УДК 004.588
DOI https://doi.org/10.32838/2663-5941/2021.4/13

Козак Є.Б.
GAN Inc.

ОРГАНІЗАЦІЯ ЗАХИЩЕНОГО ІНТЕРФЕЙСУ ПЕРЕДАЧІ ДАНИХ НА ОСНОВІ М-АРНОГО АЛГОРИТМУ «TREE-BASED ORAM»

У статті розглянуто принципи організації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM". Сформовано поняття "Tree-based ORAM", зазначається, що ORAM являє собою обумовлений симулятор оперативної пам'яті – компілятор, який перетворює алгоритми таким чином, що алгоритми, які виникають е результаті, зберігають поведінку вводу-виводу вихідного алгоритму, але схема розподілу пам'яті трансформованого алгоритму не залежить від схеми доступу до пам'яті оригінального алгоритму. Наголошено, що, оскільки хмарні обчислення стають дедалі популярнішими, конфіденційність персональних даних користувачів стає основною проблемою під час аутсорсингу обчислень. Доведено, що шаблон доступу до пам'яті програми виявляє великий відсоток її поведінки або зашифрованих даних, на яких вона обчислює. Здійснено визначення безпеки (конфіденційність) для ORAM. Зазначено, що кожен запит DRAM, зроблений під час доступу до ORAM, відбувається в певний час, і, отже, інформація не витікає. Підкреслено, що апаратно ORAM складається з двох компонентів: надійний – вбудований контролер ORAM і ненадійний – зовнішня пам'ять. Сформовано дерево ORAM та описано кожний із рівнів реалізації. Наголошено, що контролер ORAM складається з карти позицій, сховища й логіки управління, водночас карта позицій (або блок позиціонування), коротше PosMap, є таблицею пошуку, яка пов'язує кожен блок даних із випадковим листом і n деревом ORAM, а управління та оптимізація блоку позиціонування є головною умовою для ефективної роботи. Підкреслено, що безпека ORAM полягає в тому, що кожен пошук у блоці позиціонування дає новий випадковий лист для доступу до дерева ORAM, це робить послідовність шляхів дерева ORAM доступною незалежною від фактичного трасування адреси програми. Доведено, що кількість записів у блоці позиціонування масштабується лінійно з кількістю блоків даних в ORAM, що приводить до значного обсягу вбудованого сховища (від сотень кілобайтів до сотень мегабайтів).

Ключові слова: *захист, криптографія, інтерфейс, передача даних, алгоритм, Tree-based ORAM.*

Постановка проблеми. Невидима оперативна пам'ять (ORAM) – це обумовлений симулятор оперативної пам'яті – компілятор, який перетворює алгоритми таким чином, що алгоритми, які виникають в результаті, зберігають поведінку вводу-виводу вихідного алгоритму, але схема розподілу пам'яті трансформованого алгоритму не залежить від схеми доступу до пам'яті оригінального алгоритму. Нещодавно ORAM перетворився на безпечні процесори. Великою проблемою для апаратних схем ORAM є питання, як ефективно управляти картою позицій (блоком позиціонування), центральним компонентом сучасних алгоритмів ORAM. Запроваджений раніше, блок позиціонування призводить до того, що ORAM принципово не піддається масштабуванню з точки зору мікросхеми. З іншого боку, техніка під назвою «Рекурсивний ORAM» вирішує проблему області, але значно збільшує накладні витрати ORAM.

Оскільки хмарні обчислення стають дедалі популярнішими, конфіденційність конфіденційних даних користувачів стає основною проблемою при аутсорсингу обчислень. В ідеалі користувачі хотіли б «перекинути свої зашифровані дані через стіну» на хмарну службу, яка повинна виконувати обчислення цих даних без вивчення службою будь-якої інформації з цих даних. Проте добре відомо, що шифрування недостатньо для отримання конфіденційності. Доведено, що шаблон доступу до пам'яті програми виявляє великий відсоток її поведінки або зашифрованих даних, на яких вона обчислює [1].

ORAM – це криптографічний примітив, який повністю усуває витік інформації у трасі доступу до пам'яті програми (складається з читання / запису в пам'ять). Концептуально ORAM працює, підтримуючи всю пам'ять у зашифрованому та перемішаному вигляді. При кожному доступі пам'ять зчитується, а потім перетасовується. У рамках ORAM будь-який шаблон доступу до пам'яті обчислювально не відрізняється від будь-якого іншого шаблону доступу тієї ж довжини. Уперше ORAM був запропонований Голдрейхом [2], і була проведена значна подальша робота, яка призвела до більш ефективних та криптографічно безпечних схем ORAM.

Важливою складовою частиною використання ORAM постає надійне обладнання. У цьому обладнанні вбудований контролер ORAM, який перехоплює кеш останнього рівня пропускає / виселяє та перетворює їх на заплутані запити основної пам'яті (тобто шаблон адреси рандомізований, дані, зчитувані / записані, шифруються). Оскільки продуктивність програми дуже чутлива до затримки кеш-пам'яті, логіка контролера ORAM реалізована безпосередньо в апаратному забезпеченні.

Аналіз останніх досліджень і публікацій. На сьогодні, питання організації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM" здійснили чимало науковців.

Г.П. Семченко [3] дослідила особливості організації автоматизованих систем захисту даних при застосуванні програмних додатків та інформаційних сховищ хмарних сервісів у військовій сфері. Авторкою узагальнено модель захищеного інтерфейсу взаємодії програмного додатку й апаратної платформи мережевого ресурсу, що дозволяє приховати послідовність виконання процедур зчитування та запису даних на фізичному рівні роботи з оперативною пам'яттю серверу. Науковиця зазначає, що пріоритет у розвитку сучасних схем ORAM полягає не тільки в забезпеченні надійної передачі «чутливих даних», але й в оптимізації алгоритмів їх захисту за умови збереженнявеликих обсягів даних у середовищі хмарного сервісу.

Спосіб організації захищеного каналу передачі даних про стан об'єкта водопостачання у реальному часі з використанням бездротових технологій розробили В.В. Сидоренко й К.О. Буравченко [4]. Авторами продемонстровано можливість роботи системи диспетчеризації з використанням лише однієї глобальної статичної IP-адреси у сервера збору даних. Запропоновано використовувати одноплатний комп'ютер з операційною системою на точці збору сигналів.

О.А. Суліма [5] здійснила аналіз методів оцінок рівня доступу до даних, завдяки чому з'явилась можливість кількісно оцінювати небезпеку і відповідно необхідний рівень безпеки, які забезпечуються певними засобами захисту системи.

Із зарубіжних авторів варто відзначити такі роботи як: M. Islam, M. Kuzu and M. Kantarcioglu [6]; Xiao Wang, Hubert Chan and Elaine Shi [7]; C. Fletcher, M. van Dijk and S. Devadas [8]; Craig Gentry, Shai Halevi, Charanjit Jutla and Mariana Raykova [9]; M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz and D. Song [10]; Craig Gentry, Kenny A Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova and Daniel Wichs [11]; E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu and S. Devadas [12]; E. Shi, T.-H. H. Chan, E. Stefanov and M. Li [13] та інші.

Незважаючи на масштабність наукових досліджень за темою роботи, питання організації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM" є актуальним і потребує детального опрацювання.

Постановка завдання. Мета статті – здійснити дослідження щодо організації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM".

Виклад основного матеріалу дослідження. В умовах сьогодення, враховуючи сучасність і рівень розвитку інформаційних технологій, надійне обладнання (наприклад, захищений процесор) працює в ненадійному середовищі (наприклад, центр обробки даних) від імені віддаленого користувача. Процесор запускає приватну або загальнодоступну програму на приватних даних, поданих користувачем, і взаємодіє з надійним мікросхемним контролером ORAM під час пропусків кешу останнього рівня для доступу до даних у ненадійній зовнішній пам'яті.

Центр обробки даних розглядається як пасивний і активний супротивник. По-перше, центр обробки даних буде пасивно спостерігати, як процесор взаємодіє з DRAM, щоб дізнатись інформацію про зашифровані дані користувача. По-друге, він може додатково спробувати втрутитися у вміст DRAM, щоб вплинути на результат програми та / або дізнатись інформацію про зашифровані дані.

Визначення безпеки (конфіденційність) для ORAM:

Для послідовності запитів даних $\bar{a}$, нехай ORAM ($\bar{a}$) буде результуючою рандомізованою послідовністю запитів даних алгоритму ORAM. Кожен елемент у послідовності запитів даних відповідає стандартному інтерфейсу оперативної пам'яті, тобто є кортежем (адреса, операція, запис даних). Для будь-якої довжини поліномів $\bar{a}$ та $\overline{a'}$ отримані послідовності довжини поліномів ORAM ($\bar{a}$) та ORAM ($\overline{a'}$) обчислювально неможливо розрізнити, якщо | ORAM ($\bar{a}$) | = | ORAM ($\overline{a'}$) |.

Тут | ORAM ($\bar{a}$) | позначає довжину ORAM ($\bar{a}$). Іншими словами, послідовність запитів пам'яті, видима для супротивника, витікає лише з його довжиною. Важливо те, що це визначення дозволяє процесору використовувати звичайний вбудований кеш. $\bar{a}$ – це послідовність завантаження/зберігання інструкцій у програмі. При визначенні | ORAM ($\bar{a}$) |, визначається, кількість пропусків кешу останнього рівня в $\bar{a}$, але не $|\bar{a}|$. Визначення охоплює суть гарантії конфіденційності ORAM: ORAM

приховує окремі елементи у послідовності запитів даних, маючи при цьому витік невеликої кількості інформації, виставляючи довжину послідовності. З інформаційно-теоретичної точки зору перший зростає лінійно з довжиною послідовності запитів, тоді як другий зростає лише логарифмічно.

ORAM поводиться як справжня пам'ять із переважною ймовірністю з точки зору процесора. Пам'ять виконує запис відповідно до дій, якщо значення, яке процесор читає з певної адреси, є найновішим значенням, яке він записав на цю адресу (тобто справжнім та новим), то ORAM здійснює оновлення запису.

У присутності активних супротивників ORAM поводиться як діюча пам'ять і результуюча послідовність запитів ORAM обчислювально не відрізняється до того моменту, поки фальсифікація не визначиться контролером ORAM (тобто, послідовність запитів ORAM забезпечує гарантії конфіденційності описані вище).

Коли виявлено фальсифікацію, процесор отримує виняток, після чого він може знищити програму або вжити певних заходів для запобігання витоку щодо виявлення порушення цілісності.

Кожен запит DRAM, зроблений під час доступу до ORAM, відбувається в певний час, і, отже, інформація не витікає.

Апаратно ORAM складається з двох компонентів: (надійний) вбудований контролер ORAM та (ненадійний) зовнішня пам'ять.

Ненадійне зовнішнє сховище логічно структуровано у вигляді двійкового дерева, як показано на рисунку 1, яке називається деревом ORAM.

З рисунку 1 видно L = 3 рівнів і Z = 4 слотів. Припустимо, блок ***a*** (затінений чорним кольором) відображається на шляху L 1 раз, тобто L=1. У будь-який час блок ***a*** може бути розташований у будь-якій із затінених структур (тобто на шляху 1 або у сховищі).

Рівні дерева ORAM коливаються від 0 (корінь) до L (листя). Кожен вузол у дереві називається сегментом і має фіксовану кількість слотів (позначається Z), які можуть зберігати блоки, які є одиницею даних, запитуваних процесором (наприклад, рядком кешу). Слоти можуть бути порожніми в будь-який момент і заповнені фіктивними блоками. Всі блоки у дереві, включаючи фіктивні блоки, зашифровані за допомогою імовірнісної схеми шифрування, наприклад режиму лічильника AES, із рандомізованим ключем сеансу. Таким чином, будь-які два блоки (фіктивний або реальний) неможливо розрізнити після шифрування. Шлях від кореня до будь-якого листа L є шляхом L.

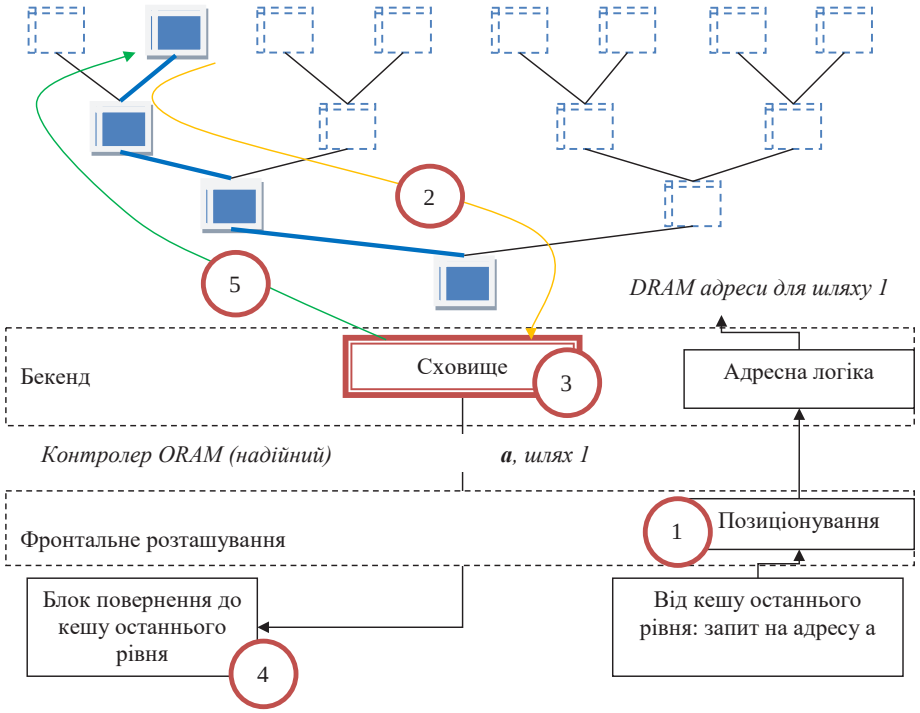

Рис. 1. Дерево (граф) ORAM

Контролер ORAM складається з карти позицій, сховища та логіки управління. Карта позицій (або блок позиціонування), коротше PosMap, є таблицею пошуку, яка пов'язує кожен блок даних із випадковим листом і n деревом ORAM. Управління та оптимізація блоку позиціонування є головною умовою, для ефективної роботи. Якщо n – максимальна кількість реальних блоків даних в ORAM, місткість блоку позиціонування становить $n^* L$ біт: одне зображення на блок. Сховище – це пам'ять, яка тимчасово зберігає невелику кількість блоків.

У будь-який час кожен блок даних у ORAM зображується у випадковому листі через блок позиціонування. Шлях ORAM підтримує такий інваріант: якщо блок зіставляється з листом L, то він повинен бути або в якомусь осередку/блоку на шляху L, або в сховищі. Блоки зберігаються в сховищі або дереві ORAM разом із поточною адресою листа та блоку.

Щоб зробити запит на блок із адресою ***a*** (коротко блок ***a***), кеш останнього рівня викликає контролер ORAM через доступ ORAM (a, op, d'), де op або зчитується, або записується, а d' – це нові дані op = записуються (кроки також показані на рисунку 1):

Крок 1. Перегляд блоку позиціонування з ***a***, встановлення позначки листа L. Випадкова генерація нового листа L' і оновлення блоку позиціонування для ***a*** з L'.

Крок 2. Читання та розшифровка всіх блоків вздовж шляху L. Додавання дійсних блоків до сховища (недійсні відкидаються). Через інваріант шляху ORAM, блок ***a*** повинен знаходитись у сховищі на цьому етапі.

Крок 3. Оновлення блоку ***a*** у сховищі, щоб мати лист L'.

Крок 4. Якщо op = прочитано, повертаємо блок ***a*** у кеш останнього рівня. Якщо op = записано, заміняємо вміст блоку ***a*** даними d'.

Крок 5. Витягуємо та зашифровуємо максимальну кількість блоків зі сховища до шляху L у дереві ORAM (щоб зберегти низьку завантаженість сховища), зберігаючи інваріант. Заповнюємо залишковий простір на шляху зашифрованими фіктивними блоками.

Безпека ORAM полягає в тому, що кожен пошук у блоці позиціонування (Крок 1) дає новий випадковий лист для доступу до дерева ORAM. Це робить послідовність шляхів дерева ORAM доступною незалежною ві д фактичного трасування адреси програми. Імовірнісне шифрування приховує, до якого блоку здійснюється доступ на шляху. Крім того, ймовірність переповнення запасів незначна, якщо $Z \geq 4$.

Кількість записів у блоці позиціонування масштабується лінійно з кількістю блоків даних в ORAM. Це призводить до значного обсягу вбудованого сховища (від сотень кілобайт до сотень мегабайт). Для вирішення цього питання у [13] запропонована схема під назвою рекурсивний ORAM, яка була вивчена при моделюванні надійних апаратних реалізацій. Основна ідея полягає в

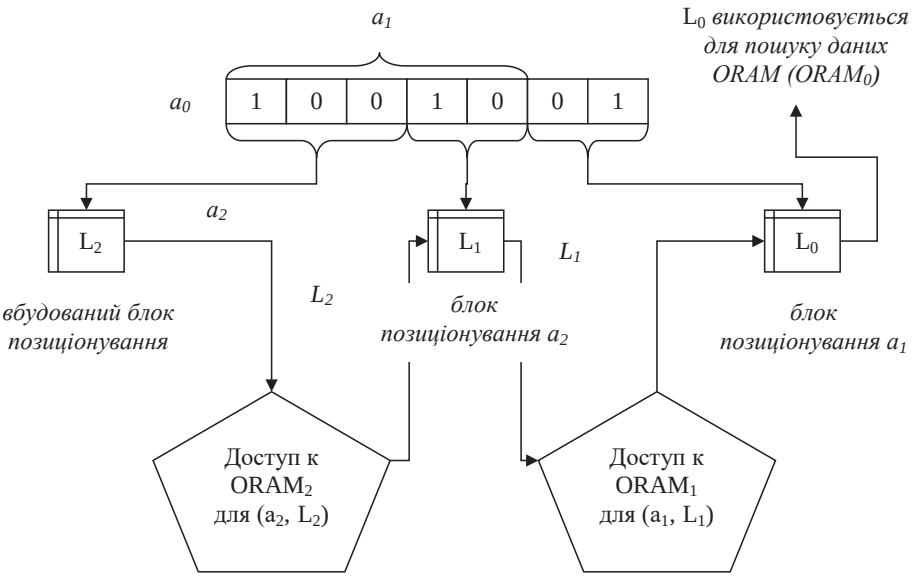

Рис. 2. Рекурсивний ORAM із розміром блоку позиціонування 4, що робить доступ до блоку даних з адресою програми $a_0 = 1001001_2$. Рекурсія зменшує ємність блоку позиціонування з 128 до 8 записів

тому, щоб зберігати блок позиціонування в окремому ORAM і зберігати новий (менший) блок позиціонування на мікросхемі. Механізм рекурсивного ORAM надзвичайно схожий на багаторівневі таблиці сторінок у традиційних системах віртуальної пам'яті.

На рисунку 2 наведено рекурсивний ORAM, який використовує дві рівні рекурсії. Тепер система містить 3 окремі дерева ORAM: дані ORAM, позначені як $ORAM_0$, і два блоки лоступу, позначені як $ORam_1$ та $ORam_2$. Блоки в ORAM схожі на таблиці сторінок, які зберігають X мітки листків, що посилаються на X блоки в $ORam_{i-1}$. Це схоже на наявність покажчиків X на таблицю сторінок наступного рівня, де X є параметром.

Припустимо, що кеш останнього рівня запитує блок a_0, що зберігається у $ORam_0$. Назва листа L_0 для блоку a_0 зберігається в блоці позиціонування $a_1 = a_0 / X$ $ORam_1$. Як і таблиця сторінок, блок a_1 зберігає листя для сусідніх блоків даних (тобто, $\{a_0, a_{0+1}, ..., a_0 + X - 1\}$ у випадку, коли a_0 кратно X). Лист L_1 для блоку a_1 зберігається в блоці $a_2 = a_0 / X^2$, що зберігається в $ORam_2$. Нарешті, лист L_2 для блоку a_2 зберігається у вбудованому блоці позиціонування. Вбудований блок позиціонування тепер подібний до таблиці кореневих сторінок.

Щоб зробити доступ до даних ORAM, спочатку потрібно переглянути вбудовані блоки позиціонування, $ORam_2$ та $ORam_1$. Таким чином, рекурсивний доступ до ORAM подібний до повної таблиці. Додаткові ORM-адреси блоку позиціонування ($ORam_3$, $ORam_4$, ..., $ORam_{H-1}$) можуть бути додані за необхідності для подальшого зменшення вбудованого блоку позиціонування. H позначає загальну кількість ORAM, включаючи дані ORAM у рекурсії та $H = log(N / p) / log X + 1$, якщо p – кількість записів у блоці позиціонування. Усі логарифми будуть з основою 2 для решти листів.

Рекурсивний ORAM збільшує загальну затримку доступу до ORAM. Не інтуїтивно, малі розміри блоків ORAM можуть сприяти більш ніж половині загальної затримки ORAM. Для ємності ORAM даних 4 ГБ 39% і 56% пропускної здатності витрачається на пошук блоку позиціонування ORAM (залежно від розміру блоку), а збільшення вбудованого блоку позиціонування лише трохи зменшує ефект.

З асимптотичної точки зору це виглядає таким чином: є єдиний ORAM шлях (без рекурсії) з розміром блоку B бітів, що передає $O(B \log N)$ бітів за доступ. У рекурсивному ORAM найкращою стратегією мінімізації пропускної здатності є встановлення константи X постійною, в результаті чого розмір блоку позиціонування ORAM становить $B_p = \Theta(\log N)$. Потім необхідна кількість ORAM становить $\Theta(\log N)$, і отримані накладні витрати на смугу пропускання дорівнюють

$$O\left(\log N + \frac{HB_p \log N}{B}\right) = O\left(\log N + \frac{\log^3 N}{B}\right)$$

Перший частина стосується даних ORAM, а друга – усіх блоків позиціонування ORAM разом узятих. Таким чином, на блок позиціонування ORAM припадає приблизно половина витрат на пропускну здатність.

Висновки. У роботі здійснено дослідження щодо організації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM". Запропоновано рекурсивний ORAM із розміром блоку позиціонування 4, що робить доступ до блоку даних з адресою програми $a_0 = 1001001_2$. Рекурсія зменшує ємність блоку позиціонування з 128 до 8 записів.

Перспективи подальших досліджень ґрунтуються на програмній реалізації захищеного інтерфейсу передачі даних на основі m-арного алгоритму "Tree-based ORAM" із застосуванням мови програмування високого рівня.

Список літератури:

1. Гулак Г.М., Бурячок В.Л., Складанний П.М., Кузьменко Л.В. Криптовірологія: загрози безпеки гарантоздатним інформаційним системам і заходи протидії шифрувальним вірусам. *Кібербезпека: освіта, наука, техніка.* 2020. № 2 (10). С. 6–28.

2. Казарин О.В. Теория и практика защиты программ. Москва, 2004. 450 с.

3. Семченко Г.П. Розробка моделі захищеного інтерфейсу передачі даних при роботі з хмарними сервісами. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2021. Вип. № 43. С. 139–144.

4. Сидоренко В.В., Буравченко К.О. Розробка способу організації захищеного каналу передачі даних у системі диспетчеризації водопостачання. *ТАРП*. 2016. № 3 (30). URL: https://cyberleninka.ru/article/n/rozrobka-sposobu-organizatsiyi-zahischenogo-kanalu-peredachi-danih-u-sistemi-dispetcherizatsiyi-vodopostachannya (дата звернення: 25.06.2021).

5. Суліма О.А. Аналіз методів оцінок рівня безпеки доступу до даних. *Моделювання та інформаційні технології.* 2017. Вип. 79. С. 35–42.

6. Islam M., Kuzu M., Kantarcioglu M. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. *Ndss*. 2012. Vol. 20. P. 12.

7. Wang X., Chan H., Shi E. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015. P. 850–861.

8. Fletcher C.W., Dijk M.V., Devadas S.A secure processor architecture for encrypted computation on untrusted programs. *Proceedings of the seventh ACM workshop on Scalable trusted computing*. 2012. P. 3–8.

9. Gentry C., Halevi S., Jutla C., Raykova M. Private database access with he-over-oram architecture. *International Conference on Applied Cryptography and Network Security.* Springer, Cham, 2015. P. 172–191.

10. Gentry C., Goldman K.A., Halevi S., Julta C., Raykova M., Wichs D. Optimizing ORAM and using it efficiently for secure computation. *International Symposium on Privacy Enhancing Technologies Symposium.* Springer, Berlin, Heidelberg, 2013. P. 1–18.

11. Stefanov E., Van Dijk M., Shi E., Fletcher C., Ren L., Yu X., Devadas S. Path ORAM: an extremely simple oblivious RAM protocol. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013. P. 299–310.

12. Shi E., Chan T.H.H., Stefanov E., Li M. Oblivious RAM with O ((logN) 3) worst-case cost. *International Conference on The Theory and Application of Cryptology and Information Security*. Springer, Berlin, Heidelberg, 2011. P. 197–214.

Kozak Ye.B. ORGANIZATION OF A SECURE DATA TRANSMISSION INTERFACE BASED ON THE M-AR ALGORITHM "TREE-BASED ORAM"

The article considers the principles of organization of a secure data transmission interface based on the m-array algorithm "Tree-based ORAM". The concept of "Tree-based ORAM" is formed, it is noted that ORAM is a conditional memory simulator – a compiler that converts algorithms so that the resulting algorithms retain the I / O behavior of the original algorithm, but the memory allocation scheme The transformed algorithm does not depend on the memory access scheme of the original algorithm. It is emphasized that as cloud computing becomes more popular, the confidentiality of users' confidential data is becoming a major issue in computing outsourcing. It has been proven that a program's memory access pattern detects a large percentage of its behavior or the encrypted data on which it computes. Security (confidentiality) definition for ORAM. It is noted that each DRAM request made during access to ORAM occurs at a certain time and, therefore, the information does not leak. It is emphasized that the hardware ORAM consists of two components: (reliable) built-in ORAM controller and (unreliable) external memory. An ORAM tree is generated and each of the implementation levels is described. It is emphasized that the ORAM controller consists of a position map, storage and control logic, with the position map (or positioning block), shorter PosMap, is a search table that associates each data block with a random sheet in the ORAM tree, and management and optimization The positioning unit is the main condition for efficient operation. It is emphasized that the security of ORAM is that each search in the positioning block gives a new random letter to access the ORAM tree, which makes the sequence of paths of the ORAM tree available independent of the actual trace of the program address. It is proved that the number of records in the positioning block scales linearly with the number of data blocks in the ORAM, which leads to a significant amount of built-in storage (from hundreds of kilobytes to hundreds of megabytes).

Key words: *protection, cryptography, interface, data transfer, algorithm, Tree-based ORAM.*