ДОДАТОК А

Лістинг програмного коду

```
#include "bcaNodeActions.h"
#include "bcaCentralLoggingSystem.h"

#include <bcaRegistrationForm.h>


namespace Ui {
class main_wnd;
}

class main_wnd : public QWidget
{
 Q_OBJECT
public:
 explicit main_wnd(QWidget *parent = nullptr);
 ~main_wnd();
private:
 Ui::main_wnd *ui;
 NodeActions* m_NodeActionsManager;

 QList<RegistrationForm*> tempUserEntities;

public:
 NodeActions* GetNodeActionsManagerEnitity();

private slots:
 void on_pb_node1_clicked();
 void on_pb_node2_clicked();
 void on_pb_node3_clicked();
 void closeEvent(QCloseEvent *event);
 void on_pb_mitm_clicked();
 void on_pb_ddos_clicked();
 void on_pb_p51_clicked();
};

main_wnd::main_wnd(QWidget *parent) :
 QWidget(parent),
 ui(new Ui::main_wnd)
{
 ui->setupUi(this);

 m_NodeActionsManager = new NodeActions;

 m_NodeActionsManager->InitLogSystem(ui->te_log);

 CLS_ENTITY = new bcaCentralLoggingSystem(ui->te_log);
}

main_wnd::~main_wnd()
{
 delete ui;
}

NodeActions *main_wnd::GetNodeActionsManagerEnitity()
{
 return m_NodeActionsManager;
```

```
}


//ui->te_log->moveCursor(QTextCursor::End);


void main_wnd::closeEvent(QCloseEvent *event)
{
 for(auto nod: m_NodeActionsManager->getConnectedNodes())
 {
 // nod->blockchain_ui->close();
 delete nod;
 }
}


void main_wnd::on_pb_node1_clicked()
{
 RegistrationForm* newUserRegistrationForm = new RegistrationForm(this);
 newUserRegistrationForm->setWindowTitle("Реєстрація нового користувача, що
володіє автомобілем " + qobject_cast<QPushButton*>(sender())->text());
 tempUserEntities.push_back(newUserRegistrationForm);
 newUserRegistrationForm->show();

 /*nodeSetForm* newNode = new nodeSetForm(this);
 newNode->setWindowTitle("1");
 newNode->show();


 for(auto nod: m_NodeActionsManager->getConnectedNodes())
 {
 connect(nod->blockchain_ptr, SIGNAL(newChainAdded()), GetNodeActionsManagerEn-
itity(), SLOT(UpdateAllBlocks()));
 }*/
}

void main_wnd::on_pb_node2_clicked()
{
 RegistrationForm* newUserRegistrationForm = new RegistrationForm(this);
 tempUserEntities.push_back(newUserRegistrationForm);
 newUserRegistrationForm->show();

 /*nodeSetForm* newNode = new nodeSetForm(this);
 newNode->setWindowTitle("2");
 newNode->show();
 for(auto nod: m_NodeActionsManager->getConnectedNodes())
 {
 connect(nod->blockchain_ptr, SIGNAL(newChainAdded()), GetNodeActionsManagerEn-
itity(), SLOT(UpdateAllBlocks()));
 }*/
}

void main_wnd::on_pb_node3_clicked()
{
 RegistrationForm* newUserRegistrationForm = new RegistrationForm(this);
 tempUserEntities.push_back(newUserRegistrationForm);
 newUserRegistrationForm->show();

 /*nodeSetForm* newNode = new nodeSetForm(this);
 newNode->setWindowTitle("3");
 newNode->show();
 for(auto nod: m_NodeActionsManager->getConnectedNodes())
 {
```

```
 connect(nod->blockchain_ptr, SIGNAL(newChainAdded()), this->GetNodeActionsMan-
agerEnitity(), SLOT(UpdateAllBlocks()));
 }*/
}

void main_wnd::on_pb_mitm_clicked()
{
 m_NodeActionsManager->getConnectedNodes().at(0)->blockchain_ptr->at-
tack_block(0);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Starting mitm to: " + m_NodeActionsManager->getConnected-
Nodes().at(0)->blockchain_ptr->getNodeIP())
 LOG_DEBUG("\n==================================\n")
 m_NodeActionsManager->TESTallNodes();
}

void main_wnd::on_pb_ddos_clicked()
{
 m_NodeActionsManager->getConnectedNodes().removeAt(0);
 ui->pb_node1->setEnabled(false);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Starting DDOS on: " + m_NodeActionsManager->getConnected-
Nodes().at(0)->blockchain_ptr->getNodeIP())
 LOG_DEBUG("\n==================================\n")
 m_NodeActionsManager->TESTallNodes();
}

void main_wnd::on_pb_p51_clicked()
{
 m_NodeActionsManager->getConnectedNodes().at(0)->blockchain_ptr->at-
tack_block(1);
 m_NodeActionsManager->getConnectedNodes().at(1)->blockchain_ptr->at-
tack_block(1);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Getting controle on: " + m_NodeActionsManager->getConnected-
Nodes().at(0)->blockchain_ptr->getNodeIP())
 LOG_DEBUG("Getting controle on: " + m_NodeActionsManager->getConnected-
Nodes().at(1)->blockchain_ptr->getNodeIP())
 LOG_DEBUG("\n==================================\n")
 m_NodeActionsManager->TESTallNodes();
}


#include "nodesetform.h"
#include "bcaCentralLoggingSystem.h"

#include "kalyna_enc/qKalynaEncryption.h"
#include <QCryptographicHash>

struct LoggingSystem
{
 bool m_IsLoggingAvailable = false;
 QTextEdit* m_TextEditToLog;
 LoggingSystem(QTextEdit* TextEditToLog);
};

struct blockchain_itc
{
 node_standalone* blockchain_ptr;
 nodeSetForm* blockchain_ui;
 blockchain_itc(node_standalone* b_ptr, nodeSetForm* b_ui):block-
chain_ptr(b_ptr),blockchain_ui(b_ui){}
};
```

```
class ConnectionEncryptor
{
 QKalynaEncryption* encryptorEntity;
public:
 ConnectionEncryptor(QString currentDeviceID, QString currendDeviceSecureID,
QString );

 //BCAPackageContents *GenerateSecuredPackage(QByteArray userData, QString
userID, QString userPassword);

 // bool ValidateSecuredPackage(BCAPackageContents* validateReceivedPackage);
};

class NodeActions : public QObject
{
 Q_OBJECT
 QList<blockchain_itc*> connected_nodes;
public:
 NodeActions();
 void TESTallNodes();
 void ShutdownSystem();
 void NodeSelected(const int nodeIndex, const bool enable);
 void DDoSAttack(const int nodeIndex, const int blockIndex);
 void MitmAttack(const int nodeIndex, const int blockIndex);
 void ChangeBlockData(const int nodeIndex, const int blockIndex);

 void addNode(blockchain_itc* blc);
 int test(blockchain_itc* blc);

 QList<blockchain_itc*> getConnectedNodes();

 void InitLogSystem(QTextEdit* textEditToShowLog);


 /*bool AuthRequest(BCAPackageContents package);
 bool CloseCarRequest(BCAPackageContents package);*/

public slots:
 void UpdateAllBlocks();
};

#endif // NODEACTIONS_H

#include "bcaNodeActions.h"

LoggingSystem::LoggingSystem(QTextEdit *TextEditToLog): m_TextEditToLog(TextE-
ditToLog)
{
 if(m_TextEditToLog == nullptr)
 {
 qDebug() << "LOGGING UNAVAILABLE";
 }
}


NodeActions::NodeActions()
{
}

void NodeActions::UpdateAllBlocks()
{
 for(auto nod: connected_nodes)
 {
 for(auto nodss: connected_nodes)
```

```
 {
 if(nod->blockchain_ptr->getLastBlock()!=nodss->blockchain_ptr->getLastBlock()
&& (nodss->blockchain_ptr->getChain().length()>nod->blockchain_ptr-
>getChain().length()))
 {
 CR_Block_BC* block = new CR_Block_BC(*nodss->blockchain_ptr->getLastBlock());
 nod->blockchain_ptr->addBlock(block);
 }
 else if(nod->blockchain_ptr->getLastBlock()!=nodss->blockchain_ptr->getLast-
Block() && (nodss->blockchain_ptr->getChain().length()<nod->blockchain_ptr-
>getChain().length()))
 {
 CR_Block_BC* block = new CR_Block_BC(*nod->blockchain_ptr->getLastBlock());
 nodss->blockchain_ptr->addBlock(block);
 }

 }
 }
}

void NodeActions::TESTallNodes()
{
 LOG_DEBUG("Нодів в мережі: " + QString::number(connected_nodes.length()))
 LOG_DEBUG("\n================================\n")
 LOG_DEBUG("Дані в блокчейні на даний момент: \n")
 for(int i=0; i<connected_nodes.length(); i++)
 {
 LOG_DEBUG("Вузол "+QString::number(i)+":\n")
 //qDebug() << connected_nodes.at(i)->blockchain_ptr->test();
 for(auto temp_v: connected_nodes.at(i)->blockchain_ptr->getChain())
 {
 LOG_DEBUG(temp_v->data()+"\n")
 }
 }
 LOG_DEBUG("================================\n")
 LOG_DEBUG("Перевірка валідності блокчейн: \n")
 for(int i=0; i<connected_nodes.length(); i++)
 {
 LOG_DEBUG("Вузол "+QString::number(i)+":\n")
 bool tnv = connected_nodes.at(i)->blockchain_ptr->isValid();
 if(tnv == true)
 {
 LOG_DEBUG(QString("Вузол %1 працює в нормальному режимі").arg(con-
nected_nodes.at(i)->blockchain_ptr->getNodeIP()));
 // findChild<QPushButton*>("pb_node"+QString::number(i+1))-
>setStyleSheet("background-color: green;");
 }
 else {
 LOG_ERROR(QString("Вузол %1 був атакований!").arg(connected_nodes.at(i)->block-
chain_ptr->getNodeIP()));
 //findChild<QPushButton*>("pb_node"+QString::number(i+1))->setStyleSheet("back-
ground-color: red;");
 }
 LOG_DEBUG(QString::number(tnv)+"\n")
 }
 LOG_DEBUG("================================\n")
}

void NodeActions::ShutdownSystem()
{
 for(auto nod: connected_nodes)
 {
 nod->blockchain_ui->close();
 delete nod;
```

```
 }
}


void NodeActions::NodeSelected(const int nodeIndex, const bool enable)
{
 nodeSetForm* newNode = new nodeSetForm(this);
 newNode->setWindowTitle("1");
 newNode->show();
 for(auto nod: connected_nodes)
 {
 connect(nod->blockchain_ptr, SIGNAL(newChainAdded()), this, SLOT(up-
dateAllBlocks()));
 }
}


void NodeActions::DDoSAttack(const int nodeIndex, const int blockIndex)
{
 connected_nodes.at(0)->blockchain_ptr->attack_block(0);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Starting mitm to: "+connected_nodes.at(0)->blockchain_ptr-
>getNodeIP())
 LOG_DEBUG("\n==================================\n")
 TESTallNodes();
}

void NodeActions::MitmAttack(const int nodeIndex, const int blockIndex)
{
 connected_nodes.removeAt(0);
 //ui->pb_node1->setEnabled(false);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Starting DDOS on: "+connected_nodes.at(0)->blockchain_ptr-
>getNodeIP())
 LOG_DEBUG("\n==================================\n")
 TESTallNodes();
}

void NodeActions::ChangeBlockData(const int nodeIndex, const int blockIndex)
{
 connected_nodes.at(0)->blockchain_ptr->attack_block(1);
 connected_nodes.at(1)->blockchain_ptr->attack_block(1);
 LOG_DEBUG("\n==================================\n")
 LOG_DEBUG("Getting controle on: "+connected_nodes.at(0)->blockchain_ptr-
>getNodeIP())
 LOG_DEBUG("Getting controle on: "+connected_nodes.at(1)->blockchain_ptr-
>getNodeIP())
 LOG_DEBUG("\n==================================\n")
 TESTallNodes();
}

void NodeActions::addNode(blockchain_itc *blc)
{
 LOG_DEBUG("HGD: " + blc->blockchain_ptr->getNodeIP());
 connected_nodes.push_back(blc);
}

int NodeActions::test(blockchain_itc *blc)
{
 return connected_nodes.indexOf(blc);
}

QList<blockchain_itc *> NodeActions::getConnectedNodes()
{
```

```
 return connected_nodes;
}

void NodeActions::InitLogSystem(QTextEdit *textEditToShowLog)
{
 CLS_ENTITY = new bcaCentralLoggingSystem(textEditToShowLog);
}

ConnectionEncryptor::ConnectionEncryptor(QString currentDeviceID, QString
currendDeviceSecureID, QString)
{
 encryptorEntity = new QKalynaEncryption(QKalynaEncryption::KALYNA_256, QKal-
ynaEncryption::ECB, QKalynaEncryption::Padding::ISO);
}

/*BCAPackageContents *ConnectionEncryptor::GenerateSecuredPackage(QByteArray
userData, QString userID, QString userPassword)
{
 QByteArray hashKey = QCryptographicHash::hash(userPassword.toUtf8(), QCryp-
tographicHash::Sha256);

 BCAPackageContents* newPackage = new BCAPackageContents;
 newPackage->

 QByteArray dd = enc.encode("test_val", hashKey);
 qDebug() << dd;
 qDebug() << enc.decode(dd, hashKey);
}*/

/*bool ConnectionEncryptor::ValidateSecuredPackage(BCAPackageContents *vali-
dateReceivedPackage)
{

}*/


#include <nodesetform.h>
#include <bcaCarManagementMenu.h>

RegistrationForm::RegistrationForm(QObject *parent) :
 QWidget(),
 ui(new Ui::RegistrationForm)
{
 ui->setupUi(this);
 parentWidget = parent;
}

RegistrationForm::~RegistrationForm()
{
 delete ui;
}

void RegistrationForm::on_pushButtonCancel_clicked()
{
 close();
}

void RegistrationForm::on_pushButtonRegister_clicked()
{
 currentUserData = new UserData;
 currentUserData->userName = ui->lineEditLogin->text();
 currentUserData->passwordHash = QCryptographicHash::hash(ui->lineEditPwd-
>text().toUtf8(), QCryptographicHash::Sha1);
```

```
 currentUserData->VehSecIdHash = QCryptographicHash::hash(ui->lineEditPhy-
sAuthCode->text().toUtf8(), QCryptographicHash::Sha1);
 currentUserData->VehVINumberHash = QCryptographicHash::hash(ui->lineEditVIN-
>text().toUtf8(), QCryptographicHash::Sha1);

 hide();

 /*nodeSetForm* newNode = new nodeSetForm(parentWidget);
 newNode->setWindowTitle(currentUserData->userName);
 newNode->show();*/

 CarManagementMenu* carManagement = new CarManagementMenu(parentWidget, curren-
tUserData);
 carManagement->show();
}

void RegistrationForm::on_pushButtonGenAuthData_clicked()
{
 ui->lineEditLogin->setText(QCryptographicHash::hash(QString(windowTitle() +
static_cast<char>(rand() % 255)).toUtf8(), QCryptographicHash::Sha1).toHex());
 ui->lineEditPwd->setEchoMode(QLineEdit::EchoMode::Normal);
 ui->lineEditPwd->setText(QCryptographicHash::hash(QString("@PW@" +
static_cast<char>(rand() % 255) + static_cast<char>(rand() % 255) + window-
Title()).toUtf8(), QCryptographicHash::Sha1).toHex());
}

#include <QObject>
#include <bcaRegistrationForm.h>

class node_standalone;

class CR_Block_BC : public QObject
{
 Q_OBJECT

 friend node_standalone;
public:
 explicit CR_Block_BC(int index, QString data, UserData* currentCarInfo =
nullptr, QString previouseHash="", QObject *parent = nullptr);
 CR_Block_BC(const CR_Block_BC &block){
 mIndex = block.mIndex; mTimeStamp= block.mTimeStamp; mData = block.mData; mHash
= block.mData; mPreviouseHash = block.mPreviouseHash; mNonce = block.mNonce;
 }


 QString hash() const;
 void setHash(const QString &hash);

 int index() const;
 void setIndex(int index);

 qint64 timeStamp() const;
 void setTimeStamp(const qint64 &timeStamp);

 QString data() const;
 void setData(const QString &data);

 QString previouseHash() const;
 void setPreviouseHash(const QString &previouseHash);

 void mineBlock(int difficulty);

private:
 QByteArray calculateHash();
```

```
private:
 int mIndex;
 qint64 mTimeStamp;
 QByteArray mData;
 QByteArray mHash;
 QByteArray mPreviouseHash;
 int mNonce;

 QString userToAuthUID;
 QByteArray userAuthData;

 UserData* carInfo;
};


CR_Block_BC::CR_Block_BC(int index, QString data, UserData* currentCarInfo,
QString previouseHash, QObject *parent) : QObject(parent),
 mIndex(index),
 mTimeStamp(QDateTime::currentMSecsSinceEpoch()),
 mData(data.toUtf8()),
 mHash(""),
 mPreviouseHash(previouseHash.toUtf8()),
 mNonce(-1),
 carInfo(currentCarInfo)
{

}

QByteArray CR_Block_BC::calculateHash()
{
 QByteArray lCode = QString().number(mIndex).toUtf8() + QString().num-
ber(mTimeStamp).toUtf8() + mData + QString().number(mNonce).toUtf8() + mPrevi-
ouseHash;
 QByteArray lArray = QCryptographicHash::hash(lCode, QCryptographi-
cHash::Sha256).toHex();

 return lArray;
}

QString CR_Block_BC::previouseHash() const
{
 return mPreviouseHash;
}

void CR_Block_BC::setPreviouseHash(const QString &previouseHash)
{
 mPreviouseHash = previouseHash.toUtf8();
}

QString CR_Block_BC::data() const
{
 return mData;
}

void CR_Block_BC::setData(const QString &data)
{
 qDebug() << this->index();
 mData = data.toUtf8();
}

qint64 CR_Block_BC::timeStamp() const
{
 return mTimeStamp;
```

```
}

void CR_Block_BC::setTimeStamp(const qint64 &timeStamp)
{
 mTimeStamp = timeStamp;
}

int CR_Block_BC::index() const
{
 return mIndex;
}

void CR_Block_BC::setIndex(int index)
{
 mIndex = index;
}

QString CR_Block_BC::hash() const
{
 return mHash;
}

void CR_Block_BC::setHash(const QString &hash)
{
 mHash = hash.toUtf8();
}

void CR_Block_BC::mineBlock(int difficulty) {
 QString lString;
 for (int i = 0; i < difficulty; ++i) {
 lString.append("0");
 }

 do {
 mNonce++;
 mHash = this->calculateHash();
 } while (mHash.mid(0, difficulty) != lString);

 qDebug() << "Block mined: " << mHash << endl;
}
```

Auth_can.h

```
#ifndef AUTH_CAN_H
#define AUTH_CAN_H

#include <QString>
#include <QByteArray>
#include <QTime>
#include <QCryptographicHash>
#include <QDebug>
#include <QRandomGenerator>
#include <QSignalMapper>
#include <Windows.h>
#include <QPlainTextEdit>
#include "qaesencryption.h"
struct IP_list_element
{
 QByteArray ip;
 int type;
 QByteArray description;
```

```
 IP_list_element(QByteArray ip, int type, QByteArray description): ip(ip),
type(type), description(description){}
};
struct user
{
 int user_id;
 QByteArray user_name_h;
 QByteArray user_pass_h;
 QList<IP_list_element*> user_ip_whitelist;
 QByteArray pwd_hid;
 QByteArray pin_temp;
 bool isAuthentifcated;
 //car info
 QByteArray VIN_h;
 QByteArray currentGPS_point;
 QByteArray keyless_go_id;
 user(int user_id, QByteArray user_name, QByteArray user_pass):user_id(user_id),
user_name_h(user_name), user_pass_h(user_pass), isAuthentifcated(false){}
 void update_info(QByteArray new_ip="0.0.0.0", QByteArray cur-
rentGPS_point="000", QByteArray keyless_go_id="XXXXXXXXXXXXX", QByteArray
PK="XXXXXXXXXXXXX");
};
class auth_server: public QObject
{
 Q_OBJECT
 QList<user*> users_db;
 QPlainTextEdit* log_widget;
public:
 void setLogWidget(QPlainTextEdit* log_w){log_widget = log_w;}
 void add_user(int user_id, QByteArray user_name, QByteArray user_pass, QByteAr-
ray VIN_hash, QByteArray currentGPS_point_hash, QByteArray keyless_go_id_hash);
 int auth_code_generate(int user_id, QString user_name, QByteArray VIN_hash,
QByteArray currentGPS_point_hash, QByteArray keyless_go_id_hash);
 bool check_pin(QByteArray VIN_hash, QByteArray currentGPS_point_hash, QByteAr-
ray keyless_go_id_hash, QByteArray PIN);
public slots:
 int new_connection_request(QByteArray IP, QByteArray credentianls);
};

#endif // AUTH_CAN_H
```

Auth_can.cpp

```
#include "auth_can.h"


 void user::update_info(QByteArray new_ip, QByteArray currentGPS_point, QByteAr-
ray keyless_go_id, QByteArray PK_hash)//only on spec auth
 {
 if(PK_hash == pwd_hid)
 {
 if(new_ip!="0.0.0.0")
 {
 IP_list_element* allowed_ip = new IP_list_element(new_ip, 0, "MERCEDES-
2imas");//TEST
 user_ip_whitelist.push_back(allowed_ip);
 }
 else if (currentGPS_point!="000") {
 this->currentGPS_point = currentGPS_point;
 }
 else if (keyless_go_id!="XXXXXXXXXXXXX") {
 this->keyless_go_id = keyless_go_id;
 }
 }
```

```
 }

 int auth_server::new_connection_request(QByteArray IP, QByteArray credentianls)
 {
 qDebug() << users_db.length();
 for(auto temp: users_db)
 {
 for(auto t_ips: temp->user_ip_whitelist)
 {
 if(t_ips->ip==IP)
 {
 if(credentianls == QCryptographicHash::hash(QByteArray(temp->user_name_h+temp-
>user_pass_h), QCryptographicHash::Sha1))
 {
 temp->isAuthentifcated = true;
 qDebug() << "Server AUth OK!";
 //server sends push message on mobile device
 return auth_code_generate(temp->user_id, temp->user_name_h, temp->VIN_h, temp-
>currentGPS_point, temp->keyless_go_id);
 }
 }
 }
 }
 return 122;
 }

 int auth_server::auth_code_generate(int user_id, QString user_name, QByteArray
VIN_hash, QByteArray currentGPS_point_hash, QByteArray keyless_go_id_hash)
 {
 for(auto temp: users_db)
 {
 if(temp->VIN_h == VIN_hash && temp->currentGPS_point == currentGPS_point_hash
&& temp->keyless_go_id == keyless_go_id_hash)
 {
 QByteArray pin_auth;
 for(int i=0; i<6; i++)
 {
 pin_auth+=QByteArray::number(rand()%10);
 temp->pin_temp = pin_auth+"@T@"+QTime::currentTime().toString().toUtf8();
 QByteArray token = QCryptographicHash::hash(QByteArray(pin_auth), QCryp-
tographicHash::Sha1);
 }
 log_widget->setPlainText(QByteArray("New message:\n Your auth PIN: "+temp-
>pin_temp.mid(0, temp->pin_temp.indexOf("@T@"))));
 //send token to mobile, mobile has access to critical PIN data, but not to gen
time
 return 101;//tells to wait for pin input
 }
 }
 return 100;//error code
 }

 void auth_server::add_user(int user_id, QByteArray user_name,QByteArray
user_pass, QByteArray VIN_hash, QByteArray currentGPS_point_hash, QByteArray
keyless_go_id_hash)
 {
 user* new_user = new user(user_id, user_name, user_pass);
 IP_list_element* allowed_ip = new IP_list_element("187.90.0.101", 0, "MERCEDES-
2imas");//TEST
 new_user->user_ip_whitelist.push_back(allowed_ip);
 new_user->VIN_h = VIN_hash;
 new_user->currentGPS_point = currentGPS_point_hash;
 new_user->keyless_go_id = keyless_go_id_hash;
 users_db.push_back(new_user);
```

```
 }


 bool auth_server::check_pin(QByteArray VIN_hash, QByteArray cur-
rentGPS_point_hash, QByteArray keyless_go_id_hash, QByteArray PIN)
 {
 qDebug() << PIN;
 for(auto temp: users_db)
 {
 if(temp->VIN_h == VIN_hash && temp->currentGPS_point == currentGPS_point_hash
&& temp->keyless_go_id == keyless_go_id_hash)
 {
 int tsm = temp->pin_temp.indexOf("@T@");
 qDebug() << QString::fromStdString(temp->pin_temp.mid(0, tsm).to-
Hex().toStdString());
 // Sleep(2000); //!!!!!!!!!!
 qDebug() << -(QTime::currentTime()).secsTo( (QTime::fromString(temp-
>pin_temp.mid(tsm+3, temp->pin_temp.length()-tsm))));
 if( (-(QTime::currentTime()).secsTo( (QTime::fromString(temp-
>pin_temp.mid(tsm+3, temp->pin_temp.length()-tsm)))))<=15)
 {
 if(QCryptographicHash::hash(PIN, QCryptographicHash::Sha1) == QCryptographi-
cHash::hash(temp->pin_temp.mid(0, tsm), QCryptographicHash::Sha1))
 {
 return true;
 }
 }
 }
 }
 return false;
}
```

Can_blocks.h

```
#ifndef CAN_BLOCKS_H
#define CAN_BLOCKS_H

#include <QString>
#include <QByteArray>
#include <QTime>
#include <QCryptographicHash>
#include <QDebug>
#include <QRandomGenerator>
#include <QSignalMapper>
#include <QInputDialog>
#include <QMessageBox>
#include <QLine>
#include <QRadioButton>
#include <QPlainTextEdit>

#include "qaesencryption.h"

#include "auth_can.h"

class arbitrage_block;
class multipurp_block;
```

```
class ignition_key_block: public QObject
{
 Q_OBJECT
public:
 QByteArray last_key_ignition_init_time;
 QByteArray my_key_id;
 void init_key();
 QString getKeyId(){return my_key_id;}
signals:
 void key_inited();
};

class abstact_block: public QObject
{
 Q_OBJECT
protected:
 int my_id;
 QByteArray AESKey;
 bool isIgnitionKeyInited;
 ignition_key_block* ignition_block;

 QList<QByteArray> token_base;
public:
 QByteArray newcome_data;
 abstact_block(): isIgnitionKeyInited(false){}
 QByteArray get_timestamp();
 void new_packet(QByteArray packet){newcome_data = packet;}
public slots:
 virtual void ign_key_inited();
 void ign_key_detached(){isIgnitionKeyInited=false;}
 QByteArray gen_key();
};

class multipurp_block: public abstact_block
{
 Q_OBJECT
public:
 multipurp_block(ignition_key_block* ignition_blk){ignition_block= igni-
tion_blk;}
 QByteArray gen_packet(int id, quint32 Na, QByteArray data);
 virtual void send_message(arbitrage_block* send_to_id, QByteArray data);
signals:
 void dataSended();
};

class attacked_block: public multipurp_block
{
 Q_OBJECT
public:
 attacked_block(ignition_key_block* ignition_blk): multipurp_block(igni-
tion_blk){ignition_block= ignition_blk;}
 void send_message(arbitrage_block* send_to_id, int Na, QByteArray data);
};

class arbitrage_block: public abstact_block
{
 Q_OBJECT
 QWidget *pwgt;
 auth_server* au_server;
 int current_multipurpose_block;
 QPlainTextEdit* log_widget;
 bool authed;
public:
```

```
 arbitrage_block(auth_server* au_serv, QWidget* parent){au_server = au_serv;
pwgt=parent; authed=false;}
 void setIgnitionBlockPointer(ignition_key_block* ignition_blk){ignition_block=
ignition_blk;}
 void blockWantsToAuth(int id);
 QByteArray requestCoordsFromGPSblock(){return "41°24'12.2N 2°10'26.5E";}
 QByteArray requestVIN(){return "2HSFMAER7XC025225";}
 QByteArray requestKeylessID(){return "2G1WB5E38E1148406";}
 bool isAuthed(){return authed;}
public slots:
 void newData();
};

struct message_new_can
{
 int resp_id;
 QByteArray timestamp;
 QByteArray Na;
 QByteArray data;
 QByteArray hash;
};

#endif // CAN_BLOCKS_H
```

Can_blocks.cpp

```
#include "can_blocks.h"


QByteArray abstact_block::gen_key()
{
 if(isIgnitionKeyInited==true)
 {
 QCryptographicHash key_hash(QCryptographicHash::Sha1);
 QByteArray temp_key = ignition_block->last_key_ignition_init_time+igni-
tion_block->my_key_id;
 AESKey = key_hash.hash(key_hash.hash(temp_key, QCryptographicHash::Sha1),
QCryptographicHash::Sha1);
 AESKey.resize(16);
 qDebug() << "A:" << AESKey;
 return AESKey;
 }
 else {
 qDebug() << "Key not yet initialized!";
 return QByteArray();
 }
}

QByteArray abstact_block::get_timestamp()
{
 QByteArray date = QByteArray::number(QDate::currentDate().toJulianDay());
 QByteArray time = QByteArray::number(QTime::currentTime().msecsSin-
ceStartOfDay());
 date = date+time;
 qChecksum(date.toStdString().c_str(), date.length());
 date = date+"[]"+QByteArray::number(qChecksum(date.toStdString().c_str(),
date.length()))+"[]"+time;
 date.resize(15);
 // qDebug() << QCryptographicHash::hash(date, QCryptographi-
cHash::Sha1).length();
 date = QCryptographicHash::hash(date, QCryptographicHash::Sha1);
 date.resize(15);
```

```
 QAESEncryption encryption(QAESEncryption::AES_128, QAESEncryption::ECB);
 qDebug() << AESKey;
 QByteArray tt = encryption.encode(date, AESKey);
 return tt;
}

void abstact_block::ign_key_inited(){
 isIgnitionKeyInited=true;
}

QByteArray multipurp_block::gen_packet(int id, quint32 Na, QByteArray data)
{
 QByteArray out_data = QByteArray::number(id)+"[]"+get_timestamp()+"[]"+QByteAr-
ray::number(Na)+"[]"+data+"[]";//int conversion!!!
 out_data+=QCryptographicHash::hash(out_data, QCryptographicHash::Sha1).mid(0,
16);
 out_data+="~~";
 token_base.push_back(QByteArray::number(Na));
 qDebug() << "Not crypted:" << out_data;
 QAESEncryption encryption(QAESEncryption::AES_128, QAESEncryption::ECB);
 QByteArray encodedData = encryption.encode(out_data, AESKey);
 qDebug() << "Crypted:" << encodedData;
 return encodedData;

}

 void multipurp_block::send_message(arbitrage_block* send_to_id, QByteArray
data)
 {
 gen_key();
 send_to_id->new_packet(gen_packet(99, QRandomGenerator::global()->generate(),
data));//static id QRandomGenerator::global()->generate()
 emit dataSended();
 }


 void ignition_key_block::init_key()
 {
 /*for(int i=0; i<8; i++)
 {
 my_key_id.push_back(rand()%255);
 }*/
 my_key_id = "12345678";
 last_key_ignition_init_time = QTime::currentTime().toString().toUtf8();
 emit key_inited();
 }

 void arbitrage_block::blockWantsToAuth(int id)
 {
 current_multipurpose_block = id;
 QPlainTextEdit* server_stats;
 for(auto temp: pwgt->children())
 {
 if(temp->objectName() == "server")
 {
 server_stats = static_cast<QPlainTextEdit*>(temp->find-
Child<QPlainTextEdit*>("te_stat_server"));
 }
 }
 for(auto temp: pwgt->children())
 {
 if(temp->objectName() == "smartph")
 {
 au_server->setLogWidget(temp->findChild<QPlainTextEdit*>("te_msg_phone"));
```

```
 }
 }
 server_stats->setPlainText(server_stats->placeholderText()+QByteArray("Auth re-
quest from block with id: "+QByteArray::number(id)));
 if(au_server->new_connection_request("187.90.0.101", QCryptographi-
cHash::hash(QByteArray("2ima12345"), QCryptographicHash::Sha1)))
 {
 for(auto temp: pwgt->children())
 {
 if(temp->objectName() == "line_serv_toab")
 {
 static_cast<QWidget*>(temp)->setStyleSheet("color: green");
 server_stats->setPlainText(server_stats->placeholderText()+QByteArray("Auth ok,
arbitration server will receive now PIN code "));
 }

 if(temp->objectName() == "ab_block")
 {
 log_widget = static_cast<QWidget*>(temp)->find-
Child<QPlainTextEdit*>("te_stat_can");
 }

 if(temp->objectName() == "line_serv_phone")
 {
 static_cast<QWidget*>(temp)->setStyleSheet("color: green");
 server_stats->setPlainText(server_stats->placeholderText()+QByteArray("Pin
sended to your smartphone, you can it out using your app"));
 }
 }
 QByteArray pin_input = QInputDialog::getText(pwgt, "Enter pin from mobile auth
app", "Pin code:", QLineEdit::EchoMode::Password).toUtf8();
 QMessageBox msg;

 if(au_server->check_pin(requestVIN(), requestCoordsFromGPSblock(), requestKey-
lessID(), pin_input))//!!!!!!!!!!!!!!!!!!!!!!
 {
 msg.setWindowTitle("Success!");
 msg.setText("Auth OK");
 for(auto temp: pwgt->children())
 {
 if(temp->objectName() == "line_mp_toab")
 {
 static_cast<QWidget*>(temp)->setStyleSheet("color: green");
 }
 if(temp->objectName() == "rb_serv_auth_mp")
 {
 static_cast<QRadioButton*>(temp)->setChecked(true);
 }
 // ab_block
 if(temp->objectName() == "ab_block")// if(temp->objectName() ==
"rb_serv_auth_ab")
 {
 static_cast<QRadioButton*>(temp->findChild<QRadioButton*>("rb_serv_auth_ab"))-
>setChecked(true);
 }
 if(temp->objectName() == "mp_block")// if(temp->objectName() == "mp_block")
 {
 static_cast<QRadioButton*>(temp->findChild<QRadioButton*>("rb_serv_auth_mp"))-
>setChecked(true);
 }
 server_stats->setPlainText(server_stats->placeholderText()+QByteArray("Blocks
Successully authentifcated, keys are avalable now!"));
 authed = true;
 }
```

```
 }
 else {
 msg.setWindowTitle("Error!");
 msg.setText("Auth failed");
 for(auto temp: pwgt->children())
 {
 if(temp->objectName() == "line_mp_toab")
 {
 static_cast<QWidget*>(temp)->setStyleSheet("color: red");
 }
 }
 }
 msg.exec();
 }
 }

 void arbitrage_block::newData()
 {
 gen_key();
 QAESEncryption encryption(QAESEncryption::AES_128, QAESEncryption::ECB);
 QByteArray decodedData = encryption.decode(newcome_data, AESKey);
 decodedData.resize(decodedData.indexOf("~~"));//deprc
 message_new_can new_msg;
 QList<QByteArray> data_msg;
 int index_one = 0;
 for(int i=0; i<6; i++)
 {
 int index_two = decodedData.indexOf("[]", 0);
 QByteArray out = decodedData.mid(0, index_two);
 decodedData = decodedData.mid(index_two+2, decodedData.length()-index_two);
 data_msg.push_back(out);
 index_one = index_two;
 }
 qDebug() << data_msg;
 new_msg.resp_id = data_msg.at(0).toInt();
 log_widget->setPlainText("New data received from block "+QByteArray::num-
ber(new_msg.resp_id));
 new_msg.timestamp = data_msg.at(1);
 new_msg.Na = data_msg.at(2);
 new_msg.data = data_msg.at(3);
 qDebug() << "rid" << new_msg.resp_id;
 qDebug() << "timestamp" << new_msg.timestamp;
 qDebug() << "Na" << new_msg.Na;
 qDebug() << "data" << new_msg.data;
 QByteArray test_string = QByteArray::num-
ber(new_msg.resp_id)+"[]"+new_msg.timestamp+"[]"+new_msg.Na+"[]"+new_msg.data+"[
]";
 QByteArray hash_nw = QCryptographicHash::hash(test_string, QCryptographi-
cHash::Sha1);
 hash_nw.resize(16);
 qDebug() << "Hash: " << hash_nw << " : " << data_msg.at(4);
 Sleep(1000);

 if(hash_nw == data_msg.at(4) && (token_base.indexOf(new_msg.Na)<0))
 {
 qDebug() << "OK!!";
 log_widget->setPlainText("Message from block "+QByteArray::num-
ber(new_msg.resp_id)+" validated and proceeded to needed block!");
 token_base.push_back(new_msg.Na);
 }
 else
 {
 for(auto temp: pwgt->children())
 {
```

```
 if(temp->objectName() == "ab_block")
 {
 log_widget = static_cast<QWidget*>(temp)->find-
Child<QPlainTextEdit*>("te_stat_can");
 log_widget->setPlainText("Error! Either message corrupted or replay attack was
secured!");
 }
 }
 QMessageBox msg;
 msg.setText("Error! Either message corrupted or replay attack was secured!");
 msg.exec();
 }
 qDebug() << "MMMM: " << token_base << " : " << new_msg.Na;
 }


 void attacked_block::send_message(arbitrage_block* send_to_id, int Na, QByteAr-
ray data)
 {
 gen_key();
 send_to_id->new_packet(gen_packet(99, Na, data));//static id
 emit dataSended();
 }
```

Parent.h

```
#ifndef PARENT_H
#define PARENT_H
#include <QWidget>
#include "auth_can.h"
#include "can_blocks.h"

namespace Ui {
class parent;
}
class parent : public QWidget
{
 Q_OBJECT
public:
 explicit parent(QWidget *parent = nullptr);
 ~parent();
private slots:
 void on_bn_keyless_on_clicked();
 void on_bn_multipurp_clicked();
 void on_bn_add_user_clicked();
 void on_bn_arb_clicked();
 void on_bn_get_gps_2_clicked();
 void on_bn_get_keyless_2_clicked();
 void on_bn_multipurp_2_clicked();
 void on_bn_multipurp_4_clicked();
 void on_bn_multipurp_5_clicked();
private:
 Ui::parent *ui;
 ignition_key_block* ignition_block;
 arbitrage_block* abl;
 multipurp_block* mpbl;
 auth_server* au_server;
};

#endif // PARENT_H
```

Parent.cpp

```
#include "parent.h"
```

```
#include "ui_parent.h"

parent::parent(QWidget *parent) :
 QWidget(parent),
 ui(new Ui::parent)
{
 ui->setupUi(this);
 ignition_block = new ignition_key_block;
 au_server = new auth_server;
 abl = new arbitrage_block(au_server, this);
 abl->setIgnitionBlockPointer(ignition_block);
 // mpbl = new multipurp_block(0);
}
parent::~parent()
{
 delete ui;
}
void parent::on_bn_keyless_on_clicked()
{
 ignition_block->init_key();
 ui->rb_keygo->setChecked(true);
 ui->rb_keygo_mp->setChecked(true);
 ui->rb_keygo_ab->setChecked(true);
 ui->bn_multipurp->setEnabled(true);
 ui->bn_multipurp_2->setEnabled(true);
}
void parent::on_bn_multipurp_clicked()
{
 abl->blockWantsToAuth(99);
}
void parent::on_bn_add_user_clicked()
{
 au_server->add_user(1, ui->le_name->text().toUtf8(), ui->le_pwd-
>text().toUtf8(), abl->requestVIN(), abl->requestCoordsFromGPSblock(), abl->re-
questKeylessID());
 ui->te_stat_server->setPlainText("New user "+ui->le_name->text()+" added");
}
void parent::on_bn_arb_clicked()
{

}
void parent::on_bn_get_gps_2_clicked()
{
 QMessageBox msg;
 msg.setText(QString::fromStdString(abl->requestCoords-
FromGPSblock().toStdString()));
 msg.setWindowTitle("Current GPS position of your car:");
 msg.exec();
}
void parent::on_bn_get_keyless_2_clicked()
{
 QMessageBox msg;
 msg.setText(QString::fromStdString(abl->requestKeylessID().toStdString()));
 msg.setWindowTitle("Keyless KeyID of your car:");
 msg.exec();
}
void parent::on_bn_multipurp_2_clicked()
{
 //abl->gen_key();
 if(abl->isAuthed())
 {
 mpbl = new multipurp_block(ignition_block);
 mpbl->gen_key();
 connect(mpbl, SIGNAL(dataSended()), abl, SLOT(gen_key()));
```

```
 connect(mpbl, SIGNAL(dataSended()), abl, SLOT(newData()));
 abl->ign_key_inited();
 mpbl->ign_key_inited();
 mpbl->send_message(abl, "RPM:1000");
 }
 else {
 QMessageBox msg;
 msg.setText("Alarm: suspected message sent without auth!");
 msg.setWindowTitle("Alert!");
 msg.exec();
 }


}
void parent::on_bn_multipurp_4_clicked()
{
 abl->new_packet(abl->newcome_data);
 emit mpbl->dataSended();
}

void parent::on_bn_multipurp_5_clicked()
{
 abl->newcome_data[5] = 7;
 abl->new_packet(abl->newcome_data);
 emit mpbl->dataSended();
}
```

ДОДАТОК Б

ВИКОРИСТАННЯ ДЕЦЕНТРАЛІЗОВАНИХ ТЕХНОЛОГІЙ ДЛЯ ЗАСОБІВ АДЕНТИФІКАЦІЇ АВТОМОБІЛЯ

І.Д. ГОРБЕНКО, д-р техн. наук,, Д. О. ФЕСЕНКО

Сучасними систем захисту автомобілів все більше цікавляться зловмисники, автомобілі стають більш технологічними, це в свою чергу відкриває нові можливості компрометації роботи вузів та систем автомобіля, тому до систем безпеки пред'являються все більш жорсткі вимоги щодо забезпечення ефективності та безпечності їх функціонування. Розглянені сучасні системи захисту від незаконного заволодіння автотранспортом, більш відомі всім як «сигналізація» намагаються стримувати атаки зловмисників, але в свою чергу можуть привносити додаткові бекдори для зловмисників зовсім ненавмисно, наприклад додаючи цікаву функцію в систему автомобіля, а згодом ця функція може мати двояке значення через проблеми з системою авдентифікації. Тож, виходячи з цього, системи безпеки автомобіля повинні мати найвищий рівень безпеки авдентифікації, для реалізації якого пропонується використання децентралізованої мережі блокчейн з вузлами для кожного автомобіля, що авдентифікують користувача групово, це дозволить відійти від стандартної клієнт-серверної архітектури, що є не достатньо захищеною. Основними шляхами вирішення зазначеної проблеми є побудування комплексної системи безпеки, що в свою чергу включає покращений та надійний захід авдентифікації на основі децентралізованої мережі блокчейн та двох комплексних схем оновлення системи передачі критичних даних автомобіля – мережі CAN. Використання даних систем дозволить поліпшити показники захищеності системи авдентифікації та інформації що курсує між блоками критичної важливості, що покращить безпечність автомобіля як від угонів, так і від можливостей створення зловмисниками аварійних ситуацій дистанційно.

Ключові слова: Blockchain, Атака, Вразливість, Децентралізація, Підміна, Automotive security.

В сучасному світі, коли використання електронних систем авдентифікації є дуже розповсюдженою технологією в багатьох сферах життя суспільства, деякі з них знаходяться у вжитку вже достатньо великий час, через це вони з часом втрачають свою захищеність та стійкість до атак. Розглянемо детальніше сучасну та критичну до безпеки проблему захищеності сучасних автомобільних систем. Основною проблемою захищеності автомобільних систем є недостатня захищеність системи авдентифікації, що дозволяє власникам отримувати доступ до транспортного засобу (фізичний доступ, доступ до програмних елементів для спрощеного використання), вразливості яких можуть привести до отримання доступу зловмисниками.

Основною проблемою систем безпеки є те, що вони побудовані на вже відомих світу технологіях, але дещо спрощених, наприклад використовуються криптографічні системи з зменшеним розміром блоків та ключів, крім того навіть дорогі та складні системи будуються на основі одного центрального серверу, з'єднання з яким може буті скомпрометовано. Було розглянено можливість використання децентралізованих технологій для покращення рівня захищеності засобів авдентифікації, визначено що найращим рішенням є використання технології блокчейн для покращення рівня довіри та мінімізації можливостей з перехоплення та модифікації даних.

Використання технології блокчейн в системі авдентифікації автомобіля дозволить розширити можливості безпеки авдентифікації для доступу до автомобіля, роблячи цей процес в той самий час простішим для користувача, бо всі дії можна виконувати за допомогою смартфону, що працює на широко відомій платформі, але ж і в одночас підвищується рівень безпеки та довіри, в одночас не вимагає додаткових грошових вкладень, є легким до налаштування та використання та є гарно захищеним як від поствантових атак так і від інших видів атак, що зараз є дуже загрозливими для безпеки сучасної автоіндустрії. Також великим плюсом, що Apple буде співпрацювати з виробниками автомобілів для CarKey, та є велика вірогідність виконання версії заводського варіанту, що зробить цю технологію ще більш доступною, простою для потенційних користувачів. В одночас системи безпеки автомобіля будуть мати гарний рівень безпеки авдентифікації, для реалізації якої використовується децентралізована мережі блокчейн з вузлами для кожного автомобіля, що авдентифікують користувача групово та 51% інших вузлів має підтвердити запит на авдентифікацію, в інакшому випадку авдентифікація буде неможлива і доступу до транспортного засобу надано не буде. Таким чином вирішуються проблем авдентифікації та безпечності передачі даних між блоками між собою, що перетворює розроблену систему в покращений та надійний захід авдентифікації на основі децентралізованої мережі блокчейн та двох комплексних оновленних схем системи передачі критичних даних всередині автомобіля.

Список літератури:

[1] 1.NISTIR 8202. Blockchain Technology Overview / NIST // NISTIR. – Gaithersburg, US Department of Commerce, 2017. – С.1-26.

[2] 2. Brown. Vehicle Security Systems: Build Your Own Alarm and Protection Systems,: Newnes, 160, 1996. – С. 7 – 155.

[3] 3. Knight A., Hacking Connected Cars: Tactics, Techniques, and Procedures / Alissa Knight – K. : Information Systems, 2019 – С. 5 – 250.

[4] 4. Car Security 101 [Електронний ресурс]. – Режим доступу: www/ URL: https://www.lifewire.com/car-security-101-534872 – 01.05.2020 р.

УДК 004;681.51;007.5 Фесенко Д.О. (ХНУРЄ)
Горбенко І.Д(ХНУРЄ)

ВИКОРИСТАННЯ ДЕЦЕНТРАЛІЗОВАНИХ ТЕХНОЛОГІЙ ДЛЯ ЗАСОБІВ АДЕНТИФІКАЦІЇ АВТОМОБІЛЯ

Розгляд проблем систем авдентифікації автомобільних систем та можливості покращення рівня безпеки шляхом використання децентралізованих систем. *Сучасними систем захисту автомобілів все більше цікавляться зловмисники, автомобілі стають більш технологічними, це в свою чергу відкриває нові можливості компрометації роботи вузів та систем автомобіля, тому до систем безпеки пред'являються все більш жорсткі вимоги щодо забезпечення ефективності та безпечності їх функціонування. Розглянені сучасні системи захисту від незаконного заволодіння автотранспортом, більш відомі всім як «сигналізація» намагаються стримувати атаки зловмисників, але в свою чергу можуть привносити додаткові бекдори для зловмисників зовсім ненавмисно, наприклад додаючи цікаву функцію в систему автомобіля, а згодом ця функція може мати двояке значення через проблеми з системою авдентифікації. Тож, виходячи з цього, системи безпеки автомобіля повинні мати найвищий рівень безпеки авдентифікації, для реалізації якого пропонується використання децентралізованої мережі блокчейн з вузлами для кожного автомобіля, що авдентифікують користувача групово, це дозволить відійти від стандартної клієнт-серверної архітектури, що є не достатньо захищеною. Основними шляхами вирішення зазначеної проблеми є побудування комплексної системи безпеки, що в свою чергу включає покращений та надійний захід авдентифікації на основі децентралізованої мережі блокчейн та двох комплексних схем оновлення системи передачі критичних даних автомобіля – мережі CAN. Використання даних систем дозволить поліпшити показники захищеності системи авдентифікації та інформації що курсує між блоками критичної важливості, що покращить безпечність автомобіля як від угонів, так і від можливостей створення зловмисниками аварійних ситуацій дистанційно.*

Виділені основні проблеми систем авдентифікації автомобіля та запропоновано варіанти їх рішення.

Ключові слова: Blockchain, Атака, Вразливість, Децентралізація, Підміна, Automotive security.

Фесенко Д.А., Горбенко И.Д. Рассмотрение проблем систем авдентификации автомобильных систем и возможности улучшения уровня безопасности путем использования децентрализованных систем. *Современными систем защиты автомобилей все больше интересуются злоумышленники, автомобили становятся более технологичными, это в свою очередь открывает новые возможности компрометации работы вузов и систем автомобиля, поэтому к системам безопасности предъявляются все более жесткие требования по обеспечению эффективности и безопасности их функционирования. Рассмотрены современные системы защиты от незаконного завладения автотранспортом, более известные всем как «сигнализация», которые пытаются сдерживать атаки злоумышленников, но в свою очередь могут привносить дополнительные бэкдоры для злоумышленников совершенно непреднамеренно, например добавляя интересную функцию в систему автомобиля, а затем эта функция может иметь двоякое значение из-за проблем с системой аутентификации. Поэтому, исходя из этого, системы безопасности автомобиля должны иметь высокий уровень безопасности аутентификации, для реализации которого предлагается использование децентрализованной сети блокчейн с узлами для каждого автомобиля, аутентифицируют пользователя в группе, что позволит отойти от стандартной клиент-серверной архитектуры, является недостаточно защищенной . Основными путями решения данной проблемы является построение комплексной системы безопасности, что в свою очередь включает улучшенный и надежный мероприятие аутентификации на основе децентрализованной сети блокчейн и двух комплексных схем обновления системы передачи критических данных автомобиля – сети CAN. Использование данных систем позволит улучшить показатели защищенности системы аутентификации и информации курсирующий между блоками критической важности, улучшит безопасность автомобиля как от угонов, так и от возможностей создания злоумышленниками аварийных ситуаций дистанционно.*

Выделены основные проблемы систем аутентификации автомобиля и предложены варианты их решения.

Ключевые слова: Blockchain, Атака, Уязвимость, Децентрализация, Підміна, Automotive security.

D. Fesenko, I. Gorbenko. Automotive authentication systems problems and possibilities to elevate security level by using blockchain technologies. *Intruders are becoming more and more interested in modern car protection systems, cars are becoming more technological, which in turn opens up new opportunities for compromising the work of universities and car systems, so security systems are subject to increasingly stringent requirements to ensure efficiency and*

safety. Modern systems of protection against illegal seizure of vehicles, better known as "alarm" try to deter attacks by intruders, but in turn can bring additional backdoors for intruders completely unintentionally, for example by adding an interesting feature to the car system, and then this feature may have a dual meaning due to problems with the authentication system. Therefore, based on this, car security systems must have the highest level of authentication security, which requires the use of a decentralized blockchain network with nodes for each car, identifying the user in groups, this will move away from the standard client-server architecture, which is not sufficiently secure . The main ways to solve this problem are to build a comprehensive security system, which in turn includes an improved and reliable authentication measure based on a decentralized blockchain network and two comprehensive schemes to update the critical data transmission system of the car – CAN network. The use of these systems will improve the security of the identification system and information flowing between critical units, which will improve the safety of the car from theft, as well as from the ability of attackers to create emergencies remotely.

The main problems of car identification systems are highlighted and variants of their solution are offered.

Keywords: Blockchain, attack, vulnerability, decentralization, Automotive security.

Постановка завдання в загальному вигляді. В сучасному світі, коли використання електронних систем авдентифікації є дуже розповсюдженою технологією в багатьох сферах життя суспільства, деякі з них знаходяться у вжитку вже достатньо великий час, через це вони з часом втрачають свою захищеність та стійкість до атак. Розглянемо детальніше сучасну та критичну до безпеки проблему захищеності сучасних автомобільних систем. Основною проблемою захищеності автомобільних систем є недостатня захищеність системи авдентифікації, що дозволяє власникам отримувати доступ до транспортного засобу (фізичний доступ, доступ до програмних елементів для спрощеного використання), вразливості яких можуть привести до отримання доступу зловмисниками.

Основною проблемою систем безпеки є те, що вони побудовані на вже відомих світу технологіях, але дещо спрощених, наприклад використовуються криптографічні системи з зменшеним розміром блоків та ключів, крім того навіть дорогі та складні системи будуються на основі одного центрального серверу, з'єднання з яким може буті скомпрометовано. Було розглянено можливість використання децентралізованих технологій для покращення рівня захищеності засобів авдентифікації, визначено що найращим рішенням є використання технології блокчейн для покращення рівня довіри та мінімізації можливостей з перехоплення та модифікації даних[2].

Використання технології блокчейн в системі авдентифікації автомобіля дозволить розширити можливості безпеки авдентифікації для доступу до автомобіля, роблячи цей процес в той самий час простішим для користувача, бо всі дії можна виконувати за допомогою смартфону, що працює на широко відомій платформі, але ж і в одночас підвищується рівень безпеки та довіри, в одночас не вимагає додаткових грошових вкладень, є легким до налаштування та використання та є гарно захищеним як від поствантових атак так і від інших видів атак, що зараз є дуже загрозливими для безпеки сучасної автоіндустрії. Також великим плюсом, що Apple буде співпрацювати з виробниками автомобілів для CarKey, та є велика вірогідність виконання версії заводського варіанту, що зробить цю технологію ще більш доступною, простою для потенційних користувачів. В одночас системи безпеки автомобіля будуть мати гарний рівень безпеки авдентифікації, для реалізації якої використовується децентралізована мережі блокчейн з вузлами для кожного автомобіля, що авдентифікують користувача групово та 51% інших вузлів має підтвердити запит на авдентифікацію, в інакшому випадку авдентифікація буде неможлива і доступу до транспортного засобу надано не буде. Таким чином вирішуються проблем авдентифікації та безпечності передачі даних між блоками між собою, що перетворює розроблену систему в покращений та більш надійний захід авдентифікації на основі децентралізованої мережі блокчейн та двох комплексних оновленних схем системи передачі критичних даних всередині автомобіля.

Метою статті є аналіз захищеності систем авдентифікації автомобіля та покращеннярівня захищеності системи авдентифікації автомобіля.

Виклад основного матеріалу дослідження. Об’єктом інформаційної діяльності є абстрактний транспортний засіб, що використовує для захисту від несанкціонованого доступу систему безпеки високого класу з постійним моніторингом стану захищеності транспортного засобу за допомогою GSM з’язку, використовуючи який дані відправляються на центральний сервер, до якого може отримати доступ власник для перегляду стану захищеності автомобіля у вигляді веб-сервісу, або з використанням програмного забезпечення, що встановлюється на смартфон.

Сучасні автомобілі є складною мережею, що складаються з багатьох незалежних вбудованих обчислювальних систем, з'єднаних по мережі. Наприклад, автомобіль, що розглядається складається з понад 200 мікропроцесорів, на яких працює до 65 мільйонів рядків коду, що робить його одним з найскладніших програмних систем. Як і у випадку з будь-якою складною системою програмного забезпечення, збереження безпеки є актуальною справою[1].

Вимоги безпеки, що висуваються до компонентів системи авдентифікації – це вимоги цілісності, доступності та достовірності, оскільки для правильної роботи всіх системи безпеки автомобіля необхідно щоб інформація, що курсує в ІС залишалася завжди доступною і цілісною та була не скомпрометованою, бо це є дуже важливим аспектом для правильного керування параметрами безпеки автомобіля та взагалі правильного функціонування всіх його систем.

В даній мережі циркулюють критичні дані у вигляді пакетів, що курсують від одного компонента системи до іншого, таким чином компоненти зв’язані між собою та можуть взаємодіяти, що дозволяє автомобілю рухатися та ефективно використовувати системи керування та безпеки.

Інформаційна система являє собою мікроконтролер з встановленими додатковими модулями зв’язку з мережею GSM та інтреграції пристрою до мережі CAN. Система побудована на основі клієнт-серверної архітектури, що складається з мікроконтролера з доступом до мережі інтернет, сервера, що виконує роль системи зберігання даних, що курсують в системі, системи моніторингу даних для виявлення проблем безпеки і загроз, що у випадку позаштатної ситуації відправляє повідомлення на смартфон власника транспортного засобу У системі безпеки, що встановлений на об’єкт інформаційної діяльності в якості криптографічного захисту інформації використовується KeeLoq, блоковий шифр, заснований на програмному компоненті "NLFSR", що являє собою регістр зсуву з нелінійним зворотним зв'язком. Односпрямований протокол передачі команди був розроблений Фредеріком Брувер, який є генеральним директором компанії Nanoteq Pty Ltd.

Алгоритм KeeLoq був розроблений в середині 80-х Джідеоном Куном з кремнієвої реалізацією Вілліема Смитта в Nanoteq Pty Ltd (підрозділ Південної Африки) і був проданий Microchip Technology, Inc. в 1995 році за 10 млн доларів. Алгоритм являє собою «плаваючий код», кодується і декодується за допомогою мікросхем NTQ105 / 106/115 / 125D / 129D і HCS2XX / 3XX / 4XX / 5XX.

Шифрування відбувається блоками по 32 біта з використанням 64 бітного ключа, один блок тексту шифрується за 528 раундів[2].

Об’єкти системи безпеки автомобіля, що підлягають захисту:

- Система зберігання даних авдентифікації, що знаходяться в блоках автомобіля

- Система зберігання даних авдентифікації, що знаходяться на пристрої власника/того кому він делегує свої обов'язки

- Дані авдентифікації, дані власника, інформація про машину та її статут, що передаються за допомогою мережі інтернет

Засоби захисту інформації, що пропонуються:

Пропонується використання децентралізованої системи блокчейн на заміну клієнт-серверній архітектурі, що дозволяє:

– не використовувати сервер для зберігання даних про автомобіль, про стан його безпеки, геопозицію та даних авдетифікації, що дозволяє ускладнити доступ до перехоплення та модифікації даних на сервері при отриманні неправомірного доступу, оскільки такими серверами в системі блокчейн можна назвати ноди мережі, які й мають виконувати весь функціонал серверів з клієнт-серверної архітектури.

- важко відстежити ноди мережі щоб отримати до них неправомірний доступ та мати можливість впливати роботу системи з боку підтвердження невалідних дій від зловмисників та виконати DDoS атаку, бо дані про мережу не відомі.

- планується встановлена кількість нодів в мережі в кількості 10, що будуть працювати в будь-якому разі, оскільки будуть підтримуватися в робочому стані автоматичним менеджером нод, який має слідкувати за станом мережі та правильністю роботи системи, таким чином забезпечується постійний пул арбітруючих пристроїв, що повинно забезпечити безпечну роботи мережі блокчейн.

Захист об'єктів системи безпеки автомобіля за допомогою запропонованих засобів захисту інформації:

Криптографічний захист даних, що дозволяє вирішити проблему щодо атак на зміну цілісності даних, унеможливлює підміну даних та перехоплення важливих даних про роботу системи:

- шифрування даних, що зберігаються в блоці безпеки автомобіля

- шифрування даних, що передаються в мережі

- валідація даних, що передаються в мережі за допомогою геш-значень

- використання захищених каналів зв'язку

З використанням мережі блокчейн в якості мережі для передачі даних стає можливим використання механізмів керування доступом, що дозволяє ефективно захищати інформацію, що курсує в системі безпеки автомобіля, даних ідентифікації, що передаються та їх верифікації за рахунок відсутності сервера, що може бути скомпрометованим та надійних засобів криптографічного захисту інформації.

В якості міток доступу використовується мобільний пристрій з достатнім рівнем довіри (в огляду на те, що криптоконтейнер, що пропонується використовується для зберігання даних банківських рахунків та іншої критичної інформації) з додатковим шифруванням блоковим симетричним «Калина», що є шифром державного стандарту та показує достатню стійкість для такої розробки.

У ході роботи була розглянута побудова захищеної системи авдентифікації для автомобільних систем, розглянена структура та будова комерційних пропозицій засобів безпеки для безпечної авдентифікації власника автомобіля. Визначені можливі загрози зі сторони зловмисників для системи авдентифікації автомобіля, найнебезпечніші з цих загроз –

модифікація даних, можливість проведення реплей атак, використання вразливих місць у коді мережі та зламування криптоалгоритмів. Була розроблена програмна реалізація тестової моделі систем безпеки з використанням функціоналу моделювання розглянутих в даній роботі видів атак.

Програмне моделювання роботи системи авдентифікації дозволяє розглянути всі єтапи роботи мережі системи, розглянути механізм генерації даних, їх синхронізації між собою, перевірки валідності даних та можливостей з їх модіфікації. Було розглянуто можливість впливу на дані авдентифікації, що передаються, за допомогою MITM атаки, що дозволило зрозуміти захищеність каналу передачі даних, метою атаки було скомпрометувати дані, що зберігалися в попередніх блоках, але за рахунок перевірки блоків з використанням геш-значень попередніх блоків, що зберігаються в наступних блоках в блокчейні. Були проведені тести атак на відключення вузлів авдентифікаії з мережі з використанням атаки DDOS, що повинна була перешкодити роботі мережі через виключення вузлу з мережі, але завдяки тому, що всі вузли були синхронізовані то атака була неефективною[3], атаки на отримання контролю над 51% блоків для нав'язування даних, що курсують в мережі, але навіть при невеликій кількості блоків, що тестувалися в цій атаці, вона стала ймовірною, коли зловмисником було отримано доступ до двох з трьох вузлів, та той отримав можливість робити нав'язування даних, але в той же час третій блок перестав потрібним чином працювати через те що не була пройдена перевірка валідності блоків, що не змогли засинхронізуватися з скомпрометованими блоками. Таким чином, можна визначити, що розроблена програмна модель системи авдентифікації автомобільної системи задовольняє вимогам, висунутим під час розгляду захисту від атак.

Були представлені вимоги до рівня безпеки для безпечного використання систем авдентифікації та пропозиції щодо досягнення відповідного рівня безпеки з використанням мережі блокчейн, розглянуті можливі загрози для системи безпеки при використанні в якості системи передачі даних мережу блокчейн.

За отриманими даними висунуті пропозиції щодо безпечного застосування технології блокчейн з огляду на розглянені можливі атаки на реалізації мережі блокчейн.

ЛІТЕРАТУРА

1) Lemke K. Embedded Security in Cars: Securing Current and Future
Automotive IT Applications / Kerstin Lemke – K. : Springer, 273. – С. 50 – 250.
Brown. Vehicle Security Systems: Build Your Own Alarm and Protection Systems,: Newnes, 160, 1996. – С. 7 – 155.
2) Knight A., Hacking Connected Cars: Tactics, Techniques, and Procedures / Alissa Knight – K. : Information Systems, 2019 – С. 5 – 250.
CAN bus [Електронний ресурс]. – Режим доступу: www/ URL:
https://en.wikipedia.org/wiki/CAN_bus – 22.04.2019 р.
3) Understanding CAN-Bus [Електронний ресурс]. – Режим доступу: www/ URL:
https://avscarsecurity.com/avs-car-security/choosing-a-car-alarm/understanding-can-bus-systems – 26.04.2019 р.
4) Car Security 101 [Електронний ресурс]. – Режим доступу: www/ URL:
https://www.lifewire.com/car-security-101-534872 – 01.05.2019 р.

Секція 2

ВИКОРИСТАННЯ ДЕЦЕНТРАЛІЗОВАНИХ ТЕХНОЛОГІЙ ДЛЯ ЗАСОБІВ АВДЕНТИФІКАЦІЇ АВТОМОБІЛЯ

Фесенко Д.О. , Горбенко І. Д.

Харківський національний університет радіоелектроніки, Харків, Україна

Можливості обходу систем захисту все більше цікавлять зловмисників, зокрема можливості модифікації та підміни даних в них, на що вони витрачають багато ресурсів для відкриття нових можливостей для компрометації роботи систем, тому до систем безпеки пред'являються все більш жорсткі вимоги щодо забезпечення ефективності та безпечності їх функціонування. Розглянені сучасні системи захисту від незаконного заволодіння автотранспортом, більш відомі всім як «сигналізація» намагаються стримувати атаки зловмисників, але в свою чергу можуть привносити додаткові бекдори для зловмисників зовсім ненавмисно, наприклад додаючи цікаву функцію в систему автомобіля, а згодом ця функція може мати двояке значення через проблеми з системою авдентифікації[2]. Тож, виходячи з цього, системи безпеки автомобіля повинні мати найвищий рівень безпеки авдентифікації, для реалізації якого пропонується використання децентралізованої мережі блокчейн[1] з вузлами для кожного автомобіля, що авдентифікують користувача групово, це дозволить відійти від стандартної клієнт-серверної архітектури, що є не достатньо захищеною. Основними шляхами вирішення зазначеної проблеми є побудування комплексної системи безпеки, що в свою чергу включає покращений та надійний захід авдентифікації на основі децентралізованої мережі блокчейн та двох комплексних схем оновлення системи передачі критичних даних автомобіля – мережі CAN[2].

Метою доповіді є розгляд особливостей використання технології блокчейн в системі авдентифікації автомобіля дозволить розширити можливості безпеки авдентифікації для доступу до автомобіля, роблячи цей процес в той самий час простішим для користувача, бо всі дії можна виконувати за допомогою смартфону, що працює на широко відомій платформі, але ж і в одночас підвищується рівень безпеки та довіри, в одночас не вимагає додаткових грошових вкладень, є легким до налаштування та використання та є гарно захищеним як від поствантових атак так і від інших видів атак, що зараз є дуже загрозливими для безпеки сучасної автоіндустрії.

Список літератури

1. NISTIR 8202. Blockchain Technology Overview / NIST // NISTIR. – Gaithersburg, US Department of Commerce, 2017. – С.1-26..
2. Brown. Vehicle Security Systems: Build Your Own Alarm and Protection Systems,: Newnes, 160, 1996. – С. 7 – 155..

Горбенко Іван Дмитрович, д.т.н., проф., 067-714-22-05,
gorbenkoi@iit.kharkov.com
Фесенко Дмитро Олександрович, магістрант, 066-8630475, dmytro.fesenko@nure.ua

ДОДАТОК В. ІНСТРУКЦІЯ СИСТЕМНОМУ ПРОГРАМІСТУ ТА ІНСТРУКЦІЮ ОПЕРАТОРУ З ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ПРОГРАММА МОДЕЛЮВАННЯ СИСТЕМИ БЕЗПЕКИ АВТОМОБІЛЯ

Інструкція системному програмісту та оператору з застосування програмного забезпечення

2020

АННОТАЦІЯ

В даному програмному документі наведено керівництво системного програміста по налаштуванню і оператору з застосування програмного забезпечення по використанню програми «CarSecurityManager.exe», призначеної для демонстрації роботи системи безпеки автомобіля.

В даному програмному документі, в розділі «Загальні відомості про програму» вказані призначення і функції програми і відомості про технічні і програмні засоби, що забезпечують виконання даної програми, а також вимоги до персоналу.

У розділі «Структура програми» наведені відомості про структуру програми, її складові частини, про зв'язки між складовими частинами і про зв'язки з іншими програмами.

В даному програмному документі, в розділі «Налаштування програми» описано дій з налаштування програми на умови конкретного застосування (настройка на склад технічних та програмних засобів, вибір функцій і ін.).

ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПРОГРАМУ

Призначення програми

Для моделювання роботи схеми попередження вторгнення для мережі CAN створено програмний продукт який дозволяє протестувати роботу розробленого протоколу автентифікації, провести моделювання атак та виявити критичні вразливості, навчити користувачів налаштуванню та використанню системи на наочному прикладі.

Програма «CarSecurityManager.exe» працює під керуванням ОС Windows 8/10.

Функції програми

Попередження вторгнення для мережі CAN для тестування роботи протоколу автентифікації та захисту блоків та інформації що курсує в системі, провести моделювання атак для виявлення критичних вразливостей, навчити користувачів налаштуванню та використанню системи на наочному прикладі.

Програма “CarSecurityManager.exe” реалізує наступні функції:

1) Моделювання роботи сервера автентифікації;
2) Моделювання роботи мобільного телефону з програмою автентифікації;
3) Моделювання роботи блоків загального призначення;
4) Моделювання роботи арбітражного блоку;
5) Моделювання атак на протокол автентифікації.

Мінімальний склад технічних засобів

Мінімальний склад технічних засобів:

Параметр PC	Значення
ОЗУ	Більше 4 Гб
Відеопам’ять	512 Мб або вище
Вільне місце на жорсткому диску	Більше 100 Мбайт

Мінімальний склад програмних засобів

Системні програмні засоби, що використовуються програмою “CarSecurityManager.exe”, повинні бути представлені локалізованої версією операційної системи Windows 8/10.

Вимоги до персоналу (системному програміста)

Системний програміст повинен мати мінімум середню технічну освіту.

У перелік завдань, які виконуються системним програмістом, повинні входити:

а) завдання підтримки працездатності технічних засобів;

б) завдання установки (інсталяції) і підтримки працездатності системних програмних засобів – операційної системи;

в) завдання установки (інсталяції) і підтримки працездатності програми “ CarSecurityManager.exe”.

СТРУКТУРА ПРОГРАМИ

Відомості про структуру програми

Програма «CarSecurityManager.exe» складається з 4 форм. Програма структурно складається з програмних частин “auth”, що являє собою програмну реалізацію модулю авдентифікації на основі технології блокчейн; “blocks”, що являє собою програмну реалізацію протоколу обміну даними про стан транспортного засобу, “kalyna_enc” – програмна реалізація шифрування Калина, та реалізації графічного інтерфейсу програми. До програми додається код програмного забезпечення у вигляді проекту з файлами за допомогою яких можна самостійно побудувати програмний продукт.

Відомості про складові частини програми

Програма «CarSecurityManager.exe» складається з програмного модулю та динамічних бібліотек для роботи програмного продукту.

НАЛАШТУВАННЯ ПРОГРАМИ

Налаштування на склад технічних засобів

Програма «CarSecurityManager.exe» не потребує будь-яких налаштувань на технічних засобах. Для всіх додаткових технічних засобів для роботи програми проводиться емуляція цих засобів тому їх фізичне налаштування не потрібно.

Налаштування на склад програмних засобів

Програма «CarSecurityManager.exe» не потребує додаткових програмних засобів.

Список необхідних файлів для роботи програми:

Файл	Розмір
CarSecurityManager.exe	20042 Кбайт
Qt5Core.dll	4 968 Кбайт
Qt5Gui.dll	5 216 Кбайт
Qt5Widgets.dll	4 423 Кбайт

№ док.	Позначення	Найменування	Дод. відом.
1		Текстові документи	
	ГЮІК.ХХХХХ11412Ст.16ПЗ	Пояснювальна записка	108 с.
2		Графічні документи	
		Презентаційний матеріал	12слайд.
3		Інші документи	
		Електронна версія ПЗ	1 шт.
		Відзив керівника	1 с.
		Рецензія	1 с.

Изм.	Лист	№ докум.	Підпис	Дата	ГЮІК. ХХХХХ1412Ст.16ВД		
Розроб.		Фесенко Д.О.			Використання децентралізованих систем для захисту автомобільних систем	Лит.	Лист 1 / Листів 1
Перевір.		Горбенко І.Д.				Т	
Н.контр.		Конєва Н.Ф.				ХНУРЕ Кафедра БІ	
Затверд.		Халімов Г.З.					