УДК 004.056.5
DOI: 10.25140/2411-5363-2021-1(23)-87-95

Ігор Меліхов, Володимир Базилевич

ЗАХИСТ АЛГОРИТМІВ І ДАНИХ НА СТОРОНІ КЛІЄНТА

Захист даних та алгоритмів є важливою складовою розробки програмного забезпечення. Не завжди є можливість використати сервер як надійне сховище даних та алгоритмів, тому існує необхідність детального вивчення та розробки способів захисту даних та алгоритмів на стороні клієнта.

У статті розглянуто прийоми захисту даних з використанням програмних засобів різних операційних систем та програмних платформ. Розроблений алгоритм захисту даних використовує симетричне шифрування AES та різноманітні способи генерації ключа для убезпечення створеної програми від зворотної розробки.

***Ключові слова**: захист даних; захист алгоритмів; симетричне шифрування; обфускація; безпека через неясність.*
Бібл.: 21.

Актуальність теми дослідження. У сучасному світі захист інформації є важливим завданням. Щороку науковці та фахівці з кібербезпеки розробляють нові алгоритми та способи захисту даних, а криптоаналітики та реверс-інженери знаходять способи їх обійти.

Найбільш надійним (і рекомендованим у переважній більшості випадків) способом захисту алгоритмів і даних є зберігання їх на стороні сервера. Звичайно, отримати віддалений доступ до сервера можливо, але це зазвичай набагато складніше і, відповідно, дорожче, аніж отримати дані на стороні клієнта. Однак бувають випадки, коли зберігати дані або виконувати алгоритми на сервері неможливо або незручно. У таких випадках виникає необхідність захисту на стороні клієнта з використанням принципу безпеки через неясність [1]. Це означає, що система може мати недоліки, проте зловмисники не зможуть ними скористатися, оскільки вони про них не знають.

Щоб реалізувати такий захист у власній системі, необхідно розуміти, які прийоми та технології можуть бути використані для цього, та за якими принципами створюються алгоритми такого захисту.

Прикладами задач, що можуть використовувати захист на стороні клієнта, можуть бути: захист комерційно важливих алгоритмів, зберігання логінів та паролів у браузерах, зберігання даних кредитних карт, шифрування конфіденційних листів та інших файлів, збільшення надійності технічних систем захисту авторських прав (англ. DRM – digital rights management), захист алгоритмів, що стосуються монетизації програм та відеоігор, зокрема покупок у застосунку або інтеграції рекламних мереж тощо.

Постановка проблеми. На жаль, досі не існує ґрунтовного дослідження питання захисту даних та алгоритмів на стороні клієнта, тож у цій роботі будуть сформульовані основні принципи побудови такого захисту.

Будемо вважати аксіомою те, що будь-які дані та алгоритми на стороні клієнта можливо отримати, яким би чином вони не були захищені. Тим не менш, можливо захистити дані так, щоб складність несанкціонованого доступу до них була вищою за потенційну вигоду від цього. Отже, щоб обрати спосіб захисту даних на стороні клієнта, слід врахувати цінність даних та алгоритмів, визначити умовну вартість несанкціонованого доступу до них та порівняти ці значення. Наприклад, якщо в однокористувацькій комп'ютерній відеогрі є можливість купувати ігрову валюту за реальні кошти, дані про ігрову валюту варто захистити, щоб зменшити ймовірність втрати прибутку. Проте захист принципово нового алгоритму програми, через розкриття якого компанія може понести значні збитки, вимагає набагато складніших підходів. Отже, важливо розуміти, наскільки надійним та складним у реалізації є той чи інший спосіб захисту даних.

Аналіз останніх досліджень і публікацій. Більшість останніх досліджень і публікацій у сфері захисту алгоритмів та даних пов'язана із захистом даних на стороні сервера. Тим не менше, ведуться окремі дослідження питань обфускації коду та симетричного шифрування.

Зокрема, Xu, H., Zhou, Y., Ming, J., та Lyu, M. у своїй статті «Layered obfuscation: a taxonomy of software obfuscation techniques for layered security» [2] провели детальний та ґрунтовний аналіз способів обфускації Android-застосунків та запропонували багатошаровий підхід до захисту алгоритмів таких застосунків.

Dong, S., Li, M., Diao, W., Liu, X., Liu, J., Li, Z., Zhang, K. та інші у своїй статті «Understanding android obfuscation techniques: A large-scale investigation in the wild» [3] не тільки детально розглянули основні способи обфускації Java-коду, а й зібрали статистичні дані щодо того, які способи обфускації найчастіше використовуються в тих чи інших типах застосунків. Ці дані можуть бути використані розробниками на практиці, полегшуючи вибір способу захисту коду власних застосунків.

Цікавою виявилася також робота Abdullah, A. «Advanced encryption standard (aes) algorithm to encrypt and decrypt data.» [4], в якій автор проводить детальний аналіз шифру AES та наводить сфери його застосунку.

Виділення недосліджених частин загальної проблеми. Більшість сучасних досліджень, що стосуються захисту алгоритмів та даних на стороні клієнта, покривають лише окремі технології, що використовуються під час створення систем такого захисту. Зокрема, досліджуються алгоритми AES, але не досліджуються способи генерації та захисту ключа, який використовується для шифрування та дешифрування даних. Крім того, не аналізуються вже відомі методи захисту даних та алгоритмів у сучасних операційних системах та програмних платформах.

Постановка завдання. У цій статті ми проаналізуємо основні способи захисту даних та алгоритмів на стороні клієнта з використанням засобів сучасних операційних систем та програмних платформ, а також розробимо власний алгоритм захисту довільних даних на стороні клієнта й захистимо його від зворотної розробки. Крім того, слід виділити загальні властивості таких алгоритмів, щоб розробники могли ефективно реалізовувати свої надійні версії таких алгоритмів.

Виклад основного матеріалу. Після детального вивчення наявних алгоритмів і програм, що використовують захист даних на стороні клієнта ми сформували список основних способів реалізації таких алгоритмів.

1. Засоби операційної системи. Більшість сучасних операційних систем реалізують певні методи зберігання даних у недоступному для користувача місці.

Для операційних систем сімейства Windows, зокрема, існує системний реєстр [5], довільний доступ до якого потребує прав адміністратора. До того ж для захисту даних Windows пропонує системну функцію CryptProtectData, яка використовує дані користувача Windows для шифрування даних. Цей підхід використовується, зокрема, у браузерах для зберігання паролів, проте вважається досить ненадійним, оскільки отримавши доступ до комп'ютера, їх можна легко дізнатися.

Операційна система Android дозволяє зберігати дані у трьох стандартних видах контейнерів: SharedPreferences, Account Manager та SQLite. Усі три засоби зберігання інформації є досить надійними, доки користувач не має root-прав, оскільки дані зберігаються у файлах, доступ до яких операційна система надає тільки застосункам, які ці файли створили. Починаючи з версії 5.0 (Lollipop), будь-які дані користувача перед записом до таких файлів зашифровуються, а перед зчитуванням розшифровуються. Проте, оскільки на більшості пристроїв можна легко змінити прошивку, а сама операційна система Android має відкритий вихідний код, вважати, що будь-який пристрій під керуванням операційної системи Android версії вищої за 5.0 реалізує цей підхід, хибно. Тому для убезпечення даних користувача можна використовувати сторонні бібліотеки, що реалізовують бази даних із вбудованим функціоналом захисту інформації. Крім того, деякі сучасні застосунки просто не запускаються, якщо користувач має root-права. Прикладом такого застосунку є мобільний додаток McDonald's[6].

Операційна система Linux має гнучкі права доступу до файлів, тож для захисту даних можна використати файл, який може читати тільки спеціально створений для цього користувач. Відповідно, для того щоб його відкрити, необхідно буде змінити права доступу за допомогою root-акаунта. Отже, крім прав доступу, все-таки варто додатково використати шифрування.

Операційні системи iOS та Mac OS мають вбудований механізм надійного зберігання даних під назвою Keychain Services[7]. Цей механізм дозволяє зберігати невеликі за обсягом дані в безпечному сховищі, до якого має доступ тільки конкретний застосунок. Оскільки системи iOS та Mac OS є закритими, доступ до даних в цьому сховищі можна отримати тільки з використанням джейлбрейку (англ. Jailbreak). Відповідно до офіційної документації, у сховищі Keychain Services можна зберігати таку конфіденційну інформацію, як паролі та дані кредитних карт.

2. Шифрування. Для шифрування даних на стороні клієнта зазвичай використовуються симетричні алгоритми шифрування. Це пов'язано з тим, що симетричне шифрування відбувається за єдиним приватним ключем. Таких алгоритмів існує достатньо велика кількість, зокрема алгоритм Advanced Encryption Standard (AES)[8] сьогодні є стандартним алгоритмом захисту інформації в США. Іншими прикладами симетричних криптосистем є шифри DES, RC4, RC5, SEAL та інші. Переважно безпеку інформації забезпечує не тільки сам алгоритм, а і приватний ключ, що використовується для шифрування та дешифрування даних. Такий ключ варто добре захищати в коді (про що мова йтиме нижче) або генерувати під час роботи програми.

3. «Екзотичні» методи. До таких методів захисту даних користувачів може належати зберігання даних в тих місцях, де їх важко знайти, шифрування власними алгоритмами тощо.

Способи захисту алгоритмів та конфіденційних даних, наявних у коді, зазвичай залежать від мови програмування, на якій реалізована програма, проте майже для кожної мови програмування існують спеціальні утиліти - обфускатори. Нікольська Ксенія Юріївна дає наступне визначення поняттю «обфускація»: «Обфускація ... – приведення вихідного тексту програми до вигляду, що зберігає його функціональність, однак робить складнішим його аналіз, розуміння алгоритмів роботи»[9]. Обфускатори – це утиліти, які беруть вихідний (або скомпільований) код програми і «заплутують» його, ускладнюючи процес зворотної розробки (реверс-інжинірингу) вашої програми. Використовуючи обфускатори з компільованими мовами програмування, треба бути обережним, оскільки це може позначитися на швидкодії програми. Дослідженням алгоритмів обфускації займаються такі науковці, як Gregory Wroblewski [10], Shafi Goldwasser та Guy N. Rothblum [11] та інші.

Крім захисту статичного коду, варто приділяти увагу також захисту даних під час виконання програми, оскільки вміст оперативної пам'яті та навіть регістрів процесора в будь-який момент часу можна визначити за допомогою програм-налагоджувачів.

Розглянемо найпоширеніші мови програмування, а також засоби, що можуть бути використані для убезпечення алгоритмів та конфіденційних даних для тієї чи іншої мови програмування:

1. C/C++ (а також будь-які інші мови програмування, що компілюються безпосередньо у виконуваний код). Фактично, першою ланкою захисту програми від реверс-інжинірингу є сам процес компіляції. Здебільшого код, який можна отримати за допомогою декомпіляторів, значно відрізняється від початкового і є досить заплутаним. Аби ускладнити роботу реверс-інженера, варто прибрати з коду зайві експорти та інформацію на-

лагоджувача, оскільки імена експортованих функцій зберігаються в програмі в текстовому вигляді. Проте, відкривши виконуваний файл у будь-якому редакторі шістнадцяткових значень, можна побачити в ньому всі рядкові константи, що використовуються в програмі, зокрема ключі шифрування даних. Для захисту таких даних треба формувати їх під час роботи програми. Для цього можна використовувати будь-які способи, які тільки можна собі уявити: побітові операції, форматні рядки sprintf, байти заздалегідь відомих значень глобальних та локальних змінних, певні параметри, унікальні для даного пристрою, хешування певних ділянок коду чи даних програми тощо. Крім того, варто заплутати й алгоритм створення таких ключів. Його можна рознести в різні частини виконуваного файлу, або навіть написати частину алгоритму на асемблері. Можна вважати, що процес захисту даних та алгоритмів для компільованих у виконуваний код мов програмування є процесом творчим та обмеженим тільки фантазією розробника. Крім того, існують готові інструменти захисту застосунків на C++, серед них варто зазначити продукти StarForce Crypto та StarForce C++ Obfuscator [12]. Вони виконують із кодом спеціальні перетворення, які значно ускладнюють процес реверс-інжинірингу.

2. Kotlin/Java/JVM. Оскільки скомпільований код JVM легко декомпілювати, а результат роботи декомпілятора наближений до вихідного коду програми, його захист потребує більших зусиль. Певний рівень захисту можна отримати з допомогою обфускаторів. Найпопулярнішою з таких утиліт є оптимізатор ProGuard [13]. Він виконує два основні завдання – оптимізація та обфускація коду. Корисними для захисту алгоритмів на стороні клієнта є обидва процеси, проте захистити константи таким чином неможливо. Для їх більш надійного зберігання можна використати будь-які симетричні алгоритми шифрування або прийоми створення рядків під час роботи програми, наведені для мови програмування C/C++. Проте найнадійнішим способом залишається делегування констант з бібліотеки, написаної на C/C++ за допомогою інтерфейсу JNI, і захист їх саме в цій бібліотеці.

3. C#/.NET. Для програм, написаних для цієї платформи, існує ряд обфускаторів, кожен з яких пропонує власні алгоритми. Існують досить ґрунтовні статті, які порівнюють найвідоміші з таких обфускаторів [14]. Проте більшість із цих інструментів є комерційними продуктами, і їх вартість досить висока. Більшість проєктів із відкритим вихідним кодом є ненадійними, оскільки знаючи алгоритм їх роботи, можна відновити і алгоритм роботи вашої програми, і необхідні константи в коді. Серед відкритих проєктів, що активно підтримуються, можна назвати Obfuscar [15].

4. JavaScript та вебтехнології. Оскільки JavaScript є скриптовою мовою програмування, для вивчення вихідного коду не потрібно використовувати декомпілятори – програма зазвичай поширюється у відкритому вигляді. При такому підході захист вихідного коду є дуже важливим. Найпростішим способом захисту алгоритмів та даних в такому випадку є шифрування власне самого коду та постачання його клієнту разом з функцією, яка вміє його дешифрувати та виконує з допомогою функції eval. Утиліти, що дозволяють виконати відповідні дії, називають крипторами. Недоліком цього методу є те, що якщо зрозуміти принцип роботи функції-дешифратора, можна швидко отримати вихідний код. Також обсяг результуючого коду збільшується в середньому на 30 %. Іншим способом захисту коду є уже знайомі нам обфускатори. Найчастіше вони змінюють імена змінних та функцій, додають псевдокод, замінюють символи на їхнє hex-значення, тощо. До того ж варто додатково мініфікувати весь код (прибрати зайві символи форматування), якщо це не зробив один з вище зазначених інструментів. Це навряд зупинить реверс-інженера, проте створить додаткові незручності. Результат маніпуляцій варто перевірити на працездатність (деякі обфускатори некоректно опрацьовують код з пропущеними знаками «;»), а також в одному (або декількох) онлайн-сервісах деобфускації, наприклад, Online JavaScript Beautifier [16]. Можна також розділити код на кілька

частин і обфускувати різними інструментами, щоб ускладнити деобфускацію. Ще одним цікавим способом захистити алгоритм JavaScript від реверс-інжинірингу – заборонити налагодження, як це продемонстровано у прикладі [17]. Звісно, цей захист також не є абсолютним, оскільки існують готові рішення, що дозволяють повноцінно використовувати налагоджувач навіть в таких веб-додатках. Таким чином досягти такого рівня захисту алгоритмів чи даних, який забезпечує C/C++ або навіть Java, зазвичай не вдається.

5. Python. При роботі з цією мовою програмування також можна використовувати кілька підходів. По-перше, оскільки це скриптова мова програмування, для неї можна використати криптори. По-друге, існують обфускатори для програм на мові Python, проте зазвичай вони менш ефективні, ніж обфускатори для інших мов програмування, через особливості синтаксису. Окрім того, Python підтримує компіляцію в байт-код[18]. На жаль, цей байт-код може бути легко декомпільований, в ньому зберігаються навіть деякі імена змінних. Єдиним достатньо надійним способом зберігання даних та алгоритмів в програмі на мові Python є використання механізму байндингів до інших мов програмування, зокрема до C/C++. Критичні дані або алгоритми зберігаються в компільованому модулі на мові C/C++ та використовуються з Python-програми за потреби.

Розробимо власний алгоритм захисту даних на стороні клієнта. Задачу можна сформулювати таким чином: необхідно розробити алгоритм захищеного збереження та відновлення довільних даних на комп'ютері користувача. Доступ до цих даних має бути обмеженим поза цією програмою.

Для вирішення цієї задачі створимо динамічну бібліотеку на мові C++, інтерфейсом до якої будуть функції читання даних з файлу та запису даних до файлу. Для безпосереднього шифрування даних скористаємося розповсюдженим алгоритмом симетричного шифрування AES[8]. Оскільки, як пише Shripal Rawal, «... ми можемо бути впевнені у безпеці AES лише якщо його правильно реалізовано і використовується надійне управління ключами»[19], використання готового алгоритму шифрування дозволяє звести задачу безпечного збереження даних до задачі генерації та захисту ключа шифрування.

Процес генерації та захисту ключа, як було зазначено вище, є досить творчим. Для того щоб зловмисник не зміг відновити алгоритм програми, у нашому алгоритмі використані такі підходи:

1. Використання значень глобальних і локальних змінних. Оскільки глобальні і локальні змінні розміщуються в різних секціях виконуваної програми, послідовність змінних з різних ділянок пам'яті важче знайти в коді.

2. Використання даних різного розміру. Додавання даних розміром у 1, 2, 4 та навіть 8 байт, а також окремих фрагментів рядків дозволяє частково спростити процес заповнення масиву значеннями, а також ускладнити аналіз такого коду. Також можна записувати не повне значення змінної, а тільки окремі байти більших за розмірами типів даних з допомогою функції *memcpy*.

3. Використання даних системи, наприклад, номер ревізії процесору чи ім'я комп'ютера. Ці дані дозволять захистити дані від можливості розшифрування на інших комп'ютерах, проте якщо для розв'язку задачі це не є необхідним, можна не використовувати їх.

4. Використання форматних рядків *sprintf* та інших способів формування рядків тексту і значень змінних під час виконання програми. Це додатково ускладнює отримання ключа під час вивчення алгоритму його формування засобами зворотної розробки.

5. Використання підробних умов, циклів, функцій і інших конструкцій, а також перезапис даних у вже заповнених комірках. Вітки коду, що ніколи фактично не виконуються, або результат їх виконання не впливає на фінальний результат, зазвичай небажані в реальних програмах, проте в цьому випадку вони заважають зворотній розробці алгоритму, отже, є бажаними і доцільними.

6. Використання асемблерних вставок. Для нашого прикладу тестовим шляхом ми дізналися, що після виконання попередньої операції в регістрі eax зберігається одне і те ж значення (побічний результат). Використовуючи асемблерні вставки, можемо отримати додаткові байти даних:

```
int eaxdata = 0;
__asm
{
    mov eaxdata, eax
}

*((DWORD*)result + 5) = eaxdata;
```

Цей підхід значно ускладнить роботу декомпіляторів, оскільки подібні операції зазвичай не генеруються компіляторами під час обробки C/C++ коду.

Після генерації базової версії ключа ми застосовуємо до нього різноманітні трансформації:

1. Математичні й побітові операції. Не зважаючи на те, що їхня поведінка досить передбачувана, вони дозволяють позбавитися від повторів та нульових байтів в отриманому ключі. Також можна використати дві копії ключа, виконати над ними різні перетворення, а далі виконати певні побітові операції з використанням згенерованих ключів.

2. Алгоритми, що виконуються над звичайними колекціями: сортування, зворотного порядку та інші. Головне, щоб результат їх виконання був однозначними. Після декомпіляції такі алгоритми досить важко ідентифікувати, тож їх використання також ускладнить зворотну розробку алгоритму.

Крім заплутування процесу генерації та трансформації ключа, варто також захистити сам алгоритм. Найпростіший спосіб це зробити – видалити усі символи налагоджування та зайві експорти. Без назв функцій зловмисник не зможе так легко відновити послідовність дій у програмі та дізнатися, яким саме чином генерується ключ. До того ж у нашому прикладі використовується бібліотека obfy [20], яка дозволяє ще більше ускладнити отриманий після компіляції код.

Варто розуміти, що кожна наступна маніпуляція зменшує швидкодію програми, проте наведемо ще кілька прикладів можливих маніпуляцій, що не були використані в прикладі, проте можуть допомогти краще захистити дані:

1. Ускладнення способу передачі ключа до функції, що реалізує алгоритм AES. Наразі найвужчим місцем системи є передача ключа з функції генерації до функції шифрування. Можна розбити ключ на окремі складові та передавати як окремі параметри, через глобальні змінні, тощо. Частину ключа можна генерувати прямо у функції шифрування, і навіть взагалі не зберігати ключа в єдиному масиві.

2. Збільшення кількості гілок коду, що виконуються, проте не впливають на результат роботи програми.

3. Використання формату статичної бібліотеки замість динамічної. Це накладе певні обмеження та вимоги на спосіб компіляції основної програми, проте дасть можливість «сховати» код генерації ключа у коді програми. Також за потреби можна взагалі відмовитися від реалізації окремого модуля і розподілити генерацію у вихідному коді програми.

4. Передавати певний набір рядків з основної програми. Це може бути версія програми, будь-які рядкові літерали, навіть локалізовані дані. Такий набір рядків може бути змінного розміру та оброблятися в циклі, для кожного рядка заміняємо певні символи ключа на символи рядка.

5. Додаткова обробка вхідних даних. Шифрування кількома алгоритмами з різними ключами у заданій послідовності може значно сповільнити роботу програми з великими обсягами даних, проте для невеликих рядків може забезпечити додатковий захист.

6. Використання додаткових перетворень ключа та/або його складових. Наприклад, можна зберігати частину ключа в зашифрованому вигляді, а під час виконання розшифровувати його, або використовувати результати хешування констант відомими алгоритмами.

Ми розробили приклад алгоритму захисту даних на стороні клієнта, даний алгоритм опубліковано на платформі GitHub [21]. Після фактичної реалізації ми змогли виділити чотири основні властивості алгоритмів захисту даних на стороні клієнта:

1. Відносність. Побудований алгоритм може мати різну стійкість до аналізу, що визначається його структурою, проте ця стійкість не може бути абсолютною. Чим складніший алгоритм, тим більше часу зловмиснику знадобиться на його аналіз, проте отримати ключ все одно можливо.

2. Закритість. Знання вихідного коду або логіки роботи конкретного алгоритму зловмисниками зробить такий алгоритм неефективним для захисту даних, тому розповсюджувати додатки з таким кодом слід лише у скомпільованому вигляді та без символів налагодження. Не слід використовувати готові алгоритми (в тому числі розроблений нами для цієї роботи) без додаткових змін констант та послідовності обробки даних.

3. Детермінованість. Побудований алгоритм має повертати однакові значення на одному комп'ютері з однаковими вхідними даними. Чи має він повертати однакові дані на різних комп'ютерах, залежить від вимог до конкретної реалізації.

4. Комплексність. Для того щоб алгоритм забезпечував достатній рівень безпеки, слід використовувати різні підходи до генерації та перетворень ключа. Чим більше різноманітних операцій використовується в алгоритмі, тим надійнішим є алгоритм, проте тим більше часу може займати його виконання.

Висновки. У процесі досліджень виділено та структурувано основні способи захисту даних та алгоритмів на стороні клієнта. Розглянуто засоби, які для цього надають популярні операційні системи, а також яким чином можна захищати дані користувача й алгоритми програм із використанням різних технологій програмування. У кожного з таких підходів є власні переваги та недоліки, проте одним із перспективних напрямів вважаємо використання власного алгоритму захисту даних, що використовує AES як алгоритм шифрування та генерує ключ для нього з використанням даних системи та власних констант. Отриманий алгоритм не можна використовувати напряму, оскільки він є публічно доступним, тому для захисту даних у власних застосунках слід створити власний алгоритм на його основі. Для того щоб спростити подальшу побудову подібних алгоритмів і забезпечити їх високий рівень захисту, сформульовано чотири основні властивості, якими має володіти будь-який алгоритм захисту даних на стороні клієнта.

На нашу думку, дане дослідження є ґрунтом для подальшого вивчення теми захисту даних і алгоритмів на стороні клієнта. Пошук найбільш ефективних і надійних способів створення та перетворення ключа, а також обфускації відповідного коду є важливим кроком до створення надійних алгоритмів захисту даних на стороні клієнта.

Список використаних джерел

1. What is Security through Obscurity? URL: https://securitytrails.com/blog/security-through-obscurity.

2. Xu Hui, Yangfan Zhou, Jiang Ming, Michael Lyu. Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. *Cybersecurity.* 2020. № 3. Pp. 1-18.

3. Understanding android obfuscation techniques: A large-scale investigation in the wild / Dong, Shuaike, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang. *International Conference on Security and Privacy in Communication Systems*. Springer, Cham, 2018. Pp. 172-192.

4. Abdullah A. Advanced encryption standard (aes) algorithm to encrypt and decrypt data. *Cryptography and Network Security.* 2017. № 16.

5. Системний реєстр. URL: https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry.

6. Додаток McDonald's. URL: https://www.mcdonalds.ua/ua/Eat/GMAL.html.

7. Офіційна документація до механізму Keychain Services. URL: https://developer.apple.com/documentation/security/keychain_services.

8. Advanced Encryption Standard. URL: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

9. Никольская К. Ю., Хлестов А. Д. Обфускация и методы защиты программных продуктов. *Вестник УрФО. Безопасность в информационной сфере.* 2015. № 2(16). С. 7-10.

10. General method of program code obfuscation. *Wroblewski, Gregory.* 2002.

11. Shafi Goldwasser, Guy N. Rothblum On best-possible obfuscation. *Theory of Cryptography Conference*. Springer, Berlin, Heidelberg, 2007.

12. Захист та обфускація вихідного коду за допомогою продуктів StarForce Crypto та StarForce C++ Obfuscator. URL: http://www.star-force.ru/solutions/source-code-protection/.

13. Оптимізатор ProGuard. URL: https://www.guardsquare.com/en/products/proguard.

14. Огляд обфускаторів для .NET. URL: https://habr.com/ru/post/97062/.

15. Відкритий обфускатор для .NET. URL: https://github.com/obfuscar/obfuscar.

16. Online JavaScript Beautifier. URL: https://beautifier.io/.

17. Anti code-protection Demo. URL: https://ohmycoding.com/demos/anti-code-protection.html.

18. Опис модуля py_compile. URL: https://docs.python.org/3/library/py_compile.html.

19. Advanced Encryption Standard (AES) and It's Working. / Shripal Rawal // International Research Journal of Engineering and Technology 3.8 (2016): 1165-1169.

20. fritzone / obfy. URL: https://github.com/fritzone/obfy.

21. IchZerowan / ClientProtect. URL: https://github.com/IchZerowan/ClientProtect.

References

1. What is Security through Obscurity? https://securitytrails.com/blog/security-through-obscurity.

2. Xu Hui, Yangfan Zhou, Jiang Ming, Michael Lyu. (2020). Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. *Cybersecurity*, 3, pp. 1-18.

3. Dong, Shuaike, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang (2018). Understanding android obfuscation techniques: A large-scale investigation in the wild. *International Conference on Security and Privacy in Communication Systems*, pp. 172-192.

4. Abdullah, A. (2017). Advanced encryption standard (aes) algorithm to encrypt and decrypt data. *Cryptography and Network Security,* 16.

5. System Registry. https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry.

6. McDonald's app. https://www.mcdonalds.ua/ua/Eat/GMAL.html.

7. Keychain Services official documentation. https://developer.apple.com/documentation/security/keychain_services.

8. Advanced Encryption Standard. https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

9. Nikolskaya, K. Yu., Hlestov, A. D. (2015). Obfuskatsiia i metody zaschity programmnykh produktov [Obfuscation and software products protection methods]. *Vestnik UrFO. Bezopasnost v informatsionnoy sfere – Bulletin of the Ural Federal District. Information security*, 2(16), pp. 7-10.

10. General method of program code obfuscation. (2002). *Wroblewski, Gregory.*

11. Shafi Goldwasser, Guy N. (2007). Rothblum On best-possible obfuscation. *Theory of Cryptography Conference*. Springer, Berlin, Heidelberg.

12. Source code obfuscation and protection using StarForce Crypto and StarForce C++ Obfuscator. http://www.star-force.ru/solutions/source-code-protection/.

13. ProGuard optimizer. https://www.guardsquare.com/en/products/proguard.

14. .NET obfuscators reveiw. https://habr.com/ru/post/97062/.

15. Open .NET obfuscator. https://github.com/obfuscar/obfuscar.

16. Online JavaScript Beautifier. https://beautifier.io/.

17. Anti code-protection Demo. https://ohmycoding.com/demos/anti-code-protection.html.

18. py_compile module description. https://docs.python.org/3/library/py_compile.html.

19. Shripal Rawal (2016). Advanced Encryption Standard (AES) and It's Working. *International Research Journal of Engineering and Technology,* 3.8, pp. 1165-1169.

20. fritzone / obfy. https://github.com/fritzone/obfy.
21. IchZerowan / ClientProtect. https://github.com/IchZerowan/ClientProtect.

UDC 004.056.5

Ihor Melikhov, Volodymyr Bazylevych

CLIENT-SIDE ALGORITHMS AND DATA PROTECTION

Algorithms and data protection is an important part of software development. It is not always possible to use the server as a reliable storage for data and algorithms, so there is a need for a detailed study and development of methods of data and algorithms protection on the client side.

It is possible to access any data and algorithms on the client side. However, it is possible to protect them so that the complexity of unsanctioned access is greater than the potential benefit from it.

Symmetric encryption and obfuscation have been studied by Xu Hui, Yangfan Zhou, Jiang Ming, Michael Lyu, A. Abdullah and others. We reviewed the latest publications related to symmetric encryption and obfuscation, as well as various articles and discussions on the protection of algorithms and data on the client side on the Internet. There are currently no detailed and comprehensive studies on the topic of the client-side data and algorithms protection.

To structure information about the client-side algorithms and data protection. To develop our own algorithm for client-side data protection. To retrieve the general principles that can be used when developing such algorithms.

This article discusses methods of data protection using different operating systems and software platforms. The developed data protection algorithm uses symmetric AES encryption and various ways of key generation to secure the created program from reverse engineering. The key features of such algorithms are relativity, closeness, determinability and complexity.

The features and principles of algorithm development stated in the article allow to effectively protect data and algorithms on the client side.

***Keywords**: data protection; algorithms protection; symmetric encryption; obfuscation; security through obscurity.*

Меліхов Ігор Олександрович – здобувач вищої освіти, Національний університет «Чернігівська політехніка» (вул. Шевченка, 95, м. Чернігів, 14035, Україна).
Melikhov Ihor – bachelor student, Chernihiv Polytechnic National University (95 Shevchenka Str., 14035 Chernihiv, Ukraine).
E-mail: IchZerowan@gmail.com
ORCID: https://orcid.org/0000-0002-3611-0549

Базилевич Володимир Маркович – кандидат економічних наук, доцент, завідувач кафедри інформаційних та комп'ютерних систем, Національний університет «Чернігівська політехніка» (вул. Шевченка, 95, м. Чернігів, 14035, Україна).
Bazylevych Volodymyr – PhD in Economics, Associate Professor, Head of the Department of Information and Computer Systems, Chernihiv Polytechnic National University (95 Shevchenka Str., 14035 Chernihiv, Ukraine).
E-mail: bazvlamar@gmail.com
ORCID: https:// orcid.org/0000-0001-8935-446X
ResearcherID: G-5764-2014
SCOPUS Author ID: 57193029322