МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Satbayev University

Институт кибернетики и информационных технологий

Кафедра «Кибербезопасность, обработка и хранение информации»

Сысолов Артём Иванович

"Разработка криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования".

ДИПЛОМНАЯ РАБОТА

Специальность 5В100200 - "Системы информационной безопасности"

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Satbayev University

Институт кибернетики и информационных технологий

Кафедра «Кибербезопасность, обработка и хранение информации»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой КБОиХИ,
канд. техн. наук., доцент
_____ Сейлова Н.А.
“_____” _____ 2020 г.

ДИПЛОМНАЯ РАБОТА

На тему: "Разработка криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования"

по специальности 5В100200 - «Системы информационной безопасности»

Выполнил  Сысолов А.И.

Научный руководитель  лектор Ибраев Р. Б.

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Satbayev University

Институт кибернетики и информационных технологий

Кафедра Кибербезопасность, обработка и хранение информации

5В100200 - Системы информационной безопасности

УТВЕРЖДАЮ
Заведующий кафедрой КБОиХИ,
канд. техн. наук., доцент
_____ Сейлова Н.А.
“_____” _____ 2020 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающемуся *Сысолову Артёму Ивановичу*
Тема: *Разработка криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования*
Утверждена *приказом Ректора Университета № 762-б от “27” 01 2020 г.*
Срок сдачи законченной работы *“26” мая 2020 г.*
Исходные данные к дипломному проекту
Перечень подлежащих разработке в дипломном проекте вопросов:
а) механизм выработки и распределения общего ключа между двумя и более участниками информационного обмена;
б) механизм аутентификации участников информационного обмена;
в) построение иерархической структуры хранения криптографических ключей.
Перечень графического материала (с точным указанием обязательных чертежей): *представлены 8 слайдов в презентации проекта.*
Рекомендуемая основная литература: *из 16 наименований*

ГРАФИК

подготовки дипломной проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1 Аналитический обзор литературы	25.02.2020	Выполнено
2 Формирование общего секрета с помощью искусственных нейронных сетей	05.03.2020	Выполнено
3 Выбор параметров и обоснование структуры ИНС	10.04.2020	Выполнено
4 Алгоритм и специфика работы программной реализации	01.05.2020	Выполнено

Подписи

консультантов и нормоконтролера на законченный дипломный проект с указанием относящихся к ним разделов проекта

Наименования разделов	Консультанты, И.О.Ф. (уч. степень, звание)	Дата подписания	Подпись
Нормоконтролер	магистр тех. наук, лектор Зиро А.	04.05.2020	

Научный руководитель  Ибраев Р. Б.

Задание принял к исполнению обучающийся  Сысолов А.И

Дата «27»_января_2020 г.

Протокол анализа Отчета подобия

заведующего кафедрой / начальника структурного подразделения

Заведующий кафедрой / начальник структурного подразделения заявляет, что ознакомился(-ась) с полным отчетом подобия, который был сгенерирован Системой выявления и предотвращения плагиата в отношении работы:

Автор: Сысолов А.И.

Название: Разработка криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования

Координатор: Ренат Ибраев

Коэффициент подобия 1:9,9

Коэффициент подобия 2:1,5

Замена букв: 0

Интервалы: 0

Микропробелы: 0

Белые знаки: 0

После анализа отчета подобия заведующий кафедрой / начальник структурного подразделения констатирует следующее:

- ☑ обнаруженные в работе заимствования являются добросовестными и не обладают признаками плагиата. В связи с чем, признаю работу самостоятельной и допускаю ее к защите;

- ☐ обнаруженные в работе заимствования не обладают признаками плагиата, но их чрезмерное количество вызывает сомнения в отношении ценности работы по существу и отсутствием самостоятельности ее автора. В связи с чем, работа должна быть вновь отредактирована с целью ограничения заимствований;

- ☐ обнаруженные в работе заимствования являются недобросовестными и обладают признаками плагиата, или в ней содержатся преднамеренные искажения текста, указывающие на попытки сокрытия недобросовестных заимствований. В связи с чем, не допускаю работу к защите.

Обоснование:
……
……
……
……
……
……

……	……
Дата	*Подпись заведующего кафедрой / начальника структурного подразделения*

Окончательное решение в отношении допуска к защите, включая обоснование:

……
……
……
……
……
……

……
Дата

……
Подпись заведующего кафедрой / начальника структурного подразделения

Протокол анализа Отчета подобия Научным руководителем

Заявляю, что я ознакомился(-ась) с Полным отчетом подобия, который был сгенерирован Системой выявления и предотвращения плагиата в отношении работы:

Автор: Сысолов А.И.

Название: Разработка криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования

Координатор: Ренат Ибраев

Коэффициент подобия 1:9,9

Коэффициент подобия 2:1,5

Замена букв: 0

Интервалы: 0

Микропробелы: 0

Белые знаки: 0

После анализа Отчета подобия констатирую следующее:

- ☑ обнаруженные в работе заимствования являются добросовестными и не обладают признаками плагиата. В связи с чем, признаю работу самостоятельной и допускаю ее к защите;

- ☐ обнаруженные в работе заимствования не обладают признаками плагиата, но их чрезмерное количество вызывает сомнения в отношении ценности работы по существу и отсутствием самостоятельности ее автора. В связи с чем, работа должна быть вновь отредактирована с целью ограничения заимствований;

- ☐ обнаруженные в работе заимствования являются недобросовестными и обладают признаками плагиата, или в ней содержатся преднамеренные искажения текста, указывающие на попытки сокрытия недобросовестных заимствований. В связи с чем, не допускаю работу к защите.

Обоснование:
……
……
……
……
……

…… Ибраев Р. Б.

Дата *Подпись Научного руководителя*

АННОТАЦИЯ

Данная работа посвящена разработке криптографического протокола генерации, распределения и управления ключами для симметричных алгоритмов шифрования. Данный протокол призван обеспечить высокий уровень безопасности информации, защищаемой с применением симметричных алгоритмов шифрования, циркулирующей в информационных системах.

В качестве основы протокола использована нейросетевая модель выработки общего ключа между двумя участниками информационного обмена. В данной модели все арифметические вычисления являются простейшими матричными операциями, имеющими полиномиальную сложность, что обуславливает возможность применения разрабатываемого криптографического протокола в системах с маломощными оконечными устройствами.

Ключевые слова: *криптографический протокол, генерация, распределение, управление, ключ шифрования, симметричный алгоритм шифрования*

АҢДАТПА

Бұл жұмыс симметриялық шифрлау алгоритмдерінің кілттерін құруға, таратуға және басқаруға арналған криптографиялық хаттаманы жасауға арналған. Бұл хаттама ақпараттық жүйелерде ақпаратты беру кезінде қауіпсіздіктің жоғары деңгейін қамтамасыз етуге арналған.

Алгоритм ақпарат алмасудың екі қатысушысы арасында ортақ кілтті құруға арналған нейрондық желі моделіне негізделген. Бұл модельде барлық арифметикалық есептеулер тұрақты уақытта орындалатын қарапайым матрицалық операциялар болып табылады, бұл жоғарыда аталған алгоритмді төмен қуатты терминал құрылғыларымен жүйелерде қолдануға мүмкіндік береді.

Түйін сөздер: *криптографиялық хаттама, генерация, тарату, басқару, шифрлау кілті, симметриялық шифрлау алгоритмі*

ANNOTATION

This work is devoted to the development of a cryptographic protocol for generating, distributing and managing keys for symmetric encryption algorithms. This protocol is designed to provide a high level of security for information protected using symmetric encryption algorithms circulating in information systems.

As the basis of the protocol, a neural network model of generating a common key between two participants in information exchange was used. In this model, all arithmetic calculations are the simplest matrix operations having polynomial complexity, which makes it possible to use the developed cryptographic protocol in systems with low-power terminal devices.

Keywords: *cryptographic protocol, generation, distribution, management, encryption key, symmetric encryption algorithm*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире неотъемлемой частью повседневной жизни являются потоки конфиденциальных данных, денежные и иные транзакции. Сегодня, как никогда, информация нуждается в механизмах обеспечения ее конфиденциальности, целостности и доступности.

С одной стороны, научно-технический прогресс и рост вычислительных мощностей как раз-таки и способствуют бурному развитию интернет-технологий, безналичному расчету, передаче конфиденциальных данных по сети. Но, с другой стороны, вышеописанные факторы, в противовес, дают злоумышленникам инструмент, что в последующем может привести к нарушению базовых принципов информационной безопасности. Поэтому тематика секретности сегодня как никогда актуальна.

Стоит отметить, что наибольшую угрозу любой информационной системе представляет человеческий фактор. Данная угроза обусловлена прежде всего некомпетентностью как сотрудников, так и обычных пользователей. В связи с этим, *целью* настоящей дипломной работы является разработка криптографического протокола, который мог бы быть максимально независимым от участников информационного взаимодействия. Кроме того, основополагающей отличительной чертой данного протокола должна быть легковесность и мобильность.

Для достижения поставленной цели необходимо создать систему (протокол), элементы которой представлены в виде отдельных модулей и при этом дополняют друг друга. Таким образом создается возможность замены компонентов, без ущерба всей системе. Отсюда вытекают основополагающие *задачи* и проблемы, которые необходимо решить:

- механизм выработки и распределения общего ключа между двумя и более участниками информационного обмена
- механизм аутентификации участников информационного обмена
- построение иерархической структуры хранения криптографических ключей

Практическим значением, разрабатываемого в данной дипломной работе протокола является возможность его широкого применения, с участием устройств различной вычислительной мощности. Такая применимость протокола, прежде всего, обусловлена снятием нагрузки на узел, отвечающий за генерацию общего ключа, чему виной - замена операции возведения в степень на простейшие арифметические действия с матрицами.

1 Аналитический обзор литературы

1.1 Управление ключами в криптографической защите информационных систем

Для защиты данных от несанкционированного доступа используются различные методы. Одним из наиболее эффективных методов является криптографическая защита информации. Здесь применяются специальные алгоритмы преобразования открытого текста в зашифрованный текст, недоступный для чтения злоумышленнику без знания т.н. ключа шифрования.

Любая криптографическая система строится с учетом того, что алгоритм криптографического преобразования информации известен всем, включая злоумышленника. Также предполагается, что противник имеет инструменты и средства для проведения криптографического анализа. Таким образом криптографическая система, в общем случае, предполагает наличие секрета, известного только отправителю и получателю сообщения. Без знания этого секрета невозможно в разумные сроки восстановить исходный открытый текст. В криптографических системах такой секрет называется ключом шифрования. Также на практике обязательным критерием устойчивой к атакам криптосистемы является наличие механизма смены ключа. Основные принципы по управлению ключами изложены в стандарте ANSI X9.17 [1].

Стандарт ANSI X9.17 основан на стандарте ISO 8732 и посвящен процедурам управления ключами, и определяет два типа ключей: ключи шифрования ключей и ключи шифрования данных. Ключи шифрования ключей используются для шифрования распределяемых ключей, а также их подписи.

Ключи шифрования ключей распределяются среди участников информационной инфраструктуры достаточно редко и для этого требуется метод распределения, отличный от метода распространения ключей шифрования данных. Одним из наиболее надежных методов распределения ключей шифрования ключей был и остается – обмен ключами при личной встрече участников.

Распределение ключей шифрования данных происходит гораздо чаще, потому что их использование гораздо более интенсивное. Следующая диаграмма иллюстрирует процесс распределения ключей в соответствии с X9.17:

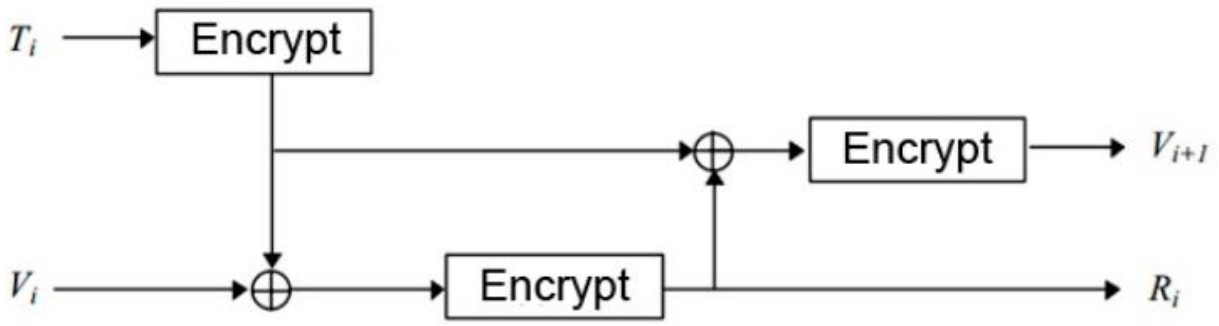

Рисунок 1- Процесс распределения ключей в соответствии с X9.17

Пусть $E_k(x)$ - результат шифрования сообщения X на ключе K с использованием некоторого криптографического алгоритма. v_0 - секретная начальная последовательность или синхропосылка. T – отметка времени. Для генерации следующего значения ключа R_i используется соотношение (1):

$$R_i = E_k(E_k(T_i) \oplus v_i) \quad (1)$$

Следующее значение v_i генерируется следующим образом:

$$v_{i+1} = E_k(E_k(T_i) \oplus R_i) \quad (2)$$

На каком-то заранее определенном этапе процесс останавливается и текущее значение R_i используется в качестве ключа шифрования. В стандарте X9.17 в качестве алгоритма шифрования предусмотрен американский криптографический алгоритм DES.

1.1 Распределение ключей по параллельным каналам

Одним из решений проблематики распространения ключевой информации, в виду достаточно высокой степени ее компрометации является разделение ключа на части, с последующей передачей по независимым каналам. Одна часть может быть передана по почте, другая - по SMS и т.д. В данном случае для перехвата ключа шифрования, злоумышленнику потребуется перехватить все его части, передаваемые по нескольким каналам сразу. Эту проблему довольно сложно решить на практике. Следующая иллюстрация описывает процесс разделения, передачи и восстановления ключевой информации для участника информационного обмена.

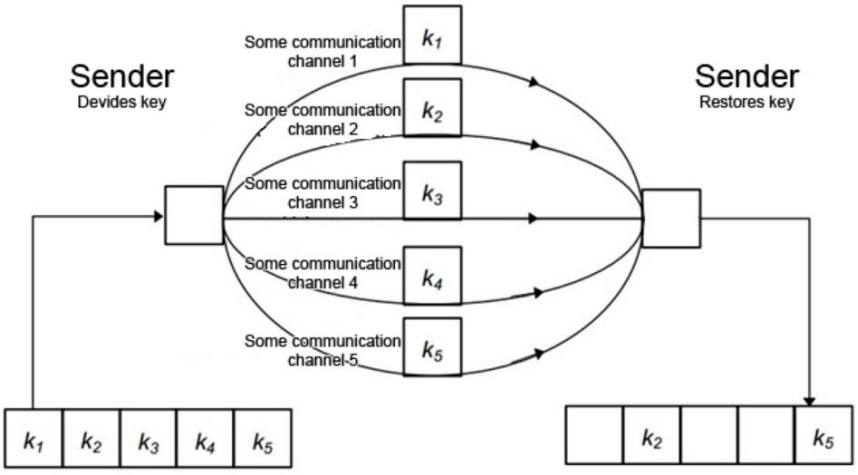

Рисунок 2 – Процесс разделения, передачи и восстановления ключевой информации для участника информационного обмена

Также стоит учитывать, что в процессе передачи некоторые фрагменты ключа могут быть искажены или вовсе утеряны, в связи с чем необходимо ввести дополнительные меры по повышению надежности передачи, а также дальнейшей проверки целостности частей ключа.

1.2.1 Распределение ключей в крупных сетях

В небольших сетях удобно использовать общие ключи шифрования ключей сразу для пары пользователей. Однако при таком способе распределения ключей в сети из N человек требуется $(N * (N - 1))/2$ обмена, что серьезно ограничивает практическую возможность использования такой схемы.

В тех случаях, когда количество участников в сети может быть достаточно большим, гораздо эффективнее базировать схему распределения на центральном сервере.

1.2.2 Использование одноразового ключа связи

Бывают случаи, когда при шифровании могут использоваться одноразовые или сеансовые ключи. Примером может послужить аудио- или видеозвонок. Сеансовый ключ, исходя из названия – это такой ключ, время жизни которого равно одной определенной сессии. Очевидным плюсом здесь является отсутствие необходимости дальнейшего хранения ключа.

Существует большое количество специальных протоколов, называемых протоколами согласования или распределения общих ключей. Их задача – сводится как раз-таки к согласованию сеансового ключа связи между участниками. Некоторые протоколы могут предусматривать присутствие участниками ранее установленного общего секрета, с помощью которого стороны аутентифицируют и генерируют сеансовый ключ. Также могут использоваться протоколы без общего секрета, такие как протокол Диффи-Хеллмана, трехэтапный протокол Шамира или TPM [2].

1.2.3 Обновление ключей шифрования

Часто, на практике, задача распределения ключей представляет собой достаточно трудоемкую задачу. В связи с этим более простое решение – это сгенерировать новый ключ на основе предыдущего. Такое решение потребует использования односторонней функции, которая также может быть общим секретным параметров для участников информационного обмена. В этом случае новый ключ генерируется согласно заранее определенному алгоритму, результат работы которого будет представлять результат работы односторонней функции.

При таком подходе параметр безопасности нового ключа определяется старым его значением. Однако здесь стоит учесть, что при компрометации как текущего, так и одного из предыдущих ключей будут скомпрометированы и последующие. Поэтому вводятся дополнительные меры безопасности, которые

подразумевают хранение односторонней функции и алгоритма для разработки ключа в секрете, что зачастую также трудно реализуемо на практике.

1.2 Хранение ключей

Зачастую в большинстве систем реализован подход, при котором пользователь запоминает секретный пароль или ключ шифрования, с помощью которого генерируется уникальный ключ. Данный подход имеет существенные недостатки: во-первых, при достаточно длинном ключе шифрования или пароле, пользователь легко может его забыть. Во-вторых, возникает необходимость каждый раз вводить секретные параметры, которые могут быть подсмотрены или перехвачены через TEMPEST.

Другое решение – хранить ключ в нечитаемом виде на физическом носителе, который активируется при подключении к считывающему устройству. В этом случае даже пользователь не знает секретного параметра, что исключает его преднамеренную или случайную компрометацию.

Любой знает, что такое физический ключ, что привносит некоторую абстракцию и делает такую защиту более легкой для понимания.

1.4 Резервирование ключей

Весь смысл шифрования данных – это невозможность их прочтения без ключа. Если ключ шифрования утерян, то практически гарантировано данные будут утеряны навсегда.

Key-escrow. При таком подходе все сотрудники записывают свой пароль, например, на бумажке и передают их системному администратору, который зашифровывает их с помощью мастер ключа. Мастер ключ должен знать только системный администратор и руководство.

Стоит отметить, что и при таком способе организации секретности ключей существует огромный риск их компрометации, т.к. не исключен сговор или некомпетентность со стороны последних.

Shared secret protocol. Единовременно генерации, ключ разбивается на части, которые в последующем отправляются руководящему составу компании. При таком подходе, полученные фрагменты не имеют никой ценности и только вместе составляют ключ.

1.5 Компрометация ключей

Как уже было отмечено, хранение ключей в секрете – наиболее сложная задача прикладной криптографии и, пожалуй, самая щепетильная проблема возникает в случае их компрометации.

В случае компрометации ключа необходимо отказаться от его использования в как можно более кратчайшие сроки.

1.6 Плановая смена ключей

Любой криптографический ключ должен использоваться определенное число раз. К основным причинам регулярной смены ключей шифрования относится:

- вероятность компрометации ключа прямо пропорциональна времени его использования;
- урон от компрометации ключа тем выше, чем дольше он использовался;
- при длительном использовании ключа на нем накапливается критическая масса, что может значительно облегчить криптанализ;
- если ключ используется длительное время, то у злоумышленника гораздо больше времени на его частичный или полный перебор.

Из вышеизложенного следует, что при любом сценарии криптографической защиты требуется стратегия, которая бы четко с математической точки зрения, обосновывала бы время жизни криптографического ключа.

1.7 Уничтожение ключей

Всякий раз, при смене криптографического ключа, старый ключ должен быть надлежащим образом уничтожен. В слово «надлежащим» здесь вкладывается куда более комплексное понятие, нежели простое перемещение ключа в корзину и ее очистка после. Этот метод плох тем, что так отчищается только заголовочный сектор диска, а данные все еще физически остаются на нем. Поэтому следует применять процедуру многократной перезаписи сектора диска последовательностью нулей и единиц. Также важно чтоб распределение такой бинарной последовательности носило равновероятностный характер.

2 Формирование общего секрета с помощью искусственных нейронных сетей

Предположим, что абоненты A и B имеют абсолютно идентичные ИНС, соединенные открытым каналом связи [5, 6] (рисунок 3). Каждая такая ИНС состоит из одного слоя персептронов. Каждый персептрон имеет n входов и ступенчатую функцию активации $\sigma(*)$ (рисунок 4). Перед началом синхронизации абоненты A и B независимо друг от друга, случайным образом, формируют вектор весовых коэффициентов (BK).

$$\bar{w}a = wa_{11}, wa_{12}, \dots, wa_{1n}, wa_{21}, wa_{22}, \dots, wa_{2n}, \dots, wa_{K1}, wa_{K2}, \dots, wa_{Kn}, \quad (3)$$

$$\bar{w}b = wb_{11}, wb_{12}, \dots, wb_{1n}, wb_{21}, wb_{22}, \dots, wb_{2n}, \dots, wb_{K1}, wb_{K2}, \dots, wb_{Kn}, \quad (4)$$

где wa_{ij} , $wb_{ij} \in [-L, L]$, $i = 1,2, \dots, K$, $j = 1,2, \dots, n$, L – целое число.

Каждый элемент этих векторов w_{ij} есть случайное целое число с дискретным равномерным законом распределения (рисунок 3).

$$P(w_{ij} = s_{ij}) = \frac{1}{2L+1}, \quad (5)$$

где $s_{ij} = -L, -L+1, \dots, -1,0,1, \dots, L-1, L$;

L – целое число

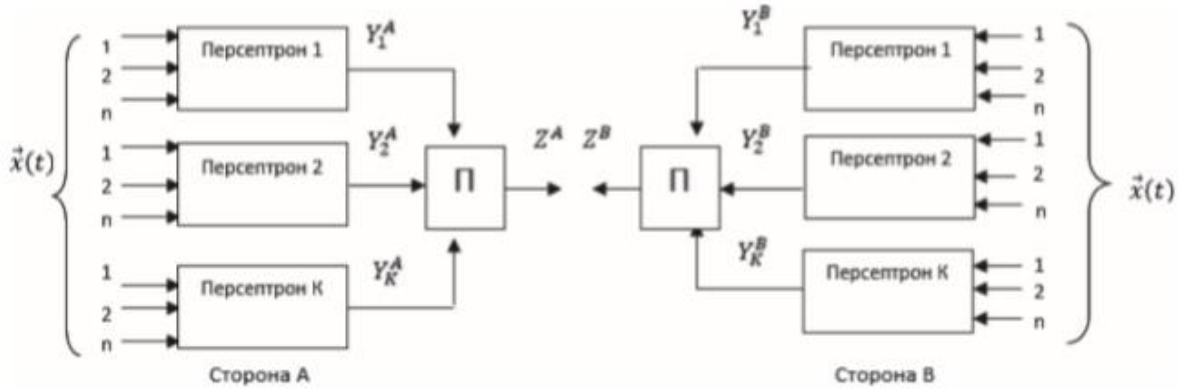

Рисунок 3 - Синхронизируемые ИНС

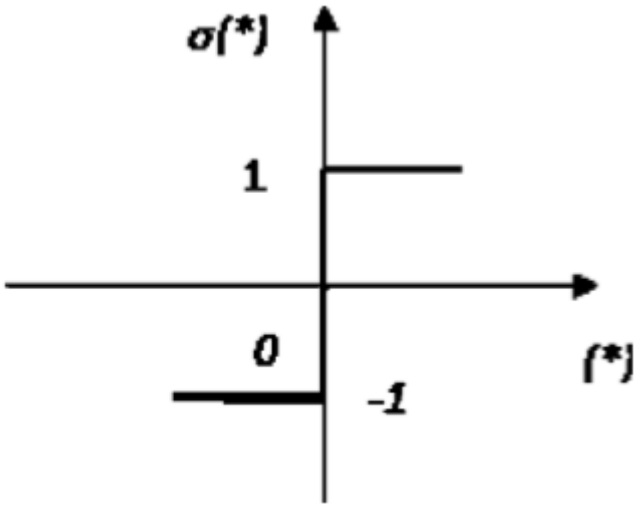

Рисунок 4 - Функция активации

Каждый шаг синхронизации начинается с подачи на входы обеих сетей выбранного случайным образом вектора:

$$\vec{x}(t) = x_{11,}x_{12}, \dots, x_{1n}, x_{21}, x_{22}, \dots, x_{2n}, \dots, x_{K1}, x_{K2}, \dots, x_{Kn}, \quad (6)$$

где $x_{ij} \in [-1,1]$ – дискретная случайная величина с равномерным распределением, $t = 1,2, \dots$ - номер такта (далее все рассмотренные величины зависят от t, но для упрощения записи эта зависимость в обозначениях отсутствует).

Для каждого из персептронов выходное значение определяется как знак от взвешенной суммы произведения входного вектора на соответствующие веса:

$$Y_i^{A/B} = \sigma(\sum_{j=1}^{n} w_{ij}^{A/B} x_{ij}), \quad (7)$$

Индекс A/B означает, что операция касается обеих частей A и B, а единичный индекс – что операция касается одной сети соответственно. Функция активации σ (*) имеет следующее определение:

$$\sigma(*) = \begin{cases} 1, \sigma(*) \geq 0, \\ -1, \sigma(*) \leq 0.' \end{cases} \quad (8)$$

После чего вычисляется выходная величина Z для каждой из сетей:

$$Z^{A/B} = \prod_{i=1}^{K} Y_i^{A/B} = \prod_{i=1}^{K} \sigma\left(\sum_{j=1}^{n} w_{ij}^{A/B} x_{ij}\right), \quad (9)$$

На основании сравнения выходных величин двух ИНС, реализован процесс синхронизации. Коррекция векторов весов обеих сетей происходит только тогда, когда обе выходные величины равны друг другу ($Z^A = Z^B$). Корректировка весовых коэффициентов производится только тогда, когда выходная величина персептрона, к которому они присоединены равна выходу сети $Z^{A/B}$. Корректировка ВК осуществляется по правилу Хэбба:

$$w_{ij}^{A/B} = \begin{cases} w_{ij}^{A/B} + Z^{A/B} x_{ij}, \text{если } Z^A = Z^B \text{ и } Z^{A/B} = Y_i^{A/B}, \\ w_{ij}^{A/B}, \text{в противном случае.} \end{cases}, \quad (10)$$

Также учитывается ограничение $w_{ij}^{A/B} \in [-L, L]$

$$w_{ij}^{A/B} = \begin{cases} \pm L, \text{если } \left|w_{ij}^{A/B}\right| > 0, \\ w_{ij}^{A/B}, \text{в противном случае.} \end{cases}, \quad (11)$$

Процесс синхронизации продолжается до тех пор, пока векторы $\vec{w}a, \vec{w}b$ не станут равными друг другу Таким образом у абонентов *A* и *B* появляется общая секретная информация, выраженная в виде последовательности десятичных чисел:

$$\vec{w}^{A/B} = w_{11,} w_{12}, \dots, w_{1n}, w_{21}, w_{22}, \dots, w_{2n}, \dots, w_{K1}, w_{K2}, \dots, w_{Kn}\ , \quad (12)$$

Общее секретное число можно сформировать из (11), например, как конкатенацию значений ВК

$$S = w_{11} \| w_{12} \| \dots \| w_{1n} \| w_{21} \| w_{22} \| \dots \| w_{2n} \| \dots \| w_{K1} \| w_{K2} \| \dots \| w_{Kn}\ , \quad (13)$$

2.1 Статистические закономерности процесса синхронизации

2.1.1 Неопределенность момента наступления полной синхронизации

Как уже отмечалось ранее, в процессе синхронизации абоненты только выходами значениями ИНС $Z^{A/B}$ по открытому каналу связи. Таким образом выходит, что они не знают равны ли ВК их ИНС и в какой именно момент необходимо остановить циклический процесс синхронизации. Исходя из этой проблемы, абоненты могут продолжать процесс синхронизации, сколь угодно долго даже не зная о том, что их ВК уже выровнялись. Это, во-первых, приводит к увеличению времени формирования общего ключа, на основании ВК двух сетей абонентов *A* и *B*. Во-вторых, увеличивает шансы злоумышленника на реализацию атак [3]. Ну и в-третьих, появляется возможность преждевременной остановки процесса синхронизации. В [4] рассмотрен подход для распознавания

процесса полной синхронизации, основанный на сравнении выходных величин двух нейросетей. Суть данного подхода основана на сравнении выходных величин $Z^A(t)$ и $Z^B(t)$ на протяжении определенного количества тактов обновлений сетей и если на протяжении достаточно большого количества тактов выходные величины равны $Z^A(t) = Z^B(t)$, то появляется высокая степень уверенности в достижении равенства ВК.

Эксперименты в которых были построены модели [4], выявили, что в процессе синхронизации пока $\vec{w}^A(t) \neq \vec{w}^B(t)$, наблюдаются такты в которых $Z^A(t) \neq Z^B(t)$, а так же такты в которых $Z^A(t) = Z^B(t)$. И действительно, на практике, весьма часто встречаются такие отрезки тактов, длина которых доходит до нескольких сотен и более совпадений $Z^A(t) = Z^B(t)$, однако же синхронизация все еще не была достигнута. Поучается, что решение этой задачи носит сугубо вероятностный характер и опирается на результаты так называемого имитационного моделирования. Однако же, в любом случае процесс такого формирования общего ключа необходимо закончить сравнением $\vec{w}^A(t_c)$ и $\vec{w}^B(t_c)$, чтобы А и В были твердо уверены в одинаковости полученных ВК. В данном дипломном проекте, в частности в программной реализации, рассматривается способ с применением хеширования, то есть сравниваются хеш-значения векторов $h[\vec{w}^A(t_c)]$ и $h[\vec{w}^B(t_c)]$. Также возможно шифровать и расшифровывать какие-либо случайные последовательности, сгенерированные одной из сторон, с использованием в качестве ключей полученные хеши от ВК.

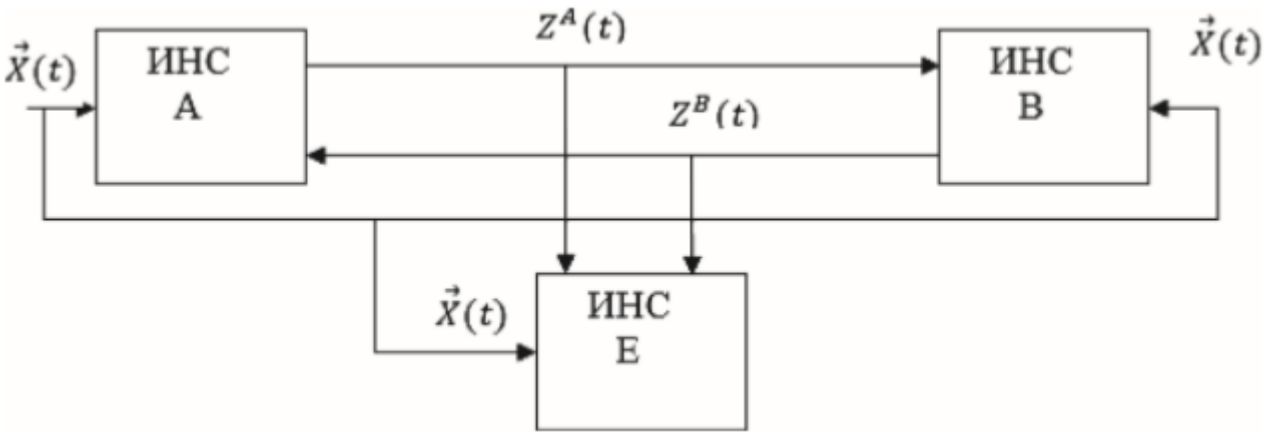

Рисунок 5 - Схема взаимодействия сетей: t – номер такта синхронизации, $\vec{X}(t)$ – вектор синхронизирующих случайных воздействий, $Z^A(t)$, $Z^B(t)$ – выходные величины сетей А и В соответственно

Также стоит отметить проблематику выбора параметров, синхронизируемых ИНС: *K, n, L*. Выбор этих параметров необходимо осуществлять исходя из соображений криптографических, не забывая о конфиденциальности сформированного ключа, а также о времени, затрачиваемом на его формирование, то есть учет количества требуемых тактов на полную синхронизацию. Так как все вышеперечисленное зависит [5] от параметров сетей и той модели поведения, которую выберет криптоаналитик или

третья сторона E, то целесообразным является рассмотрение статистических закономерностей процесса синхронизации с введением криптоаналитика *E.*

2.1.1.1 Модели поведения криптоаналитика

Рассматривая основные известные атаки на формируемый ключ со стороны криптоаналитика или же третьей «прослушивающей» канал связи стороны, по которому синхронизируемые сети обмениваются информацией, можно изложить некую общую схему, которая представлена на (рисунке 5). Архитектура и параметры всех сетей идентичны (рисунок 3).

2.1.1.2 Силовая атака

Силовая атака, он же брутфорс, он же полный перебор – метод взлома, который является самым универсальным, простым и широко распространённым, однако в то же время и самым долгим. Данный метод можно классифицировать, как метод исчерпывающего поиска решения путем полного перебора всевозможных значений. Сложность данного метода обусловлена количеством всевозможных значений, принимаемых формируемым ключом. Для того чтоб ввести количественную оценку сложности перебора множества значений ключа, необходимо произвести арифметический расчет максимального количества значений ключа. Всевозможное количество значений рассчитывается по формуле:

$$n = (2L + 1)^{n*K}, \tag{14}$$

Для примера, выберем нейросеть с параметрами *n=25, K=3, L=8*. Подставляя соответствующие значения в формулу, получим количество комбинаций, равное $4.2 \cdot 10^{84}$. При скорости перебора, равной 1 миллиарду комбинаций в секунду, в предположении, что будет запущено сразу 1 миллион потоков для параллельных вычислений, на полный перебор всех комбинаций уйдет $8 \cdot 10^{63}$ лет, что во представляет значение куда большее нежели предположительное времени актуальности зашифрованной информации. Из этого следует, что данный метод не подходит для взлома, даже при относительно небольших значениях параметров сети.

2.1.1.3 Простая атака

При простой атаке для взлома ключа криптоаналитику необходимо иметь свою собственную нейросеть, имеющую такую же структуру, как и сети легитимных абонентов.

При начале сеанса выработки ключа криптоаналитик подключает свою сеть к каналу связи, и, при помощи перехватываемых векторов синхронизирующих случайных воздействий и выходных величин сетей A и B, обучает сеть E по следующим правилам обучения:

1) Если A и B получили разные выходные величины ($z^A \neq z^B$), тогда E не меняет своих весов.

2) Если сети A и B получили одинаковые выходные величины ($z^A = z^B$) и к тому же ($z^A = z^B = z^E$), что является обязательным и необходимым условием, то E производит корректирование своих весовых коэффициентов.

3) Если выходы легитимных пользователей идентичны, а криптоаналитик получает отличный от них результат ($z^A = z^B \neq z^E$), то E пропускает свою коррекцию.

Вышеизложенные правила приводят к тому, что на практике коррекций весов у криптоаналитика E гораздо меньше, чем у абонентов A и B, что приводит к задержке обучения. И действительно, вероятность совпадения выходов у сетей A и B равна $p_{AB} = \frac{1}{3}$, в то время как вероятность совпадения выхода сразу у всех трех сетей равна $p_{ABE} = \frac{1}{27}$. Такое отставание и является главным фактором безопасности процесса синхронизации.

2.1.1.4 Геометрическая атака

Для ослабления эффекта отставания в количестве коррекций применяется так называемая геометрическая атака. Суть которой заключается в следующем: при возникновении ситуации 3, то есть при ($z^A = z^B \neq z^E$), предварительно делается коррекция выходной величины того персептрона сети E, у которого величина $\sum_{i=1}^{n} w_{ij}^E x_{ij}$ наименьшая по абсолютному значению. Это приводит к изменению знака выбранного персептрона на противоположный, как, собственно, и знака всей сети, что переводит сеть в случай ($z^A = z^B = z^E$). Эффективность такой атаки в среднем выше, чем эффективность простой атаки, поэтому на практике целесообразно считать основной именно такую модель поведения криптонаналитика E.

2.2 Конфиденциальность сформированного общего ключа

Исходя из того, что процесс синхронизации двух сетей является случайным, то и степень конфиденциальности носит вероятностный характер и рассчитывается с помощью математических вероятностных моделей.

Обозначим число заранее заданных тактов синхронизации через d, количество фактических тактов, через которые наступила полная синхронизация сетей A и B через t_{AB}, а сетей A и E через t_{AE}. Величины t_{AB} и t_{AE} – дискретные случайные величины, законы распределения которых зависят от параметров сетей L, n, K. В некоторых источниках, например, [7], для оценки уровня безопасности процесса синхронизации рассматривается параметр, названный коэффициентом безопасности:

$$r = T_{AE}/T_{AB} \tag{15}$$

где T_{AB} – среднее время до полной синхронизации сетей A и B,

T_{AE} – среднее время до полной синхронизации сетей A и E.

С помощью этого параметра можно проследить тенденцию зависимости безопасности от параметров сети. Однако для криптографических применений важна не безопасность в среднем, а конкретно в каждом сеансе формирования, поэтому далее рассматриваются другие критерии. В процессе синхронизации могут произойти следующие события:

1. Сети A и B достигли синхронизма (их веса стали равны друг другу), E смог обучить свою сеть. Для этого события выражение вероятности запишется в виде $P(t_{AB} \le d, t_{AE} \le d)$. Это событие является неблагоприятным, так как сформированный ключ сразу дискредитируется.

2. Сети A и B достигли синхронизма, E не успел обучить свою сеть. Для этого события выражение вероятности запишется в виде $P(t_{AB} \le d, t_{AE} > d)$. Это событие является благоприятным, так как ключ был сформирован и не дискредитирован.

Остальные события нас не интересуют, так как в случае их наступления A и B не достигают синхронизма, следовательно, ключ не будет сформирован.

Вероятность сложного события $P(t_{AB} \le d, t_{AE} \le d)$ пренебрегая взаимосвязью частных событий можно представить следующим образом:

$$\begin{aligned} P(t_{AB} \le d, t_{AE} \le d) &= P(t_{AB} \le d)P(t_{AE} \le d | t_{AB} \le d) \\ &\approx P(t_{AB} \le d)P(t_{AE} \le d) \end{aligned} \tag{16}$$

Аналогично преобразуется выражение $(t_{AB} \le d, t_{AE} > d)$. Такие преобразования позволяют упростить расчетную запись путем перехода к одномерным вероятностям.

Основываясь на вышеизложенном, задачи анализа безопасности есть ни что иное, как расчет вероятности удачной синхронизации сетей A и B ($P(t_{AB} \le d)$) и конечно же наступления или не наступления синхронизации сети E с сетями A и B ($P(t_{AB} \le d, t_{AE} > d)$) при выбранных значениях d.

3 Выбор параметров и обоснование структуры ИНС

Анализируя работы и некоторые результаты других авторов можно сформировать некоторое представление и, собственно, обоснование параметров, синхронизируемых ИНС.

К примеру, как отмечают в [2], для того, чтоб вынести решение касательно обоснования, необходимо и достаточно выбрать некий диапазон параметров, которые, во-первых, зададут диапазон значений ВК $\vec{w}^{A/B}$, а во-вторых, будут влиять на время синхронизации. Увеличение параметра L с одной стороны, приводит к криптостойкости, так как параметр L отучает за диапазон возможных значений, которые могут принимать отдельно взятые ВК $w_{ij} \in W$, но с другой стороны, опять же, сильно влияет на время согласования ИНС. Из этого следует, что если вопрос криптостойкости стоит особенно остро, то конечно же стоит отдать предпочтение первому пункту, то есть увеличить значение L.

Таким образом, при достижении полной синхронизации сети A и B формируют общую, одинаковую, последовательность десятичных чисел (8), которая состоит из целых положительных и отрицательных чисел из интервала [–L, L]. Для использования такой последовательности в качестве ключа, над ней следует произвести некоторые преобразования или просто представить в виде бинарной последовательности, не исключая возможность удаления лишних нулей старших разрядов, что позволит рассеять статистические закономерности полученного ключа.

Таким образом становится естественным вопрос касательно разрядности двоичного числа при переходе от десятичного к двоичному, т.е. $w \to key_2$. Как отмечалось, диапазон значений десятичных чисел w_{ij} изменяется в пределах [-L, L] и получается, что диапазон возможных значений w_{ij} равен $2L + 1$. Так же стоит учитывать, что избыточная разрядность нежелательна, т.к. это приведет к избыточному количеству нулей в ключевой последовательности, а, следовательно, создаст дополнительные возможности для криптоанализа. Один из вариантов базируется на исключении нулевой последовательности старших разрядов двоичного числа (таблица 1). Вторым вариантом может быть перевод, при котором максимальное количество разрядов двоичного числа будет равно длине наибольшего значения $w_{ij} \in w$ (таблица 2). Дальнейшие преобразования могут подразумевать конкатенацию получившихся таким образом двоичных значений переменной длины. При заданных параметрах остается вопрос касательно знака «-», который как правило стоит в старшем разряде двоичного числа. В данной работе, для решения этой проблемы, предполагается произвести смещение очередного w_{ij} на L, т.е. $w_{ij} \coloneqq w_{ij} + L$. Таким образом новые значения вектора весов w будет лежать на интервале от 0 до 2L.

Таблица 1 - Перевод десятичного числа в двоичное с исключением нулей старших разрядов

$w_{ij\,(10)}$	0	1	2	3	4	5	6	…	20	…
$w_{ij\,(2)}$	0	1	10	11	100	101	110	…	10100	…

Таблица 2 - Перевод десятичного числа в двоичное без удаления нулей старших разрядов при максимальном значении L=10

$w_{ij\,(10)}$	0	1	…	9	10	11	12	…	19	20
$w_{ij\,(2)}$	00000	00001	…	01001	01010	01011	01100	…	10011	10100

Корреляция параметров ИНС со временем. Ниже приведены результаты экспериментов с изменением одного и фиксацией двух других параметров ИНС.

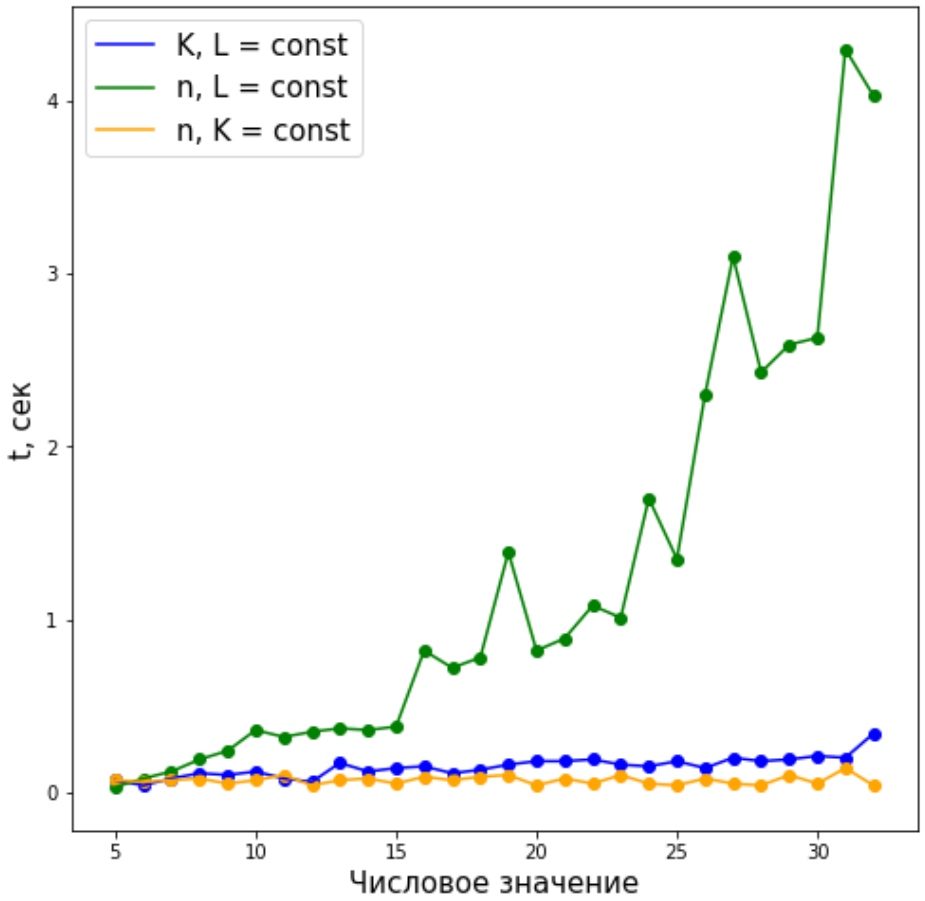

Рисунок 6 - Корреляция параметров ИНС со временем

Таким образом, исходя из значений, представленных на графике выше, становится очевидным целесообразность уменьшения числового значения параметра *K* и увеличение параметров ИНС – *n* и *L* соответственно.

Необходимое количество тактов синхронизации при различных параметрах ИНС. Так как вероятность наступления синхронизации носит сугубо

вероятностный характер, то наибольшую точность в выявлении закономерности показывают эмпирические данные. В следующем эксперименте, графики которого приведены на (рисунках 7-10), параметры нейросети фиксировались и эксперимент повторялся по 10 раз для каждого такого эксперимента. Для визуального отличия данных разных экспериментов графики имеют разный цвет.

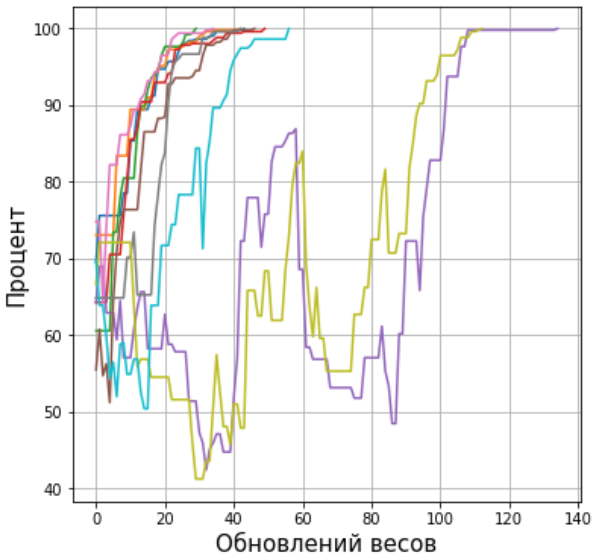

Рисунок 7 - K=2, n=16, L=8

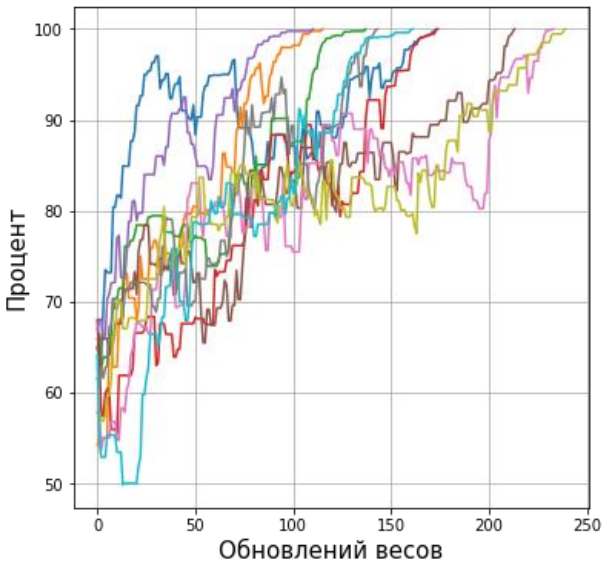

Рисунок 8 - K=8, n=8, L=10

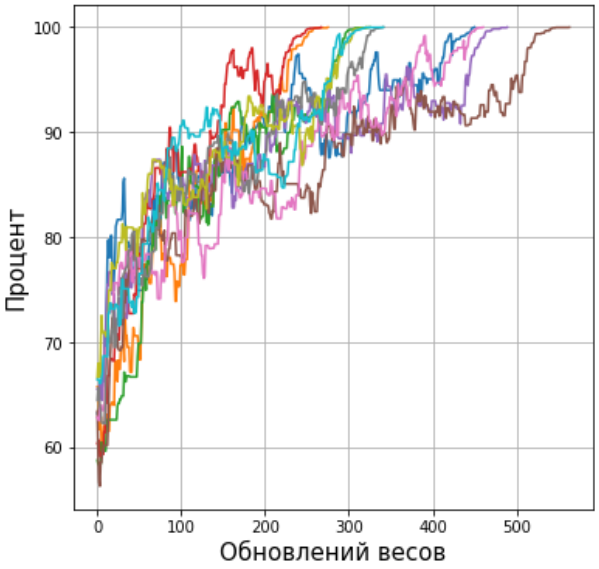

Рисунок 9 - K=16, n=16, L=10

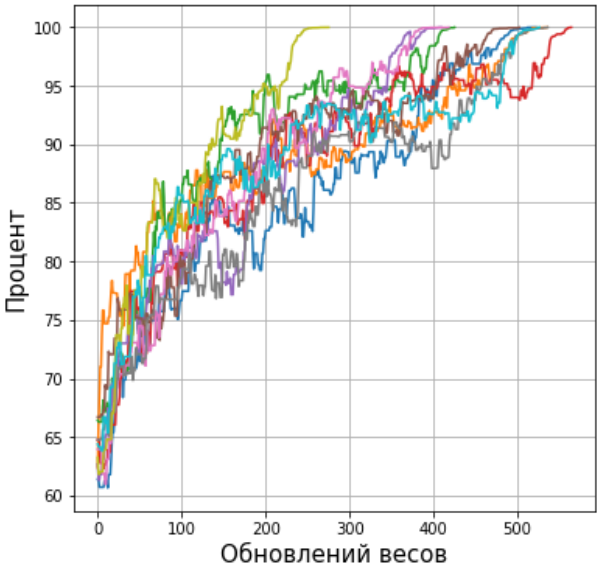

Рисунок 10 - K=24, n=24, L=16

Вышеприведенные иллюстрации еще раз подчеркивает вероятностный характер наступления синхронизации двух нейросетей.

3.1 Web Sockets

Веб-сокеты (Web Sockets) — это протокол прикладного уровня, который позволяет создавать tcp-соединение между клиентом и сервером для обмена данными в режиме реального времени. Веб-сокеты, в отличие от многих протоколов типа запрос-ответ (request-response), позволяют создавать двунаправленный канал, тем самым образуя так называемое дуплексное соединение.

Исходя из вышеизложенного вытекает очевидное преимущество соединения посредством веб-сокетов — это возможность слушать порт постоянно, без разрыва соединения, что дает возможность получать актуальные данные мгновенно, с поправкой на скорость передачи в канале связи.

На практике существует два основных, логически разделенных этапа работы web-сокетов:

1. Установка соединения (Opening handshake)
2. Передача данных

В качестве приложения к данному дипломному проекту было реализовано 3 программных продукта. Все три построены на схеме, в которой клиент взаимодействует с сервером. Две из которых используют web-сокеты, как основной инструмент взаимодействия с сервером.

Ниже представлена таблица (3), которая поверхностно описывает все три программные продукта.

Таблица 3 - Информация о программных продуктах, реализованных к дипломному проекту

№	Язык программирования	В основе взаимодействия	Формат передаваемых данных	Взаимодействие посредством интернет браузера	Скорость передачи данных
1	Python	ajax-запросы	JSON	Да	Низкая
2	JavaScript	web-сокеты	JSON	Да	Высокая
3	Python	web-сокеты	Байты	Нет	Очень высокая

4 Алгоритм и специфика работы программной реализации

Данная программная реализация представляет собой программный продукт, представленный двумя основными частями:

1. Часть, отвечающая за выработку и распределение, мастер-ключа или «USB-token creator»
2. Клиент-серверная часть

Для корректной работы программного продукта необходимо:

1. USB-накопитель, который предполагается использовать в качестве токена.
2. База данных, в которой будет храниться информация о легальных пользователях системы.
3. Сервер, на котором будет запущена БД, а также программный код для обработки новых подключений.
4. Дисковое пространство на сервере, для хранения мастер-ключей.
5. Так же необходима станция, у которой были бы доверительные отношения с сервером, для запуска USB-token creator'а.
6. Доступ к серверу по Белому IP-адресу.

4.1 Алгоритм работы «USB-token creator»

Целесообразно начать описание работы с части создания Мастер-ключа:

1. Генерируется случайный ключ K_{rand}, длиной 196 бит.
2. Вычисляется значение FLASH_ID и FLASH_NAME.
3. Пользователь придумывает пароль K_{user}.
4. Выбирается алгоритм хеширования HASH.
5. Создается пара файлов на usb-токене и на сервере, с одним и тем же именем key_name=$HASH(\ FLASH\_NAME\ )$.
6. На сервере в файл key_name записывается $K_{master-key} = K_{rand}$. Путь к файлу key_name заносится в бд.
7. На usb-токен записывается результат шифрования $E_{K_{user}}(K_{rand})$.

здесь *FLASH_ID* – это так называемый Volume Serial Number или серийный номер тома. Создается он на основе довольно сложной комбинации года, часа, месяца, секунды и сотой доли секунды в процессе форматирования. Так же стоит отметить, что поскольку серийный номер тома генерируется во время форматирования, то он будет меняться при каждом последующем форматировании накопителя; *FLASH_NAME* – имя накопителя, распознаваемое операционной системой.

4.2 Пример использования «USB-token creator»

В данном примере использовался самый обычный накопитель на 4Gb памяти, хотя на практике предполагается использование номинала емкости не более чем 512мб.

Рисунок 11- Пример usb-накопителя

Для начала необходимо отформатировать usb-накопитель, который предполагается использовать в качестве usb-токена.

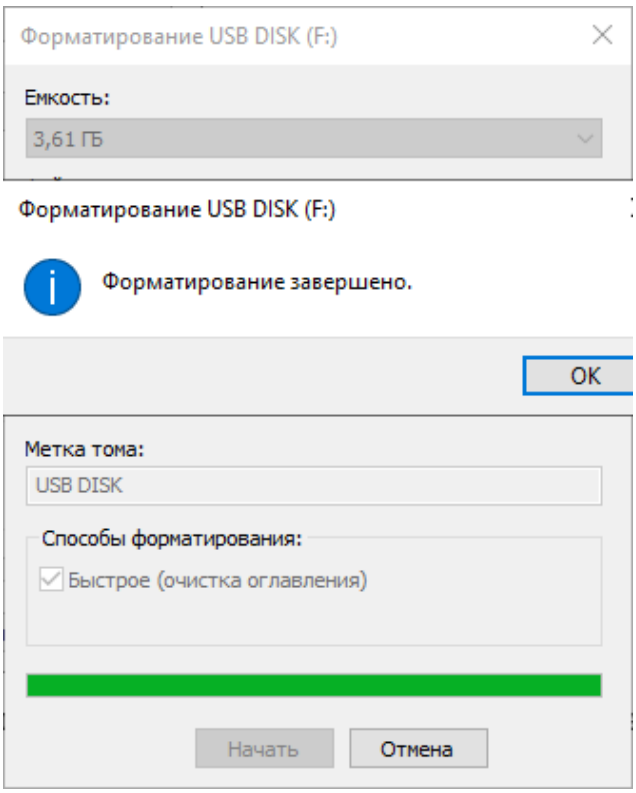

Рисунок 12 - Форматирование usb-накопителя

Имя usb-накопителя необходимо запомнить, т.к. оно пригодится на этапе создания мастер ключа.

Следующий этап – это регистрация пользователя в БД. Для упрощения, таблица пользователей представлена в самом лаконичном виде

Добавляем пользователя. Проверяем:

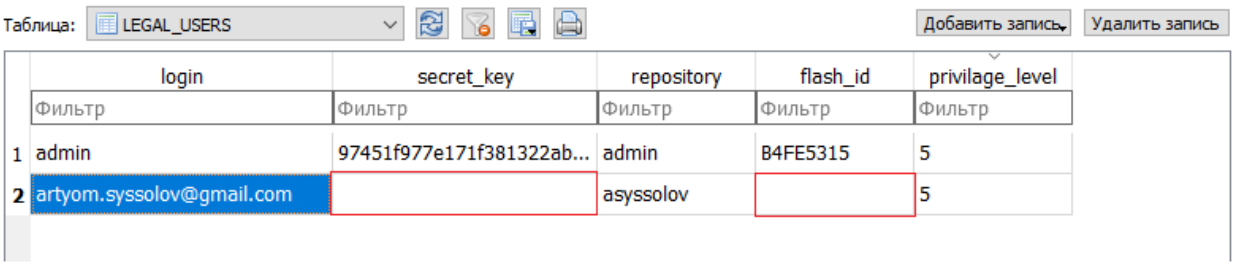

Рисунок 13 - Таблица легальных пользователей в БД

Как можно заметить, поля *secret_key* и *flash_id* на этом этапе необходимо оставить пустыми. Открываем «USB-token creator». Появляется консоль:

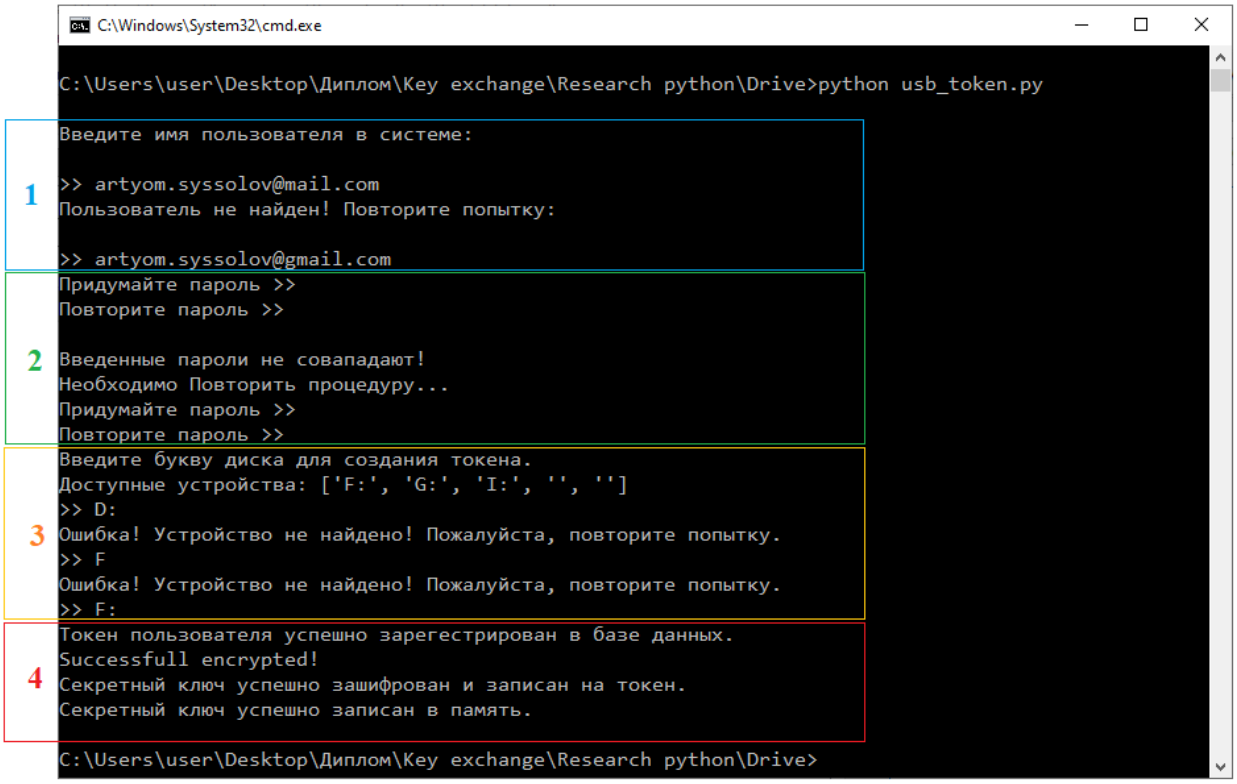

Рисунок 14 - Пример создания токена на usb-накопителе

Алгоритм создания и записи мастер-ключа:

1. Система просит ввести логин пользователя
2. Далее необходимо придумать пароль. Данный пароль должен знать только пользователь. Без знания этого пароля невозможно расшифровать мастер-ключ, хранящийся на usb-накопителе.
3. Важно: в целях безопасности, в консоли, введенный пароль не отображается.
4. Необходимо выбрать имя того накопителя, на который предполагается запись мастер-ключа.
5. Система уведомляет нас об успешности операции выработки, зашифрования и записи ключа на usb-накопитель и на дисковое пространство сервера или сообщает об ошибках - в противном случае.

Проверка usb-накопителя:

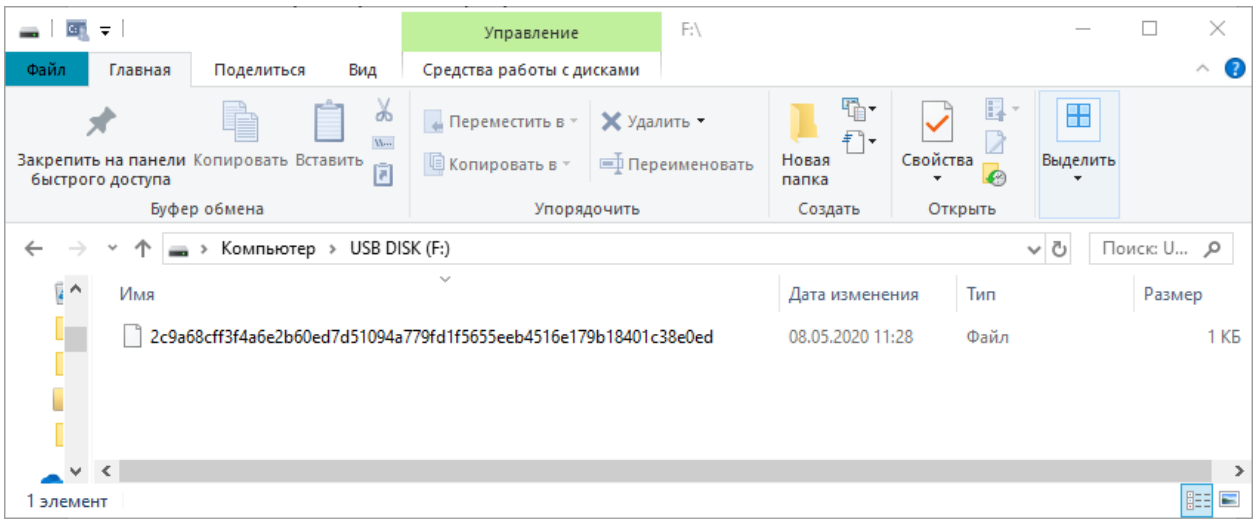

Рисунок 14 - Содержимое usb-накопителя после создания и записи мастер-ключа

Как видно на флеш-токене появился сгенерированный ключ.
Сожержание БД после успешного создания usb-токена:

```
1  SELECT * FROM LEGAL_USERS;
```

	login	secret_key	repository	flash_id	privilage_level
1	admin	97451f977e171f381322ab3693825001faecebc95...	admin	B4FE5315	5
2	artyom.syssolov@gmail.com	2c9a68cff3f4a6e2b60ed7d51094a779fd1f5655ee...	asyssolov	F2548C65	5

Рисунок 15 - БД после создания мастер ключа

ВАЖНО: в *secret_key* хранится имя ключа. Еще раз отмечу, что серверной части и части, отвечающей за выработку, мастер-ключа должны быть доверительные отношения, а также предполагается, что они находиться в защищенном сегменте сети.

4.3 Алгоритм работы клиент-серверной программы

1. Клиент подключается к серверу.
2. Производится выработка общего ключа K_{cs} по нейросетевому алгоритму [2] между клиентом и сервером.
3. После успешного формирования общего ключа клиент отправляет свой логин серверу, как результат зашифрования $E_{K_{cs}}(CLIENT\_LOGIN)$.
4. Сервер сопоставляет значения в базе данных значению $D_{K_{cs}}(E_{K_{cs}}(CLIENT\_LOGIN))$, в случае, если такой пользователь найден сервер

вытаскивает из БД значение FLASH_ID и генерирует случайное значение $rand \in [2^{100}, 2^{500}]$ и отпраляет клиенту $E_{K_{cs}}$ (FLASH_ID | rand).

5. Клиентская часть извлекает значение FLASH_ID и $rand$ из $D_{K_{cs}}(E_{K_{cs}}$ (FLASH_ID | rand)). Затем сопоставляет значение FLASH_ID всем USB-накопителям подключенным к машине, на которой запущена клиентская часть и в случае, если такое устройство найдено, то пытается найти на нем ключ с именем $HASH(\text{FLASH}_{\text{ID}})$, получает от туда $K_{master-key}$ как результат операции $D_{K_{user}}\left(E_{K_{user}}\left(K_{master-key}\right)\right)$. После чего клиент совершает операцию $E_{K_{master-key}}$ (FLASH_ID | rand) и отправляет серверу $E_{K_{cs}}(E_{K_{master-key}}$ (FLASH_ID | rand)).

6. Сервер производит операцию $D_{K_{master-key}}(D_{K_{cs}}(E_{K_{cs}}(E_{K_{master-key}}$ (FLASH_ID | rand)))) и если извлеченное значение совпадает с тем, что было отправлено клиенту ранее, то такой пользователь считается аутентифицированным.

Пример использования. Запускаем сервер:

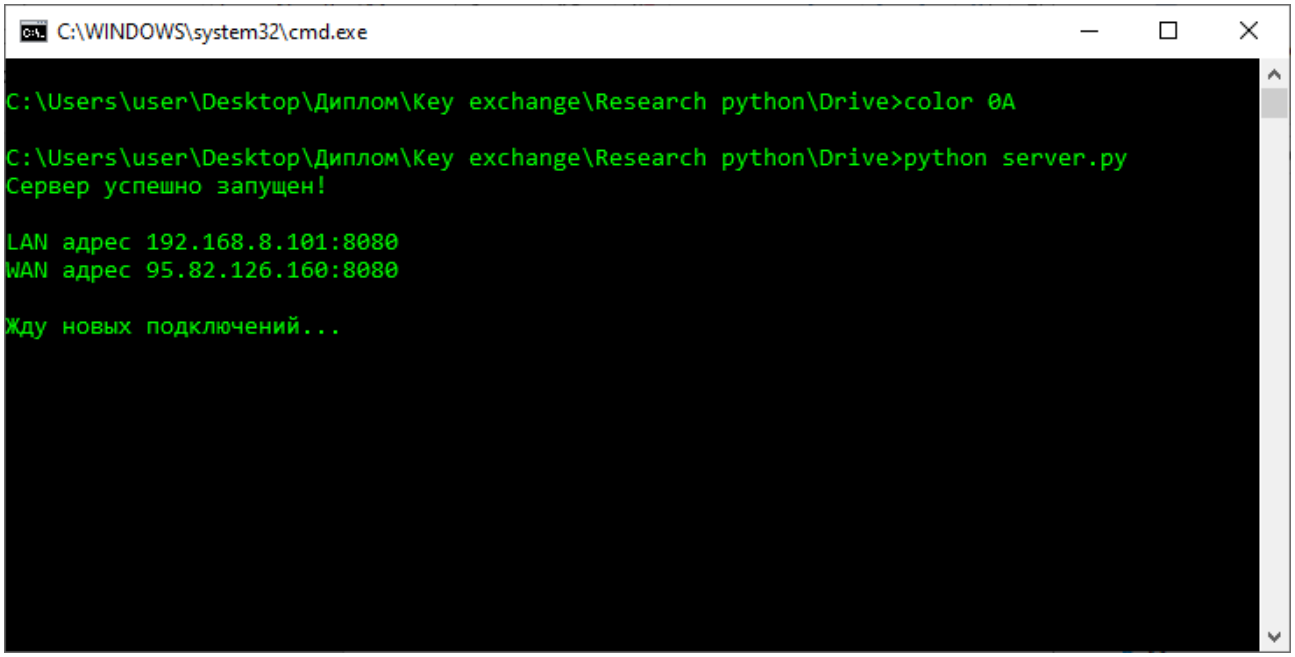

Рисунок 16 - Консоль сервера

Запускаем клиента. Производится импорт необходимых модулей и отрисовка графических компонентов окна программы. Дабы пользователи не сочли эти доли секунды задержкой или того хуже – багом, данный процесс сопровождается надписью «Загрузка…» на экране:

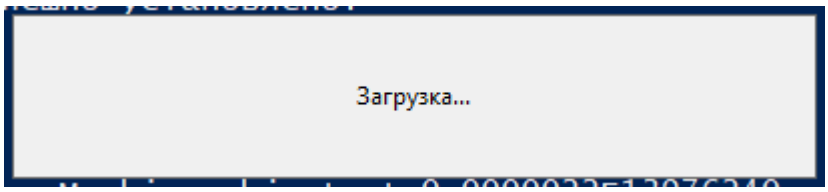

Рисунок 17 - Индикатор загрузки

Все необходимое для работы программы подгрузилсось, теперь клиент инициализирует подключение к серверу. Этот процесс сопровождается надписью «Соединение…»

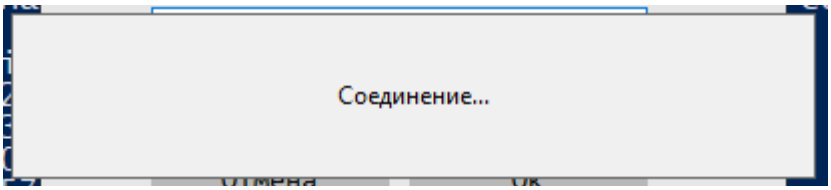

Рисунок 18 - Индикатор подключения

В случае успешного соединения в консоли сервера появится вывод о новом подключении. Так же сервер в ответ на подключение сообщает параметры для выработки общего ключа по алгоритму [2]:

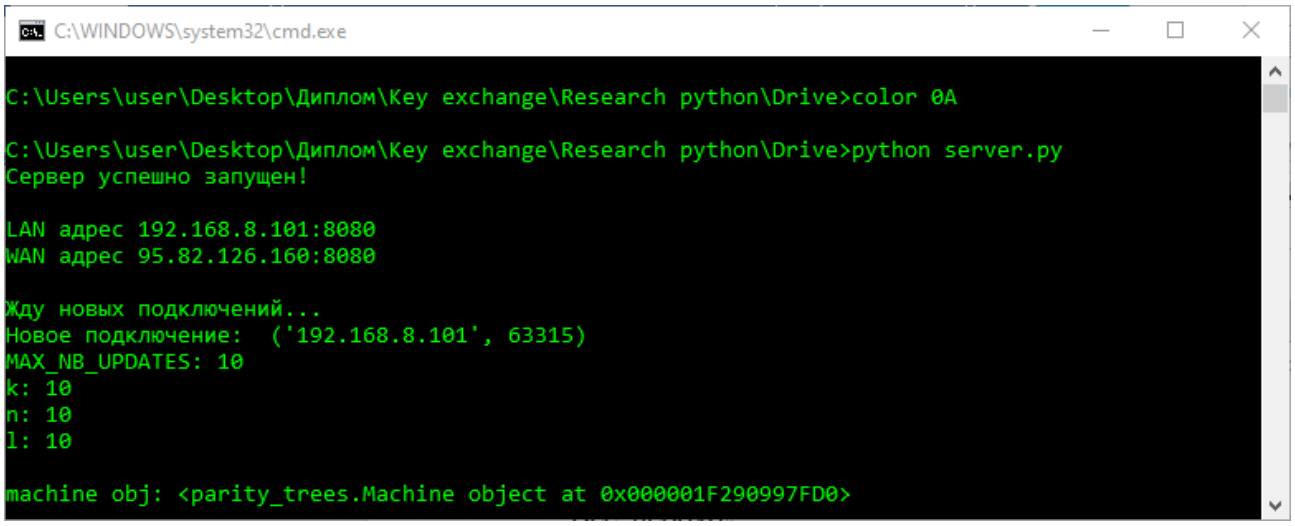

Рисунок 19 - Консоль сервера при подключении очередного клиента

Затем программа уведомляет клиента о том, что происходит процесс выработки общего ключа:

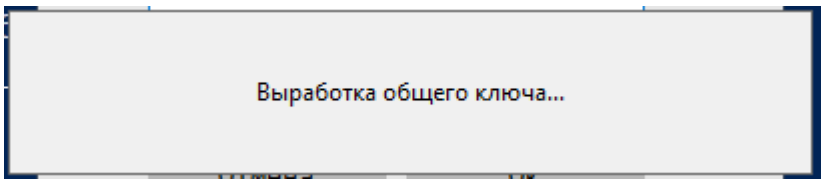

Рисунок 20 - Индикатор процесса выработки общего ключа K_{cs}

В случае успеха появится диалоговое окно, как показано ниже:

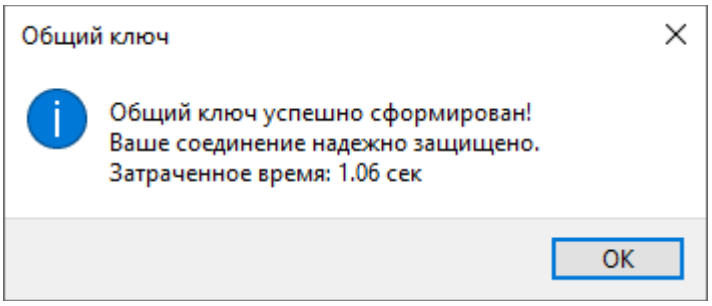

Рисунок 21 - Диалоговое окно после успешного подключения

Далее предполагается нажатие кнопки «ок». Диалоговое сразу же закроется.

Главное окно авторизации выглядит следующим образом:

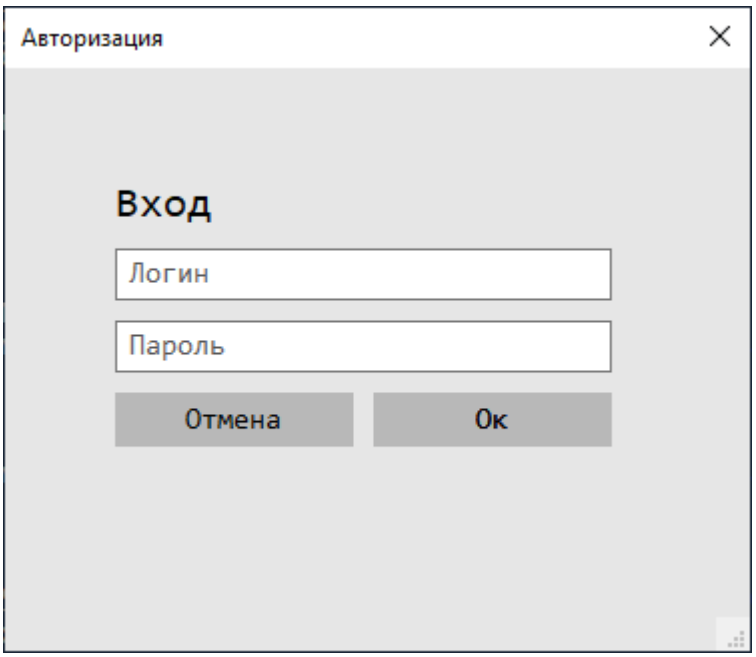

Рисунок 22 - Главное окно программы

где в поле «Логин» необходимо ввести имя пользователя, при регистрации в БД, а в поле «Пароль» – ключ для расшифрования мастер ключа, хранящегося на usb-токене.

Попытка пройти авторизацию:

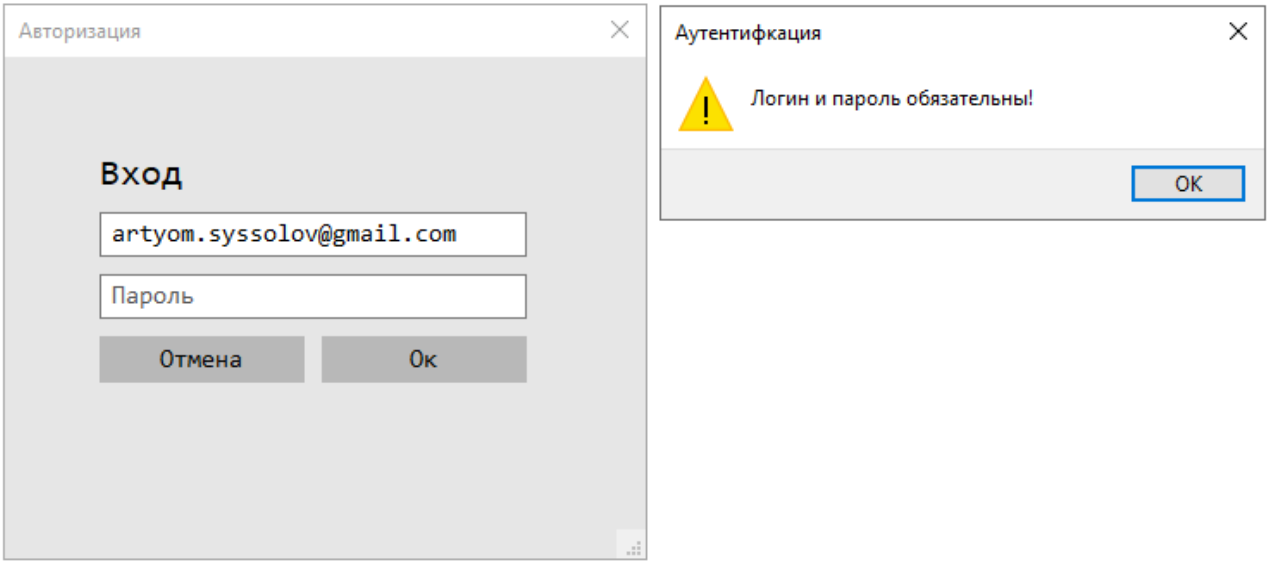

Рисунок 23 - Реакция программы на некорректно заполненные поля

В случае, если пользователь забудет ввести пароль или логин, то система сообщит об этом. При этом данные на сервер отправляться не будут, что позволяет в какой-то степени снизить нагрузку на серверную часть.

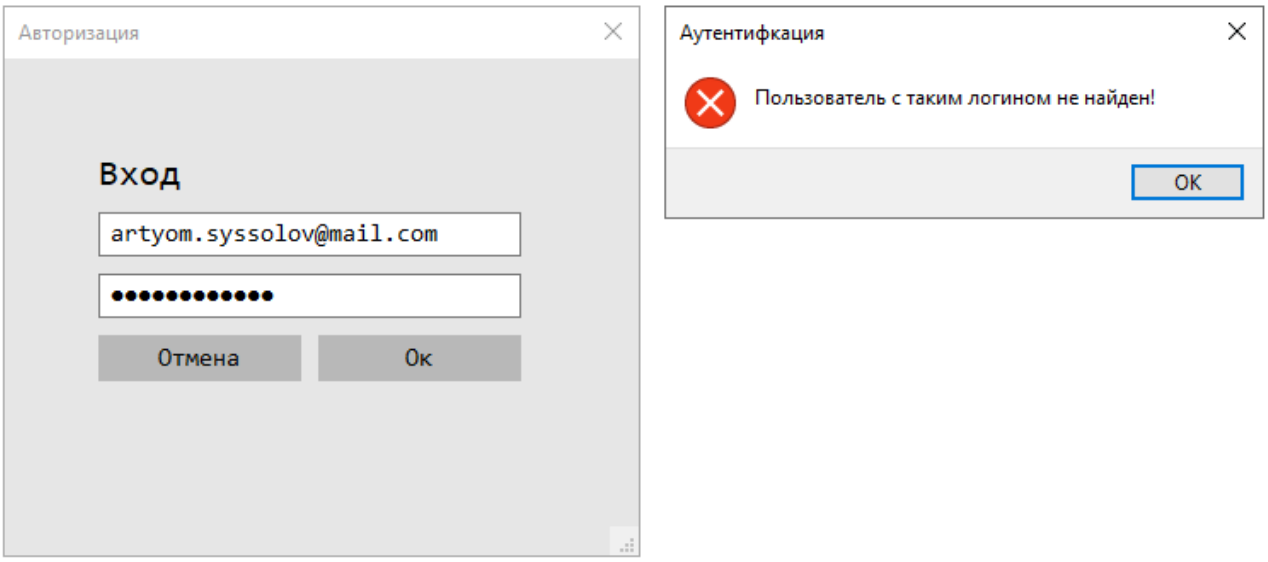

Рисунок 24 - Уведомление сервером о некорректности введенного логина

Система говорит, что такого пользователя нет. И действительно некорректно указан почтовый домен. Исправим и попробуем снова:

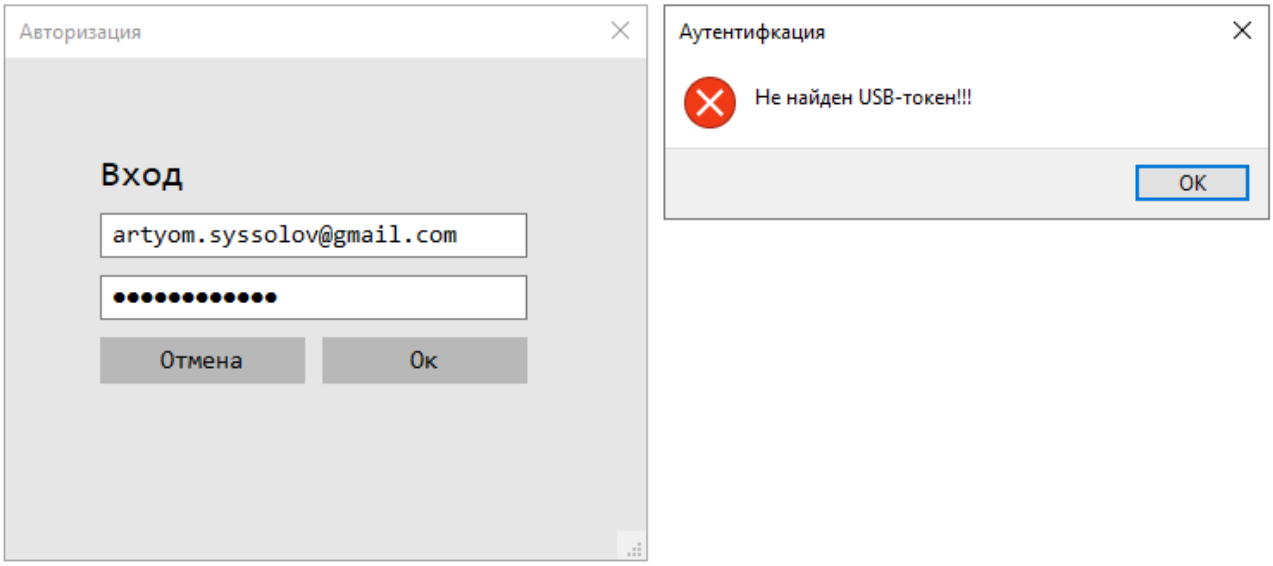

Рисунок 25 - Уведомление программой об отсутствии usb-токена

Уведомление системы об ошибке. На этот раз не найден usb-токен. После того как usb-токен подключен, как показано на иллюстрации необходимо повторить операцию:

Рисунок 26 - Пример подключения usb-токена

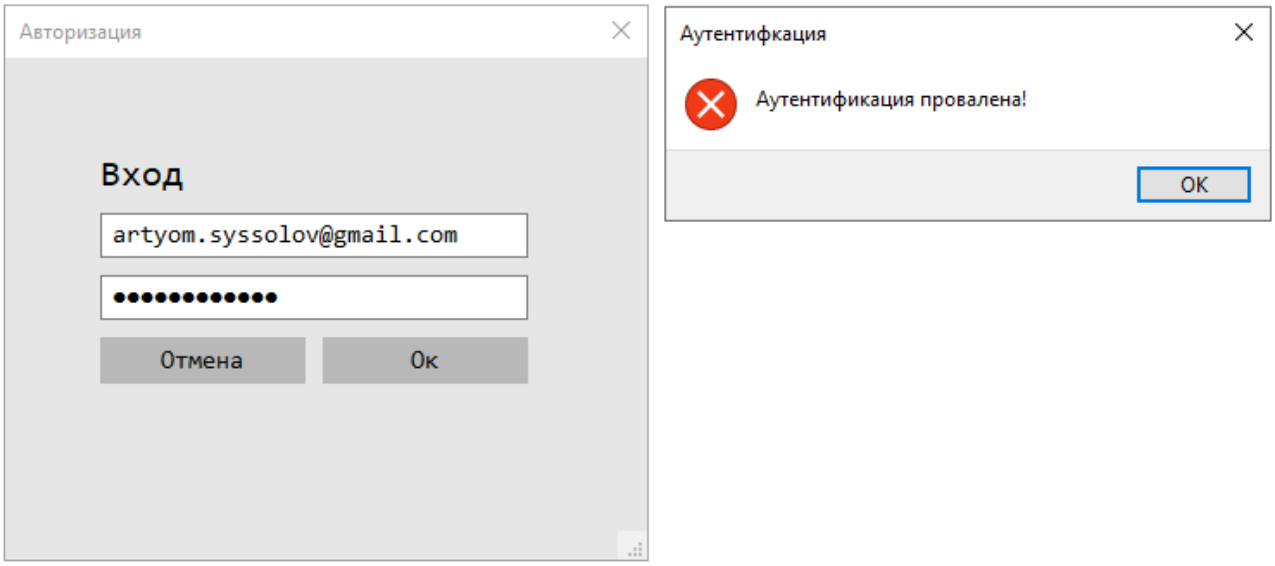

Рисунок 27 - Сервер уведомляет пользователя о некорректно введенном пароле

На этот раз сервер сообщает, что неверно введен пароль. Необходимо ввести корректный логин:

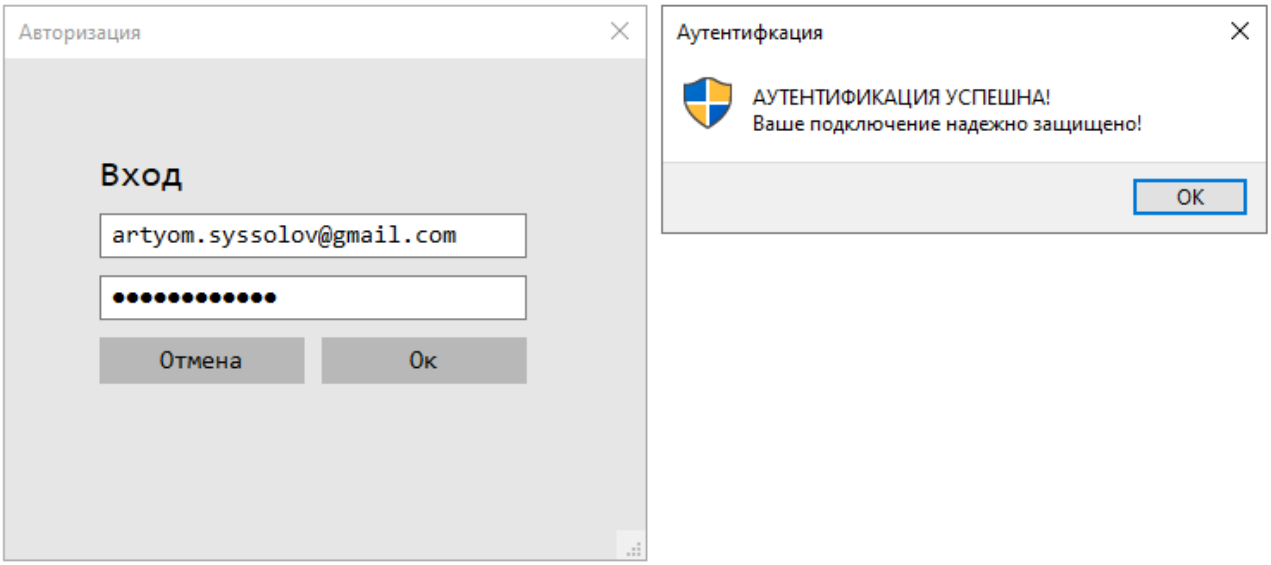

Рисунок 28 - Уведомление об успешности аутентификации

Операция аутентификации прошла успешно, а также был выработан общий сессионный ключ для шифрования потоковых данных.

ЗАКЛЮЧЕНИЕ

Современный мир – это мир бизнеса и больших корпораций. В этом мире ежесекундно производится миллионы транзакций и иных операций. Здесь нарушение хотя бы одного из основных аспектов информационной безопасности может привести к фатальным последствиям. Поэтому сегодня вопросы безопасного соединения, а также вопросы аутентичности участников информационного взаимодействия составляют одну из ключевых проблем, решить которую практически достаточно трудно.

Разработанный в данном дипломном проекте протокол, исходя из проведенных тестов, в полной мере соответствует всем требованиям в вопросах обеспечения конфиденциальности, а также аутентичности участников информационного взаимодействия. Реализованный программный продукт являет собой прикладное решение, которое может быть развернуто и внедрено в той инфраструктуре, где вопросы безопасности стоят на первом месте.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Symmetric key management // Электронная версия на сайте http://cryptowiki.net/index.php?title=Symmetric_key_management

2 Голиков В.Ф. Формирование общего секрета с помощью искусственных нейронных сетей/М.Л. Радюкевич, В.Ф. Голиков//Системный анализ и прикладная информатика. -2019. номер 2.- С. 49-56

3 Kanter, I. The Theory of Neural Networks and Cryptography, Quantum Computers and Computing / I. Kanter, W. Kinzel. - 2005.Vol. 5, n. 1. – P. 130-140.

4 Kinzel, W. Neural Cryptography / W. Kinzel, / I. Kanter // 9th International Conference on Neural Information Processing, Singapore, 2002.

5 Ruttor, A. Dynamics of neural cryptography / A. Ruttor, I. Kanter, and W. Kinzel // Phys. Rev. E, 75(5):056104, 2007.

6 Golikov V. F., Brich N. V., Pivovarov V. L. «On some problems in the distribution of cryptographic keys using artificial neural networks», System Analysis and Applied Informatics, № 1–3, 2014.

7 Plonkovski, M. Cryptographic transformation of information based on neural network technology / M. Plonkovski, P. P. Urbanovich // Proceedings of BSTU. Series VI. Physics and Mathematics and Informatics; by ed. I. M. Zharsky. – Minsk: BSTU, 2005.

8 Прохоренок Н.А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.

Noel Rappin and Robin Dunn. wxPython in Action, March 2006 - 584 с.

wxPython API Documentation // Электронная версия на сайте https://docs.wxpython.org/

9 Арсак, Ж. Программирование игр и головоломок / Ж. Арсак. - М.: 2014. - 501 с.

10 Кнут, Д.Э. Искусство программирования (Том 1. Основные алгоритмы): моногр. / Д.Э. Кнут. - М.: [не указано], 2000. - 700 с.

11 Кнут, Д.Э. Искусство программирования (Том 2. Получисленные алгоритмы) / Д.Э. Кнут. - М.: [не указано], 2000. - 225 с.

12 Кнут, Д.Э. Искусство программирования (том 3) / Д.Э. Кнут. - М.: [неуказано], 1978

13 Роберт Мартин Чистый код. Создание, анализ и рефакторинг. – СПб.: Питер, 2015– 464 с.

14 Фергюсон, Нильс, Шнайер, Брюс.Ф43 Практическая криптография.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2004. — 432 с.: ил. — Парал. тит. Англ.

Приложение А

В данном приложении показана наглядная демонстрация данных в процессе формирования общего сессионного ключа между клиентом и сервером, а также аутентификации пользователя сервером. Перехват трафика осуществлялся с помощью утилиты Wireshark.

Для простоты и наглядности в данном примере сервер был запущен на loopback интерфейсе.

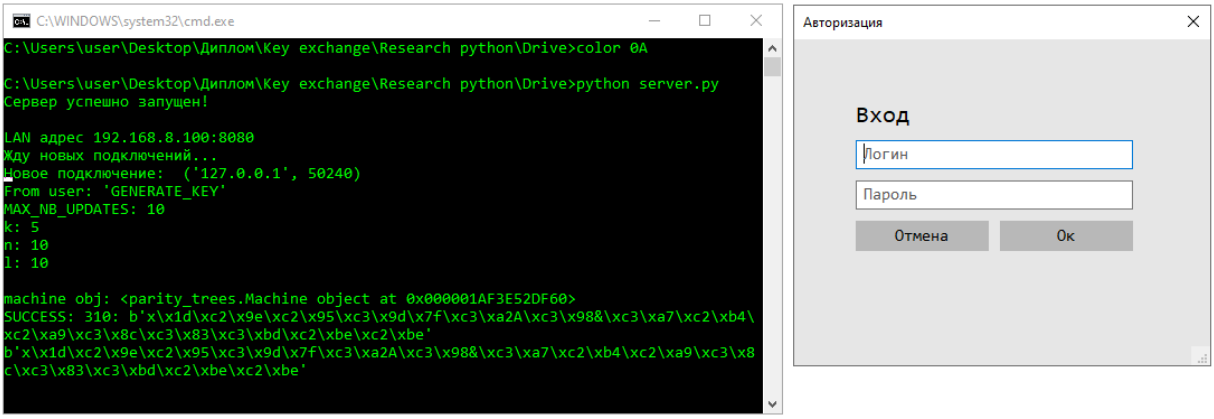

Рисунок А.1 - Запущенный сервер и клиент

Как можно заметить из рисунка выше, клиент, после подключения отправил серверу команду «GENERATE_KEY». В ответ на что сервер отправил клиентскому приложению значение «CAN_START» и параметры для инициализации TPM [2]. Как было отмечено ранее в данном дипломном проекте, после успешного формирования общего ключа K_{cs}, весь трафик шифруется на ключе K_{cs}.

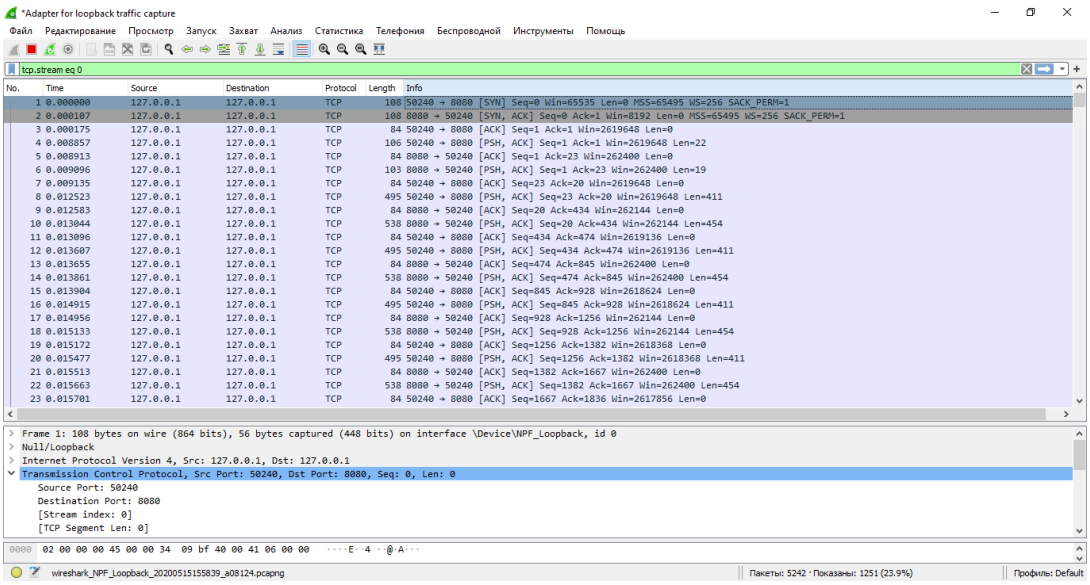

Рисунок А.2 - Перехваченные утилитой Wireshark пакеты

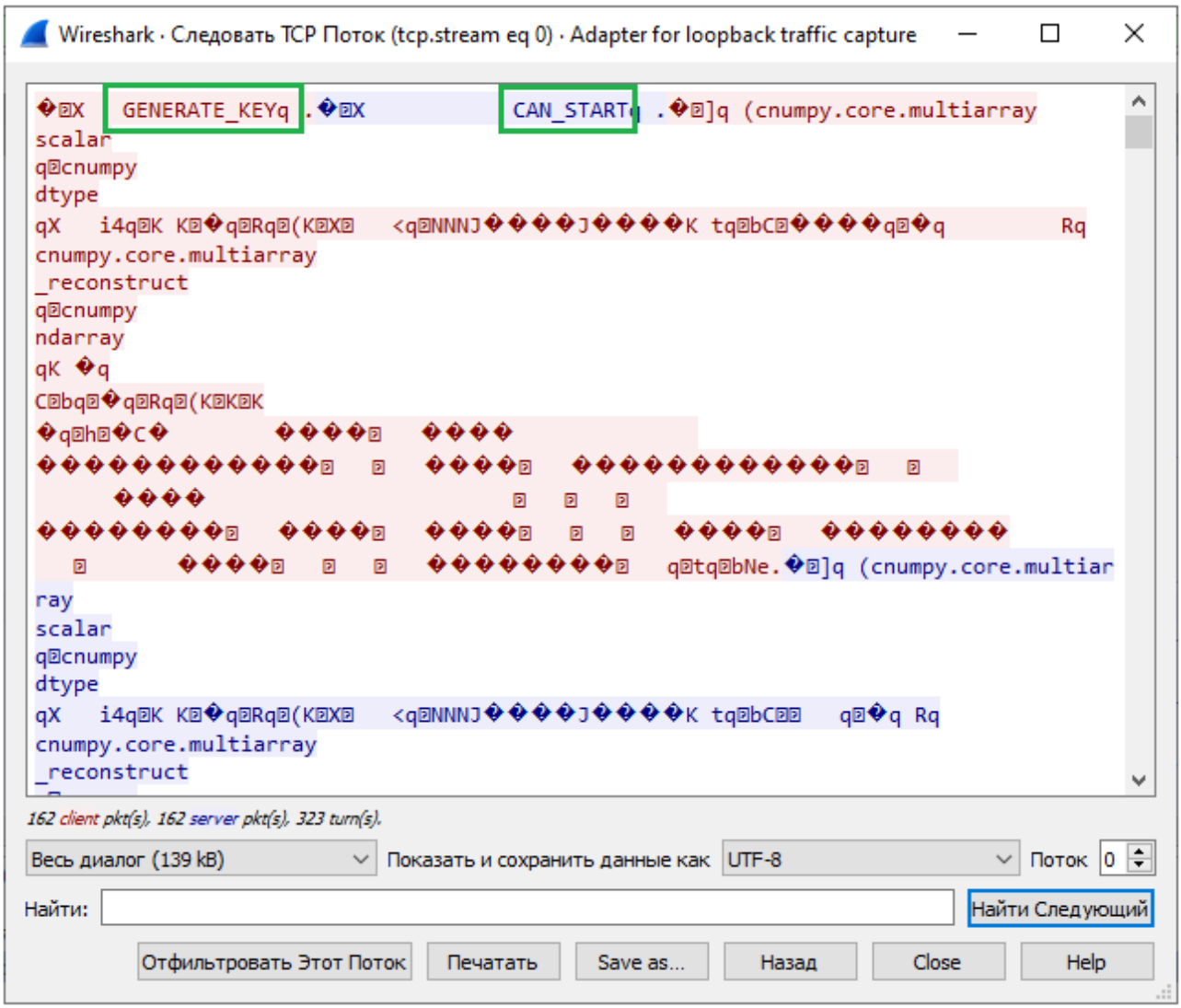

Рисунок А.3 - Перехваченный tcp-пакет в Wireshark

Рисунок А.3 отчетливо демонстрирует значение «GENERATE_KEY» отправленное в направлении клиент-сервер и значение «CAN_START» в направлении сервер-клиент. После чего началась выработка общего ключа, выраженная передачей массивов в обе стороны.

Последующие перехваченные пакеты уже передаются в полностью зашифрованном виде.

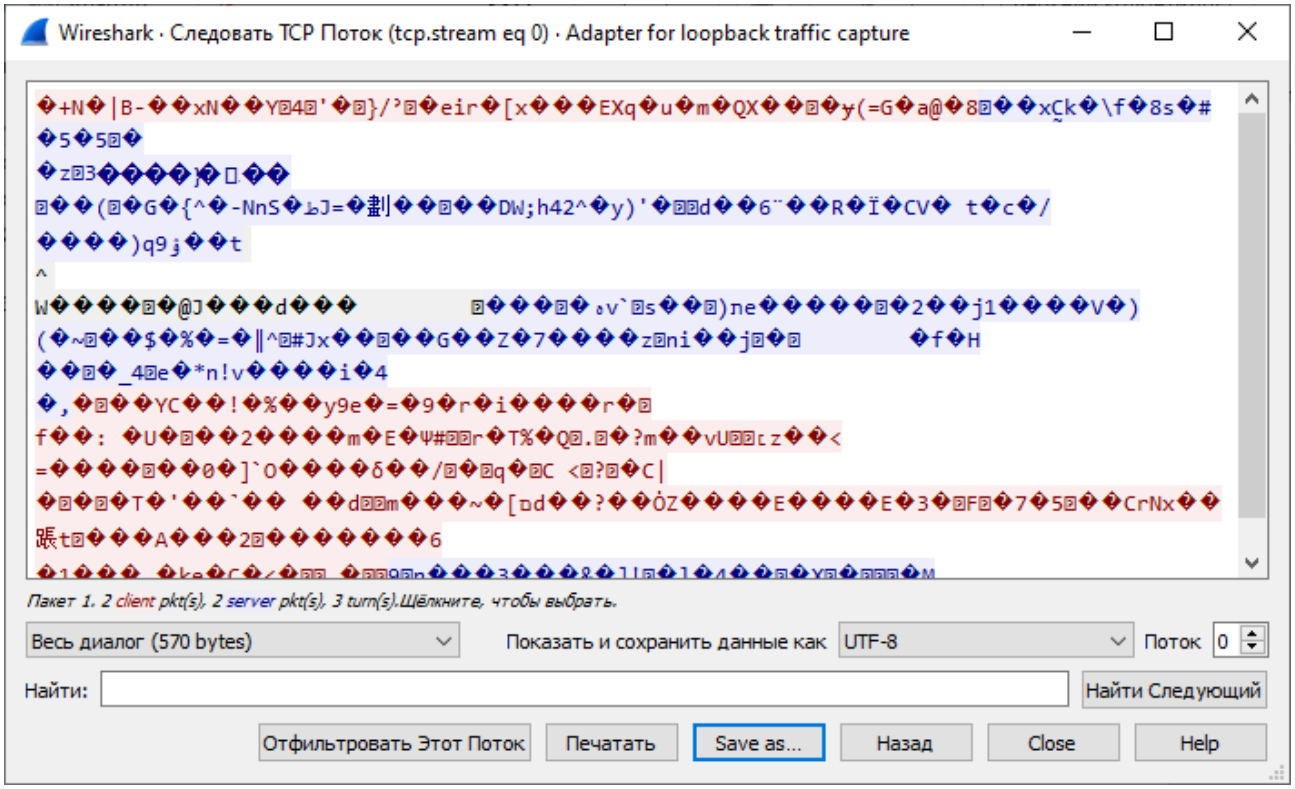

Рисунок А.4 - Перехваченный tcp-пакет в Wireshark после формирования общего ключа между клиентом и сервером

Приложение Б

Листниг модуля выработки общего ключа по алгоритму TPM [2] с участием криптоаналитика

```
# библиотека для работы с массивами и матрицами + поодержка математичских вычслений
import numpy as np

# библиотека для построения графиков
import matplotlib.pyplot as plt

import hashlib
import time
import sys

import pandas as pd
from tqdm import tqdm_notebook as tqdm

def theta(t1, t2):
    # вернуть 1 если аргументы равны и 0 в противном случае
    return 1 if t1 == t2 else 0


# функции обновления весов:
def hebbian(W, X, sigma, tau1, tau2, l):
    k, n = W.shape
    for (i, j), _ in np.ndenumerate(W):
        W[i, j] += X[i, j] * tau1 * theta(sigma[i], tau1) * theta(tau1, tau2)
        W[i, j] = np.clip(W[i, j] , -l, l)

def anti_hebbian(W, X, sigma, tau1, tau2, l):
    k, n = W.shape
    for (i, j), _ in np.ndenumerate(W):
        W[i, j] -= X[i, j] * tau1 * theta(sigma[i], tau1) * theta(tau1, tau2)
        W[i, j] = np.clip(W[i, j], -l, l)

def random_walk(W, X, sigma, tau1, tau2, l):
    k, n = W.shape
    for (i, j), _ in np.ndenumerate(W):
        W[i, j] += X[i, j] * theta(sigma[i], tau1) * theta(tau1, tau2)
        W[i, j] = np.clip(W[i, j] , -l, l)

# перевод массива в десятичную систему счисления
```

```
# num_sys   - система счисления
# key       - ключ в десятичнй системе счисления
# Задается смешение на +L для всех весовых коэфицентов, т.к.  L∈{-L, L} => теперь L∈{0, L*2}
# Теперь вектор весов W представляется в полиномиальном виде: key = sum (W[i] * num_sys ^ i)

def get_key_deg(w, l):
    num_sys = l*2 + 1               # вычисляем систему счисления
    dim = w.shape                   # получаем размерность вектора весов
    res = w.reshape(dim[0]*dim[1])    # получаем длину результирующего вектора: 1измерение * 2измерение

    key = 0
    for i, v in enumerate(res):      # пробегаемся в цикле по всем весам
        v += l                      # задаем смещение
        key += v * (num_sys**i)      # прибавляем к ключу значение i-го веса умнож. на сист сч. в степени позиции (i)
#         key += v * (i)

    key = str(key)
    step = 3

    length = len(key)//step # определяем близжайшее, кратное шагу число
    key = key[:length*step] #доведение до длины кратной шагу
    new_key = '' #
    for i in range(length):
        # пробегаемся по всему ключу
        pos = i*step
        # срез по три числа -> их ковертация в int -> mod 255(16) -> char
        num = chr(int(key[pos:pos+step])%0xFF)
        new_key += num # конкатенация

    # byte_key = bytes(new_key, 'ascii')
    byte_key = bytes(new_key, 'utf-8')    # перевод в байты
    byte_key[:16]                       # срез первых 16 битов

    return byte_key


def get_key_bin(W, L, binary=True, splice=False, ret_stat=False):
    W = W.reshape(( W.shape[0]*W.shape[1]))
```

```
#    print(W.shape[0])
   key = ''
   max_len = len("{0:b}".format(L+L))
   stat = {'zeros': 0, 'ones': 0}
#    print("len of l: {}".format(max_len))

   for w in W:
      w += L

      n=w
      w  = "{0:b}".format(w)

      if not splice:
         w = "{}{}".format('0'*(max_len-len(w)), w)
#          print("{}) {}".format(n, w))

      key += w

   step = 3

   key = int(key, 2)
   key = str(key)
   print(key)

   length = len(key)//step # определяем близжайшее, кратное шагу число
   key = key[:length*step] #доведение до длины кратной шагу
   new_key = '' #
   for i in range(length):
      # пробегаемся по всему ключу
      pos = i*step
      # срез по три числа -> их ковертация в int -> mod 255(16) -> char
      num = chr(int(key[pos:pos+step])%0xFF)
      new_key += num # конкатенация

   # byte_key = bytes(new_key, 'ascii')
   byte_key = bytes(new_key, 'utf-8')    # перевод в байты
   byte_key[:16]                         # срез первых 16 битов

   return byte_key

class Machine: # класс Machine
```

```
  # Машина четности деревьев. Генерирует двоичную цифру (tau) для входного случайного вектора (X).
   # Машиа может быть описана следующими параметрами
   # k - количество скрытых нейронов
   # n - количество входных нейронов сети
   # l - задает числовой даипазон для каждого веса W[i] ∈ W
   # W - матрица весов, между входным и скрытыми слоями сети. Ее измерение: [K, N]

   def __init__(self, k=3, n=4, l=6):

      # Аргументы:
      # k - количество скрытых нейронов
      # n - количество входных нейронов, соеденeнных с каждым скрытым нейроном
      # l - адает числовой даипазон для каждого веса ({-L, ..., -2, -1, 0, 1, 2, ..., +L })

      self.k = k
      self.n = n
      self.l = l
      self.W = np.random.randint(-l, l + 1, [k, n]) # инициализируем веса случайным образом

   def get_output(self, X):
      # возвращает бинарную цифру tau для входного случайного вектора
      # Аргументы:
      # X - входной случайный вектор

      k = self.k
      n = self.n
      W = self.W
      X = X.reshape([k, n]) # делаем форму вектора X с [1, k*n] на [k, n]

      sigma = np.sign(np.sum(X * W, axis=1)) # Вычисляем знак от суммы весов помноженных на соотвественный входной X
      tau = np.prod(sigma) # вычисляем произведенеие всех выходных значений sigma

      self.X = X
      self.sigma = sigma
      self.tau = tau
```

```
        return tau

    def __call__(self, X):
        # при вызове объекта класса Machine - объект будет возвращать результат от get_output(X)
        return self.get_output(X)

    def do_hash(self, number):
        # вычисляет hash от переданного числа по алгоритму 'sha256'
        m = hashlib.sha256()
        m.update(str(number).encode())
        return m.digest()

    def get_key_hash(self):
        # возвращает hash от ключа, где ключ вычисляется путем представления вектора весов в виде полинома
        key = get_key(self.W, self.l)
        key = str(key).encode()
        hash_key = self.dwo_hash(key)
        return hash_key

    def update(self, tau2, update_rule='hebbian'):
        # Обновляет веса по правилу обновления
        # Аргументы:
        # tau2 - выходное значение второй сети, с которой происходит синхронизация
        # update_rule - правило обновления весов W
        # Может быть одним из ['hebbian', 'anti_hebbian', random_walk']

        X = self.X
        tau1 = self.tau
        sigma = self.sigma
        W = self.W
        l = self.l

        if (tau1 == tau2):
            if update_rule == 'hebbian':
                hebbian(W, X, sigma, tau1, tau2, l)
            elif update_rule == 'anti_hebbian':
                anti_hebbian(W, X, sigma, tau1, tau2, l)
            elif update_rule == 'random_walk':
                random_walk(W, X, sigma, tau1, tau2, l)
```

```
        else:
            raise Exception("Invalid update rule. Valid update rules are: " +
                "\'hebbian\', \'anti_hebbian\' and \'random_walk\'.")

# Задаем параметры сети:
k = 24
n = 24
l = 10

# Правило обновления весов:
update_rules = ['hebbian', 'anti_hebbian', 'random_walk']
update_rule = update_rules[0]

# Создаем три машины : Алиса, Боб и Ева. Здесь Ева будет пытаться синхронизироваться с Бобом и Алисой
print("Создаем машину со значениями: k={}, n={}, l={}".format(k, n, l))
print("Правило обновлния весов: {}".format(update_rule))

# Создание объектов:
Alice = Machine(k, n, l)
Bob = Machine(k, n, l)
Eve = Machine(k, n, l)

oldAW = Alice.W.copy()
oldBW = Bob.W.copy()

# Генератор случайного вектора, в диапазоне от -L до L, размерностью: [k, n]
def random():
    return np.random.randint(-l, l + 1, [k, n])

# Функция, которая будет оценивать уровень синхронизации между двумя сетями
def sync_score(m1, m2):
    # Аргументы:
    # m1 - объект типа <Machine> №1
    # m2 - синхронизируемый объект типа <Machine> №2 с объектом №1
    # возвращет число в диапазоне: [0, 1]
    return 1.0 - np.average(1.0 * np.abs(m1.W - m2.W)/(2 * l))

# Синхронизация весов
sync = False # Флаг, котрый проверяет синхронизацию
nb_updates = 0 # Счетчик количества обновлений весов м\у Алисой и Бобом
```

```
nb_eve_updates = 0 # Счетчик количества обновлений весов м\у Алисой\Бобом и Евой
start_time = time.time() # Время при старте синхронизации

# Массивы для построения графиков, отражающих динамику синхронизации:
sync_history = [] # Массив, хранящий nb_updates
eve_sync_history = [] # Массив, хранящий nb_eve_updates

# xs = []
# xs.append(Alice.W.copy())
# xs.append(Bob.W.copy())

while(not sync): # до тех пор, пока не произойдет синхронизация

    X = random() # Создаем случайный вектор X измерения: [k, n]
#     xs.append(X)
    tauA = Alice(X) # Получаем выход машины Алисы
    tauB = Bob(X) # Получаем выход машины Боба
    tauE = Eve(X) # Получаем выход машины Евы

    Alice.update(tauB, update_rule) # Обновляем веса Евы исходя из выхода сети Боба
    Bob.update(tauA, update_rule) # Обновляем веса Боба исходя из выхода сети Алисы

    # Веса сети Евы будут обновляться, только если выходы: tauA = tauB = tauE
    if tauA == tauB == tauE:
#     if tauA == tauE:
        Eve.update(tauA, update_rule)
        nb_eve_updates += 1
#         eve_sync_history.append(eve_score)

    nb_updates += 1 # увеличиваем счетчик обновлений

    score = 100 * sync_score(Alice, Bob) # Вычисляем синхронизацию машин Алисы и Боба (в процентах)
    eve_score = 100 * sync_score(Eve, Alice) # Вычисляем синхронизацию машин Евы и Алисы (в процентах)

    sync_history.append(score) # Добавляем полученный результат 'score' к истории sync_history
    eve_sync_history.append(eve_score) # <-
```

```
    sys.stdout.write("\rСинхронизация на {}%  /  Обновлений = {} / Обновлений Евы = {}".format(round(score, 1), nb_updates, nb_eve_updates))
    if score == 100: # If synchronization score is 100%, set sync flag = True
        sync = True

end_time = time.time() #
time_taken = time.time() - start_time # Вычисляем затраченное время, как разницу текущего и start_time

# печатаем результаты:
print ('\nМашины синхронизированы.')
print ('Затрачено времени = {} сек'.format(time_taken))
print ('Всего обновлений = {}.'.format(nb_updates))

# Посмотрим на сколько Ева синхронизировалась с машинами Алисы и Боба:
eve_score = 100 * sync_score(Alice, Eve)
if eve_score >= 100:
    print("Ой-ей! Ева синхронизировала свою машину с машинами Алисы и Боба :(")
else:
    print("Машина Евы синхронизировалась всего на {}% с машинами Алисы и Боба! Обновлений у Евы: {}.".format(round(eve_score, 1), nb_eve_updates))

# get_key_bin(Alice.W, l, binary=True, ret_stat=True, splice=True)
key_alice = get_key_deg(Alice.W, Alice.l) # получам ключ Алисы
key_bob  = get_key_deg(Bob.W, Bob.l)      # получам ключ Боба
# key_alice = get_key_bin(Alice.W, Alice.l) # получам ключ Алисы
# key_bob  = get_key_bin(Bob.W, Bob.l)      # получам ключ Боба

# Сравниваем ключи Алисы и Боба, а затем печатаем 'true' если да и false в ином случае
print("Ключ Алисы = ключу Боба: {};".format(key_alice==key_bob))
# Выводим длину ключа в количестве позиций десятичного числа:
print("Длина ключа: {} бит".format(len(str(key_alice))))
print("Key: {}...".format(key_alice))
```

Листниг серверной части

```
import socket, threading, time
import re, os, random
import pickle
```

```
from tags import *
from parity_trees import generate_key
from tools import Cryptor, Connector



def select_user(login):
	with Connector() as cur:
		QUERY = '"SELECT secret_key, repository, flash_id
				FROM LEGAL_USERS WHERE
login==\'{}\'"'.format(login)

		res = cur.execute(QUERY).fetchone()
		if res:
			secret_key, repository, flash_id = res

		print(secret_key, repository, flash_id)

# select_user('root')

class ServerThread(threading.Thread):
	# static variable

	def __init__(self, clientAddress, clientSocket):
		threading.Thread.__init__(self)
		self.daemon		= True

		print ("Новое подключение: ", clientAddress)

		self.repository = 'repository'
		self.mKeysDir = 'master-keys'
		self.clientAddress  = clientAddress
		self.clientSocket    = clientSocket

		self.BYTES_NUM = 2048
		self.shared_key = None
		self.login		= None
		self.secret_key	= None
		self.repository	= None
		self.flash_id = None
		self.cryptor  = None
```

```
            self.IM_LEGAL = False

        def run(self):
            # print ("Connection from : ", self.clientAddress)
            while True:
                try:
                    # out_data = "Hello from Server!"
                    command = self.clientSocket.recv(self.BYTES_NUM)

                    if self.shared_key:
                        command = self.cryptor.decrypt(command)

                    command = self.load_data(command)
                    print("From user: '{}'".format(command))
                    self.command_handler(command)


                except socket.error as sock_err:
                    print(sock_err)
                    self.clientSocket.close()
                    break
                except Exception as exc:
                    print(exc)
                finally:
                    pass

                # threads[self.clientAddress[0]] = ClientThread(self.clientAddress,
self.clientSocket).start()
            # generate_key(self.csocket, spawned=False, name="SERVER")

        def command_handler(self, command):
            LEGAL_CMD = ['CONTENT', 'REGISTER', 'GENERATE_KEY',
'##AUTH##']
            #          0        1        2            3


            if LEGAL_CMD[0] in command:
                print("From client -> '{}'".format(command))
                content = os.listdir(self.repository)


                content = self.pack_data(content)
```

```
                self.clientSocket.sendall(content)

            if LEGAL_CMD[1] in command:
                # add checker [if user legal] <- !!!!!!!!
                users.append((self.clientAddress[0],       str(self.clientAddress[1]),
None))
                print(users)
                sockets[self.clientAddress] = self.clientSocket
                # out_data = self.pack_data('NO')
                out_data = self.pack_data('OK')
                self.clientSocket.sendall(out_data)

            if LEGAL_CMD[2] in command: # GENERATE_KEY
                # params = [16,16,8,100,'hebbian']
                response = 'CAN_START'
                out_data = self.pack_data(response)
                self.clientSocket.sendall(out_data)
                key = generate_key(self.clientSocket)
                self.shared_key = key
                self.cryptor = Cryptor()
                self.cryptor.set_key(key)
                print(key)

            if LEGAL_CMD[3] in command: # ##AUTH##
                self.login = command.replace(LEGAL_CMD[3], '')
                print(self.login)
                with Connector() as cur:
                    #              0         1         2
                    QUERY = '''SELECT secret_key, repository, flash_id
                                FROM          LEGAL_USERS          WHERE
login=='{}'''.format(self.login)

                    res = cur.execute(QUERY).fetchone()

                    response = '##UNF##' # UNF = user not found
                    if res:
                        self.secret_key      = res[0]
                        self.repository      = res[1]
                        self.flash_id = res[2]

                        response = self.secret_key # имя ключа HASH
```

```
                        self.sync_num = str(random.randint(2**100, 2**500))

                        response = self.pack_data(response + '##SYNC##' +
self.sync_num)
                        response = self.cryptor.encrypt(response)
                        self.clientSocket.sendall(response)

                        if not res: raise Exception('Пользователь не найден!')

                        with  open(os.path.join(self.mKeysDir,self.secret_key),  'rb')
as SMKey:
                                SMKey = SMKey.read()
                                print(SMKey)

                        digest = self.clientSocket.recv(self.BYTES_NUM)
                        digest = self.cryptor.decrypt(digest)
                        tmpCryptor = Cryptor()
                        tmpCryptor.set_key(SMKey)
                        digest = tmpCryptor.decrypt(digest)

                        response = '##UNCKNOWN_ERROR##'
                        try:
                                digest = self.load_data(digest)
                                print("-------> [digest]:", digest)
                                if          digest['digest'][0]==self.flash_id          and
digest['digest'][1]==self.sync_num:
                                        print('SUCCESSFULL!')
                                        self.IM_LEGAL = True
                                        response = '##OK##'
                        except:
                                response = '##AUTH_FAILED##'
                                print('Что-то не так с паролем!')


                        response = self.pack_data(response, crypto=True)
                        self.clientSocket.sendall(response)

        def pack_data(self, data, crypto=False):
                data = pickle.dumps(data)

                if crypto:
                        data = self.cryptor.encrypt(data)
```

```
                return data

        def load_data(self, data, crypto=False):
                if crypto:
                        data = self.cryptor.decrypt(data)
                data = pickle.loads(data)

                return data

def my_ip():
        import requests
        session = requests.Session()
        result = session.get('http://icanhazip.com/')
        return result.text.split('\n')[0]

# WAN_IP = my_ip()
LAN_IP  = socket.gethostbyname(socket.gethostname())
HOST = "0.0.0.0"
# HOST = "127.0.0.1"
# HOST = ""
PORT = 8080
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((HOST, PORT))
# server.bind((socket.gethostbyname(socket.gethostname()), PORT ))

print("Сервер успешно запущен!")
print("\nLAN адрес {}:{}".format(LAN_IP, PORT))
# print("WAN адрес {}:{}\n".format(WAN_IP, PORT))
print("Жду новых подключений...")

try:
        while True:

                server.listen(1)
                clientSock, clientAddress = server.accept()
                newthread = ServerThread(clientAddress, clientSock)
                newthread.start()



except KeyboardInterrupt as kbi:
```

```
        print("Отмена. Сервер выключен...")
```

Листинг модуля клиентской части

```
import time, sys
import wx

app = wx.App()

# loading = wx.lib.busy.BusyInfo("Pre-Rolling media - Pleasewait!")
loading = wx.BusyInfo('Загрузка...')

try:
        with loading:
                from wx import (Button, BoxSizer, StaticText)
                from wx.lib.pubsub import pub

                start = time.time()
                import socket, threading, pickle, re


                from tags import *
                from parity_trees import generate_key
                from tools import (Cryptor, find_flash, do_hash, exec_cmd, DEVICE_ID,
VOL_SERIAL_NUM)
                print("[{}]: {}".format('IMPORTS', time.time()-start))
except Exception as exc:
        print(exc)
        sys.exit(0)
finally:
        del loading



class Updater(threading.Thread):
        def __init__(self):
                super().__init__()
                self.daemon = True

                # self.SERVER = "192.168.8.101"
                self.SERVER = "localhost"

                # self.SERVER = "45.79.123.107" # Linode
```

```
            # self.SERVER = "95.163.214.65"
            self.PORT = 8080
            print("Инициализация порта...")

            self.BYTES_NUM = 2048
            self.all_files = []
            self.shared_key = None

            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


            self.verbose = True

            self.LEGAL_CMD = ['CONTENT', 'REGISTER', 'GENERATE_KEY',
'##AUTH##']
            #              0       1       2            3

    def set_parrent(self, parent):
            self.parent = parent

    def establish_connection(self):
            while True:
                    try:
                            self.socket.connect((self.SERVER, self.PORT))
                            print('Соединение с сервером успешно установлено!')
                            break
                    except Exception as exc:
                            print(exc)
                            time.sleep(1)
            return

    def run(self):
            with wx.BusyInfo('Соединение...'):
                    self.establish_connection()

            with wx.BusyInfo('Выработка общего ключа...'):
                    satrt_g = time.time()
                    self.key_generating()
                    end_g = round(time.time()-satrt_g, 2)

            if self.verbose:
```

```
                msg = 'Общий ключ успешно сформирован!\nВаше соединение надежно защищено.'
                msg += '\nЗатраченное время: {} сек'.format(end_g)
                wx.MessageDialog(self.parent, msg, 'Общий ключ', wx.OK|wx.ICON_ASTERISK).ShowModal()


        while True:
            try:
                print("[Updater]: I'm still alive")
            except Exception as exc:
                print(exc)
            finally:
                time.sleep(3)

    def retriveContent(self):
        command = self.pack_data('CONTENT')
        self.socket.sendall(command)
        content = self.socket.recv(self.BYTES_NUM)
        self.all_files = self.load_data(content)
        print(self.all_files)

        if len(self.all_files)==0: return
        for file in self.all_files:
            if file not in self.parent.files.GetItems():
                # self.anchor_elem.Clear()
                self.parent.files.Append(file)
        for index, item in enumerate(self.parent.files.GetItems()):
            if item not in self.all_files:
                self.parent.files.Delete(index)

    def key_generating(self):
        command = self.pack_data('GENERATE_KEY')
        self.socket.sendall(command)

        response = self.socket.recv(self.BYTES_NUM)
        response = self.load_data(response)
        if response=='CAN_START':
            key = generate_key(self.socket, spawned=True)
            self.shared_key = key
            ## <--------< creating CRYPTO object  >-------->
            self.cryptor = Cryptor()
```

```
                self.cryptor.set_key(key)
                ## >--------< creating CRYPTO object  >--------<
            print(key)
            return

    def auth(self, login, passwd):
            try:
                with wx.BusyInfo('Аутентификация...'):
                    # теперь данные шифруются
                    command = self.pack_data(self.LEGAL_CMD[3]+login,
crypto=True)
                    self.socket.sendall(command)

                    response = self.socket.recv(self.BYTES_NUM)
                    response = self.load_data(response, crypto=True)
                    try:
                        self.enc_key_file, self.sync_num =
response.split('##SYNC##')
                    except Exception as exc:
                        print("[SYNC ERROR]: ", exc)
                        raise Exception('Получены некорректные данные с
сервера')

                print("[self.key_name]: {}".format(self.enc_key_file))
                print("[self.sync_num]: {}".format(self.sync_num))

                if self.enc_key_file=='##UNF##':
                    raise Exception('Пользователь с таким логином не
найден!')

                # self.master_key
                path2mKey, serial_num = None, None
                try:
                    path2mKey, serial_num = find_flash(self.enc_key_file)
                except Exception as exc:
                    print('EXCEPTION FLASH: ', exc)
                    raise Exception("Не найден USB-токен!!!")

                # if path2mKey==None or serial_num==None:
                #     raise Exception("Не найден USB-токен!!!")
                    print("Master key: '{}'\nserial_num:
'{}'".format(path2mKey, serial_num ))
```

```
                with open(path2mKey, 'rb') as masterKey:
                    masterKey = masterKey.read()
                    tmpCryptor = Cryptor()
                    tmpCryptor.set_key(passwd)
                    masterKey = tmpCryptor.decrypt(masterKey)
                    # masterKey = self.load_data(masterKey)
                    print(len(masterKey), masterKey)

                    # <--< зашифрование serialNumber+rand_num на мастер ключе >-->
                    # с последующей отправкой
                    tmpCryptor.set_key(masterKey)

                    digest = {'digest': [serial_num, self.sync_num]}
                    digest = self.pack_data(digest)
                    digest = tmpCryptor.encrypt(digest)    # шифрование на мастер ключе
                    digest = self.cryptor.encrypt(digest) # шифрование на нейросетевом ключе
                    self.socket.sendall(digest)
                    # >--< зашифрование serialNumber на мастер ключе >--<

                    response = self.socket.recv(self.BYTES_NUM)
                    response = self.load_data(response, crypto=True)

                    if response=='##OK##':
                        msg = 'АУТЕНТИФИКАЦИЯ УСПЕШНА!\nВаше подключение надежно защищено!'
                        dlg = wx.MessageDialog(self.parent, msg, 'Аутентифкация', wx.OK|wx.ICON_MASK)
                        dlg.ShowModal()
                    if response=='##AUTH_FAILED##':
                        raise Exception('Аутентификация провалена!')
                    if response=='##UNCKNOWN_ERROR##':
                        raise Exception('На сервере произошла ошибка!')

            except Exception as exc:
                print(exc)
                dlg = wx.MessageDialog(self.parent, str(exc), 'Аутентифкация', wx.OK|wx.ICON_ERROR)
                dlg.ShowModal()
```

```
        def pack_data(self, data, crypto=False):
                data = pickle.dumps(data)
                if crypto:
                        data = self.cryptor.encrypt(data)

                return data

        def load_data(self, data, crypto=False):
                if crypto:
                        data = self.cryptor.decrypt(data)
                data = pickle.loads(data)

                return data

class LoginGUI(wx.Dialog):
        def __init__(self, parent, title):
                super(LoginGUI, self).__init__(parent, title = title, size = (800,600))

                self.fontSize = 10

                # with wx.BusyInfo('Загрузка...'):
                start = time.time()
                self.InitUI()
                print("[{}]: {}".format('InitUI', time.time()-start))
                start = time.time()
                self.Centre()
                print("[{}]: {}".format('Centre', time.time()-start))
                start = time.time()
                self.Show()
                print("[{}]: {}".format('Show', time.time()-start))

        def set_thread(self, thread_obj):
                self.thread_obj = thread_obj

        def InitUI(self):
                self.Bind(wx.EVT_CLOSE, self.onCloseFrame)
                self.SetWindowStyle( wx.RESIZE_BORDER | wx.SYSTEM_MENU |
wx.CAPTION


                 | wx.CLOSE_BOX | wx.CLIP_CHILDREN
                 | wx.FRAME_NO_TASKBAR )
```

```
        self.SetBackgroundColour((230,230,230))
        # font =wx.Font(12, wx.FONTFAMILY_SWISS, wx.NORMAL, wx.FONTSTYLE_NORMAL, False, u'Consolas')
        font = wx.Font(11, wx.FONTFAMILY_SWISS, wx.NORMAL, wx.NORMAL, False, u'Consolas')
        font2 = wx.Font(16, wx.FONTFAMILY_ROMAN, wx.NORMAL, wx.NORMAL, False, u'Consolas')


        main_sizer = wx.FlexGridSizer(rows = 3, cols = 2, hgap = 2, vgap = 12)

        components = wx.BoxSizer(wx.VERTICAL)
        btns_ok_cnl = wx.BoxSizer(wx.HORIZONTAL)


        input_width = 248
        btn_width    = input_width//2-5
        border          = 100
        window_width    = input_width + border*2 -60
        window_height   = 130 + border*2
        self.SetSize(wx.Size(window_width, window_height))


        enter_lbl = wx.StaticText(self, label="Вход", style=wx.ALIGN_CENTER)
        enter_lbl.SetFont(font2)
        # <-----------< поле для ввода логина >------------>
        # user_lbl = wx.StaticText(self, label="Логин:")
        self.login = wx.TextCtrl(self, size=(input_width, -1))
        self.login.SetHint('Логин')
        self.login.SetFont(font)
        # >-----------< поле для ввода логина >------------<

        # <-----------< поле для ввода пароля >------------>
        # p_lbl = wx.StaticText(self, label="Пароль:")
        self.password = wx.TextCtrl(self, size=(input_width, -1), style=wx.TE_PASSWORD|wx.TE_PROCESS_ENTER)
        self.password.SetHint('Пароль')
        self.password.SetFont(font)
        # self.password.SetWindowStyleFlag(wx.NO_BORDER)
        # >-----------< поле для ввода пароля >------------<
```

```
            # <-----------< кнопка "OK" >------------>
            ok_btn = wx.Button(self, wx.ID_ANY, label='Ок', size=(btn_width, -1))
            ok_btn.SetBackgroundColour((184,184,184))
            ok_btn.SetWindowStyleFlag(wx.NO_BORDER)
            ok_btn.Bind(wx.EVT_BUTTON, self.onLogin)
            ok_btn.SetFont(font)
            # >-----------< кнопка "OK" >------------<

            # <-----------< кнопка "Отмена" >------------>
            cancel_btn = wx.Button(self, wx.ID_ANY, label='Отмена',
size=(btn_width, -1))
            cancel_btn.SetBackgroundColour((184,184,184))
            cancel_btn.SetWindowStyleFlag(wx.NO_BORDER)
            cancel_btn.Bind(wx.EVT_BUTTON, self.onCloseFrame)
            cancel_btn.SetFont(font)
            # >-----------< кнопка "Отмена" >------------<

            # <-------< компановка элментов >------->
            btns_ok_cnl.Add(cancel_btn, 0, wx.ALL, 5)
            btns_ok_cnl.Add(ok_btn,     0, wx.ALL, 5)

            components.Add(enter_lbl,       0, wx.ALL, border=5)
            components.Add(self.login,      0, wx.ALL, border=5)
            components.Add(self.password,   0, wx.ALL, border=5)
            components.Add(btns_ok_cnl,  0, wx.ALL, border=0)
            # >-------< компановка элментов >-------<

            # <--< добавление скомпанованного слоя в главный слой >-->
            # лавный layout, который содержит все остальное
            # создан для того, чтоб добавить отступ по краям
            main_box = wx.BoxSizer(wx.VERTICAL)
            main_box.Add(components, 0, wx.ALL, border=border//2)
            # >--< добавление скомпанованного слоя в главный слой >--<

            self.SetSizer(main_box)

            return


        def onLogin(self, event):
            usr_passwd = self.password.GetValue()
            usr_login = self.login.GetValue()
```

```
                print("Login: {}\nPasswd: {}\n{}".format(usr_passwd, usr_login, '-'*10))

                if bool(usr_login) and bool(usr_passwd):
                        self.thread_obj.auth(usr_login, usr_passwd)
                else:
                        dlg = wx.MessageDialog(self, 'Логин и пароль обязательны!',
'Аутентифкация', wx.OK|wx.ICON_EXCLAMATION)
                        dlg.ShowModal()

                if usr_passwd == 'exit':
                        print( "You are now logged in!")
                        pub.sendMessage("frameListener", message="show")
                        self.Destroy()
                else:
                        print( "Username or password is incorrect!")

        def scale_bitmap(self, bitmap, bitmap_size):
                width, height = bitmap_size[0], bitmap_size[1]
                image = wx.ImageFromBitmap(bitmap)
                image = image.Scale(width, height, wx.IMAGE_QUALITY_HIGH)
                result = wx.BitmapFromImage(image)
                return result

        def onCloseFrame(self, event):
                print("[onCloseFrame]")
                self.Destroy()
                sys.exit(0)

if __name__ == '__main__':
        start = time.time()
        # app = wx.App()
        loginGUI = LoginGUI(None, title = 'Авторизация')
        print("[{}]: {}".format('create Thread', time.time()-start))

        servConn = Updater()

        servConn.set_parrent(loginGUI)
        servConn.start()

        loginGUI.set_thread(servConn)
        app.MainLoop()
```

Продолжение Приложения Б

Листниг модуля создания токена

```
import os
import getpass
from tools import (Cryptor, Connector, do_hash, exec_cmd, DEVICE_ID, VOL_SERIAL_NUM)

def create_token(user_login, path2drive, user_passwd):
	cryptor = Cryptor()
	# while len(user_passwd)<cryptor.block_size:
	#	user_passwd += user_passwd

	# user_passwd = user_passwd[:cryptor.block_size]
	# print(user_passwd)

	cryptor.set_key(user_passwd)
	randKey = cryptor.getRandomSeq()

	user_passwd += user_passwd

	master_keys = 'master-keys'

	deviceIDs = exec_cmd(DEVICE_ID)
	deviceVSNs = exec_cmd(VOL_SERIAL_NUM)

	for i in range(len(deviceIDs)):
		if path2drive==deviceIDs[i]:
			deviceHash = do_hash(deviceVSNs[i])

			with Connector() as cur:
				QUERY = '"UPDATE LEGAL_USERS SET secret_key='{0}', flash_id=\'{1}\'

					WHERE login==\'{2}\'"'.format(deviceHash, deviceVSNs[i], user_login)
				try:
					cur.execute(QUERY)
					print('Токен пользователя успешно зарегестрирован в базе данных.')
				except:
					raise Exception('[TOKEN ALREADY IN USE]')
return
```

```
                with open(os.path.join(path2drive, deviceHash), 'wb') as
encUsrKeyFile:
                    encUsrKey = cryptor.encrypt(randKey)
                    encUsrKeyFile.write(encUsrKey)
                    print('Секретный ключ успешно зашифрован и записан на
токен.')

                with open(os.path.join('master-keys', deviceHash), 'wb') as
UsrKeyFile:
                    UsrKeyFile.write(randKey)
                    print('Секретный ключ успешно записан в память.')



def check_user(login):
    QUERY = '''SELECT count(login) FROM LEGAL_USERS
        WHERE login='{}' GROUP BY login'''.format(login)

    with Connector() as cur:
        count = cur.execute(QUERY).fetchone()

    if count:
        return True
    return False


def register_token():
    login = None
    passwd = None
    drive = None


    print('\nВведите имя пользователя в системе:')
    while True:
        login = input('\n>> ')
        if check_user(login):
            break
        print('Пользователь не найден! Повторите попытку:')


    while True:
```

```
            text = 'Придумайте пароль >> '
            passwd1 = getpass.getpass("{:23}".format(text))
            passwd2 = getpass.getpass("{:23}".format('Повторите пароль >> '))

            if passwd1!=passwd2:
                print('\nВведенные пароли не совападают!')
                print('Необходимо Повторить процедуру...')

                text = 'Введите пароль еще раз >> '
            else:
                passwd = passwd1.encode('utf-8')
                break

    all_drives = exec_cmd(DEVICE_ID)
    print("Введите букву диска для создания токена.")
    print("Доступные устройства: {}".format(all_drives))

    while True:
        drive_name = input('>> ')

        if drive_name in all_drives:
            drive = drive_name
            break

        print('Ошибка! Устройство не найдено! Пожалуйста, повторите
попытку.')

    create_token(login, drive, passwd)

try:
    register_token()
except KeyboardInterrupt:
    print('\nОтменено!')


except Exception as exc:
    print(exc)
```

Листинг модуля работы с матрицами для выработки общего ключа

```
# библиотека для работы с массивами и матрицами + поодержка математичских
вычслений
```

```
import numpy as np

import hashlib
import time
import sys
import pickle   # для перевода в бинарный вид

from tags import MAX_NB_UPDATES, CHCK_SYNC_KEXC, k, n, l

#
_____________________________________________________________________________
_______
def theta(t1, t2):
        # вернуть 1 если аргументы равны и 0 в противном случае
        return 1 if t1 == t2 else 0

#
_____________________________________________________________________________
_______
# class Hebbian():
# функции обновления весов:
def hebbian(W, X, sigma, tau1, tau2, l):
        k, n = W.shape
        for (i, j), _ in np.ndenumerate(W):
                W[i, j] += X[i, j] * tau1 * theta(sigma[i], tau1) * theta(tau1, tau2)
                W[i, j] = np.clip(W[i, j] , -l, l)

def anti_hebbian(W, X, sigma, tau1, tau2, l):
        k, n = W.shape
        for (i, j), _ in np.ndenumerate(W):
                W[i, j] -= X[i, j] * tau1 * theta(sigma[i], tau1) * theta(tau1, tau2)
                W[i, j] = np.clip(W[i, j], -l, l)

def random_walk(W, X, sigma, tau1, tau2, l):
        k, n = W.shape


        for (i, j), _ in np.ndenumerate(W):
                W[i, j] += X[i, j] * theta(sigma[i], tau1) * theta(tau1, tau2)
                W[i, j] = np.clip(W[i, j] , -l, l)
```

```
#
_____
_______
# перевод массива в десятичную систему счисления
# num_sys   - система счисления
# key       - ключ в десятичнй системе счисления
# Задается смешение на +L для всех весовых коэфицентов, т.к.  L∈{-L, L} => теперь L∈{0, L*2}
# Теперь вектор весов W представляется в полиномиальном виде: key = sum (W[i] * num_sys ^ i)
def get_key(W, L, splice=True):                    # возвращает 128 битный ключ в байтовом виде
        W = W.reshape(( W.shape[0]*W.shape[1]))
        key = ''
        max_len = len("{0:b}".format(L+L))
        stat = {'zeros': 0, 'ones': 0}
#     print("len of l: {}".format(max_len))

        for w in W:
                w += L

                n=w
                w  = "{0:b}".format(w)

                if not splice:
                        w = "{}{}".format('0'*(max_len-len(w)), w)

                key += w

        step = 3

        key = int(key, 2)
        key = str(key)

        length = len(key)//step # определяем близжайшее, кратное шагу число
        key = key[:length*step] #доведение до длины кратной шагу
        new_key = '' #


        for i in range(length):
                # пробегаемся по всему ключу
                pos = i*step
```

```
            # срез по три числа -> их ковертация в int -> mod 255(16) -> char
            num = chr(int(key[pos:pos+step])%0xFF)
            new_key += num # конкатенация

        # byte_key = bytes(new_key, 'ascii')
        byte_key = bytes(new_key, 'utf-8')    # перевод в байты
        byte_key[:16]                    # срез первых 16 битов

        return byte_key

#
_____
_______
# Генератор случайного вектора, в диапазоне от -L до L, размерностью: [k, n]
def random(l, k, n):

        return np.random.randint(-l, l + 1, [k, n])

#
_____
_______
#
_____
_______
class Machine: # класс Machine
        # Машина четности деревьев. Генерирует двоичную цифру (tau) для входного случайного вектора (X).
        # Машиа может быть описана следующими параметрами
        # k - количество скрытых нейронов
        # n - количество входных нейронов сети
        # l - задает числовой даипазон для каждого веса W[i] ∈ W
        # W - матрица весов, между входным и скрытыми слоями сети. Ее измерение: [K, N]

        def __init__(self, k=3, n=4, l=6):

                # Аргументы:
                # k - количество скрытых нейронов
                # n - количество входных нейронов, соедененных с каждым скрытым нейроном
                # l - адает числовой даипазон для каждого веса ({-L, ..., -2, -1, 0, 1, 2, ..., +L })
```

```
        self.k = k
        self.n = n
        self.l = l
        self.W = np.random.randint(-l, l + 1, [k, n]) # инициализируем веса случайным образом

        self.nb_upd = 0

    def get_output(self, X):
        # возвращает бинарную цифру tau для входного случайного вектора
        # Аргументы:
        # X - входной случайный вектор

        k = self.k
        n = self.n
        W = self.W
        X = X.reshape([k, n]) # делаем форму вектора X с [1, k*n] на [k, n]

        sigma = np.sign(np.sum(X * W, axis=1)) # Вычисляем знак от суммы весов помноженных на соотвественный входной X
        tau = np.prod(sigma) # вычисляем произведенеие всех выходных значений sigma

        self.X = X
        self.sigma = sigma
        self.tau = tau

        return tau

    def __call__(self, X):
        # при вызове объекта класса Machine - объект будет возвращать результат от get_output(X)
        return self.get_output(X)

    def do_hash(self, number):
        # вычисляет hash от переданного числа по алгоритму 'sha256'
        m = hashlib.sha256()


        m.update(str(number).encode())
        return m.digest()
```

```
        def get_key_hash(self):
                # возвращает hash от ключа, где ключ вычисляется путем представления вектора весов в виде полинома
                key = get_key(self.W, self.l)
                # key = str(key).encode()
                hash_key = self.do_hash(key)
                return hash_key

        def update(self, tau2, update_rule='hebbian'):
                # Обновляет веса по правилу обновления
                # Аргументы:
                # tau2 - выходное значение второй сети, с которой происходит синхронизация
                # update_rule - правило обновления весов W
                # Может быть одним из ['hebbian', 'anti_hebbian', random_walk']

                X = self.X
                tau1 = self.tau
                sigma = self.sigma
                W = self.W
                l = self.l

                if (tau1 == tau2):

                        if update_rule == 'hebbian':
                                hebbian(W, X, sigma, tau1, tau2, l)
                                self.nb_upd += 1

                        elif update_rule == 'anti_hebbian':
                                anti_hebbian(W, X, sigma, tau1, tau2, l)
                        elif update_rule == 'random_walk':
                                random_walk(W, X, sigma, tau1, tau2, l)
                        else:
                                raise Exception("Invalid update rule. Valid update rules are: " +
                                                "\'hebbian\', \'anti_hebbian\' and \'random_walk\'.")

def to_bytes(data):
        data = str(data)
        return bytes(data,'UTF-8')

def to_str(data):
```

```
        data = data.decode()
        return str(data)

# def send_data(tau, X)
uj
def generate_key(socket, spawned=False, name=""):
        # импортируем необходимые параметры
        from tags import BYTES_NUM, MAX_NB_UPDATES,
CHCK_SYNC_KEXC, k, n, l

        print( "MAX_NB_UPDATES: {}\nk: {}\nn: {}\nl:
{}\n".format(MAX_NB_UPDATES, k, n, l) )

        # Правило обновления весов:
        # ОРГАНИЗОВАТЬ ВОЗМОЖНОСТЬ ИМПОРТИРОВАНИЯ !!!!!!!!!!!!!
        update_rules = ['hebbian', 'anti_hebbian', 'random_walk']
        update_rule = update_rules[0]

        # !!!!!!!!!!!!!!!!!!!!!!!!!!!!
        k = 5
        n = 10
        l = 10

        # Создание объекта:
        machine = Machine(k, n, l)
        print("machine obj:", machine)

        sync = False # Флаг, котрый проверяет синхронизацию
        nb_updates = 0 # Счетчик количества попыток обновления весов

        # temp
        SYNC_CHK = False


        # for X in new_xs:
        while(not sync):
                in_data = None


                tauB = 1
                key_hash = None
                # далее небольшое дублирование кода :(
```

```
                # однако так нагляднее :)
                if spawned: # РЕШИТЬ ПРОБЛЕМУ STR -> BYTES
                        k_hash = None

                        if nb_updates == MAX_NB_UPDATES:
                                k_hash = machine.get_key_hash()
                                # print(get_key(machine.W, machine.l))
                                MAX_NB_UPDATES += 50

                        # тот, кто инициализировал соединение, тот и генерирует ветор
<X>
                        X = random(l, k, n) # Создаем случайный вектор X измерения:
[k, n]
                        tauA = machine(X) # Получаем выход машины

                        out_data = list()
                        out_data.append(tauA)
                        out_data.append(X)
                        out_data.append(k_hash)
                        out_data = pickle.dumps(out_data)
                        socket.sendall(out_data)

                        in_data = socket.recv(BYTES_NUM)
                        in_data = pickle.loads(in_data)
                        tauB = in_data[0]
                        X = in_data[1]
                        key_hash = in_data[2]

                if not spawned: # РЕШИТЬ ПРОБЛЕМУ STR -> BYTES
                        k_hash = None

                        in_data = socket.recv(BYTES_NUM)
                        in_data = pickle.loads(in_data)
                        tauB = in_data[0]
                        X = in_data[1]
                        key_hash = in_data[2]

                        if in_data[2] != None:
                                # print( "{}: {}".format(nb_updates, "Not None"))
                                k_hash = machine.get_key_hash()

                        tauA = machine(X) # Получаем выход машины
```

```
                    out_data = list()
                    out_data.append(tauA)
                    out_data.append(X)
                    out_data.append(k_hash)
                    out_data = pickle.dumps(out_data)
                    socket.sendall(out_data)

            # print( "{}: {}".format(nb_updates, get_key(machine.W, machine.l) ))
            if key_hash == machine.get_key_hash():
                    # print(get_key(machine.W, machine.l))
                    print( "SUCCESS: {}: {}".format(nb_updates,
get_key(machine.W, machine.l) ))
                    # time.sleep(60)
                    break

            tauB = int(tauB)
            machine.update(tauB, update_rule) # Обновляем веса исходя из выхода
сети второй сети
            # print(nb_updates, tauA, tauB)
            nb_updates += 1 # увеличиваем счетчик обновлений



    # return machine.W
    return get_key(machine.W, machine.l)
```

Листинг вспомогательных модулей

```
from Crypto.Cipher import AES
from Crypto import Random

import random, re, hashlib, os, sqlite3, smtplib


from subprocess import Popen,CREATE_NEW_CONSOLE,PIPE


class Sender():
    def __init__(self, FROM, PASSWD):
        self.FROM = FROM
        self.PASSWD = PASSWD
    def __enter__(self):
```

```
            self.server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
            self.server.ehlo()
            self.server.login(self.FROM, self.PASSWD)

            return self.server

        def __exit__(self, exc_type, exc_value, exc_trace):
            self.server.close()

class Connector:
        def __inin__(self):
            self.conn      = None
            self.cursor = None

        def __enter__(self):
            self.conn = sqlite3.connect('database.db')
            self.cursor = self.conn.cursor()

            return self.cursor

        def __exit__(self, exc_type, exc_val, exc_tb):
            self.cursor.close()
            self.conn.commit()
            self.conn.close()

class Cryptor:
        def __init__(self):
            self.block_size = AES.block_size

        def set_key(self, key):
            if type(key)==type('str') or len(key)!=self.block_size:
                # проверка, является ли ключ строкой
                # и на кратность длине блока AES
                while len(key)<self.block_size:
                    key += key
                key = key[:self.block_size]
            self.key = key

        def getRandomSeq(self):
            rand = Random.new().read(AES.block_size)
            return rand
```

```
        def encrypt(self, plain_txt):
                iv = self.getRandomSeq()
                cipher = AES.new(self.key, AES.MODE_CFB, iv)
                enc_msg = iv + cipher.encrypt(plain_txt)
                print('Successfull encrypted!')

                return enc_msg


        def decrypt(self, enc_txt):
                iv = enc_txt[:AES.block_size]
                enc_txt = enc_txt[AES.block_size:]

                cipher = AES.new(self.key, AES.MODE_CFB, iv)
                plain_txt = cipher.decrypt(enc_txt)
                print('Successfull decrypted!')

                return plain_txt


def get_serial(FLASH_models):

        def get_clear(value):
                value  = re.sub(r'[^ \w\.\n]', '', value)
                value = re.sub(r'\s{2,}', '', value)
                value = value.strip()
                return value

        # <------------< извлечение Model флеш-накопителя>------------>
        modelsCMD='cmd /C wmic diskdrive get Model'
        proc=Popen(modelsCMD,stdout=PIPE,shell=False)
        models=proc.communicate()[0]
        models = models.decode("utf-8")
        models = models.split('\n')
        print(models)
        # >------------< конец извлечение Model >------------<

        # <------------< извлечение SerialNumber флеш-накопителя>------------>
        serialNumbersCMD ='cmd /C wmic diskdrive get SerialNumber'
        proc=Popen(serialNumbersCMD,stdout=PIPE,shell=False)
        serialNumbers=proc.communicate()[0]
        serialNumbers = serialNumbers.decode("utf-8")
```

```
        serialNumbers = serialNumbers.split('\n')
        # >------------< конец извлечения SerialNumber >------------<

        # print(len(models), len(serialNumbers))
        for i in range(1, len(models)):
                model = get_clear(models[i])
                serialNumber = get_clear(serialNumbers[i])
                print("{}\t\t{}".format(model, serialNumber))

                if model==FLASH_models:
                        return serialNumber

        raise Exception('Не найден флеш-токен!')


VOL_SERIAL_NUM = 'wmic logicaldisk where drivetype=2 get VolumeSerialNumber'
DEVICE_ID = 'wmic logicaldisk where drivetype=2 get deviceid'

def do_hash(value, hexdigest=True):
        # используется sha256 алгоритм хеширования
        # если hex=False вернет хеш в байтовом виде
        # иначе как строку
        m = hashlib.sha256()
        m.update(str(value).encode())
        hashed = m.hexdigest()

        if not hexdigest:
                hashed = m.digest()
        return hashed

def get_clear(value):
        value  = re.sub(r'[^ \w\.\:\n]', '', value)
        value = re.sub(r'\s{2,}', '', value)
        value = value.strip()
        return value

def exec_cmd(command):
        proc=Popen(command,stdout=PIPE,shell=False)
        values = proc.communicate()[0]
        values = values.decode("utf-8")
        values = values.split('\n')[1:]
```

```
    clear_vlaues = []
    for value in values:

        clear_vlaues.append( get_clear(value) )

    return clear_vlaues

def find_flash(flash_hash):

    deviceid = exec_cmd(DEVICE_ID)
    vol_serial_num = exec_cmd(VOL_SERIAL_NUM)


    for i in range(len(deviceid)):
        device = get_clear(deviceid[i])
        serial_num = get_clear(vol_serial_num[i])
        print(device)

        if do_hash(serial_num)==flash_hash:
            content = os.listdir(device)
            print("USB-token найден!\nСодержимое: {}".format(content))
            return os.path.join(device, flash_hash), serial_num

    raise Exception('USB-token не найден!')
```