLS – алгоритми шифрування, ґеш-функції та інфраструктура формування та підтримки сертифікатів. [1] Кожен із цих елементів та їхня роль у створенні надійного механізму безпеки розглянуті нижче.

Алгоритми шифрування. Шифрування є базовим елементом безпеки TLS, який забезпечує захист переданої інформації від перехоплення. У межах протоколу застосовуються два типи криптографічних методів:

- симетричне шифрування: використовується для захисту переданих даних після встановлення з'єднання. Обидві сторони сесії застосовують один спільний ключ, що пришвидшує обробку та зменшує навантаження на систему;
- асиметричне шифрування: застосовується на початковому етапі встановлення з'єднання для захисту обміну ключами. Цей метод базується на використанні двох ключів: відкритого (public) та закритого (private).

Ґеш-функції. Ґеш-функції відіграють важливу роль у гарантуванні цілісності переданих даних і створенні цифрових підписів у TLS. Вони використовуються для формування Message Authentication Codes (MAC), які дозволяють виявляти будь-які зміни в даних під час передачі. Основні алгоритми наведено у табл. 1.

Ключі та сертифікати. Для забезпечення автентифікації та довіри між сторонами у TLS використовується механізм цифрових сертифікатів та криптографічних ключів. Формування та життєвий цикл сертифікатів відкритих ключів забезпечується центрами сертифікації відкритих ключів. Разом з тим, безпечне зберігання особистих ключів а також зберігання сертифікатів відкритих ключів у системах кінцевих користувачів часто покладається на функціонал криптопровайдерів.

Наявність інфраструктури відкритих ключів у TLS гарантує, що сервер або клієнт, з яким встановлюється з’єднання, дійсно є тим, за кого себе видає, що зменшує ризик атак типу «людина посередині» (man-in-the-middle, MITM) [2].

Таблиця 1

Алгоритми, що використовуються у TLS

Тип алгоритму	Назва алгоритму	Призначення/примітка	TLS-версії
Симетричне шифрування	AES-128-GCM / AES-256-GCM	Стандарти шифрування з автентифікацією	TLS 1.2, TLS 1.3
	AES-128-CBC / AES-256-CBC	Старі режими блочного шифрування (менш безпечні)	TLS 1.0 – TLS 1.2
	ChaCha20-Poly1305	Альтернатива AES для пристроїв безапаратного AES	TLS 1.2, TLS 1.3
	3DES	Застарілий алгоритм, небезпечний	TLS 1.0, TLS 1.1
Ассиметричне шифрування / обмін ключами	RSA	RSA алгоритм для обміну ключами	TLS 1.0–1.2
	DH	DH Алгоритм обміну ключами	TLS 1.0–1.2
	ECDH	Версія DH на еліптичних кривих	TLS 1.0–1.2
	DHE/ECDHE	Динамічні (ефемерні) версії DH/ECDH для forward secrecy	TLS 1.2, TLS 1.3
	Kyber	Post-quantum KEM (експериментальний, у гібридних обмінах)	TLS 1.3 (через розширення)
Геш-функції / MAC	SHA-1	Застаріла, не рекомендована до використання	TLS 1.0–1.2
	SHA-2 (SHA-256, SHA-384)	Сучасні стандарти для гешування / HMAC	TLS 1.2, TLS 1.3
	HMAC		TLS 1.2, 1.3
	BLAKE2	Не в основному стандарті, можливий через розширення	(експериментальні)
Алгоритми підпису (ЕЦП)	RSA-PSS	Рекомендований стандарт підпису у TLS 1.3	TLS 1.2, TLS 1.3
	ECDSA	Підписи на еліптичних кривих	TLS 1.2, TLS 1.3
	EdDSA (Ed25519, Ed448)	Сучасні, компактні та швидкі ЕЦП	TLS 1.3 (через розширення)
	Dilithium, Falcon	Post-quantum підписи (у режимі тестування / PQC експерименти)	TLS 1.3 (через розширення)

Етапи TLS-протоколу. Протокол TLS як правило визначається трьома етапами: TLS-Handshake, data transfer і connection termination [1], кожен з яких використовує окремі сімейства алгоритмів, що дає можливість у клієнтських застосунках використовувати криптопровайдери з обмеженими наборами алгоритмів. Але на практиці набори криптографічних алгоритмів не обмежуються.

Вразливості протоколу TLS

Попри широке впровадження, TLS не позбавлений вразливостей, які можуть виникати як через недоліки в дизайні протоколу, так і через помилки в реалізації алгоритмів конкретними криптографічними бібліотеками. Ці вразливості можуть бути експлуатовані зловмисниками для проведення атак, що ставить під загрозу безпеку переданих даних. Попри постійні оновлення версій протоколу, які усувають ці вразливості, стверджувати про відсутність вразливостей у протоколі неможливо – деякі вразливості просто можливо просто ще не були виявлені. Наведемо найбільш відомі й критичні вразливості, що були виявлені в TLS.

Однією з ключових проблем ранніх версій TLS (1.0 та 1.1) була вразливість до атак типу BEAST (Browser Exploit Against SSL/TLS) [9], яка використовувала недоліки в реалізації

режиму шифрування CBC для отримання доступу до зашифрованих даних. Ця атака стала можливою через передбачуваність ініціалізаційного вектора (IV) в цих версіях протоколу.

Також достатньо цікавою вразливістю є можливість виконання атаки типу POODLE (Padding Oracle On Downgraded Legacy Encryption) [10], яка використовує можливість за певних обставин знизити версію протоколу до SSL 3.0. Відповідно така маніпуляція дозволяє зловмиснику атаки, що були можливі для версії SSL 3.0, і тому числі padding oracle атаки. Це стало можливим через підтримку зворотної сумісності в TLS, що дозволяє зловмиснику нав'язати використання застарілих і менш безпечних версій протоколу.

Прикладом критичної помилки в реалізації OpenSSL, яка дозволяла зчитувати до 64 кілобайтів пам'яті сервера, включаючи конфіденційні дані, такі як приватні ключі та паролі, є вразливість Heartbleed (CVE-2014-0160) [11]. Ця помилка виникла через неправильну обробку запитів heartbeat, що дозволяло зловмиснику отримувати доступ до пам'яті сервера без авторизації. Після цього, у 2013 р., атака Lucky13 (CVE-2013-0169) продемонструвала, як навіть незначні відмінності в часі обробки повідомлень можуть бути використані для витоку інформації [12]. Ця атака експлуатує час обробки padding в режимі CBC, що дозволяє зловмиснику поступово відновлювати зашифровані дані.

З-поміж іншого, Bleichenbacher атаки, вперше описані в 1998 р., залишаються актуальними й сьогодні. Вони експлуатують помилки в обробці помилок при дешифруванні RSA, що дозволяє зловмиснику отримати доступ до зашифрованих даних без знання приватного ключа [13]. Незважаючи на численні оновлення протоколу, деякі реалізації TLS залишаються вразливими до цих атак.

Безпека протоколу TLS залежить не лише від його специфікації, але й від реалізації конкретними криптографічними провайдерами. Вибір бібліотеки повинен враховувати не лише продуктивність, але й рівень безпеки, підтримку сучасних алгоритмів і швидкість реагування на виявлені вразливості. Регулярне оновлення бібліотек, відмова від застарілих алгоритмів і впровадження сучасних практик безпеки є ключовими факторами для забезпечення надійного захисту даних в мережевих комунікаціях [2].

Вплив криптографічних провайдерів на роботу протоколу TLS

Криптографічний провайдер (Cryptographic Service Provider, CSP) – це програмний або апаратний модуль, який реалізує криптографічні операції, необхідні для забезпечення безпеки в протоколі TLS. Він виступає як посередник між застосунками, що потребують шифрування, та криптографічними алгоритмами, які забезпечують захист даних.

Основні функції криптографічного провайдера:

- реалізація алгоритмів симетричного та асиметричного шифрування;
- створення, збереження та управління криптографічними ключами;
- генерація та перевірка цифрових підписів;
- виконання операцій ґешування та автентифікації.

Використання CSP дозволяє застосовувати TLS без необхідності глибокого розуміння складних криптографічних процесів з боку розробників. Замість цього вони можуть звертатися до стандартних бібліотек або API, які забезпечують високий рівень безпеки та продуктивності.

Різні операційні системи та програмні середовища мають власні криптографічні провайдери, які відповідають за безпеку TLS-з'єднань. Серед найвідоміших рішень виділяють наступні:

1. *OpenSSL* – один із найпопулярніших криптографічних провайдерів з відкритим кодом. Він використовується у вебсерверах (наприклад, Apache, Nginx), поштових серверах, VPN-системах та інших мережевих додатках. OpenSSL підтримує широкий набір криптографічних алгоритмів, зокрема AES, ChaCha20, RSA, ECC, та SHA-256 [14].

До головних переваг OpenSSL можна віднести підтримку сучасних стандартів шифрування, постійний перегляд кодової бази та оновлення, які реагують на нові загрози. Крім того, для OpenSSL існує можливість використання в різних операційних системах.

2. *BoringSSL* – це варіація OpenSSL, що була розроблена корпорацією Google для використання у власних продуктах, таких як Chrome та Android. Він оптимізований для швидкості та безпеки, а також спрощений для усунення потенційних вразливостей [15].

Особливості BoringSSL полягають у зменшеному розмірі коду та відсутності застарілих функцій, регулярних оновленнях безпеки, а також кращій інтеграції з мобільними платформами.

3. *Криптопровайдери від Microsoft, а саме сімейство CryptoAPI - CNG* – це криптографічна платформа у операційній системі Windows (починаючи з відносно застарілих версій). Вона забезпечує доступ до шифрувальних операцій через API. Крім того, вона використовується у Windows для автентифікації, підпису цифрових документів та шифрування TLS-з'єднань.

Серед основних характеристик CryptoAPI вбудована підтримка TLS у Windows, можливість роботи з сертифікатами X.509 й інтеграція з апаратними модулями безпеки (HSM, TPM).

Крім цих провайдерів, існують й інші криптографічні бібліотеки, такі як, наприклад, LibreSSL (форк OpenSSL, що зосереджується на безпеці) та WolfSSL (легковаговий криптографічний провайдер, орієнтований на IoT-пристрої).

Хоча всі криптографічні провайдери реалізують схожі алгоритми, їхня продуктивність, безпека та оптимізація суттєво відрізняються залежно від імплементації.

Безпека TLS-протоколу в криптопримітивах

Реалізація TLS значною мірою залежить від криптографічних бібліотек, які використовуються для його впровадження. Різні бібліотеки мають свої особливості, які можуть як підвищувати, так і погіршувати безпеку протоколу.

OpenSSL є однією з найпоширеніших криптографічних бібліотек, яка підтримує широкий спектр алгоритмів і протоколів. Однак її складність і підтримка застарілих алгоритмів можуть призводити до вразливостей, якщо не вжити належних заходів безпеки. Наприклад, підтримка слабких cipher suites, таких як RC4 або 3DES, може бути використана зловмисниками для проведення атак.

BoringSSL, розроблена Google, є форком OpenSSL з акцентом на безпеку і спрощення. Вона виключає підтримку застарілих і небезпечних алгоритмів, що зменшує поверхню атаки. Однак така стратегія може обмежувати гнучкість у деяких випадках, коли потрібна підтримка специфічних алгоритмів або протоколів.

CryptoAPI від Microsoft забезпечує інтеграцію TLS в операційну систему Windows. Хоча цей криптопровайдер гарантує стабільність і підтримку з боку виробника, обмеження в налаштуваннях і повільне впровадження оновлень можуть призводити до затримок у виправленні відомих вразливостей. Крім того, обмежена підтримка сучасних алгоритмів, таких як Curve25519 або SHA-3, може знижувати загальний рівень безпеки.

Аналіз імплементації основних компонентів TLS різними криптопровайдерами

Розглянемо можливості імплементації основних компонентів TLS різними криптопровайдерами.

У контексті сучасних підходів до реалізації TLS, одним із ключових критеріїв залишається продуктивність – зокрема, наскільки ефективно провайдер використовує апаратні ресурси, оптимізацію під конкретні архітектури процесорів та підтримку новітніх криптографічних стандартів.

Можливості імплементації симетричного шифрування (AES, ChaCha20). У випадку симетричного шифрування, OpenSSL завдяки модульній структурі забезпечує максимально ефективне використання AES-NI на процесорах Intel і AMD, що підтверджено численними

бенчмарками (наприклад, openssl speed-evp aes-256-gcm). Наприклад, при використанні AES-256-GCM з підтримкою AES-NI, продуктивність досягає приблизно 2251 МБ/с на блоках розміром 8192 байти. [14] Реалізація ChaCha20, починаючи з версії 1.1.0, була значно оптимізована для мобільних ARM-архітектур, що особливо помітно у середовищах без AES-NI [15].

BoringSSL, який є форком OpenSSL, активно видаляє зайвий функціонал на користь оптимізації. Реалізація ChaCha20-Poly1305 у BoringSSL є однією з найшвидших на ARM-архітектурах, забезпечуючи перевагу над AES за відсутності апаратного прискорення. Швидкість роботи алгоритму для блоків розміром 8192 байти складає до 1707.5 MB/c [16], у той час, як за використання OpenSSL – лише до 719 MB/c [14].

Натомість CryptoAPI часто працює через абстрактний інтерфейс, який обмежує прямий контроль над апаратною оптимізацією. Вимірювання продуктивності з використанням BCryptEncrypt та BCryptOpenAlgorithmProvider показують значно більше навантаження в порівнянні з OpenSSL або BoringSSL, особливо при великій кількості одночасних з'єднань. Варто зазначити, що до значних переваг CryptoAPI слід віднести реалізацію AES, яка повністю сумісна з сертифікатами Windows, однак може мати нижчу продуктивність через обмеження API.

Асиметричне шифрування (RSA, ECC, ECDHE). У контексті асиметричного шифрування, OpenSSL демонструє стабільну продуктивність із підтримкою апаратного прискорення для великих RSA-ключів через Intel QuickAssist та інші HSM. Наприклад, при використанні ключів RSA-2048 швидкість підпису становить приблизно 100 операцій в секунду, а перевірки — 22000 операцій в секунду [17]. Після версії 3.0 було покращено підтримку криптографії на кривих Brainpool та Curve25519, що важливо для сучасного TLS 1.3.

BoringSSL навмисне виключив підтримку деяких функцій, як-от CMS і повну підтримку PKCS#11, але натомість сконцентрувався на високоефективній реалізації X25519 та ECDHE, що робить його першочерговим вибором для клієнтських застосунків, таких як Android Chrome. Ці оптимізації зменшують затримку під час встановлення TLS-з'єднання, що критично важливо для користувацького досвіду в мобільному середовищі. Згідно з результатами дослідження [18] швидкість підпису з використанням X25519 складає 56680 операцій за секунду, перевірки – 18020 операцій в секунду [18].

CryptoAPI, попри підтримку сучасних кривих через CNG, на практиці може бути обмеженим через відсутність можливості тонкого налаштування протоколу — зокрема, неможливість вибору кривих, не схвалених Microsoft, без використання сторонніх бібліотек або CAPI wrapper'ів. Наприклад, Curve25519 не входить до переліку підтримуваних алгоритмів CryptoAPI без використання сторонніх бібліотек, однак у Windows 10 вже підтримується з CNG[19].

Ґеш-функції (SHA-256, SHA-3). Для ґеш-функцій, OpenSSL продовжує демонструвати високу швидкодію завдяки SIMD-оптимізованим реалізаціям. Підтримка SHA-3 була додана у версіях після 1.1.1 і є цілком придатною для масштабних сценаріїв. У певних бенчмарках [20] видно, що SHA-2 і SHA-3 реалізовані з урахуванням паралельності обробки, що знижує латентність при обробці великих об'ємів трафіку. При цьому реалізації ґешування в OpenSSL дозволяє обробляти до 696 МБ/с, залежно від вибраного алгоритму [20].

BoringSSL, хоч і менш гнучкий щодо вибору ґеш-функцій, у межах TLS 1.3 суворо дотримується використання SHA-256 або SHA-384, що спрощує і прискорює обчислення, одночасно мінімізуючи ризики неправильного конфігурування. Властиво, що реалізація ґешування в BoringSSL демонструє кращі результати за OpenSSL, обробляючи до 873 МБ/с [16].

У випадку CryptoAPI, незважаючи на додавання SHA-3 у нові версії Windows 10/11 (через BCryptCreateHash), підтримка обмежена до певних конфігурацій, а продуктивність суттєво нижча. CryptoAPI демонструє нижчі швидкості – до 320 МБ/с для SHA3-256, через менш ефективну реалізацію та обмеження в API [21]. Це робить CryptoAPI менш привабливим у високонавантажених TLS-серверах.

У підсумку, хоча всі три криптопровайдери підтримують основні алгоритми, OpenSSL та BoringSSL мають значну перевагу у швидкодії та адаптивності до сучасних криптографічних вимог. OpenSSL, як більш універсальний та широко підтримуваний інструмент, залишається стандартом де-факто у серверних реалізаціях TLS, тоді як BoringSSL, орієнтований на клієнтські мобільні платформи, забезпечує найкращу продуктивність у цих умовах. CryptoAPI, натомість, має більшу інтеграцію з екосистемою Windows, але поступається у гнучкості та швидкості.

Таким чином, криптографічні провайдери значною мірою впливають на продуктивність та безпеку реалізації TLS, забезпечуючи різні рівні оптимізації, підтримку апаратного прискорення та інтеграцію з операційними системами. Розуміння особливостей імплементації алгоритмів у криптографічних провайдерах дозволяє ефективно обирати рішення залежно від вимог до продуктивності, безпеки та сумісності.

Проілюструємо результати досліджень у вигляді табл. 2.

Таблиця 2

Порівняльний аналіз ефективності криптопровайдерів щодо реалізації протоколу TLS

Критерій порівняння	Криптопровайдери		
	OpenSSL	BoringSSL	Crypto API
Загальні характеристики	Потужна, кросплатформена бібліотека з відкритим кодом, яка підтримує більшість сучасних криптографічних алгоритмів. Активно розвивається, широко використовується у серверних продуктах.	Форк OpenSSL, розроблений Google з акцентом на безпеку, спрощення коду та видалення застарілих або ризикованих функцій. Найчастіше застосовується в мобільних і браузерних клієнтах.	Вбудований компонент Windows, інтегрований у системні служби. Забезпечує базовий набір криптоопераций із пріоритетом сумісності, стабільності та сертифікації (наприклад, FIPS).
Реалізація симетричного шифрування (AES, ChaCha20)	Забезпечує високопродуктивне симетричне шифрування через апаратне прискорення (AES-NI). Реалізація ChaCha20 оптимізована під ARM-процесори, що забезпечує ефективність на мобільних пристроях.	Висока швидкодія реалізації ChaCha20-Poly1305, оптимізована для мобільних платформ. Код суттєво спрощений, що мінімізує ймовірність вразливостей у реалізації симетричних алгоритмів.	Працює через абстрактні інтерфейси (наприклад, BCryptEncrypt), що знижує продуктивність. Підтримує AES, але реалізація менш ефективна, особливо у сценаріях з великою кількістю одночасних потоків.
Реалізація асиметричного шифрування (RSA, ECDSA, ECC)	Повна підтримка RSA, ECDSA, X25519, з можливістю апаратного прискорення (наприклад, через Intel QAT). Після версії 3.0 — підтримка нових кривих, включно з Brainpool.	Орієнтація на сучасні криві (X25519, P-256) із видаленням застарілих та малозатребуваних алгоритмів. Забезпечує швидке встановлення TLS-з'єднань, особливо в мобільних браузерах.	Підтримує сучасні еліптичні криві через інтерфейс CNG, однак гнучкість налаштувань обмежена. Деякі криві недоступні без сторонніх обгорток або налаштування реєстру.
Реалізація геш-функцій (SHA-1, SHA-2, SHA-3)	Реалізація з SIMD-оптимізацією дозволяє досягати високої швидкодії при обробці великих обсягів даних. Повна підтримка SHA-2 та SHA-3, можливість паралельної обробки.	Використовуються лише SHA-256 та SHA-384 у межах TLS 1.3. Така обмеженість зменшує ймовірність помилок, водночас забезпечуючи стабільну і швидку роботу.	Підтримка SHA-2 стабільна, SHA-3 реалізовано лише в нових версіях Windows 10/11. Продуктивність нижча в порівнянні з OpenSSL, особливо в контексті TLS-серверів
Безпека використання	Висока, за умови правильної конфігурації: бібліотека містить підтримку сучасних алгоритмів, але залишає можливість використання застарілих та слабких cipher suites (наприклад, RC4, 3DES), якщо явно не вимкнути.	Висока за рахунок видалення слабких або deprecated алгоритмів, відсутня підтримка RC4, SSLv3 тощо. Знижений ризик помилок конфігурації.	Стабільна, але залежна від політики оновлень Windows. Підтримка обмежена лише сертифікованими примітивами. Застарілі алгоритми можуть залишатися доступними в системі за замовчуванням.

Під час аналізу сучасних криптографічних провайдерів виявлено, що рішення з відкритим кодом, такі як OpenSSL та BoringSSL, демонструють високу продуктивність, адаптивність до новітніх криптографічних стандартів і швидку реакцію на виявлені вразливості. Особливу увагу заслуговує BoringSSL, оптимізований для мобільних платформ, де критично важливими є затримки з'єднання та ефективне використання ресурсів. Водночас реалізації від Microsoft, як-от CryptoAPI, забезпечують глибоку інтеграцію в екосистему Windows, проте мають обмеження в налаштуваннях і підтримці сучасних криптографічних примітивів, що знижує їхню придатність у високонавантажених сценаріях.

Загальний рівень безпеки протоколу TLS залежить не лише від його формальної специфікації, але й від правильності та актуальності реалізації з боку постачальників криптографічних функцій. Наявність вразливостей, таких як Heartbleed, POODLE, Lucky13 або BEAST, свідчить про необхідність постійного оновлення бібліотек, відмови від застарілих версій TLS і шифрувальних наборів, а також впровадження сучасних захистів, зокрема механізмів forward secrecy та strict certificate validation.

Висновки

У результаті дослідження встановлено, що протокол TLS є фундаментальним елементом захисту інформації під час її передавання в мережевих середовищах. Його надійність забезпечується за рахунок поєднання симетричних і асиметричних алгоритмів шифрування, а також криптографічних ґеш-функцій, що разом створюють багаторівневу архітектуру безпеки. Ефективність і стійкість TLS значною мірою залежить від якості реалізації криптографічних провайдерів, які виконують обчислювальні операції, керують криптографічними ключами, а також забезпечують обробку цифрових сертифікатів.

У реалізаціях криптографічного забезпечення доцільно орієнтуватися на OpenSSL або BoringSSL як найбільш актуальні, продуктивні та гнучкі рішення з широкою підтримкою сучасних криптографічних стандартів. Для підвищення безпеки мережевої інфраструктури важливо забезпечити регулярне оновлення криптографічних бібліотек, контроль за налаштуванням TLS-з'єднань, перевірку сертифікатів та обмеження підтримки слабких алгоритмів. Необхідно також досліджувати можливості апаратного прискорення криптографічних операцій для забезпечення масштабованості та високої ефективності систем захисту.

З огляду на динамічний розвиток інформаційних технологій, особливу актуальність набуває адаптація TLS до умов постквантової криптографії. Зважаючи на загрозу, яку становлять квантові обчислення для сучасних криптосистем, провайдери криптографічних послуг та криптобібліотеки мають інтегрувати квантово-стійкі алгоритми в архітектуру TLS та забезпечити їхню ефективну підтримку.

У перспективі доцільним є розвиток інструментів моніторингу безпеки TLS, аналізу роботи криптопровайдерів у реальному часі, а також підготовка до поступового переходу на квантово-стійкі криптографічні протоколи як запоруки збереження конфіденційності та цілісності даних у майбутніх поколіннях мережевих технологій.

Список літератури:

1. Dierks T., and Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246) // Internet Engineering Task Force (IETF). 2008 https://doi.org/10.17487/RFC5246

2. Lee Jae-Ho. Analysis of SSL Communication Process in CNG Crypto Library // The Journal of Korean Institute of Communications and Information Sciences. 2017. Vol. 42, no. 5. P. 1027–1037. https://doi.org/10.7840/kics.2017.42.5.1027

3. Lee Jaeho, and Wallach Dan S. Removing Secrets from Android's TLS // Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), ACM, 2019. https://doi.org/10.1145/3321705.3329810

4. Naser Abu, et al. Performance Evaluation and Modeling of Cryptographic Libraries for MPI Communications // Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2022. P. 1192–1201. https://doi.org/10.1109/IPDPS53621.2022.00104

5. Howard Bryan. Applying Cryptography Using the CNG API in Windows Vista // MSDN Magazine, July 2007. https://learn.microsoft.com/en-us/archive/msdn-magazine/2007/july/applying-cryptography-using-the-cng-api-in-windows-vista

6. Bernstein, Daniel J., et al. OpenSSLNTRU: Faster post-quantum TLS key exchange // 31st USENIX Security Symposium (USENIX Security 22). 2022. P. 359–376. https://www.usenix.org/conference/usenixsecurity22/presentation/bernstein

7. Schneier, B. Applied Cryptography: Protocols, Algorithms, and Source Code in C (20th Anniversary ed.). Wiley, 2015

8. Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 8446) // Internet Engineering Task Force (IETF). 2018. https://doi.org/10.17487/RFC8446

9. Rizzo J., and Duong T. BEAST: Surprising crypto attack against HTTPS // Presented at Ekoparty Security Conference. 2011. https://hal.science/hal-01154820/document

10. Moeller B. This POODLE Bites: Exploiting The SSL 3.0 Fallback // Google Security Blog. 2014. https://security.googleblog.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

11. Durumeric Z., Kasten J., Adrian D., Halderman J. A., Bailey M., Li F., ... and Ensafi R. The Matter of Heartbleed // Proceedings of the 2014 Conference on Internet Measurement Conference. 2014. P. 475–488. https://doi.org/10.1145/2663716.2663755

12. AlFardan N. J., and Paterson K. G. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols // 2013 IEEE Symposium on Security and Privacy. 2013. P. 526–540. https://doi.org/10.1109/SP.2013.13

13. Bleichenbacher D. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard // Advances in Cryptology – CRYPTO '98. 1998. P. 1–12. Springer. https://doi.org/10.1007/BFb0055716

14. Calomel.org. AES-NI SSL Performance Benchmarks, 2022 https://calomel.org/aesni_ssl_performance.html

15. OpenSSL Software Foundation. OpenSSL: Cryptography and SSL/TLS Toolkit. https://www.openssl.org

16. GitHub. Performance degradation in the FIPS-BoringSSL version being used by Envoy, 2021. https://github.com/envoyproxy/envoy/issues/19037

17. OpenSSL Cookbook 3rd Edition - 1.4 Performance. Feisty Duck | SSL/TLS and PKI training and books, 2020. https://www.feistyduck.com/library/openssl-cookbook/online/openssl-command-line/performance.html

18. BoringSSL Gerrit. Use packed representation for large Curve25519 table, 2023. https://boringssl-review.googlesource.com/c/boringssl/+/60107

19. Microsoft Docs. ECC Curve Support in CNG. Microsoft, 2023. https://learn.microsoft.com/en-us/windows/win32/seccng/cng-named-elliptic-curves

20. Security and So Many Things. Hashing Methods Benchmark, 2021. https://asecuritysite.com/openssl/openssl_full2b

21. GitHub. BenchmarkDotNet Crypto Hash Test, 2023. https://github.com/dotnet/BenchmarkDotNet

Надійшла до редколегії 10.02.2025

Відомості про авторів:

Тельнова Аліна Анатоліївна – Харківський національний університет радіоелектроніки, бакалавр кафедри безпеки інформаційних технологій факультет комп'ютерної інженерії та управління; Україна; e-mail alina.telnova@nure.ua; ORCID: https://orcid.org/0009-0001-3574-7425

Балагура Дмитро Сергійович – канд. техн. наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри безпеки інформаційних технологій, факультет комп'ютерної інженерії та управління, Україна; e-mail: dmytro.balahura@nure.ua; ORCID: https://orcid.org/0000-0002-6327-6405

Фроленко Владислав Олегович – Харківський національний університет радіоелектроніки, аспірант кафедри безпеки інформаційних технологій факультет комп'ютерної інженерії та управління, Україна; e-mail: vladyslav.frolenko@nure.ua; ORCID: https://orcid.org/0009-0004-3730-3432

Сухотеплий Владислав Миколайович – Харківський національний університет Повітряних Сил імені Івана Кожедуба, старший викладач кафедри радіоелектронних систем пунктів управління Повітряних Сил, Україна; e-mail: vladislav181168@gmail.com; ORCID: https://orcid.org/0000-0002-2566-4167

Флоров Сергій Володимирович – канд. техн. наук, Університет митної справи та фінансів, доцент кафедри кібербезпеки та інформаційних технологій, факультет інноваційних технологій, Україна; e-mail: pre.pod@hotmai.com; ORCID: https://orcid.org/0000-0002-4682-7666