УДК 621.311.1.018.3

Ю. В. Баришев, канд. техн. наук, доц; А. В. Скіп

МЕТОД ШИФРУВАННЯ В ЛАНЦЮГАХ ПОСТАЧАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті представлено сучасний підхід до забезпечення безпеки ланцюгів постачання ПЗ, метою якого є захист кодової бази шляхом застосування криптографічних методів на ранніх етапах розробки. Запропонований метод передбачає шифрування вмісту репозиторію вихідного коду з використанням симетричного алгоритму та захист ключа шифрування за допомогою асиметричного алгоритму. Особлива увага приділяється інтеграції системи керування секретами HashiCorpVault та LDAP-сервера для централізованої автентифікації й авторизації користувачів, гнучкому керуванню криптографічними ключами, а також впровадженню механізмів ротації ключів у середовищі CI/CD, що дозволяють оперативно реагувати на інциденти безпеки та підтримувати актуальність ключової інфраструктури. Така комбінація технологій гарантує, що лише автентифіковані користувачі з належними правами доступу можуть отримати доступ до захищеного коду, мінімізуючи ризики компрометації як через зовнішні атаки, так і через внутрішні загрози. Запропоноване рішення відповідає принципам ZeroTrust та найменших привілеїв і не створює значного навантаження на процес розробки, інтегруючись у DevSecOps-процеси та забезпечуючи наскрізний захист від моменту створення вихідного коду до його розгортання в промисловому середовищі. Запропонований підхід підвищує рівень конфіденційності, цілісності та контрольованості кодової бази. Новизна рішення полягає у наскрізному застосуванні шифрування на ранніх етапах розробки та інтеграції керування ключами з корпоративною інфраструктурою безпеки, усуваючи типове відокремлення захисту коду від інших процесів кібербезпеки. Серед обмежень – залежність від наявності інфраструктури керування секретами (Vault, LDAP) та фокусування на рівні репозиторію, що потребує подальшого вдосконалення для детальнішого розмежування прав доступу. Перспективи подальших досліджень пов'язані з розробкою механізмів гранулярного керування доступом у захищених репозиторіях.

Ключові слова: *криптографія, шифрування, захист коду, розмежування прав доступу, LDAP, CI/CD, DevSecOps, захист ланцюга постачання.*

Вступ

В сучасних умовах кібербезпеки все більшої ваги набуває захист ланцюгів постачання програмного забезпечення – від вихідного коду до готового продукту. Одним із ключових елементів безпеки програмних постачань є обов'язкове використання криптографії для забезпечення конфіденційності та цілісності даних. Зокрема, шифрування вихідного коду та артефактів збірки дозволяє запобігти несанкціонованому доступу до них навіть у разі компрометації інфраструктури.

Водночас використання криптографічних методів захисту даних породжує завдання керування ключами: розподілу, зберігання та безпечного використання. Саме тому наразі в ланцюгах постачання програмного забезпечення такі методи використовуються частково – на етапах, де кодова база вже розроблена і залучені лише DevOps та DevSecOps фахівці. При цьому робочі станції розробників та сервер інтегрування коду на етапі створення кодової бази залишаються незахищеними, саме тому постає актуальне завдання інтегрування методів криптографічного захисту інформації на етапі розроблення коду. Впровадження сучасних методів шифрування в репозиторіях коду й процесах CI/CD у поєднанні з управлінням ключами може суттєво підвищити стійкість до атак [1].

Метою цього дослідження є покращення захисту кодової бази шляхом використання методів криптографії в ланцюгах постачання програмного забезпечення на етапі розробки цієї кодової бази.

Для досягнення мети необхідно виконати такі завдання:

- проаналізувати методи захисту кодової бази в ланцюгах постачання та засоби

криптографічного захисту даних;

- розробити метод захисту кодової бази;
- розробити архітектуру засобу для доведення концепції.

Аналіз відомих рішень

Проблематика захисту ланцюгів постачання програмного забезпечення (ПЗ) активно досліджується останніми роками у зв'язку зі зростанням кількості атак на процеси розроблення та постачання програмних компонентів [3]. Окрему увагу приділено питанням використання криптографії в цьому контексті. Загальновідомо, що симетричні алгоритми шифрування, такі як AES [4], забезпечують високу продуктивність під час шифрування великих обсягів даних, тоді як асиметричні алгоритми зручні для безпечного розповсюдження ключів, хоча і поступаються в швидкодії [5]. Саме тому у багатьох рішеннях застосовується гібридне шифрування – поєднання симетричного та асиметричного підходів: відкритий ключ використовується для шифрування секретного (симетричного) ключа, який далі застосовується для основного шифрування даних [6].

У галузі зберігання та захисту даних широко відомі інструменти повнодискового шифрування, зокрема VeraCrypt [7] та BitLocker [8]. VeraCrypt – відкритий програмний засіб, який, дозволяє створювати зашифровані віртуальні диски чи розділи. Він підтримує низку алгоритмів шифрування (AES, Serpent, Twofish, Camellia тощо) і геш-функцій (RIPEMD-160, SHA-256, Whirlpool та ін.) для забезпечення конфіденційності та цілісності даних [7]. BitLocker – вбудований у Windows інструмент, що реалізує шифрування дисків із використанням алгоритму AES. Обидва засоби ефективно забезпечують базовий захист конфіденційності даних на носіях, хоча відрізняються реалізацією: BitLocker працює прозоро для користувача як частина операційної системи, тоді як VeraCrypt вимагає ручного керування контейнерами, зате дозволяє приховувати саме існування секретних даних (при відключенні том VeraCrypt невидимий системі) [7, 8]. У контексті захисту інфраструктури розробки ці інструменти можуть застосовуватися для шифрування дисків серверів, репозиторіїв, резервних копій тощо, зменшуючи ризик компрометації даних при фізичному доступі або крадіжці обладнання.

Іншою важливою складовою є керування секретами та ключами шифрування. В цьому дослідженні розглянуто продукт HashiCorpVault, як центр управління секретами, який безпечно зберігає зашифровані дані, ключі, паролі, сертифікати тощо і надає гнучкі механізми доступу на основі політик безпеки [9]. Система Vault підтримує різні методи автентифікації користувачів і сервісів (наприклад, LDAP, GitHubOAuth, JWT і ін.) та видає тимчасові токени доступу, що мінімізує ризики компрометації статичних облікових даних [9].

Окремо слід відзначити роль LDAP (LightweightDirectory Access Protocol) [10] у побудові корпоративних систем контролю доступу. LDAP-сервер (наприклад, ActiveDirectory) часто виступає єдиним репозиторієм облікових записів користувачів та їхніх груп/ролей. Інтеграція Vault з LDAP дозволяє зв'язати криптографічні політики з наявними ролями: при вході користувача через LDAP Vault автоматично призначає йому відповідні політики доступу до секретів на основі його групових належностей.

Необхідно відзначити, що тема захисту репозиторіїв вихідного коду шляхом шифрування також піднімалася в окремих практичних рішеннях. Існують, наприклад, надбудови до систем контролю версій (Git), які реалізують прозоре шифрування вмісту репозиторію. Одне з них – git-remote-gcrypt [11], що використовує GNU PrivacyGuard (GPG) для шифрування всіх даних репозиторію на стороні сервера. В літературі зазначено, що стандартні Git-сервіси не розраховані на зберігання чутливої інформації у відкритому вигляді, тому додаткове прозоре шифрування всього вмісту репозиторію є доцільним у випадках підвищених вимог безпеки [12]. Інструмент git-remote-gcrypt фактично реалізує схему, подібну до розглядуваної в цій роботі: дані шифруються симетричним ключем, який, у свою чергу, захищається за

допомогою асиметричних ключів GPG, розповсюджених між учасниками проєкту. Однак, git-remote-gcrypt має низку недоліків – обмежена кількість операційних систем що підтримуються, відсутність механізму автентифікації, а також обмежена можливість для масштабування у великих командах. Одним із критичних недоліків у великих командах є додавання нового користувача – після цього весь репозиторій підлягає повторному шифруванню. Варто додати відсутність простої ротації ключів шифрування – за компрометації одного із ключів – його потрібно відкликати, а це означає повторне шифрування всього репозиторію ключами усіх учасників проєкту.

Ще одним недоліком відомих рішень є відсутність способів інтегрування захисту до процесів інформаційної безпеки підприємства. Так відомими підходами не передбачено застосування політик розмежування прав доступу, загальних для всього підприємства, що ускладнює цей процес, фактично розбиваючи його на два різні домени, які не синхронізуються між собою: захист кодової бази, захист даних підприємства. Останнє породжує ризики некоректного конфігурування безпеки через людський фактор, а також збільшує час на внесення змін до політики розмежування прав доступу. Більше того, такий підхід під час розслідування інцидентів ускладнить ідентифікацію джерела інциденту та ускладнить аналіз дій користувачів, оскільки потребуватиме постійного зіставлення користувачів репозиторію з користувачами інформаційної системи підприємства.

Метод захисту кодової бази

Як показав аналіз відомих рішень – підхід, що передбачає комбіноване використання симетричного та асиметричного методів шифрування має найбільші перспективи для досягнення мети цього дослідження. При цьому метод повинен передбачати механізми інтегрування захисту кодової бази із рештою процесів кібербезпеки підприємства. Для цього пропонується метод, який передбачає такі кроки.

Крок 1. Створення або використання облікових записів користувача в службах розмежування прав доступу, як-то LDAP.

Крок 2. Генерування пар ключів для асиметричного шифрування для кожного користувача. При цьому збереження публічних ключів передбачається в загальнодоступних сховищах даних, а приватних – в захищеному сховищі.

Крок 3. При створенні нового репозиторію з кодовою базою відбувається створення криптографічного ключа симетричного шифрування, який використовується для зашифрування кодової бази.

Крок 4. Створення правил розмежування прав доступу користувачів до репозиторію з кодовою базою. Якщо доступ дозволено – то ключ симетричного шифрування зашифровується публічним ключем користувача, якому надається доступ.

Крок 5. Збереження зашифрованих ключів симетричного шифрування в захищеному сховищі, синхронізованому із сервером служби розмежування прав доступу. Доступ до приватного ключа зі сховища буде надаватись користувачеві лише у випадку доведення його автентичності службі розмежування прав доступу.

Крок 6. Читання репозиторію користувачем шляхом розшифрування симетричного ключа, яким зашифровано репозиторій, на основі приватного ключа користувача.

Крок 7. Запис та оновлення кодової бази шляхом зашифрування сирцевого коду на основі симетричного ключа та його подальше збереження в репозиторії.

Завдяки використання служби розмежування прав доступу та захищеного сховища приватних ключів запропонований метод дозволяє усунути недолік відомих підходів, пов'язаний із різнорідністю політик розмежування прав доступу, якщо служба розмежування прав доступу буде використана й для інших завдань управління доступу. більше того, недолік щодо потреби у повторному зашифруванні та заміні всіх ключів у випадку компрометації приватного ключа користувача пом'якшується, оскільки ймовірність цієї ситуації зменшується внаслідок централізованого захищеного сховища ключів. Водночас

цей недолік залишається актуальним у випадку компроментування симетричного ключа, проте за рахунок меншої прив'язки до дій користувачів, ротація ключів може відбутись без їх участі (оскільки приватні ключі зберігаються у захищеному сховищі, а не на носіях користувачів).

Архітектура засобу захисту програмного забезпечення

Щоб довести можливість практичної реалізації запропонованого методу, авторами запропонована архітектура засобу захисту програмного забезпечення, який використовує цей метод, наведена на рис. 1.

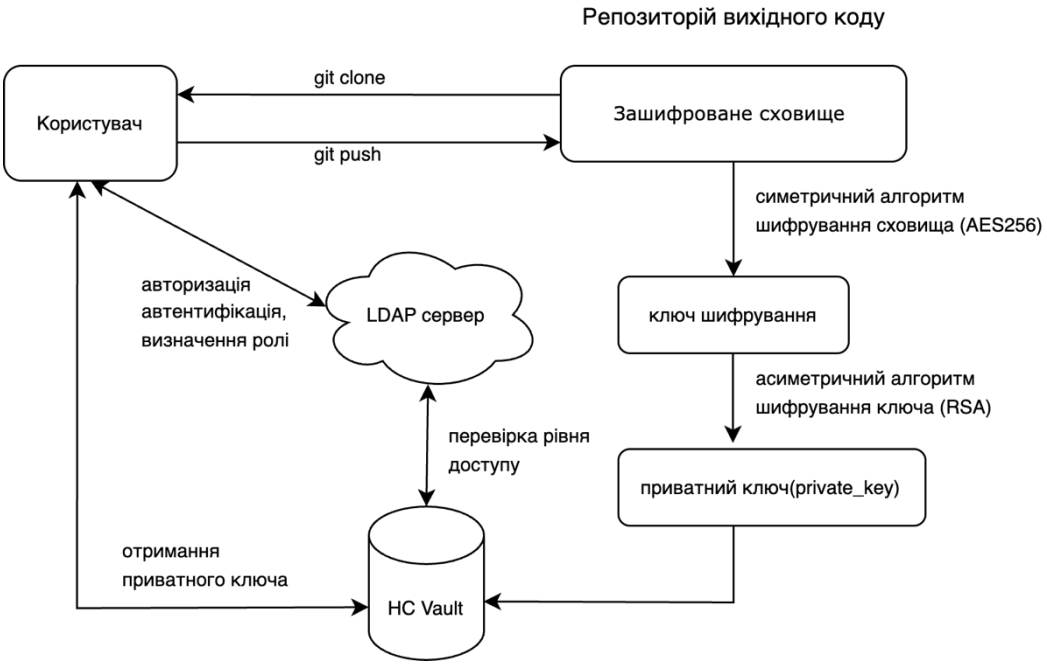

Рис. 1. Архітектура засобу захисту програмного забезпечення

Розглянемо детальніше, як запропонований в попередньому підрозділі метод реалізується за допомогою цього засобу.

Зашифрований репозиторій вихідного коду. Репозиторій зберігається на сервері у зашифрованому вигляді. Для шифрування вмісту використовується симетричний алгоритм AES-256, який є сучасним стандартом блочного шифрування даних [4]. Кожен файл або весь том репозиторію може бути зашифрований ключем AES. У результаті, навіть, маючи доступ до файлової системи сервера репозиторію, зловмисник не зможе прочитати вихідний код без знання ключа.

Захист ключа сховища (AES) за допомогою RSA. Щоб надати авторизованим користувачам можливість доступу до зашифрованого репозиторію, необхідно безпечно зберігати і передавати ключ. Для цього застосовано асиметричне шифрування RSA: генерується пара ключів public_key/private_key. Відкритий ключ використовується для шифрування симетричного ключа AES. Закритий ключ потрібен для розшифрування і отримання AES. Такий підхід відповідає типовій схемі гібридного шифрування, де поєднуються висока швидкість AES та зручність RSA у розповсюдженні ключів [5]. Розрядність RSA-ключів обирається відповідно до сучасних вимог (не менш 2048 біт, а краще 3072 біт і більше, враховуючи довгострокову безпеку).

Зберігання ключів в Vault. Для безпечного зберігання RSA ключів використовується HashiCorpVault – централізоване сховище секретів. Vault налаштований таким чином, що приватний ключ RSA доступний лише автентифікованим і авторизованим користувачам. Сам приватний ключ ніколи не зберігається відкрито на сервері репозиторію; він знаходиться

лише у сховищі Vault у зашифрованому вигляді, а доступ до нього контролюється політиками.

Автентифікація користувача через LDAP. Користувач (розробник програмного коду), який бажає отримати доступ до репозиторію (наприклад, виконати gitclone або gitpull), спочатку проходить автентифікацію через LDAP-сервер. Це означає, що він вводить свої облікові дані, які перевіряються в системі директорій (наприклад, ActiveDirectory). LDAP повертає інформацію про успішність автентифікації та, за необхідності, про групи/роль користувача. На рис. 1 показано взаємодію: користувач відправляє свої дані для входу на LDAP-сервер, проходить перевірку, після чого від сервера отримуються атрибути, зокрема роль доступу.

Отримання доступу до закритого ключа (private_key) через Vault. Vault інтегровано з LDAP-автентифікацією, тобто користувач може увійти до Vault, використовуючи свої LDAP-облікові дані (власне Vault виступає клієнтом LDAP). При цьому Vault зіставляє групи користувача з попередньо налаштованими політиками доступу. Такий підхід спрощує управління правами: достатньо керувати групами у LDAP, а Vault автоматично застосує відповідні політики доступу. У результаті після успішного входу користувач отримує від Vault тимчасовий токен або інший механізм доступу, з допомогою якого може запитати закритий ключ private_key. Vault перевіряє токен, політику і у разі дозволу надає копію закритого ключа для використання.

Доступ до репозиторію (gitclone/push) з розшифруванням. Отримавши приватний ключ із Vault, клієнт (користувач) може розшифрувати симетричний ключ AES (доставлений, наприклад, у вигляді файлу або конфігурації репозиторію, зашифрованої відкритим ключем). Після цього клієнтський інструмент використає AES для розшифрування вмісту репозиторію. Таким чином, коли розробник виконує gitclone, на його стороні відбувається отримання зашифрованих даних репозиторію з сервера, розшифрування цих даних з використанням ключа AES (який виводиться з Vault через RSA-розшифрування). При операції gitpush процес відбувається у зворотному порядку: перед відправкою на сервер вихідний код шифрується ключем AES і в такому вигляді зберігається на сервері. сервер ніколи не бачить відкритого (незашифрованого) вмісту – він оперує лише зашифрованими артефактами.

Варто зауважити, що для реалізації описаної методології на практиці можуть бути використані наявні інструменти та компоненти. Наприклад, шифрування сховища AES-256 можна реалізувати засобами операційної системи (BitLocker для Windows, LUKS/dm-crypt для Linux, VeraCrypt для обох ОС). Зберігання ключів і керування доступом реалізується через розгорнутий сервер HashiCorpVault, налаштований на роботу з LDAP. У процес інтегрується скрипт або утиліта, що автоматизує витяг ключа з Vault і його застосування при виконанні Git-операцій.

Запропоноване рішення охоплює кілька аспектів безпеки: конфіденційність зберігання коду, контроль доступу розробників, ізоляцію секретних ключів та керування ними протягом життєвого циклу. Використання алгоритму AES-256 для шифрування вихідного коду забезпечує високий рівень стійкості. Ключ довжиною 256 біт є криптостійким і не може бути підібраний перебором за прийнятний час. Таким чином, навіть якщо зловмисник отримає файлова копія репозиторію (наприклад, через компрометацію резервної копії або диска сервера), без ключа він не зможе відновити вихідний код. Важливо підкреслити, що це рішення фокусується на конфіденційності; питання цілісності та автентичності коду вирішуються іншими засобами (наприклад, цифровим підписом комітів або перевіркою цілісності), які поза межами цього дослідження

Асиметричний захист ключа AES усуває потребу передавати симетричний ключ кожному користувачу по захищених каналах вручну. Ключі RSA зберігаються тільки у Vault. Безперечна перевага – розмежування прав доступу: у запропонованому рішенні користувач фактично отримує доступ лише до розшифрованого AES після автентифікації. У криптографічному сенсі, комбінація AES+RSA добре вивчена: відомо, що RSA

використовується у першу чергу для керування ключами, тоді як AES – для масиву даних [5].

Використання HashiCorpVault дозволяє включити централізоване керування секретами у процес розробки. Однією з ключових переваг є аудит і контроль доступу: Vault веде журнал всіх звернень до секретів, отже адміністратор може відстежити, хто і коли отримував ключ. Рішення дозволяє працювати з кількома репозиторіями – для кожного можна зберігати свою пару ключів і призначати різні політики на різні LDAP-групи.

Інтеграція з LDAP забезпечує масштабованість управління доступом. Адміністраторам не потрібно вручну додавати кожного користувача в Vault – достатньо підтримувати актуальність груп в LDAP. При цьому, якщо співробітник залишає компанію або змінює проєкт, його обліковий запис відключається чи переноситься до іншої групи в LDAP, і відповідно він миттєво втрачає або отримує доступ до певних секретів в Vault. Цей механізм узгоджується з підходами ZeroTrust та принципом найменших привілеїв, адже тільки автентифікований і авторизований суб'єкт може отримати секрет, і ніхто за межами цієї системи – ні компрометований сервер репозиторію, ні сторонній спостерігач – не в змозі його отримати.

Процес регулярної заміни криптографічних ключів є критичним для довгострокової безпеки. Ключ не повинен використовуватися довше визначеного часу або обсягу даних. У нашому рішенні це стосується і симетричного ключа AES, і пари RSA-ключів. Ротація RSA-ключів може здійснюватися раз на 1–2 роки, або частіше у разі підозри компрометації/втрати. Vault спрощує цю задачу технічно, але організаційно важливо мати процедуру повідомлення усіх учасників і оновлення конфігурацій (щоб ніхто не залишився з старим приватним ключем і, відповідно, без доступу).

Впровадження симетричного (AES) та асиметричного (RSA) шифрування вихідного коду здатне незначно вплинути на швидкодію CI/CD-процесів. Симетричний алгоритм AES є високопродуктивним завдяки апаратному прискоренню (розшифрування вимірюється в гігабайтах за секунду), тому розшифрування навіть великого репозиторію займає долі секунди. RSA використовується лише для шифрування/розшифрування ключів AES, тож його вплив на час виконання мінімальний.

Окрім власне розшифрування коду, певні затримки може внести інтеграція з системою керування секретами. Перед збіркою або тестуванням агент CI повинен автентифікуватися у Vault і отримати ключ для розшифрування. Кожен такий запит через API Vault додає невелику затримку до конвеєра. Наприклад, при використанні Jenkins з плагіном Vault, пайплайн на початку виконує вхід до Vault (за допомогою AppRole, JWT тощо) і зчитує секрет – ці дії займають додаткові частки секунди. Якщо для автентифікації Vault налаштовано LDAP, то кожен логін може спричинити звернення до LDAP і додаткову затримку (як правило, десятки мілісекунд залежно від продуктивності LDAP-сервера). Також розшифрування коду вимагає ресурсів на самому CI-агенті – CPU для AES, операції введення/виводу для читання і запису файлів. В цілому, шифрування репозиторію додає коротку підготовчу стадію в конвеєрі (розшифрування перед збиранням), яка збільшує час виконання операцій розгортання заради суттєвого посилення безпеки.

Застосування Vault для зберігання ключів шифрування зміщує фокус загроз саме на цю систему. Vault стає єдиною точкою зламу: у разі компрометації зловмисник отримує "головний ключ" до всіх систем і секретів, що в ньому зберігаються. До основних ризиків належать несанкціонований доступ до приватних ключів та інших секретів, який може статися через вразливості або людський фактор. Наприклад, конфігураційні помилки здатні надати доступ до даних: надто широкі ACL-політики Vault або привілеї для CI-ролей можуть дозволити отримати ключі зловмисником. Відсутність шифрування трафіку або перевірки сертифікатів (наприклад, якщо вимкнено TLS) відкриває шлях для перехоплення секретів посередником. Такі витоки дозволяють зловмиснику отримати приватні ключі шифрування, обійти механізми захисту вихідного коду і потенційно непомітно викрасти або модифікувати його. Таким чином, любий витік або злам Vault є критичним: організація змушена вважати

скомпрометованими всі секрети й ключі, ініціювати масову ротацію, аналіз журналів доступу та проводити розслідування.

Для безпечного використання Vault у контексті CI/CD необхідно дотримуватися принципів багаторівневого захисту та найкращих практик управління секретами:

- Централізований аудит і моніторинг. Увімкнення аудит-логівVault є обов'язковим кроком: Vault реєструє кожен запит на отримання або розшифрування секрету, що створює повний ланцюжок подій для аналізу. Централізований аудит дозволяє відстежити, хто і коли отримував доступ до приватних ключів, і вчасно помітити аномалії.
- Строгі політики доступу (leastprivilege). Кожен CI/CD-процес або агент повинен мати токен Vault, обмежений лише тими секретами, які потрібні для його роботи. Наприклад, процес збірки front-end не повинен мати доступу до секретів back-end або до самих приватних ключів шифрування, якщо вони йому не потрібні. Таке розмежування прав ізолює потенційний витік: компрометація одного токена дасть змогу атакувальнику отримати лише обмежений набір даних.
- Ротація ключів і токенів. Основна мета ротації – забезпечити безперервний процес оновлення версій ключів для підвищення стійкості до витоків. Vault підтримує автоматичну ротацію внутрішніх ключів і секретів: можна налаштувати політики, щоб кожен виданий токен або динамічний секрет мав короткий TTL і відкликався після використання. Таким чином, навіть якщо токен CI/CD зазнає витоку, він швидко стає недійсним.
- Безпечна інтеграція у pipelines. Не менш важливо правильно інтегрувати Vault у процес CI/CD, щоб виключити випадковий витік секретів. Для зменшення ризиків витоку секретів через логи розгортання чи збірки використовуються механізми маскування секретів у логах ланцюга розгортання. Наприклад, JenkinsVaultPlugin автоматично приховує значення секретів у консолі – навіть якщо випадково спробувати їх надрукувати, вони будуть замасковані.

Важливо відзначити, що інтеграція таких підходів у CI/CD забезпечує безпечне зберігання секретів. Тобто жодні паролі чи ключі не додаються в скрипти ланцюга постачання – вони вичитуються динамічно з Vault.

Отже, наведені приклади демонструють, що запропонована схема сумісна з вимогами DevSecOps: вона дозволяє автоматизувати доставку ПЗ без ручного управління ключами, інтегрується з популярними інструментами, і при цьому підвищує рівень безпеки на кожному етапі – від написання коду (захищений репозиторій) до розгортання (контрольований доступ до секретів на етапах розгортання).

Висновки та перспективи подальших досліджень

У статті проаналізовано підхід до захисту ланцюга постачання програмного забезпечення, заснований на наскрізному використанні криптографії – від зберігання вихідного коду до процесів безперервної інтеграції. Виконаний аналіз дозволив виявити спільний недолік відомих рішень – відокремлення завдання розмежування прав доступу до кодової бази від решти завдань управління доступом на підприємстві. Запропоноване рішення поєднує симетричне шифрування для захисту вмісту репозиторію та асиметричне шифрування для захисту ключів, а також інфраструктурні компоненти, які допомагають інтегрувати виконання завдань із захисту програмного забезпечення до загальних процесів кібербезпеки підприємства.

Розглянуто практичні аспекти реалізації цієї схеми. Показано, що інтеграція з корпоративним LDAP спрощує управління доступом у великих командах, уніфікує автентифікацію та дозволяє використати наявні політики безпеки організації. Використання Vault дозволяє досягти принципу мінімальних привілеїв – доступ до критичних криптографічних ключів надається лише в обсязі та на час, необхідний для роботи, що знижує ризик їх компрометації.

Поєднання перевірених криптоалгоритмів (AES, RSA) з сучасними інструментами

керування ключами (Vault) та використовуваними системами автентифікації (LDAP) дозволяє побудувати багаторівневий захист без істотного впливу на продуктивність і зручність роботи розробників. Така схема забезпечує конфіденційність вихідного коду, контроль за поширенням ключів і гнучке управління доступом, що є критичним у світі, де supplychain атаки стають все більш витонченими. Впровадження описаного підходу може стати важливим кроком для організацій, які прагнуть мінімізувати ризики компрометації своїх програмних продуктів на етапі їх створення та доставки. Подальші дослідження можуть бути спрямовані на гранулювання прав доступу у захищених репозиторіях.

СПИСОК ЛІТЕРАТУРИ

1. Venkata Thej Deep Jakkaraju. Homomorphic Encryption Driven CI CD Pipelines for Zero-Trust Builds. *International journal of communication networks and information security*. 2022. №14. P. 1129–1139. URL: https://doi.org/10.5281/zenodo.15723391 (date of access: 04.08.2025).

2. Hashi Corp. Vault Hashi Corp. S. l. : s. n. URL: https://developer.hashicorp.com/vault.

3. OWASP top 10 API security risks – 2023. S. l. : s. n. 2023. URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (date of access: 04.08.2025).

4. Report on the Development of the Advanced Encryption Standard (AES) / J. Nechvatal et al. *Journal of research of the national institute of standards and technology*. 2001. Vol. 106, №3. P. 511. URL: https://doi.org/10.6028/jres.106.023 (date of access: 04.08.2025).

5. Sa`idu S. An Improved Cryptographic Scheme Using AES & RSA Algorithms for Maximum Security in File Encryption and Decryption. *International journal of scientific development and research*. 2022. Vol. 7, №6. P. 494–500. URL: https://ijsdr.org/papers/IJSDR2206079.pdf (date of access: 05.08.2025).

6. Barker E. Recommendation for key management. Gaithersburg : s. n., 2020. 170 p. URL: https://doi.org/10.6028/NIST.SP.800-57pt1r5 (date of access: 05.08.2025).

7. Vera Crypt. S. l. : s. n.. URL: https://veracrypt.io/en/Documentation.html (date of access: 05.08.2025).

8. Microsoft Inc. BitLocker overview. S. l. : s. n. URL: https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/ (date of access: 05.08.2025).

9. Hashi Corp vault secrets management: 18 biggest pros and cons. Contino. URL: https://www.contino.io/insights/hashicorp-vault (date of access: 04.08.2025).

10. Open LDAP S. l. OpenLDAP Software 2.6 Administrator's Guide. URL: https://www.openldap.org/doc/admin26/ (date of access: 05.08.2025).

11. How to create encrypted git repositories with git-remote-gcrypt. LinuxConfig. URL: https://linuxconfig.org/how-to-create-encrypted-git-repositories-with-git-remote-gcrypt#:~:text=Git%20is,%20by%20far,%20the,with%20this%20goal%20in%20mind (date of access: 04.08.2025).

12. Git-remote-gcrypt. S. l.: Joey Hess, Sean Whitton. URL: https://github.com/spwhitton/git-remote-gcrypt (date of access: 05.08.2025).

Стаття надійшла до редакції 01.09.2025.
Стаття пройшла рецензування 29.09.2025.

Баришев Юрій Володимирович – канд. техн. наук, доцент кафедри захисту інформації, e-mail: yuriy.baryshev@vntu.edu.ua.

Скіп Андрій Володимирович – аспірант кафедри захисту інформації, e-mail: phd@askip.me.
Вінницький національний технічний університет.