УДК 004.7

Г.Г. Киричек, канд. техн. наук, доц.,
О.Р. Рудьковський, В.С. Тимошенко, магістри
Запорізький національний технічний університет, м. Запоріжжя, Україна
kirichek@zntu.edu.ua

Система підтримки обчислень у децентралізованих мережах

В роботі наведено етапи реалізації системи з підтримки роботи сервісних додатків та вдосконалення обчислень у децентралізованих мережах різного призначення. Розглянуті питання підвищення надійності сервісів, збільшення швидкості їх роботи та переходу на новий протокол передачі інформації.

Ключові слова: обробка інформації, алгоритм, шифрування, хешування, ізольоване середовище, децентралізація, база даних.
DOI: 10.31474/1996-1588-2018-2-27-11-16

Вступ

У сучасному світі неможливо представити роботу комп'ютерів, ноутбуків, планшетів та смартфонів без підключення до мережі. Разом з цим збільшується кількість трафіку і даних, які треба зберігати чи обробляти. З розвитком технологій все гостріше постають питання подальшого розвитку мереж та забезпечення їх безпеки. Тому питання доступності інформації, підвищення швидкодії її отримання та використання, підвищення ефективності та якості проведення обчислень є дуже актуальними на даний час.

Вирішенням усіх вище зазначених проблем є децентралізація мереж та додатків. Тому, на даний час, проводиться багато досліджень у цьому напрямку та вже є практичні впровадження [1].

Постановка задачі

На даний час майже всі додатки працюють як клієнт-серверні сервіси. Вони дозволяють обмінюватись повідомленнями, передавати файли, отримувати новини та публікувати їх. Сервіси взаємодіють як з користувачами так і з іншими сервісами, що часто використовують зловмисники при викраденні публічної інформації, даних банківських карток, приватних даних, файлів та ін., тому захист даних є одним з пріоритетних питань кампаній при наданні сервісів. Ще одним з питань є стрімкий розвиток інтернету речей. За останніми прогнозами у наступні роки кількість пристроїв, підключених до мережі Інтернет, збільшиться у декілька разів тому питання їх персональної ідентифікації є важливим. Згідно з темпами росту впровадження інтернету речей, діапазон адрес IPv6 може скінчитись набагато раніше [2]. Тому завдання, які треба вирішити для надійної роботи мережі Інтернет є наступними: розробка нових протоколів передачі інформації, замість IPv6 та перехід на децентралізовану модель роботи сервісів [2-3]. Заміна сервісів їх децентралізованими аналогами дозволить вирішити наступні питання: підвищити надійність самих сервісів; збільшити швидкість роботи сервісів та здійснити перехід на новий протокол передачі інформації.

Метою роботи є реалізація системи підтримки обчислень у децентралізованих мережах з підтримкою роботи сервісних додатків. Об'єктом дослідження є процес реалізації системи підтримки обчислень у децентралізованих мережах. Предметом – моделі, методи та інструментальні засоби підтримки обчислень у децентралізованих мережах. Основні завдання роботи: побудова моделей та алгоритмів роботи окремих модулів системи; розробка децентралізованої мережі, що забезпечує обмін даними, децентралізоване зберігання інформації та запуск сервісів у мережі; розробка протоколу передачі та обробки інформації для децентралізованих мереж різного призначення; реалізація системи з підтримки роботи сервісів та проведення її тестування на працездатність [2].

Програмні засоби та рішення

Алгоритми хешування є поширеними та мають декілька реалізацій (MD5, SHA, RIPEMD та ін). Основними характеристиками алгоритмів хешування є: розрядність; складність обчислення; криптографічна стійкість; швидкість обчислення та кількість колізій [4]. Наразі алгоритм MD5 не є криптостійким. SHA об'єднує різні алгоритми, такі як SHA-1, SHA-2, SHA-3, є дуже популярним та широко використовуються у сучасному світі. Алгоритми SHA характеризуються тим, що внесення навіть невеликих змін у вхідному рядку значно сильно змінюють хеш. Оглянувши алгоритми хешування обрано сімейство SHA-2 оскільки вони є захищеними, достатньо дослідженими, побудовані на основі структури Меркла-Демґарда і мають велику швидкість роботи [5].

На даний момент існує багато алгоритмів шифрування, таких як AES, RSA, 3DES, ГОСТ 28147-89 та інші. Різні децентралізовані мережі використовують різні алгоритми шифрування. Часто можна зустріти використання двох або більшу кількість алгоритмів для передачі та зберігання інформації. На даний момент безпечними та популярними алгоритмами шифрування є симет-

ричні - AES, 3DES, Twofish та асиметричні - RSA, Діффі-Гелльмана, DSA, ECC та ECDH.

В роботі використовуються два алгоритми шифрування – ECDH для генерації спільного ключа та AES для подальшого шифрування даних. Таке поєднання дає велику стійкість системи, оскільки: ключі шифрування не передаються через незахищені лінії зв'язку; алгоритм AES є найбільш захищений на даний момент; вхідні дані для генерації спільного ключа є стійкими та обчислюються дуже швидко; існує можливість шифрування великих обсягів даних [6-7].

Існує два протоколи для передачі інформації – TCP та UDP. Обидва протоколи працюють на транспортному рівні моделі OSI. Кожен протокол має свої особливості та переваги. Інколи додатки підтримують роботу використовуючи обидва протоколи. Перевагами TCP є: надійна доставка сегментів; упорядкування сегментів при отриманні; робота з сесіями; контроль за швидкістю передачі. Перевагою UDP є велика швидкість передачі відносно TCP. Протокол UDP використовується тоді, коли необхідно швидко отримувати дані. При використанні UDP логічний зв'язок не створюється, отже будь-хто може надсилати дані додатку. Оглянувши обидва протоколи вирішено використовувати протокол TCP через його надійність, що дуже важливо у децентралізованих мережах.

Реалізація системи

Система підтримки обчислень є децентралізованою та оверлейною. Вона має високі вимоги до безпеки даних, які зберігаються та передаються у рамках системи. Оскільки система є оверлейною, то існує потреба у власній ідентифікації клієнтів, тому в якості адреси можна використовувати будь-яку послідовність символів будь-якої довжини. Це дозволяє уникнути закінчення діапазону доступних адрес, за рахунок додавання додаткових символів, не змінюючи всю систему.

Для досягнення необхідного рівня безпеки використовуються надійні алгоритми шифрування даних. Оскільки обрано алгоритм шифрування AES, який є симетричним алгоритмом, то існує слабке місце при передачі ключів шифрування. Для вирішення цього питання використовуємо алгоритм, за допомогою якого генерується спільний ключ без передачі його через мережу [7]. За допомогою алгоритму Elliptic Curve клієнти генерують власні закритий та відкритий ключі. Приватний ключ зберігається клієнтом та ніколи не передається мережею, а відкритий - є адресою клієнта у децентралізованій мережі. Кожен клієнт знає публічний ключ кожного клієнта, з ким встановлено зв'язок напряму. За допомогою алгоритму ECDH клієнти, маючи свої публічні та приватні ключі та публічний ключ другого клієнта, можуть згенерувати спільний ключ, який є однаковим при генерації на іншій стороні, при використанні іншої пари ключів та іншого публічного ключа. Цей підхід усуває необхідність передавати ключ через мережу та дозволяє поєднати плюси симетричних та асиметричних алгоритмів шифрування одночасно прибравши їх недоліки. При цьому використання публічного ключа в якості адреси клієнта, гарантує неможливість зіставлення віртуальної адреси з фізичним розташуванням клієнта або у мережі Інтернет. Даний алгоритм генерує відкритий ключ великого розміру, що достатньо на довгий час, а при необхідності, значного розширення діапазону, можна досягти додаванням лише одного символу до адреси.

При передачі інформації клієнти використовують різні операційні системи (взаємодія смартфону та комп'ютера) та формати даних для досягнення одного результату. Для вирішення цих питань одночасно використовують два методи: серіалізації та інкапсуляції. При серіалізації структури даних перетворюються в один спільний формат – текстовий. Отримавши текст клієнт перетворює його у необхідну структуру. Для серіалізації використовуємо формат даних JSON. Він є досить простим для використання і має підтримку у сучасних мовах програмування. Для серіалізації складних структур даних використовуємо інкапсуляцію: самі дані вкладаються у об'єкт, який додатково має опис цих даних. За допомогою цього опису клієнт однозначно визначить тип даних та коректно їх опрацює. Оскільки різні додатки реалізовані різними людьми, а отримувача вказує сам відправник, то, для усунення проблем із використанням єдиного опису для різних сервісів, програміст сам повинен подбати про унікальність опису. Наприклад, клієнт надіслає дані, які зберігаються в окремому класі та наведені у лістингу 1. Клієнт попередньо створює асоціацію цього класу з його кодом, наприклад, "weather.request". В прикладі, поле country містить значення "UA", а city – "zaporizhzhia". У серіалізованому вигляді цей клас описано у лістингу 2.

Лістинг 1 – Приклад класу

```
from dc.package import Package
class Weather(Package):
    def __init__(self):
        self.country = ''
        self.city = ''
    def fields(self):
        return ['country', 'city']
```

Лістинг 2 – Клас у серіалізованому вигляді

```
{"type":" weather.request",
 "data": {"country":"UA",
          "city":"zaporizhzhia"}}
```

Отримавши дані у такій формі, клієнт легко відтворює об'єкт потрібного класу. Дані інкапсу-

люються у окремий об'єкт, що містить додаткову інформацію, яка необхідна клієнту (специфічні дані з якими працює окрема реалізація клієнта). Фінальний вид пакету у скороченому вигляді наведено у лістингу 3.

Лістинг 3 – Фінальний вид пакета

```
{"sender":"...",
"nodes":["...","..."],
"length":"...",
"data":{"data":{…},
        "id":"...",
        "number":1,
        ”total”:1,
        ”crc”:…}}
```

Пакет має багато додаткової інформації, тому передача малих об'ємів даних відбувається з меншою швидкістю. Дані пакетів шифруються і доступні лише отримувачу та відправнику, а інформація щодо маршруту є загальнодоступною. Тому будь-хто може отримати пакет та надіслати його отримувачу не маючи доступу до самих даних. Модель системи наведена на рисунку 1.

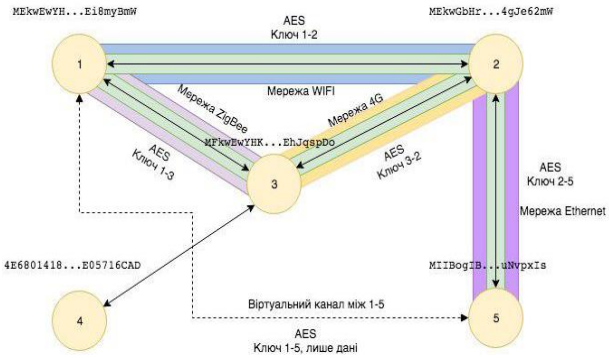

Рисунок 1 – Модель системи

Важлива частина системи це команди. Командою є звичайний клас, що передається у рамках системи і додатково має необхідні методи, які реалізують правильну поведінку. Робота системи реалізована на командах: маршрутизація, запуск та публікація додатків, отримання даних з пам'яті та ін. Це робить тестування та розширення функціоналу простішим і значно зменшує кількість помилок у роботі команд та системи в цілому [2].

Основне призначення команд – виконання певних дій для клієнтів. А важлива відмінність від додатків є те, що вони не працюють децентралізовано. Для запуску команди клієнт повинен мати адресу іншого клієнта, який виконає цю команду.

Надіслати команду може той, хто знає необхідну інформацію щодо команди (формат даних та її код). Виконати команду може лише той клієнт, який має зареєстровану команду з цим кодом. У разі отримання команди з однаковим кодом, але призначеної іншому клієнту, поведінка невідома, тому що клієнт не визначить правильно надіслана команда чи ні. За допомогою команд реалізується ланцюг маршрутизації, публікуються додатки та отримується інформація з розподіленої пам'яті.

Для коректної доставки даних, клієнт перед передачею повинен власноруч побудувати необхідний ланцюг. Для цього він виконує наступні дії: визначає клієнта з якого почнеться пошук; надіслає запит щодо наявності отримувача у списку підключених клієнтів; опрацьовує відповідь від клієнта. Якщо отримувача знайдено, процес переривається, інакше відправник отримує список усіх підключених клієнтів та починає пошук знов використовуючи іншого клієнта як кінцевого.

Використання цього алгоритму дозволяє контролювати ланцюг клієнтів. Клієнт може вилучати проміжні вузли для підвищення надійності системи. Реалізований ланцюг діє певний час після чого його потрібно перебудовувати. Тому ланцюг з часом змінюється і це робить його більш надійним через фізичну неможливість перехоплення даних на певних ділянках.

Важливою частиною системи є розподілена пам'ять. У ній зберігається вся інформація з якою оперує система, дані користувачів та власне додатки. Дані у пам'яті зберігаються «як є», тому про шифрування та дешифрування цих даних дбає команда або додаток. Отримати дані може будь-хто, але кожен запис у пам'яті має унікальний ідентифікатор. Тому для отримання даних необхідно знати цей ідентифікатор. Іншою особливістю даних у пам'яті є їх незмінність. Після внесення даних до пам'яті їх редагування становиться неможливим. Розподілена пам'ять є розподіленою хеш-таблицею. Фізично даними є окремий файл на носії, тому максимальний розмір розподіленої пам'яті залежить від кількості клієнтів та від того, скільки кожний клієнт дозволив займати місця на носії для роботи пам'яті. Мінімального та максимального розміру даних, які можуть бути записані у пам'ять не існує.

Додатком у системі є програма написана з використанням будь-якої мови програмування, що є унікальним при створенні децентралізованих додатків. Робота програм у децентралізованій системі та на локальному комп'ютері нічим не відрізняється оскільки вони ідентичні. Додатки взаємодіють з системою так: надсилають відповіді та нові дані клієнтам; викликають команди; запускають нові додатки; зчитують та записують дані з розподіленої пам'яті. Додатково вони можуть використовувати графічні адаптери для складних обчислень, робити запити до мережі Інтернет або працювати з файловою системою.

Щоб гарантувати безпеку даних додатки запускаються у спеціальному ізольованому середовищі. У якості середовища використовуємо Docker, досягаючи виконання додатків розроблених на будь-якій мові програмування та різних

операційних систем. Використання ізольованого середовища дозволяє відокремити додаток від фізичного комп'ютера під час виконання, захистивши дані та файли користувача. Такий підхід дозволяє запускати паралельно декілька додатків, забезпечити їх ізольовану роботу один від одного та від клієнтської операційної системи. Docker також дозволяє досягти максимальної переносимості додатків та використовує відкриті технології для легкого тестування додатків в ізольованому середовищі перед публікацією у системі.

Реалізація системи виконана у вигляді набору різних модулів, які працюючи разом виконують поставлені задачі. Використання модульної структури надає можливість змінювати частини системи не втрачаючи працездатність усієї системи в цілому у випадках: перенесення додатку на іншу операційну систему, або не підтримки вже існуючого коду цілком або частково [2].

Система має наступні модулі: мережі; хешування; шифрування; серіалізації; повідомлень; команд; додатків; пам'яті; розширень та модуль маршрутизації. Деякі модулі містять мало програмного коду тому їх можна замінити на безпосередні виклики необхідних функцій у тілі системи, але сам процес перенесення коду на інші системи стане більш складним і потребує набагато більше часу. До цих модулів відносимо модулі шифрування, хешування, серіалізації та мережі. Модуль розширень теж можна віднести до цього набору модулів, але він не інтегрується у інші модулі системи і повинен працювати окремо. Інші модулі набагато складніші і виконують основну роботу системи. Це модулі повідомлень, команд, додатків, пам'яті, розширень та маршрутизації. Будь-який модуль системи використовується у іншому модулі чи сам користується іншим модулем для отримання розрахунків або даних (рис. 2).

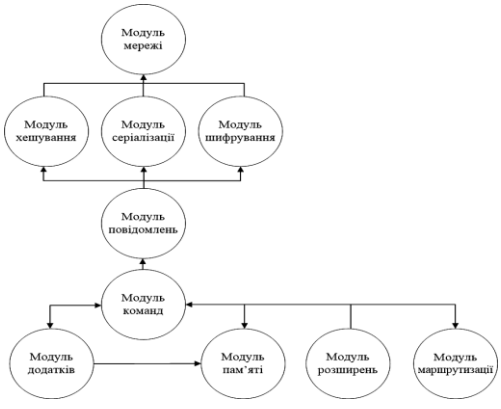

Рисунок 2 – Взаємозв'язок модулів системи

Розбиття системи на окремі модулі дозволяє легко тестувати поведінку модулів, так як кожний окремий є відносно невеликим у порівнянні з усією системою, час його розробки та складність мінімальні і це значно підвищує надійність системи [2]. Система може працювати у двох режимах: транзитному та клієнтському. У транзитному режимі додаток лише передає данні від відправника до отримувача, система не змінює та не отримує самі дані і працює лише з заголовком. У клієнтському - додаток отримує дані від інших користувачів та ініціалізує передачу. Також він одночасно працює і у транзитному режимі для передачі повідомлень, що не призначені поточному клієнту.

Для правильної роботи системи окремі модулі повинні працювати лише з інтерфейсом модуля. Кожен модуль складається з мінімального набору класів. Для коректної роботи модулів програміст повинен їх правильно реалізувати. Робота модулів тісно пов'язана з впровадженням залежності. Наприклад маємо інтерфейс модуля хешування (лістинг 4) та реалізацію цього інтерфейсу (лістинг 5).

Лістинг 4 – Інтерфейс модуля хешування

```
from abc import abstractmethod
class Hashing():
    @abstractmethod
    def encode(self, object):
        pass
```

Лістинг 5 – Реалізація модуля хешування

```
from dc.module.hashing import Hashing
from Crypto.Hash import SHA256
class HashingImplementation(Hashing):
    def encode(self, object):
        try:
            h = SHA256.new(data=object)
        except ValueError as e:
            h                          =
SHA256.new(data=bytes(object, 'utf-8'))
        return h.hexdigest()
```

Реалізацію використання та реєстрації залежності наведено у лістингу 6.

Лістинг 6 – Реєстрація залежності та приклад використання

```
from dc.injection import inject
from dc.module.hashing import Hashing
from   module.hashing   import   Hash-
ingImplementation
inject.get_instance().bind(Hashing,
HashingImplementation)
# використання:
in-
ject.get_instance().get(Hashing).encode
(…)
```

Реалізація є повністю відокремленою від інтерфейсу і при виникненні помилки у модулі її можна виправити без змін інших модулів, не впливаючи на алгоритм роботи системи в цілому.

Етапи роботи системи

При підключенні до системи клієнт повинен мати IP-адресу вже підключеного до системи клієнта. Якщо підключення відбувається до клієнта, який не має підключення до інших клієнтів, створюється нова ізольована мережа із власною пам'яттю, додатками та користувачами. Після відкриття сокету та встановленням зв'язку, кожен клієнт надсилає іншому свій публічний ключ. Після отримання ключа, який є адресою клієнта у мережі, він генерує спільний ключ.

При відправці даних, клієнт збирає їх у певний клас, який представляє ці дані у віртуальному вигляді та отримує адресу отримувача. Після заповнення класу, відправник ініціює підготовку до відправки. У цей час дані перетворюються у пакет – серіалізуються та за необхідністю сегментуються. Після сегментації дані передаються модулю маршрутизації, який визначає ланцюг вузлів, через які прокладено маршрут їх доставки. Якщо ланцюг вже побудовано і він є актуальним, кожен сегмент окремо інкапсулюється у транзитний пакет і передається модулю мережі, який і виконує передачу даних. Якщо ланцюг не знайдено, клієнт зі списку вже підключених до нього клієнтів, обирає іншого клієнта, враховуючи необхідні критерії та питає про зв'язок з отримувачем. В разі існування зв'язку з отримувачем, відправник додає цього клієнта у список вузлів та вирішує, чи відповідає отриманий ланцюг певним критеріям. За необхідністю дані шифруються. Сегмент шифрується обов'язково спільним ключем з отримувачем. Під час отримання пакета транзитним клієнтом, він зчитує ланцюг вузлів, видаляє себе з ланцюга та надсилає пакет далі згідно маршруту. Отримавши пакет, отримувач може дешифрувати його згенерувавши спільний ключ. Після отримання всіх сегментів, вони об'єднуються у один пакет і десеріалізуються. Далі надсилаються команді чи додатку, або виконуються, якщо дані є командою.

У разі необхідності запуску сервісу клієнт, за допомогою команди, надсилає запит на виконання у систему, найближчу до клієнта з яким він має пряме з'єднання. Ця система є «роутером» і працює у ролі проміжного елементу між клієнтом, який ініціював запуск та працюючою програмою. Саме через цей вузол іде обмін інформацією між клієнтом та додатком. Проміжний клієнт копіює додаток з пам'яті та надсилає широкомовно пакет із запитом, що містить інформацію щодо додатка. Отримавши запит, кожен клієнт приймає рішення зможе він виконати додаток чи ні. Якщо зможе, то клієнт надсилає відповідь «роутеру».

Після отримання відповіді, додаток пересилається цьому клієнтові, а інші отримають відмову. Клієнт, який виконує додаток, запускає ізольоване середовище та копіює у нього додаток. Після цього додаток запускається, надає необхідний сервіс та завершує виконання, про що інформує «роутер». Після отримання повідомлення про завершення роботи додатка, «роутер» повідомляє клієнта і видаляється. Під час роботи додатка він може надсилати відповіді на запити клієнта про виконання додатка до «роутера» та від «роутера» до клієнта. Ланцюг у такому випадку не будується, а процес передачі такий як і під час передачі між різними клієнтами.

Результати тестування у ізольованому та звичайному середовищі зображено на рисунку 3, де: пунктирна лінія – ізольоване, безперервна – звичайне середовище.

Рисунок 3 – Результати тестування

Як видно з результатів, час виконання одного й того ж додатка відрізняється [2]. Більшу частину випробувань додаток у ізольованому середовищі працює швидше.

Висновки

В роботі наведено етапи реалізації системи підтримки обчислень в децентралізованих мережах різного призначення. Пропонується використання децентралізованої системи для запуску сервісів на ізольованій операційній системі, що дає можливість обробляти дані швидше і без необхідності мати власні обчислювальні потужності. В подальшому планується розширити роботу системи шляхом інтеграції у мобільні пристрої та інші існуючі інтернет сервіси.

Список літератури

1. IETF Tools (1999). Mobile Ad hoc Networking (MANET): [Електронний ресурс] . – Режим доступу: http://tools.ietf.org/html/rfc2501 (дата звернення: 10.01.2019).

2. Киричек Г.Г. Децентралізована система підтримки обчислень / Г.Г. Киричек, О.Р. Рудьковський, В.С. Тимошенко // Вчені записки ТНУ ім. В.І. Вернадського. Серія: Технічні науки. - 2018. - Том 29 (68) № 6 - С. 161-166.

3. A Next-Generation Smart Contract and Decentralized Application Platform [Електронний ресурс]. – Режим доступу: https://github.com/ethereum/wiki/wiki/White-Paper (дата звернення: 10.01.2019).

4. Ахметов Б.С. Прикладная криптология: методы шифрования / Б.С. Ахметов, А.Г. Корченко, В.П. Сиденко. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – 496 с.

5. Klima V. Tunnels in Hash Functions: MD5 Collisions Within a Minute [Електронний ресурс] / V.Klima. – 2006. – Режим доступу: https://eprint.iacr.org/2006/105.pdf (дата звернення: 10.01.2019).

6. Eastlake D. US Secure Hash Algorithms [Електронний ресурс] / D. Eastlake, T. Hansen. – 2006. – Режим доступу: https://tools.ietf.org/html/rfc4634 (дата звернення: 10.01.2019).

7. Announcing the advanced encryption standard (AES) [Електронний ресурс]. – 2001. – Режим доступу: http://www.impic.org/papers/Aes-192-256.pdf (дата звернення: 10.01.2019).

References

1. IETF Tools (1999). Mobile Ad hoc Networking (MANET):. URL: http://tools.ietf.org/html/rfc2501 (Accessed 10.01.2019).

2. Kirichek G.G., Rudkovsky O.R., Timoshenko V.S. Decentralized computing support system. 2018. Scientific notes of TNU named V.I. Vernadsky Series: Engineering. Vol. 29 (68). No. 6. P. 161-166.

3. A Next-Generation Smart Contract and Decentralized Application Platform. URL: https://github.com/ethereum/wiki/wiki/White-Paper (Accessed 10.01.2019).

4. Akhmetov B.S., Korchenko A.G., Sidenko V.P. Applied Cryptology: Encryption Methods. Almaty: KazNRTU named K.I. Satpayev, 2015. 496 p.

5. Klima V. Tunnels in Hash Functions: MD5 Collisions Within a Minute. 2006. URL: https://eprint.iacr.org/2006/105.pdf (Accessed 10.01.2019).

6. Eastlake D., Hansen T. US Secure Hash Algorithms. 2006. URL: https://tools.ietf.org/html/rfc4634 (Accessed 10.01.2019).

7. Announcing the advanced encryption standard(AES). 2001. URL: http://www.impic.org/papers/Aes-192-256.pdf (Accessed 10.01.2019).

Надійшла до редакції 10.12.2018

Г.Г. КИРИЧЕК, А.Р. РУДЬКОВСКИЙ, В.С. ТИМОШЕНКО
Запорожский национальный технический университет (Украина)

СИСТЕМА ПОДДЕРЖКИ ВЫЧИСЛЕНИЙ В ДЕЦЕНТРАЛИЗОВАННЫХ СЕТЯХ

В работе приведены этапы реализации системы по поддержке работы сервисных приложений и совершенствования вычислений в децентрализованных сетях различного назначения. Рассмотрены вопросы повышения надежности сервисов, увеличение скорости их работы и перехода на новый протокол передачи информации.

Ключевые слова: обработка информации, алгоритм, шифрование, хеширование, изолированная среда, децентрализация, база данных.

G.G. KIRICHEK, O.R. RUDKOVSKYI, V.S. TYMOSHENKO
Zaporizhzhia National Technical University (Ukraine)

SUPPORT SYSTEMS OF CALCULATIONS IN DECENTRALIZED NETWORKS

In this work we propose to use decentralized network for executing desktop applications which were developed using native program languages and which were developed to run on complete isolated operating system. It allows to process data faster and without having to have own computing power. Objectives of work are: development of new protocols for transfer information instead of IPv6; transition to a decentralized model of service work. Purpose of work is conducting research and implementing a system of computing support in decentralized networks, by developing a decentralized network protocol supporting the work of service applications. The proposed method for running applications in decentralized networks is been based on launching isolated environment, which allow to access hardware and use any technologies in programming in same time. Decentralized network structure provides guaranties in security, accessibility and scalability. The authors obtained a mechanism for processing data securely, for the purpose of its further usage in Internet of Things where independent devices can works together without spending a lot of time and efforts from developers; in medicine to make solutions automatically and as "virtual doctor". System security and calculation speed is growing with the number of nodes, but time to launch is decreasing. System work algorithm has been presented.

Keywords: data processing, algorithm, encryption, hashing, isolated environment, decentralization, database.