TELKOMNIKA, Vol.17, No.3, June 2019, pp.1360~1366
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i3.11758

KANSA: high interoperability e-KTP decentralised database network using distributed hash table

Rolly Maulana Awangga*, Nisa Hanum Harani, Muhammad Yusril Helmi Setyawan
Applied Bachelor of Informatics Engineering Politeknik Pos Indonesia,
Sariasih St. No.54 Bandung 40151, 022-2009562
*Corresponding author, e-mail: awangga@poltekpos.ac.id

Abstract

e-KTP is an Indonesian Identity Card based on Near Field Communicator technology. This technology was embedded in every e-KTP card for every Indonesian citizen. Until this research, e-KTP technology never to be utilized by any stack-holder neither government agencies nor nongovernment organization or company. e-KTP Technology inside the card never been used and go with conventional with manual copy it with photocopy machine or take a photograph with it. This research was proposing an open standard to utilized e-KTP Technology. The open standard will bring e-KTP technology used as is and used broadly in many government agencies or much commercial company. This research was proposing decentralized network model especially for storing e-KTP data without breaking privacy law. Besides providing high specs of the server, a decentralized model can reduce the cost of server infrastructure. The model was proposing using Distributed Hast Table which was used for peer-to-peer networks. The decentralized model promised high availability and the more secure way to save and access the data. The result of this model can be implemented in many network topology or infrastructure also applicable to implement on Small Medium Enterprise Company.

Keywords: *decentralised, DHT, e-KTP, interoperability, P2P*

1. Introduction

Today Indonesian Identity Card (e-KTP) have NFC Technology inside it. It is a new technology safer than standard RFID [1] which has much evaluation [2-8]. The NFC Technology in e-KTP never been used until this time. Many government organisations have been used with conventional way into e-KTP to process many citizen service. There is also no regulation and socialisation to use e-KTP technology to integrate into existing government system. Beside the cost of e-KTP reader distribution and manufacturer, there was also infrastructure to built with big amount of money. This research focusing how to build decentralised network system for using e-KTP reader [9]. The network we called KANSA which is provide lower price and easy to develop and integrate with existing system.

KANSA use Distributed Hash Table (DHT) concept to built decentralised network [10]. DHT was used as base of Peer-to-per network [11] which we know as new concept of torrent [12], magnet URI [13]. With the rapid development of the Internet, it needs more computing power resource [14]. For example, it uses peer to peer network applications where storage capacity and network bandwidth can share between users. DHT (Distribution Hash Table) is a system of efficient mechanisms for data placement and search the resources in the P2P network. The DHT-based system can apply in a backup file, which allows users to store their data files in a computer connected to the Internet and store all the time. In a DHT-based file backup system can enable users to store their data files on computers connected internet and keep it all the time [15]. DHT is providing information search services for applications P2P (Peer To Peer) using a pair (key, value). Here the ‘key’ to act as input and returns ‘value’ as output [16]. In the Peer to Peer (P2P), a single peer to peer further connected in a distributed storage system. A group of solitary resolution chart illustrates the distributed hash table of assets to the node in the P2P network where assets can be collected in the network and hence back. In the DHT offered the position of nodes and services that can extend, but not just in addition to any maneuver lookup (an activity that is used to place the asset in the system) [17]. Distributed hash table (DHT) provides an efficient recovery method known as Checkpoint recovery and topology built using this table in a decentralized approach [18].

Received November 10, 2018; Revised February 1, 2019; Accepted March 2, 2019

AES-CBC is one method of encryption, known as the Rijndael algorithm. The AES-CBC method involves Initialization Vector (IV) for the XOR operation and requires 128 bits of data in single encryption. In the process the encryption and description, a plaintext must have a length of at least 128 bits of data XOR and IV to be operated on text first before a key encrypts it. The encryption process creates ciphertext. Then it will be decrypted by the first key. After that, the process will produce the plaintext decryption [19]. Before encrypting blocks, XOR with the ciphertext from the previous ciphertext block. Analysis of AES-CBC method aims to find errors during the encryption process. The simulation was performed to analyze the reduction of the chip size [20]. Mini-AES is a simple version of the application of the algorithm Rijndael (AES). Mini-AES AES has all the parameters are reduced significantly but still retains the original. Despite having all the parameters of AES, Mini-AES cannot represent AES security [21].

PBKDF2 a key derivation function based on a password [22]. In implementation has a high impact on the security of a particular application. To improve efficiency, such as the CPU also on other hardware that has a special purpose as a modern GPU, Application-Specific Integrated Circuit (ASIC) or Field-Programmable Gate can be used. PBKDF2 can efficiently implement on the FPGA [23]. Password-based authentication mode is the authentication mechanism has been widely used, after entering the user password, and then will compare the encryption calculation with a value of checks to authenticate user identity, application of the Internet is generally used by the encryption key password to protect the key communications sessions. The public key encryption standard 5 PKC (Public Key Cryptography Standard 5) is a secret key standard developed by the company RSA. It defines two single passwords is PBKDF1 Key-Based Lock function, and the other is PBKDF2 [24].PBKDF2 has implemented in Openoffice [25], developed beside SZRP protocol [26], compares with Bcrypt and Scrypt algorithm [27] and the weaknesses show in [28]. e-KTP network infrastructure is a new problem and unresolved until now. This research proposing KANSA as e-KTP infrastructure as open standard. This standard can used in many government or non government organisation without breaking any law.

2. Research Method

The methods used in this study are DHT, AES-CBC, and PBKDF2. All three methods will combine and use on this project. The general methodof Distributed Hash Table (DHT) is structured overlay network is the average latency per successful search and percentage of successful searches. For searches based on DHT service with the top search for a failed search, both are average latency per successful search as well as the percentage of success search is the key to evaluating performance. Distributed Hash Tables (DHT) has a primary interface to meet the search process of structured P2P systems [16]. In DHT offer the expandable position of nodes and services, but do not guard beside malevolent node maneuvering lookups (the activity used to situate the assets in the system). Malevolent nodes on a lookup pathway can merely undermine query. Systems like Halo (based on the chord) decrease such assaults by using superïňĆuous lookup queries. To supply better declaration, Reputation for Directory services is a foundation for boosting lookups in superfluous DHT by trailing how well further nodes service lookup desires. ReDS procedure can to practically any superfluous DHT containing halo. Moreover, also recognize mutual recognition and elimination of lousy lookup lanes in a way that does not bet on the dividing of reputation counts and dividing is susceptible to attacks that create it unïňĄt for the majority utilization of ReDS [17]. This goal is an achieved by framing WSN with distributed hash table (DHT) which provides an efficient recovery method is known as Checkpoint recovery and the topology is constructed using this table in a decentralized approach [18].

AES-CBC is symmetric key algorithm, one method of encryption known as the Rijndael algorithm [19]. AES working in which both the sender and the receiver use a single key for encryption and decryption. AES can define data block and key lengths. Data block length to 128 bits, and the key lengths to 128,192, or 256 bits. AES method is like iterative algorithm, and every iteration is called a round. The round can define as Nr, and the total number of rounds is 10, 12, or 14 when the key length is 128, 192, or 256 bits, respectively. There are four transformations each round in AES: Sub Bytes, ShiftRows, Mix Columns, and AddRoundKey [20]. The steps to perform a encryption operation is first the plaintext must have length of multiple data 128 bits XOR with IV. And then it will be operated on the first plaintext before encrypted by the key. After that procees encryption creates a chipertext. After that a chipertext will descrypted by the key first. After description process, the XOR and IV are perfomed to result produce plaintext [19]. The process of AES-CBC shows in Figure 1.

PBKDF2 a key derivation function based on a password. In algorthm PBKDF2 there is DK=PBKDF2 where, the Pass is the password, Salt is a salt value, c is the iteration count, dkLen is the length of DK. The Salt is a fixed random value ensuring a huge variety of derived keys for each password such that an attacker cannot use a brute-force attacking. And the c, iteration count, specifies the cycles of the pseudorandom function PRF against brute-force attack. PRF is the underlying pseudorandom function [22]. Figure 2 shows illustration of pbdkf2 algorithm.

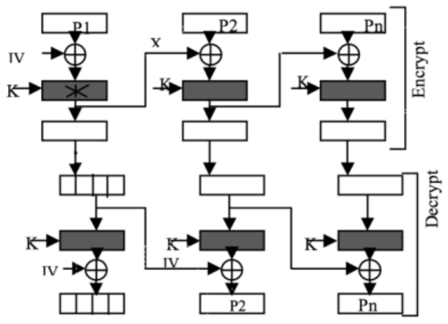


Figure 1. Encryption in AES-CBC [7]

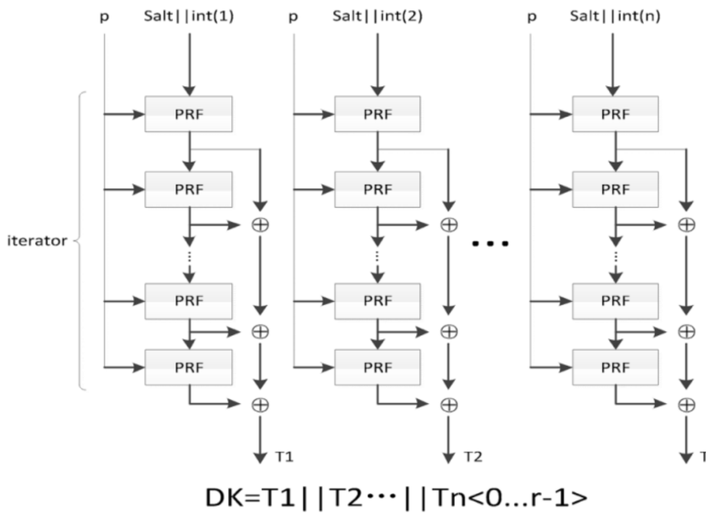
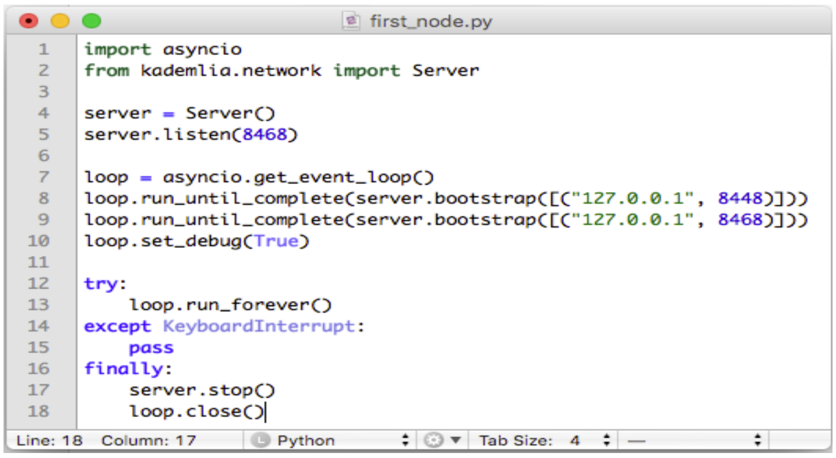


Figure 2. PBKDF2 scheme [9]

3. Results and Analysis

Every Node in the network acts as database server. The node can be a computer or mini PC or other operating system which can run the source code of KANSA node. The code shows in Figure 3. The node provides NoSQL database which is containing key and value. The key is e-KTP NFC ID encrypted by PDKF2 and AES-CBC. The Value may consist Name or another field which is needed by stack-holder. The code uses asyncio and kademlia library to run DHT network. In this research simulates two nodes in one computer. Although running simulation on same computer, the port must different to avoid collision. The host is 127.0.0.1 and the port 8448 and 8468.

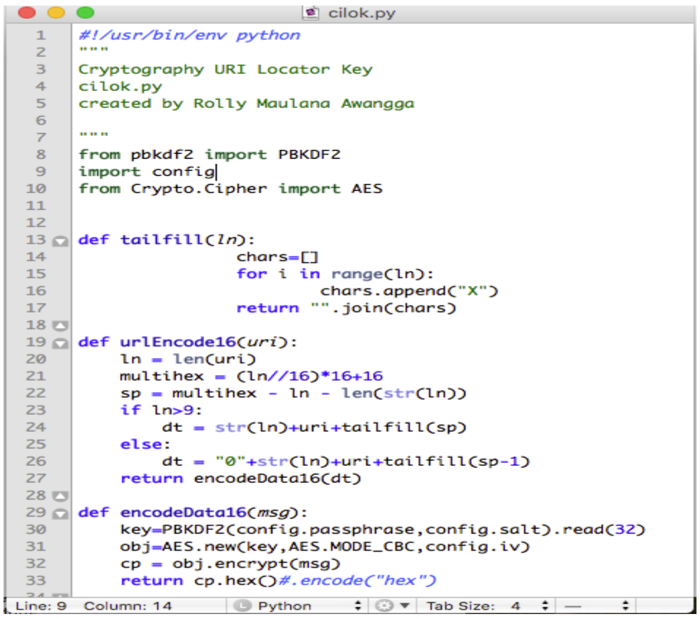


```python
import asyncio
from kademlia.network import Server

server = Server()
server.listen(8468)

loop = asyncio.get_event_loop()
loop.run_until_complete(server.bootstrap([("127.0.0.1", 8448)]))
loop.run_until_complete(server.bootstrap([("127.0.0.1", 8468)]))
loop.set_debug(True)

try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    server.stop()
    loop.close()
```

Figure 3. The source code of every node in network using python

3.1. PBKDF2 and AES-CBC

The key stored in DHT network was encrypted using collaboration of PDKF2 and AES-CBC. The algorithm shows in Figure 4. The encryption algorithm uses pbkdf2 and AES library. The pbkdf2 uses passphrase, salt and iv with 16 character for each in length. In Figure 4 shows three function tailfill, urlEncode16 and encodeData 16. The tailfill function act to fill up the string uri until 16 character. The encodeData16 function is to generate cipher text using PBKDF2 and AES CBC from string msg. The urlEncode16 uses tailfill dan encodeData16 Function to generate cipher text.


cilok.py

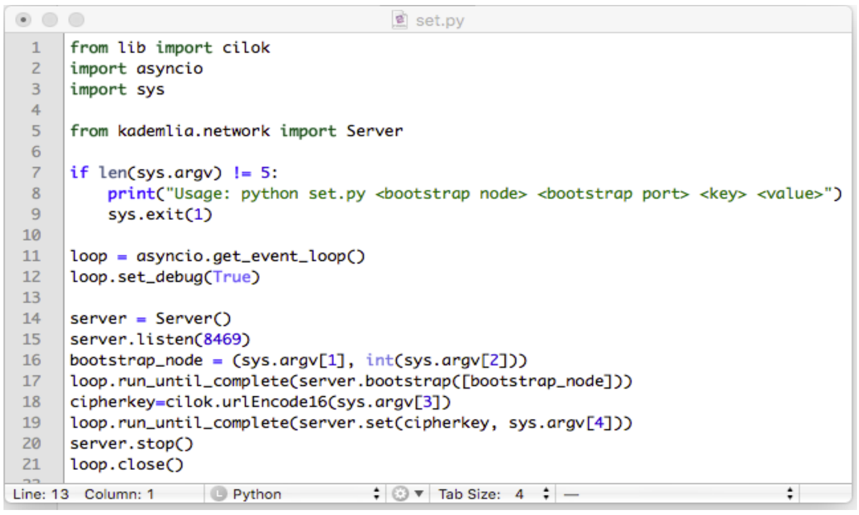
```
#!/usr/bin/env python
"""
Cryptography URI Locator Key
cilok.py
created by Rolly Maulana Awangga

"""
from pbkdf2 import PBKDF2
import config
from Crypto.Cipher import AES


def tailfill(ln):
                chars=[]
                for i in range(ln):
                        chars.append("X")
                return "".join(chars)

def urlEncode16(uri):
    ln = len(uri)
    multihex = (ln//16)*16+16
    sp = multihex - ln - len(str(ln))
    if ln>9:
        dt = str(ln)+uri+tailfill(sp)
    else:
        dt = "0"+str(ln)+uri+tailfill(sp-1)
    return encodeData16(dt)

def encodeData16(msg):
    key=PBKDF2(config.passphrase,config.salt).read(32)
    obj=AES.new(key,AES.MODE_CBC,config.iv)
    cp = obj.encrypt(msg)
    return cp.hex()#.encode("hex")
```

Line: 9 Column: 14 Python Tab Size: 4

Figure 4. The encryption algoritm

3.2. Distributed Hash Table

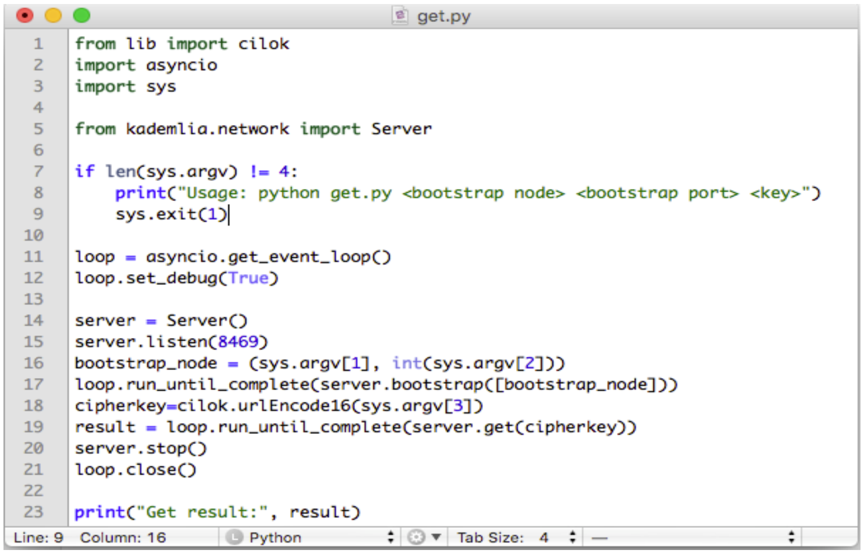
The e-KTP data stored in Distributed Hash Table using set code. To get value data, get code will be use. The set algorithm shows in Figure 5 and get algorithm shows in Figure 6.


set.py

```
from lib import cilok
import asyncio
import sys

from kademlia.network import Server

if len(sys.argv) != 5:
    print("Usage: python set.py <bootstrap node> <bootstrap port> <key> <value>")
    sys.exit(1)

loop = asyncio.get_event_loop()
loop.set_debug(True)

server = Server()
server.listen(8469)
bootstrap_node = (sys.argv[1], int(sys.argv[2]))
loop.run_until_complete(server.bootstrap([bootstrap_node]))
cipherkey=cilok.urlEncode16(sys.argv[3])
loop.run_until_complete(server.set(cipherkey, sys.argv[4]))
server.stop()
loop.close()
```

Line: 13 Column: 1 Python Tab Size: 4

Figure 5. The set algorithm



```
from lib import cilok
import asyncio
import sys

from kademlia.network import Server

if len(sys.argv) != 4:
    print("Usage: python get.py <bootstrap node> <bootstrap port> <key>")
    sys.exit(1)

loop = asyncio.get_event_loop()
loop.set_debug(True)

server = Server()
server.listen(8469)
bootstrap_node = (sys.argv[1], int(sys.argv[2]))
loop.run_until_complete(server.bootstrap([bootstrap_node]))
cipherkey=cilok.urlEncode16(sys.argv[3])
result = loop.run_until_complete(server.get(cipherkey))
server.stop()
loop.close()

print("Get result:", result)
```

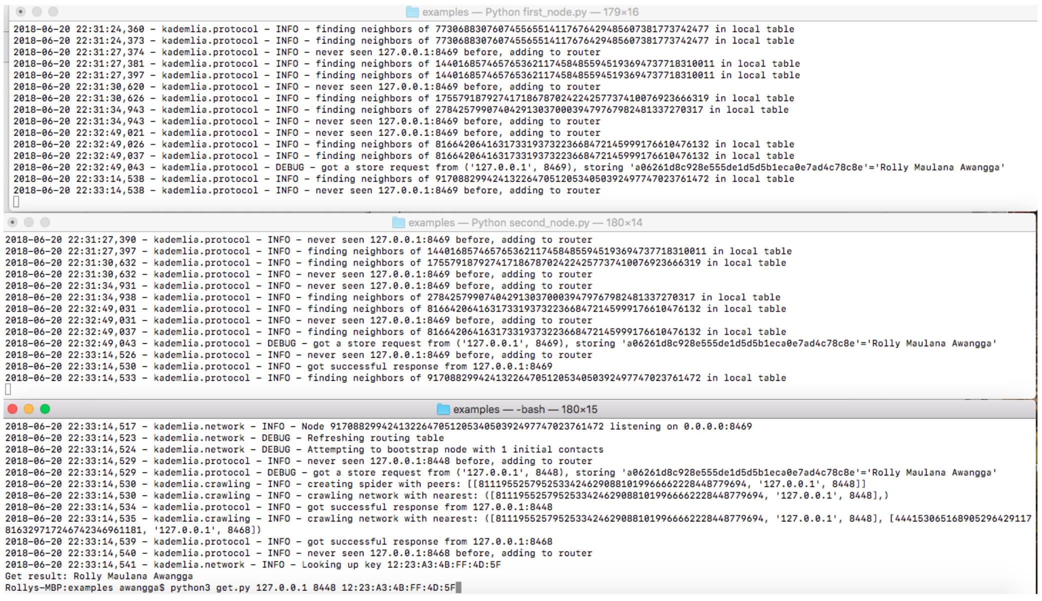
Figure 6. The get algorithm

3.3. Simulation and Testing

Simulation start with two nodes. The simulation have several scenario shows in Table 1. First scenario with two nodes go live. There is no problem with get and set data with two nodes go live. In second scenario, we put down first node and the data get the result. In third scenario we put down second node and the data get the result. The get simulation shows in Figure 7 and set scenario shows in Figure 8. The simulation has successfully make sure the data will be available if there minimum a node alive.

Table 1. Testing DHT

The Data	Node 1	Node 2
Get Result	Up	Up
Get Result	Down	Up
Get Result	Up	Down

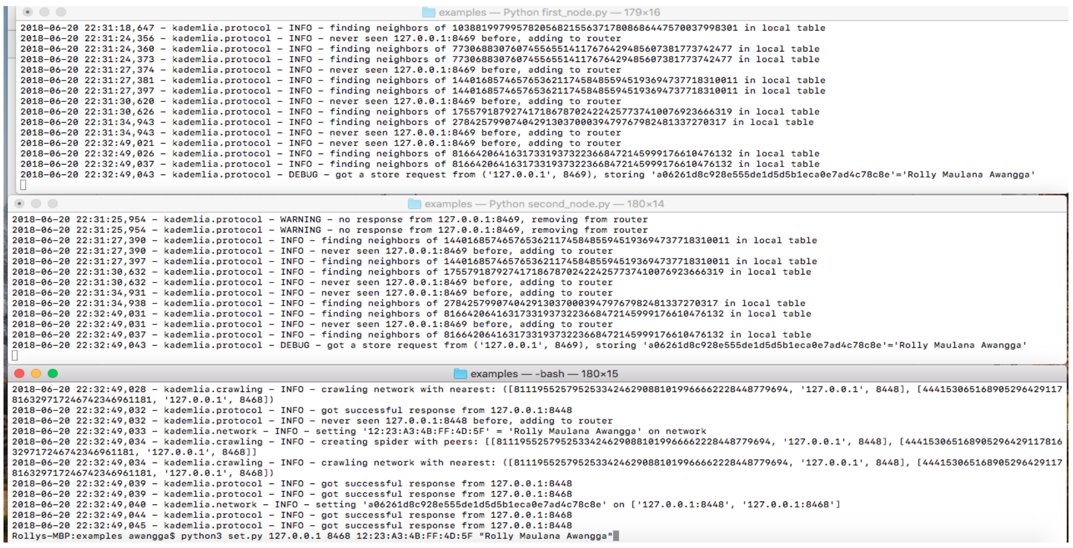


```
2018-06-20 22:31:24,360 - kademlia.protocol - INFO - finding neighbors of 773068830760745565514117676429485607381773742477 in local table
2018-06-20 22:31:24,373 - kademlia.protocol - INFO - finding neighbors of 773068830760745565514117676429485607381773742477 in local table
2018-06-20 22:31:27,374 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:27,381 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:27,397 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:30,620 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:30,626 - kademlia.protocol - INFO - finding neighbors of 175579187927417186787024224257737410076923666319 in local table
2018-06-20 22:31:34,943 - kademlia.protocol - INFO - finding neighbors of 27842579907404291303700039479767982481337270317 in local table
2018-06-20 22:31:34,943 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,021 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,026 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,037 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,043 - kademlia.protocol - DEBUG - got a store request from ('127.0.0.1', 8469), storing 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e'='Rolly Maulana Awangga'
2018-06-20 22:33:14,538 - kademlia.protocol - INFO - finding neighbors of 917088299424132264705120534050392497747023761472 in local table
2018-06-20 22:33:14,538 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
```

```
2018-06-20 22:31:27,390 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:27,397 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:30,632 - kademlia.protocol - INFO - finding neighbors of 175579187927417186787024224257737410076923666319 in local table
2018-06-20 22:31:30,632 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:34,931 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:34,938 - kademlia.protocol - INFO - finding neighbors of 27842579907404291303700039479767982481337270317 in local table
2018-06-20 22:32:49,031 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,031 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,037 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,043 - kademlia.protocol - DEBUG - got a store request from ('127.0.0.1', 8469), storing 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e'='Rolly Maulana Awangga'
2018-06-20 22:33:14,526 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:33:14,530 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8469
2018-06-20 22:33:14,533 - kademlia.protocol - INFO - finding neighbors of 917088299424132264705120534050392497747023761472 in local table
```

```
2018-06-20 22:33:14,517 - kademlia.network - INFO - Node 917088299424132264705120534050392497747023761472 listening on 0.0.0.0:8469
2018-06-20 22:33:14,523 - kademlia.network - DEBUG - Refreshing routing table
2018-06-20 22:33:14,524 - kademlia.network - DEBUG - Attempting to bootstrap node with 1 initial contacts
2018-06-20 22:33:14,529 - kademlia.protocol - INFO - never seen 127.0.0.1:8448 before, adding to router
2018-06-20 22:33:14,529 - kademlia.protocol - DEBUG - got a store request from ('127.0.0.1', 8448), storing 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e'='Rolly Maulana Awangga'
2018-06-20 22:33:14,530 - kademlia.crawling - INFO - creating spider with peers: [[81119552579525334246290881019966662228448779694, '127.0.0.1', 8448]]
2018-06-20 22:33:14,530 - kademlia.crawling - INFO - crawling network with nearest: ([81119552579525334246290881019966662228448779694, '127.0.0.1', 8448],)
2018-06-20 22:33:14,534 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8448
2018-06-20 22:33:14,535 - kademlia.crawling - INFO - crawling network with nearest: ([81119552579525334246290881019966662228448779694, '127.0.0.1', 8448], [444153065168905296429117
816329717246742346961181, '127.0.0.1', 8468])
2018-06-20 22:33:14,539 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8468
2018-06-20 22:33:14,540 - kademlia.protocol - INFO - never seen 127.0.0.1:8468 before, adding to router
2018-06-20 22:33:14,541 - kademlia.network - INFO - Looking up key 12:23:A3:4B:FF:4D:5F
Get result: Rolly Maulana Awangga
Rollys-MBP:examples awangga$ python3 get.py 127.0.0.1 8448 12:23:A3:4B:FF:4D:5F
```

Figure 7. The get scenario


examples — Python first_node.py — 179×16

```
2018-06-20 22:31:18,647 - kademlia.protocol - INFO - finding neighbors of 1038819979957820568215563717808686447570037998301 in local table
2018-06-20 22:31:24,356 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:24,360 - kademlia.protocol - INFO - finding neighbors of 773068830760745565514117676429485607381773742477 in local table
2018-06-20 22:31:24,373 - kademlia.protocol - INFO - finding neighbors of 773068830760745565514117676429485607381773742477 in local table
2018-06-20 22:31:27,374 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:27,381 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:27,397 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:30,620 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:30,626 - kademlia.protocol - INFO - finding neighbors of 175579187927417186787024224257737410076923666319 in local table
2018-06-20 22:31:34,943 - kademlia.protocol - INFO - finding neighbors of 278425799074042913037000394797679824813372703l7 in local table
2018-06-20 22:31:34,943 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,021 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,026 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,037 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,043 - kademlia.protocol - DEBUG - got a store request from ('127.0.0.1', 8469), storing 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e'='Rolly Maulana Awangga'
```

examples — Python second_node.py — 180×14

```
2018-06-20 22:31:25,954 - kademlia.protocol - WARNING - no response from 127.0.0.1:8469, removing from router
2018-06-20 22:31:25,954 - kademlia.protocol - WARNING - no response from 127.0.0.1:8469, removing from router
2018-06-20 22:31:27,390 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:27,390 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:27,397 - kademlia.protocol - INFO - finding neighbors of 1440168574657653621174584855945193694737718310011 in local table
2018-06-20 22:31:30,632 - kademlia.protocol - INFO - finding neighbors of 175579187927417186787024224257737410076923666319 in local table
2018-06-20 22:31:30,632 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:34,931 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:31:34,938 - kademlia.protocol - INFO - finding neighbors of 278425799074042913037000394797679824813372703l7 in local table
2018-06-20 22:32:49,031 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,031 - kademlia.protocol - INFO - never seen 127.0.0.1:8469 before, adding to router
2018-06-20 22:32:49,037 - kademlia.protocol - INFO - finding neighbors of 816642064163173319373223668472145999176610476132 in local table
2018-06-20 22:32:49,043 - kademlia.protocol - DEBUG - got a store request from ('127.0.0.1', 8469), storing 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e'='Rolly Maulana Awangga'
```

examples — -bash — 180×15

```
2018-06-20 22:32:49,028 - kademlia.crawling - INFO - crawling network with nearest: ([811195525795253342462908810199666622284487796694, '127.0.0.1', 8448], [444153065168905296429117
8163297172467423469611181, '127.0.0.1', 8468])
2018-06-20 22:32:49,032 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8448
2018-06-20 22:32:49,032 - kademlia.protocol - INFO - never seen 127.0.0.1:8448 before, adding to router
2018-06-20 22:32:49,033 - kademlia.network - INFO - setting '12:23:A3:4B:FF:4D:5F' = 'Rolly Maulana Awangga' on network
2018-06-20 22:32:49,034 - kademlia.crawling - INFO - creating spider with peers: [[811195525795253342462908810199666622284487796694, '127.0.0.1', 8448], [4441530651689052964291178l6
3297172467423469611181, '127.0.0.1', 8468]]
2018-06-20 22:32:49,034 - kademlia.crawling - INFO - crawling network with nearest: ([811195525795253342462908810199666622284487796694, '127.0.0.1', 8448], [444153065168905296429117
8163297172467423469611181, '127.0.0.1', 8468])
2018-06-20 22:32:49,039 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8448
2018-06-20 22:32:49,039 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8468
2018-06-20 22:32:49,040 - kademlia.network - INFO - setting 'a06261d8c928e555de1d5d5b1eca0e7ad4c78c8e' on ['127.0.0.1:8448', '127.0.0.1:8468']
2018-06-20 22:32:49,044 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8468
2018-06-20 22:32:49,045 - kademlia.protocol - INFO - got successful response from 127.0.0.1:8448
Rollys-MBP:examples awangga$ python3 set.py 127.0.0.1 8468 12:23:A3:4B:FF:4D:5F "Rolly Maulana Awangga"
```

Figure 8. The set scenario

4. Conclusion

The research proposed KANSA as open standard of network interoperability to utilise the technologi in Indonesian Citizenship Card. KANSA use decentralised model to decreasing server infrastructure cost. KANSA use DHT to store the data and successfully act as reliable and high availability to store the data. PBKDF2 and AES-CBC was used to secure the data from man of the middle and other attacker who want to steal the data. Further more development can be developed on DHT network side and encryption side. In the DHT side, the method for identifying and searching every node in the network can be optimise with another method of search algorithm. In encryption side, another encryption algorithm can make the data more secure and save from unattended user.

Acknowledgement

This research was funding by Indonesia Ministry of Research and Higher Education in PDP scheme.

References

[1] Pane SF, Awangga RM, Azhari BR, Tartila GR. RFID-based conveyor belt for improve warehouse operations. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2019; 17(2): 794-800.
[2] Gao L, Zhang L, Lin F, Ma M. Secure RFID Authentication Schemes based on Security Analysis and Improvements of the USI Protocol. *IEEE Access*. 2019.
[3] Kitsos P. Security in RFID and sensor networks. Auerbach Publications. 2016.
[4] Arneson MR, Bandy WR. *System and method for randomization for robust rfid security*. US Patent App. 15/374,889. 2017.
[5] Arulmozhi P, Rayappan J, Raj P. *A Lightweight Memory-Based Protocol Authentication Using Radio Frequency Identification (RFID)*. Advances in Big Data and Cloud Computing 2019, Proceedings of ICBDCC18. 2019: 163–172.
[6] Q Qian Q, Jia YL, Zhang R. A Lightweight RFID Security Protocol Based on Elliptic Curve Crytography. *IJ Network Security*. 2016; 18(2): 354-361.
[7] Mishra Y, Marwah GK, Verma S. Arduino Based Smart RFID Security and Attendance System with Audio Acknowledgement. *International Journal of Engineering Research and Technology*. 2015.
[8] Yang L, Wu Q, Bai Y, Zheng H, Lin S. An improved hash-based rfid two-way security authentication protocol and application in remote education. *Journal of Intelligent & Fuzzy Systems*. 2016; 31(5): 2713–2720.
[9] Awangga RM, Harani NH, Setyawan MYH. KAFA: A novel interoperability open framework to utilize indonesian electronic identity card. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2019; 17(2): 712-718.

[10] Voulgaris S, Dobson M, Van Steen M. Decentralized network-level synchronization in mobile ad hoc networks. *ACM Transactions on Sensor Networks (TOSN)*. 2016; 12(1): 5.

[11] M Zghaibeh M. O-Torrent: A fair, robust, and free riding resistant P2P content distribution mechanism. *Peer-to-Peer Networking and Applications*. 2018; 11(3): 579-591.

[12] Rodriguez-Gomez RA, Macia-Fernandez G, Casares-Andres A. On Understanding the Existence of a Deep Torrent. *IEEE Communications Magazine*. 2017; 55(7): 64-69.

[13] Xinxing Z, Zhihong T, Luchen Z. *A measurement study on mainline dht and magnet link*. 2016 IEEE First International Conference on Data Science in Cyberspace (DSC). 2016: 11–19.

[14] Awangga RM. *Sampeu: Servicing web map tile service over web map service to increase computation performance*. IOP Conference Series: Earth and Environmental Science. 2018; 145 (1): 012057.

[15] Nguyen DN, Tran XH, Nguyen HS. *A cluster-based file replication scheme for dht-based file backup systems*. Advanced Technologies for Communications (ATC), 2016 International Conference. 2016: 204–209.

[16] Varyani N, Nikhil S, Shekhawat VS. *Latency and Routing Efficiency Based Metric for Performance Comparison of DHT Overlay Networks*. Advanced Information Networking and Applications Workshops (WAINA), 2016 30th International Conference. 2016: 337–342.

[17] Vijayaraj A, Suresh RM, Devi K. *Load balancing algorithm using reputation-ReDS in the magnified distributed hash table*. Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015 International Conference. 2015: 1–5.

[18] Senthil M, Sugashini K, Abirami M, Vaigai N. *Identification and recovery of repaired nodes based on distributed hash table in WSN*. Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015 International Conference. 2015: 1–4.

[19] Awangga RM, Fathonah NS, Hasanudin TI. *Colenak: GPS tracking model for post-stroke rehabilitation program using AES-CBC URL encryption and QR-Code*. Information Technology, Information Systems and Electrical Engineering (ICITISEE), 2017 2nd International conferences. 2017: 255–260.

[20] Vaidehi M, Rabi BJ. *Design and analysis of aes-cbc mode for high security applications*. Current Trends in Engineering and Technology (ICCTET), 2014 2nd International Conference. 2014: 499–502.

[21] Wulamarisman CR, Windarta S. *Distinguishing attack and second preimage attack on Mini-AES CBC-MAC.* Advanced Informatics: Concept, Theory and Application (ICAICTA), 2014 International Conference. 2014: 326–331.

[22] Li X, Cao C, Li P, Shen S, Chen Y, Li L. *Energy-Efficient Hardware Implementation of LUKS PBKDF2 with AES on FPGA*. Trustcom/BigDataSE/I SPA, 2016 IEEE. 2016: 402–409.

[23] Chen X, Li X, Chen Y, Li P, Xing J, Li L. *A modified PBKDF2-based MAC scheme XKDF*. TENCON 2015-2015 IEEE Region 10 Conference. 2015: 1–6.

[24] Li N, Dang X, Zhang Y. *Realizing high-speed PBKDF2 based on FPGA*. Intelligent Transportation, Big Data and Smart City (ICITBS), 2015 International Conference. 2015: 580–583.

[25] Li X, Zhao C, Pan K, Lin S, Chen X, Chen B, Le D, Guo D. On the security analysis of PBKDF2 in openoffice. *Journal of Software*. 2015; 10(2): 116-127.

[26] Rahman MT, Mahi MJ. *Proposal for SZRP protocol with the establishment of the salted SHA-256 Bit HMAC PBKDF2 advance security system in a MANET*. Electrical Engineering and Information & Communication Technology (ICEEICT), 2014 International Conference. 2014: 1–5.

[27] Ertaul L, Kaur M, Gudise VAKR. *Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms*. Proceedings of the International Conference on Wireless Networks (ICWN), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016: 66.

[28] Visconti A, Bossi S, Ragab H, Calò A. *On the weaknesses of PBKDF2*. International Conference on Cryptology and Network Security. 2015: 119–126.