

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.03 —

Информационная безопасность
автоматизированных систем

Специализация —

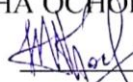
Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНЫЙ МОДУЛЬ ВЫРАБОТКИ ОБЩЕГО КЛЮЧА С
АУТЕНТИФИКАЦИЕЙ НА ОСНОВЕ ОТКРЫТОГО КЛЮЧА

Студент

 Постников Николай Андреевич

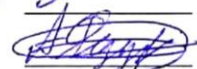
Руководитель

 Махов Е.В.

Нормоконтролёр

 Иванов А.П.

Рецензент

 Федулаев А.В.

Работа допущена к защите (протокол заседания кафедры от 27.06.18 № 16)

Заведующий кафедрой

 С.Л. Зефирков

Работа защищена с отметкой отлично (протокол заседания ГЭК
от 27.06.18 № 3)

Секретарь ГЭК

 А.П. Иванов

Подпись Е.В. Махова удостоверяю
инспектор 1 кат. ВК 



Проберга-1004106
 22.06.18г.

Оглавление

Введение.....	8
1 Обзор алгоритма выработки общего секретного ключа с аутентификацией на основе открытого ключа.....	9
1.1Схема выработки общего ключа «Лимонник - 3»	9
1.2 Эллиптические кривые.....	11
1.3Схемы выработки общего ключа с аутентификацией.....	11
1.3.1 Схема выработки общего ключа.....	11
1.3.2 Начальное состояние.....	13
1.3.3 Последовательность действий схемы выработки общего ключа.....	13
1.4 Криптографические алгоритмы, сопутствующие применению алгоритмов ЭП и функции хэширования.....	16
1.4.1 Алгоритмы HMAC_GOSTR3411_2012_256.....	16
1.4.2 Алгоритм согласования ключей VKO_GOSTR3410_2012_256.....	17
1.5 Сертификаты инфраструктуры открытых ключей.....	19
1.6 Алгоритм ГОСТ Р 34.11 – 2012.....	22
1.7Алгоритм ГОСТ Р 34.10 – 2012.....	22
2 Структура программного обеспечения специализированного вычислителя.....	24
2.1 Состав специализированного вычислителя.....	24
2.2 Семейство ARM Cortex.....	25
2.3 Семейство микроконтроллеров STM32.....	27
3 Разработка алгоритма модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа.....	30
4 Программная реализация модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа.....	41
5 Тестирование разработанного программного модуля.....	48

5.1	Тестирование функции orkA()	48
5.2	Модульное тестирование функции orkB ()	50
5.2.1	Тестирование функции VKO_GOST ()	50
5.2.2	Тестирование функции Tba ()	53
5.2.3	Тестирование функции tag ()	57
5.2.4	Тестирование функции tagB ()	61
5.2.5	Тестирование функции HMAC ()	64
5.2.6	Тестирование функции Chech_valid ()	66
5.2.7	Тестирование функции unbox_Cert ()	70
5.2.8	Тестирование функции Hesh341112 ()	74
5.3	Интеграционное тестирование функции orkB ()	78
6	Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие	84
6.1	Анализ условий труда специалиста по защите информации, влияющи сердечно-сосудистую систему специалиста по защите информации	84
6.2.1	Электромагнитные излучения	84
6.2.2	Освещенность	86
6.2.3	Шум	87
6.2.4	Микроклимат	88
6.2.5	Электробезопасность	89
6.2.6	Эргономика	90
6.2	Определение методов и средств физической культуры, снижающие воздействие факторов, влияющих на сердечно-сосудистую систему специалиста по защите информации	94
6.3	Анализ раздела	95
	Заключение	96
	Список используемых источников	97
	Приложение А Исходный код программы	98

Обозначения

- V^* — множество всех двоичных строк конечной размерности, включая пустую строку;
- $A||B$ — конкатенация строк $A, B \in V^*$, т.е. строка из $V|A|+|B|$, в которой левая подстрока из $V|A|$ совпадает со строкой A , а правая подстрока из $V|B|$ совпадает со строкой B ;
- $[T]_{i,j}$ — для битовой строки $T = (t_0, \dots, t_n)$ подстрока $T_j = (t_i, \dots, t_j)$, $0 \leq i \leq j \leq n$;
- $\text{MAC}_k(T)$ — код аутентификации сообщения T с ключом k ;
- $\text{KDF512}(T)$ — значение функции вычисления производного ключа $\text{KDF512}: V^* \rightarrow V^{512}$ на основе строки T ;
- $\pi(Q)$ — функция, отображающая точку на эллиптической кривой в битовую строку;
- $(\cdot)$ — опциональный параметр схемы (может являться строкой нулевой длины);
- $\oplus$ — операция покомпонентного сложения по модулю 2 двух двоичных строк одинаковой длины;
- V_n — конечномерное векторное пространство над $GF(2)$ размерности n с операцией сложения $\oplus$; при n равном 0 пространство V_0 состоит из единственного пустого элемента длины 0; если U — элемент V_n , то в битовом представлении $U = (u_{n-1}, u_{n-2}, \dots, u_1, u_0)$, $u_i \in \{0,1\}$;
- V_8^r — множество байтовых векторов длины r , $r \geq 0$; при $n = 0$ пространство V_0 состоит из единственного пустого элемента длины 0; если W — элемент V_8^r , $r > 0$, то в байтовом представлении $W = (w^0, w^1, \dots, w^{r-1})$, где $w^0, w^1, \dots, w^{r-1}$ принадлежат пространству V_8 ;
- $A|B$ — конкатенация байтовых векторов A и B ; если $A = (a^0, a^1, \dots, a^{r^1-1})$, $A \in V_8^{r^1}$, $B = (b^0, b^1, \dots, b^{r^2-1})$, $B \in V_8^{r^2}$, то $A|B = (a^0, a^1, \dots, a^{r^1-1}, b^0, b^1, \dots, b^{r^2-1})$, $A|B \in V_8^{r^1+r^2}$;

— H_{256} — хэш-функция H с длиной выхода, равной 256 битам, определенная в ГОСТ Р 34.11–2012 [1];

— H_{512} — хэш-функция H с длиной выхода, равной 512 битам, определенная в ГОСТ Р 34.11–2012 [8].

Термины и определения

- Код аутентификации сообщения на основе хэш-функции HMAC – механизм обеспечения аутентичности информации на основе симметричного ключа, построенного с использованием хэш-функции [1];
- Закрытый ключ – элемент секретных данных, специфичный для субъекта и используемый только субъектом в процессах согласования ключей, расшифрования ключей и формирования цифровой подписи [1];
- Открытый ключ – элемент данных, математически связанный с закрытым ключом и используемый передающей стороной в процессах согласования ключей, шифрования ключей, а также проверяющей стороной в процессе проверки цифровой подписи [1];
- Электронная подпись – строка бит, полученная в результате процесса формирования подписи. Строка имеет внутреннюю структуру, зависящую от конкретного механизма формирования подписи [1];
- Хэш-код – строка бит, являющаяся выходным результатом хэш-функции [1];
- Хэш-функция – функция, отображающая строки бит в строки бит фиксированной длины и удовлетворяющая следующим свойствам:
 - по значению функции сложно вычислить исходные данные, отображаемые в это значение;
 - для заданных исходных данных сложно вычислить другие исходные данные, отображаемые в то же значение функции [1].

Сокращения

В работе применяются следующие сокращения:

- ЭП – электронная подпись;
- СВ – специализированный вычислитель;
- ЗИ – защита информации;
- ДСЧ – датчик случайных чисел;
- МК – микроконтроллер;
- ПЗУ – постоянное запоминающее устройство;
- ИБ – информационная безопасность.

Введение

Распределение ключей — одна из фундаментальных задач криптографии. Решением является открытое распределение ключей, которое позволяет договориться о использовании общего секретного ключа абонентов. Затем ключ можно использовать для шифрования информации путем применения алгоритмов криптографического закрытия информации для скрытия от злоумышленника передаваемых потоков данных. Комбинация протокола выработки общего ключа с протоколом аутентификации нашла свое применение в аппаратуре спутниковой навигации мониторинга автотранспорта «Абонентский навигационный комплект модернизированный».

В выпускной квалификационной работе был реализован программный модуль выработки общего ключа с аутентификацией на основе открытого ключа. Для разработки программного модуля выработки общего ключа с аутентификацией на основе открытого ключа необходимо рассмотреть алгоритм выработки общего секретного ключа с аутентификацией на основе открытого ключа, разработать структуру программного обеспечения специализированного вычислителя, выполнить разработку алгоритма модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа, выполнить программную реализацию модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа, а также выполнить тестирование разработанного программного модуля.

1 Обзор алгоритма выработки общего ключа с аутентификацией на основе открытого ключа

1.1 Схема выработки общего ключа «Лимонник - 3»

Техническим комитетом 026 «Криптографическая защита информации» в 2016 году разработан проект первой редакции документа «Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования» [2]. В документе предложена схема «Лимонник - 3», которая обеспечивает выработку общего секретного ключа с одновременной двусторонней аутентификацией между двумя сторонами, осуществляющими взаимодействие в открытой сети и не имеющими заранее распределенной общей секретной информации. Схема задает алгоритмические решения на основе протоколов выработки общего ключа. Ввиду универсальности предлагаемых механизмов, схема может использоваться как составная часть протоколов защищенного обмена информацией в сетях связи различного назначения.

В работе рассматривается схема выработки общего ключа с возможностью использования эллиптических кривых и с двусторонней аутентификацией на основе схемы обмена ключей по алгоритму Диффи-Хеллмана.

Схема «Лимонник - 3», которая имеет следующий алгоритм:

- Сторона А случайным образом выбирает значение закрытого ключа k_A , $k_A \in \{1, \dots, q_B - 1\}$, вычисляет точку эллиптической кривой $K_A = k_A P_B$;
- Сторона А посылает стороне В идентификатор Id_A , сертификат $Cert_A$, открытый ключ K_A ;
- Сторона В проверяет валидность сертификата $Cert_A$. Если это условие не выполнено, то сторона В завершает сеанс связи, возвращая ошибку, информирующую о неверном сертификате;
- Сторона В проверяет, что $K_A \in E_B$ (принадлежит эллиптической

кривой), а выражение $(c_B K_A)$ - не бесконечно удаленная точка. Если эти условия не выполнены, сторона В завершает сеанс связи, возвращая ошибку, информирующую о неверном выборе параметров;

— Сторона В случайным образом выбирает значение закрытого ключа k_B , $k_B \in \{1, \dots, q_A - 1\}$, вычисляет точку эллиптической кривой $K_B = k_B P_A$;

— Сторона В вычисляет точки эллиптической кривой $Q_{AB} = c_A \cdot k_B S_A$ и $R_{AB} = s_B(c_B K_A)$;

— Сторона В вычисляет значение $T_{AB} = \text{KDF}(\pi(Q_{AB}) \parallel \pi(R_{AB}) \parallel \text{Id}_A \parallel \text{Id}_B \parallel \text{OI} \parallel K)$;

— Сторона В формирует общие сеансовые ключи $K_{AB} = [T_{AB}]0,255$, $M_{AB} = [T_{AB}]256,511$;

— Сторона В вычисляет тэг подтверждения ключа $\text{tag}_B = \text{MAC}_{M_{AB}}(h_2 \parallel \pi(K_B) \parallel \pi(K_A) \parallel \text{Id}_B \parallel \text{Id}_A)$;

— Сторона В посылает стороне А идентификатор Id_B , сертификат Cert_B , открытый ключ K_B , тэг подтверждения ключа tag_B ;

— Сторона А проверяет валидность сертификата стороны В. Если это условие не выполнено, то сторона А завершает сеанс связи, возвращая ошибку, информирующую о неверном сертификате;

— Сторона А проверяет, что $K_B \in E_A$ (принадлежит эллиптической кривой), а выражение $(c_A K_B)$ - не бесконечно удаленная точка. Если эти условия не выполнены, то сторона А завершает сеанс связи, возвращая ошибку, информирующую о неверном выборе параметров;

— Сторона А вычисляет точки эллиптической кривой $Q_{BA} = s_A(c_A K_B)$ и $R_{BA} = c_B \cdot k_A S_B$;

— Сторона А вычисляет значение $T_{BA} = \text{KDF}(\pi(Q_{BA}) \parallel \pi(R_{BA}) \parallel \text{Id}_A \parallel \text{Id}_B \parallel \text{OI} \parallel K)$;

— Сторона А формирует общие сеансовые ключи $K_{BA} = [T_{BA}]0,255$, $M_{BA} = [T_{BA}]256,511$;

— Сторона А вычисляет тэг подтверждения ключа $\text{tag}'_B = \text{MAC}_{M_{BA}}(h_2 \parallel \pi(K_B) \parallel \pi(K_A) \parallel \text{Id}_B \parallel \text{Id}_A)$ и проверяет, что $\text{tag}_B = \text{tag}'_B$. Если это условие не

выполнено, то сторона А завершает сеанс связи, возвращая ошибку, информирующую о невозможности аутентификации;

— Сторона А вычисляет тэг подтверждения ключа $\text{tag}_A = \text{MAC}_{M_{BA}} (h_3 \parallel \pi(K_A) \parallel \pi(K_B) \parallel \text{Id}_A \parallel \text{Id}_B)$;

— Сторона А посылает стороне В тэг подтверждения ключа tag_A ;

— Сторона В вычисляет тэг подтверждения ключа $\text{tag}'_A = \text{MAC}_{M_{AB}} (h_3 \parallel \pi(K_A) \parallel \pi(K_B) \parallel \text{Id}_A \parallel \text{Id}_B)$ и проверяет, что $\text{tag}_A = \text{tag}'_A$. Если это условие не выполнено, то сторона В завершает сеанс связи, возвращая ошибку, информирующую о невозможности аутентификации.

После завершения схемы абоненты А и В находятся в состоянии проведенной двусторонней аутентификации с выработанным общим ключом $K = K_{AB} = K_{BA}$. Ключи M_{AB} , M_{BA} более не используются и уничтожаются [2].

1.2 Эллиптические кривые

Предлагается использовать операции в группе точек эллиптической кривой E над конечным простым полем характеристики p , определенным в соответствии с краткой формой Вейерштрасса EW , $a, b \in GF(p)$ $= \{(x, y): y^2 \equiv x^3 + ax + b \pmod{p}\}$.

В работе используются обозначения параметров эллиптической кривой E в форме Вейерштрасса $(p, a, b, m, q, x_p, y_p)$ в соответствии с ГОСТ Р 34.10 - 2012. Используемые эллиптические кривые должны удовлетворять требованиям ГОСТ Р 34.10-2012 при $2^{508} < q < 2^{512}$.

Для вычисления кода аутентификации сообщения используется функция $\text{HMAC_GOST R3411_2012_256}$ [3].

Функция $\pi(Q)$ вычисляется в соответствии с (краткой) формой Вейерштрасса: если $Q = (x_Q, y_Q)$, то $\pi(Q) = x$.

1.3 Схемы выработки общего ключа с аутентификацией

1.3.1 Схема выработки общего ключа

Схема выработки общего ключа с аутентификацией была разработана компанией «КриптоПро» на основе схемы «Лимонник – 3». В схеме используется

эллиптическая кривая E . Для эллиптической кривой E порядки группы и подгруппы равны ($q = m$). Параметры эллиптической кривой считаются известными обеим сторонам и согласованными до начала протокола.

Долговременными ключами сторон A и B являются ключевые пары (s_A, S_A) и (s_B, S_B) , соответственно. Они определяются соотношениями: $S_A = s_A \cdot P$, $S_B = s_B \cdot P$, где $s_A, s_B \in \{1, \dots, q - 1\}$, q – порядок для кривой E , P – порождающая точка для кривой E . Стороны также обладают сертификатами $Cert_A$ и $Cert_B$ открытых ключей S_A и S_B , соответственно.

Сторонам A и B соответствуют идентификаторы ID_A и ID_B . Идентификаторы являются уникальными.

Выработка общего ключа и аутентификация осуществляются в рамках сеанса связи, с которым может быть ассоциирована доступная сторонам открытая общая информация OI , которой являются сертификаты сторон ($OI = Cert_A | Cert_B$).

В процессе инициализации специализированного вычислителя в ПЗУ устройства прошиваются:

- долговременная ключевая пара (S_a, s_a) ;
- сертификат стороны A ($Cert_A$);
- открытый ключ проверки ЭП сертификата.

Предусмотрена процедура смены прошиваемых параметров в процессе эксплуатации.

Для уникальности значений алгоритма VKO_GOSTR3410_2012_256 генерируется 8 октетов UKM_A в начале каждого сеанса связи при помощи ДСЧ. UKM_A используется для образования ключей.

В схеме на основе «Лимонник - 3» предусмотрено использование абонентами двух эллиптических кривых E_A и E_B , определенных в соответствии с краткой формой Вейерштрасса. Эллиптическая кривая E_A задается параметрами $(p_A, a_A, b_A, m_A, q_A, xP_A, yP_A)$, а эллиптическая кривая E_B задается параметрами $(p_B, a_B, b_B, m_B, q_B, xP_B, yP_B)$. Соответствующие параметры равны $s_A = m_A/q_A$, $s_B = m_B/q_B$.

Долговременные ключевые пары абонентов (s_A, S_A) и (s_B, S_B) определяются соотношениями $S_A = s_A P_A$, $S_B = s_B P_B$, где $0 < s_A < q_A$, $0 < s_B < q_B$, и удостоверяются сертификатами $Cert_A, Cert_B$.

При успешном завершении выполнения схемы стороны находятся в состоянии проведенной аутентификации с выработанным общим ключом.

Предусмотрена проверка на следующие специфические ошибки:

- невалидность сертификата;
- некорректность параметров;
- невозможность аутентификации;
- невозможность подтверждения ключа.

1.3.2 Начальное состояние параметров схемы

Абонент А хранит:

- Id_A – идентификатор абонента А;
- заранее распределенное общее секретное значение K_A .

Абонент В хранит:

- Id_B – идентификатор абонента В;
- заранее распределенное общее секретное значение K_B .

Параметры эллиптических кривых E_A , E_B считаются известными обоим абонентам и согласованными до начала схемы.

Предполагается, что выработка общего ключа и аутентификация осуществляются в рамках сеанса связи, с которым может быть ассоциирована доступная обоим абонентам открытая общая информация OI .

1.3.3 Последовательность действий схемы выработки общего ключа

– Вычисления на стороне А:

– Сторона А случайным образом выбирает значение закрытого ключа k_A , где $k_A \in \{1, \dots, q - 1\}$, и вычисляет точку эллиптической кривой $K_A = k_A \cdot P$, где P – заданный параметр в соответствии с алгоритмом ГОСТ 34.10 – 2012;

– Сторона А посылает стороне В набор $(ID_A, Cert_A, K_A, UKM_A)$ (*1 пересылка*);

– Вычисления на стороне В:

- Сторона В проверяет валидность сертификата $Cert_A$;

Если это условие не выполнено, сторона В завершает протокол, возвращая ошибку стороне А, информирующую о невалидности сертификата ключа $Cert_A$;

- Сторона В проверяет, что открытый ключ $K_A \in E$ (принадлежит эллиптической кривой) и $K_A \neq O$ (бесконечно удаленная точка). Если это условие не выполнено, сторона В завершает протокол, возвращая ошибку стороне А, информирующую о неверном выборе параметров;

- Сторона В извлекает из сертификата $Cert_A$ долговременный открытый ключ S_A ;

- Сторона В случайным образом выбирает закрытый ключ k_B , где $k_B \in \{1, \dots, q - 1\}$, и вычисляет точку эллиптической кривой $K_B = k_B \cdot P$, где P – заданный параметр в соответствии с алгоритмом ГОСТ 34.10 – 2012;

- Сторона В вычисляет точки эллиптической кривой $Q_{AB} = VKO_GOST(k_B, S_A)$ и $R_{AB} = VKO_GOST(s_B, K_A)$;

- Сторона В вычисляет значение $T_{AB} = H(Q_{AB} \parallel R_{AB} \parallel ID_A \parallel ID_B \parallel CertA \parallel CertB)$, где H – Хэш-функция в соответствии с ГОСТ Р 34.11 - 2012 с длиной хэш-кода 512 бит;

- Сторона В вычисляет общие сеансовые ключи $K_{AB} = [T_{AB}]_{0,255}$ и $M_{AB} = [T_{AB}]_{255,511}$;

- Сторона В вычисляет метку подтверждения ключа $tag_B = HMAC_{256}(M_{AB}, h_2 \parallel \pi(K_B) \parallel \pi(K_A) \parallel ID_B \parallel ID_A \parallel UKM_A \parallel UKM_B)$;

- Сторона В посылает стороне А набор $(ID_B, Cert_B, K_B, tag_B, UKM_B)$ (2 пересылка);

- Вычисления на стороне А:

- Сторона А проверяет валидность сертификата $Cert_B$. Если это условие не выполнено, сторона А завершает протокол, возвращая ошибку стороне В, информирующую о невалидности сертификата ключа $Cert_B$;

- Сторона А проверяет, что $K_B \in E$ (принадлежит эллиптической кривой) и $K_B \neq \mathcal{O}$ (бесконечно удаленная точка). Если это условие не выполнено, сторона А завершает протокол, возвращая ошибку стороне В, информирующую о неверном выборе параметров;
- Сторона А извлекает из сертификата $Cert_B$ долговременный открытый ключ S_B ;
- Сторона А вычисляет точки эллиптической кривой $Q_{BA} = VKO_GOST(s_A, K_B)$ и $R_{BA} = VKO_GOST(k_A, S_B)$;
- Сторона А вычисляет значение $T_{BA} = H(Q_{BA} || R_{BA} || ID_A || ID_B || CertA || CertB)$, где H – Хэш-функция в соответствии с ГОСТ Р 34.11 - 2012 с длиной хэш-кода 512 бит;
- Сторона А вычисляет общие сеансовые ключи $K_{BA} = [T_{BA}]_{0,255}$ и $M_{BA} = [T_{BA}]_{255,511}$;
- Сторона А вычисляет метку подтверждения ключа $tag'_B = HMAC_{256}(M_{BA}, h_2 || \pi(K_B) || \pi(K_A) || ID_B || ID_A || UKM_A || UKM_B)$ и проверяет условие $tag'_B = tag_B$. Если это условие не выполнено, сторона А завершает протокол, возвращая ошибку стороне В, информирующую о невозможности аутентификации;
- Сторона А вычисляет метку подтверждения ключа $tag_A = HMAC_{256}(M_{BA}, h_3 || \pi(K_A) || \pi(K_B) || ID_A || ID_B || UKM_A || UKM_B)$;
- Сторона А посылает стороне В набор (tag_A) (3 пересылка);
- Вычисления на стороне В:
 - Сторона В вычисляет метку подтверждения ключа $tag'_A = HMAC_{256}(M_{AB}, h_3 || \pi(K_A) || \pi(K_B) || ID_A || ID_B || UKM_A || UKM_B)$ и проверяет условие $tag'_A = tag_A$. Если это условие не выполнено, сторона В завершает протокол, возвращая ошибку стороне А, информирующую о невозможности аутентификации;
- Вычисления на сторонах А и В:

– Стороны А и В уничтожают ключи M_{BA} и M_{AB} , соответственно. Стороны А и В находятся в состоянии проведенной аутентификации с выработанным общим ключом $\bar{K} = K_{AB} = K_{BA}$;

Схема выработки общего ключа с аутентификацией на основе открытого ключа представлена на рисунке 1.1.

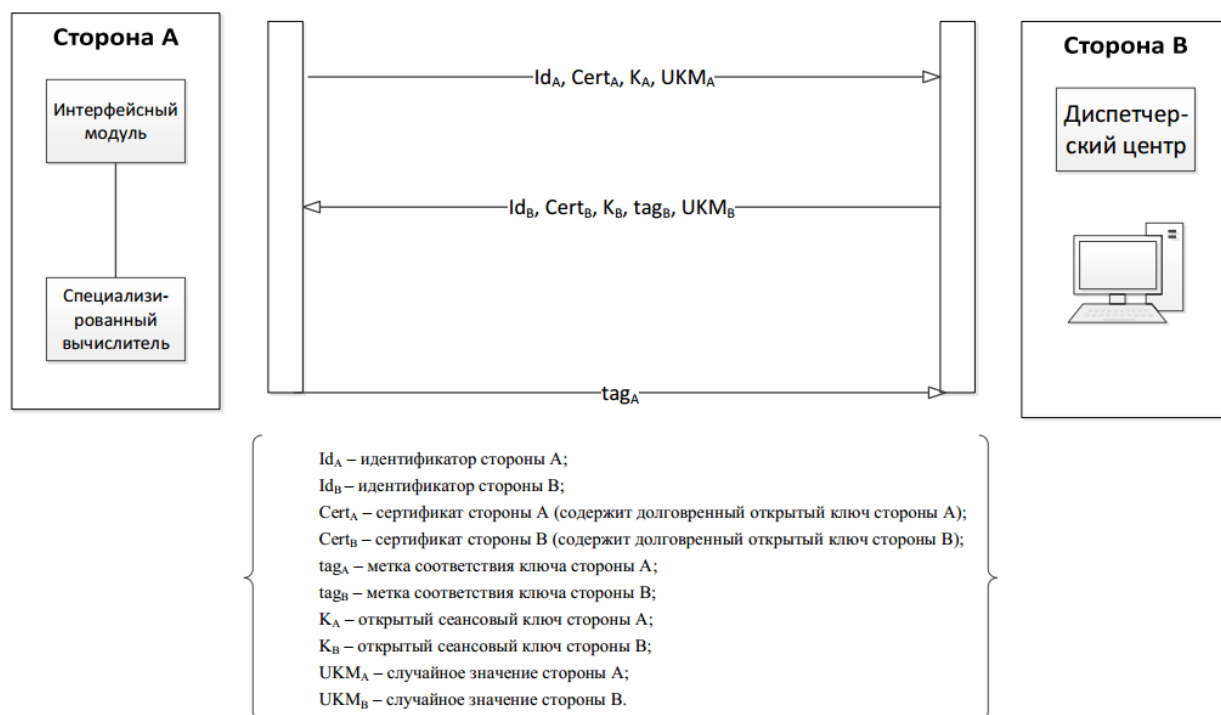


Рисунок 1.1 – Схема выработки общего ключа с аутентификацией на основе открытого ключа

1.4 Криптографические алгоритмы, сопутствующие применению алгоритмов

ЭП и функции хэширования

1.4.1 Алгоритмы HMAC_GOSTR3411_2012_256

Были определены алгоритмы вычисления кода аутентификации сообщения HMAC на основе хэш-функции H , определенной в ГОСТ Р 34.11–2012 [1] с различными длинами выходных значений.

Алгоритм HMAC_GOSTR3411_2012_256 [3] предназначен для вычисления кода аутентификации сообщения HMAC на основе хэш-функции H с длиной выхода, равной 256 битам, определенной в ГОСТ Р 34.11 – 2012, и имеет

идентификатор *id-tc26-HMAC-gost-3411-12-256*, «1.2.643.7.1.1.4.1».

Результатом работы алгоритма является значение функции *НМАС* (K, T), вычисление которой для данных T произвольной длины на ключе K длины n бит состоит в формировании байтовой строки K^* длиной 64 байта и выполнении преобразований над K^* и данными T с использованием хэш-функции в соответствии с ГОСТ 34.10 -2012.

Допускаются любые значения длины n из интервала от 256 до 512.

Для формирования ключа K^* при $n < 512$ следует использовать строку K^* равной байтовому представлению битовой строки $K|A$, где $A = (0,0,...,0) \in V_{512-n}$; если $n = 512$, использовать K^* равной байтовому представлению K . Значение *НМАС*256 (K, T) определено выражением:

$$HMAC256(K, T) = H256(K^* \oplus opad | H256(K^* \oplus ipad | T)), \quad (1)$$

где K — ключ, T — данные произвольной длины, $opad$, $ipad$ — константные значения, необходимые для формирования отличающихся друг от друга ключей на разных этапах хэширования. В байтовом представлении $ipad = (0x36 | 0x36 | \dots | 0x36) \in V_{864}$, $opad = (0x5C | 0x5C | \dots | 0x5C) \in V_{864}$.

В алгоритме *НМАС* используется вычисление хэш-суммы в соответствии с ГОСТ 34.10 - 2012 с размером выходных данных 256 бит [3].

1.4.2 Алгоритм согласования ключей VKO_GOSTR3410_2012_256

В разделе 1.4.2 определены алгоритмы согласования с использованием ключей, определенных в соответствии с ГОСТ Р 34.10–2012.

VKO_GOSTR3410_2012_256 [3] является алгоритмом согласования ключей с длиной 256 бит на основе хэш-функции с длиной выхода, равной 256 битам, определенной в ГОСТ Р 34.11–2012. Алгоритм выполняется с использованием ключей, определенных в соответствии с ГОСТ Р 34.10 – 2012.

Алгоритм предназначен для получения ключа шифрования либо ключевого

материала длины 256 бит, далее используемых в криптографических протоколах. Ключ либо ключевой материал, обозначаемые $KEKVKO$ (x , y , UKM), вырабатываются стороной обмена из своего закрытого ключа x , открытого ключа $y \cdot P$ противоположной стороны и величины UKM , рассматриваемой как число.

Алгоритм можно использовать как для статических, так и для эфемерных ключей сторон при битовой длине открытого ключа n , где $n \geq 512$, в том числе и для случая, когда ключи одной из сторон являются статическими, а другой – эфемерными. Допускается осуществлять выбор ненулевого значения UKM любой битовой длины, не превосходящей $n/2$. Использование UKM с битовой длиной не менее 64 рекомендуется в том случае, когда ключи хотя бы одной из сторон являются статическими.

Алгоритм $VKO_GOSTR3410_2012_256$ определяется формулой:

$$VKO_GOSTR3410_2012_256(x, y, UKM) = H(K(x, y, UKM)), \quad (2)$$

где H – хэш-функция в соответствии с ГОСТ 34.11 – 2012, а функция K определяется формулой:

$$K(x, y, UKM) = (m/q \cdot UKM \cdot x \bmod q) \cdot (y \cdot P), \quad (3)$$

где m – порядок группы точек эллиптической кривой, q – порядок циклической подгруппы, P – ненулевая точка подгруппы [3].

Таким образом, открытый ключ — это точка на эллиптической кривой, а личный ключ — натуральное число, которое показывает, сколько раз саму с собой нужно сложить базовую точку. Открытый ключ состоит из двух значений — это координаты X и Y , каждая из которых имеет размер 256 бит.

Алгоритм функции $VKO_GOSTR3410_2012_256$ представлен на рисунке 1.2.

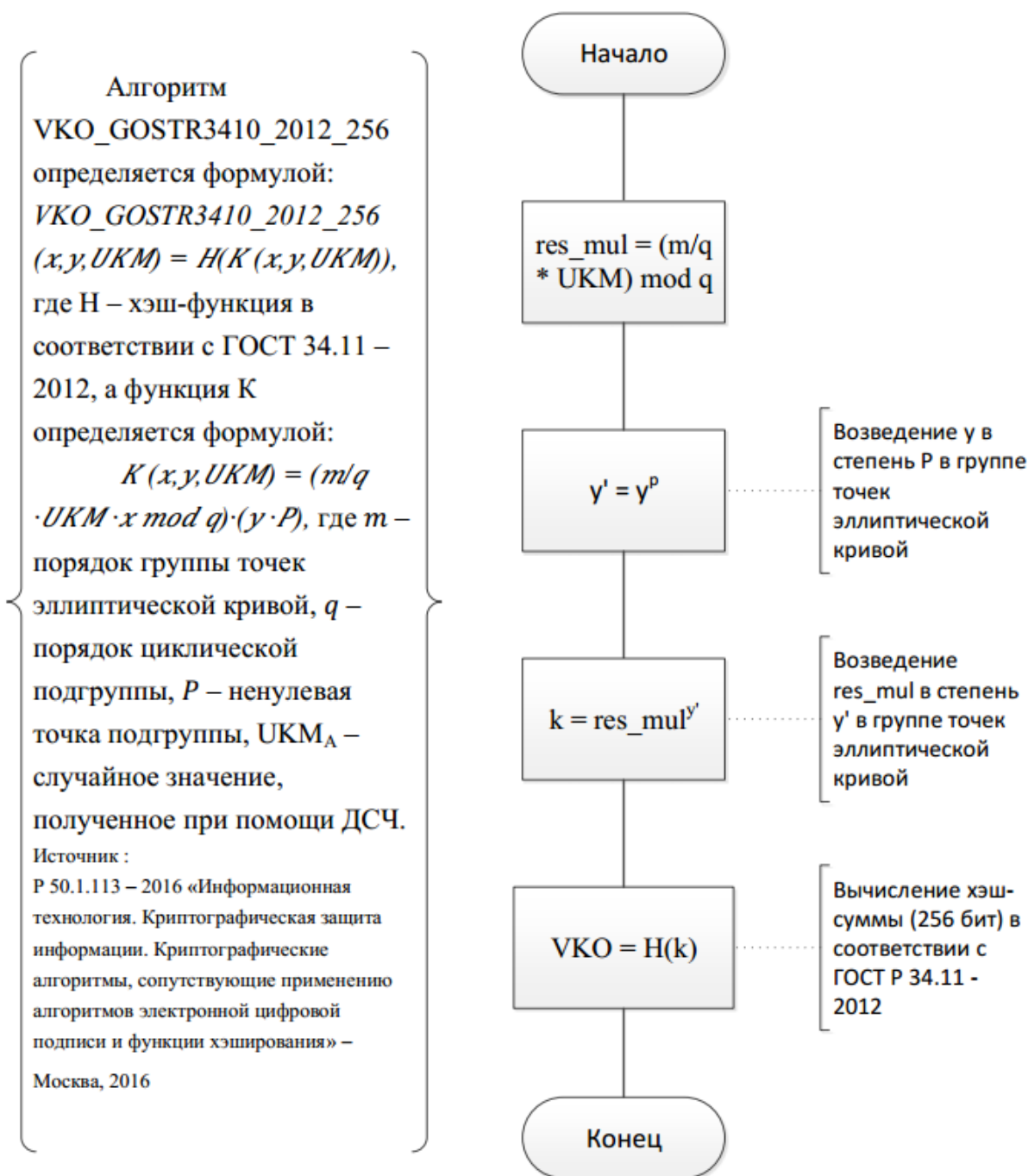


Рисунок 1.2 – Алгоритм функции VKO_GOSTR3410_2012_256

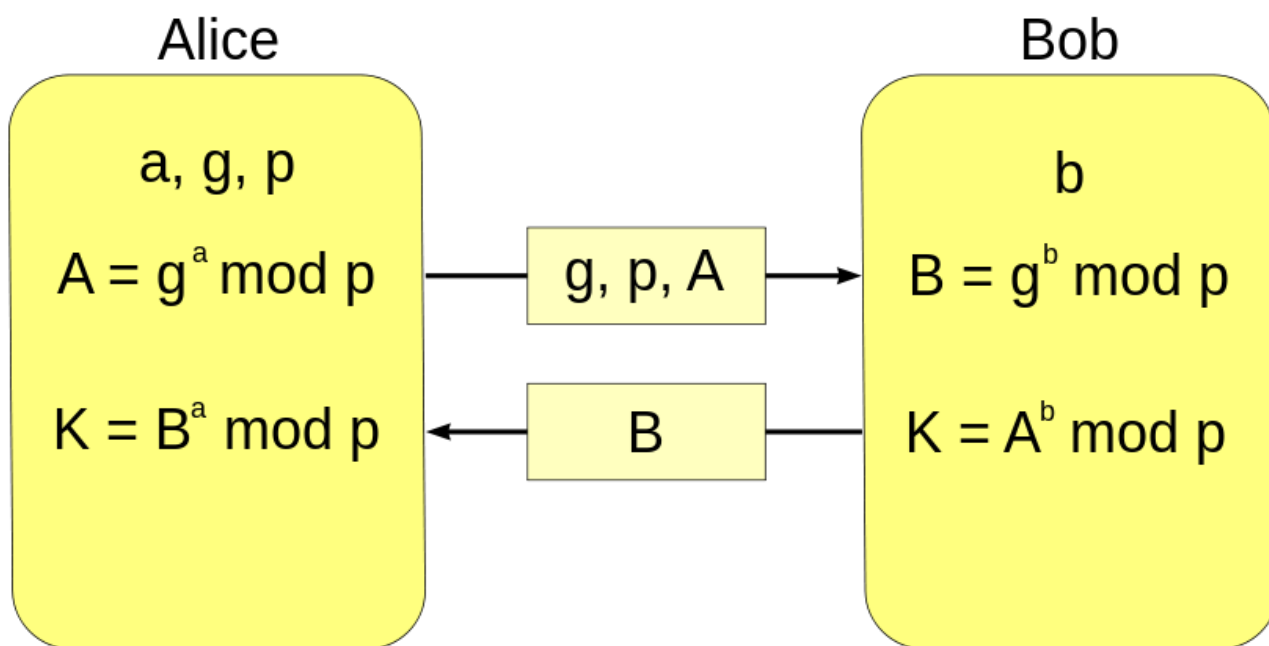
1.5 Сертификаты инфраструктуры открытых ключей

В разделе 1.5 описывается сертификат X.509, который используется в инфраструктуре открытых ключей (далее – PKI). В PKI определены идентификаторы объектов и язык для описания абстрактного синтаксиса данных в области телекоммуникаций и компьютерных сетей ASN.1 для открытых ключей, полученных путем извлечения из сертификата

Инфраструктура управления открытыми ключами получила соответствующее название в результате перевода с английского языка слов «Public Key Infrastructure». Задача PKI заключается в политике выдачи цифровых сертификатов, а также их дальнейшей поддержки. К приложениям, использующим PKI, относят: электронную почту, электронные документы, подписанные ЭП, и др.

Сертификат - это файл, содержащий информацию о человеке или организации. Сертификат подписан ЭП в соответствии с ГОСТ 34.10 - 2012. Основным содержанием сертификата является открытый ключ шифрования (S_a/S_b). У любого сертификата есть его создатель, а значит у создателя есть закрытый ключ шифрования (s_a/s_b). Фактически, сертификат - это оболочка над открытым ключом, где помимо фактического ключа содержится множество дополнительной информации: имя персоны (организации), страна, срок действия сертификата, версия, серийный номер и так далее.

Совместная работа Уитфилда Диффи и Мартина Хеллмана, опубликованная в 1976 году, о распределении открытых ключей была весомым вкладом в криптографию. Известный как «Обмен ключами Диффи — Хеллмана», метод экспоненциального обмена ключей был первым предложенным способом распределения ключа для установления сеанса связи между авторизованными пользователями канала связи. Схема открытого распределения ключей Диффи — Хеллмана (где, g и p – несекретные числа, известные Alice и Bob, K —общий секретный ключ) представлена на рисунке 1.3.



$$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$$

Рисунок 1.3 – Схема открытого распределения ключей Диффи — Хеллмана

Таким образом, протокол Диффи — Хеллмана устранил одну из важных проблем криптографии – распределение ключей, однако оригинальный алгоритм Диффи — Хеллмана может быть подвержен модификации данных в канале связи, включая атаку «Человек посередине», поэтому для усиления криптографической надежности была предложена PKI [7].

Основная задача PKI:

- обеспечить конфиденциальность и целостность передаваемых данных;
- обеспечить аутентификацию сторон;
- обеспечить неотказуемость передачи информации.

PKI реализован на основе вычислительной архитектуры клиент-сервер, а значит только по инициативе клиента может проводиться проверка информации, предоставляемой PKI.

Компоненты PKI:

- Удостоверяющий центр – является главным компонентом инфраструктуры открытых ключей, отвечающий за формирование цифровых сертификатов.

– Сертификаты открытого ключа - это пользовательские данные и открытые ключи, подписанные при помощи электронной подписи. Выдавая сертификат, удостоверяющий центр подтверждает, что лицо, указанное в сертификате, имеет закрытый ключ, соответствующий этому открытому ключу.

1.6 Алгоритм ГОСТ Р 34.11 – 2012

Алгоритм ГОСТ Р 34.11 – 2012 [1] определяет процедуру вычисления хэш-функции для любой последовательности двоичных символов, которые используются в криптографических методах защиты информации. Стандарт должен обеспечивать целостность информации и аутентичность ЭП. Хэш-функция, определенная в этом стандарте, используется для внедрения систем ЭП на основе криптографических преобразований, согласно ГОСТ Р 34.10 – 2012 [8].

Алгоритмы ГОСТ Р 34.11 – 2012 используются для вычисления 256-битного или 512-битного хэш-кода исходных данных произвольной длины.

В разработанном программном обеспечении осуществляется проверка значения подписи и открытого ключа на предмет их соответствия стандарту ГОСТ Р 34.10 – 2012 до начала использования сертификата. Удостоверяющие центры и программное обеспечение следят за тем, чтобы закрытый ключ ЭП не использовался дольше допустимого срока действия.

Алгоритм вычисления хэш-функции в соответствии с ГОСТ Р 34.11 – 2012 представлен на рисунке 1.4.

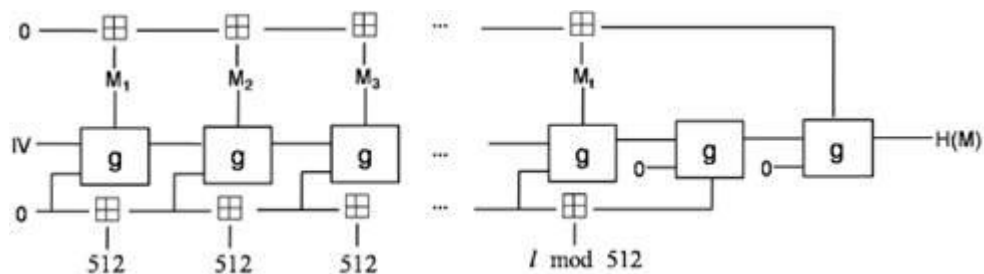


Рисунок 1.4 – Алгоритм ГОСТ Р 34.11 – 2012

1.7 Алгоритм ГОСТ Р 34.10 – 2012

Алгоритм ГОСТ Р 34.10 – 2012 [8] описывает процессы формирования и проверки ЭП, осуществляемой с использованием операций в группе точек

эллиптической кривой, определенной над конечным простым полем. Алгоритм подписи ГОСТ Р 34.10 - 2012 с длиной хэш-кода 256 бит используется для формирования ЭП в форме двух 256-битных чисел, r и s .

Удостоверяющие центры используют алгоритмы подписи в соответствии с ГОСТ Р 34.10 – 2012 для формирования подписи сертификатов и списков отзыва сертификатов. Алгоритмы подписи используются с функциями хэширования в соответствии с ГОСТ Р 34.11 - 2012.

Схема подписанного сообщения в соответствии с ГОСТ Р 34.10 – 2012 представлена на рисунке 1.5.



Рисунок 1.5 – Схема подписанного сообщения в соответствии с ГОСТ Р 34.10 – 2012

2 Структура программного обеспечения специализированного вычислителя

2.1 Состав специализированного вычислителя

В состав программного обеспечения СВ входят следующие программы:

- программа-диспетчер (диспетчер команд), реализующая получение команд и вызов обработчиков в соответствии с полученной командой;
- программы реализующие функции технического закрытия информации;
- программы обработчиков команд;
- программы, реализующие функции многозадачной среды;
- служебные программы (драйверы, обработчики прерываний и т.д.).

Структура программного обеспечения специализированного вычислителя представлена на рисунке 2.1.

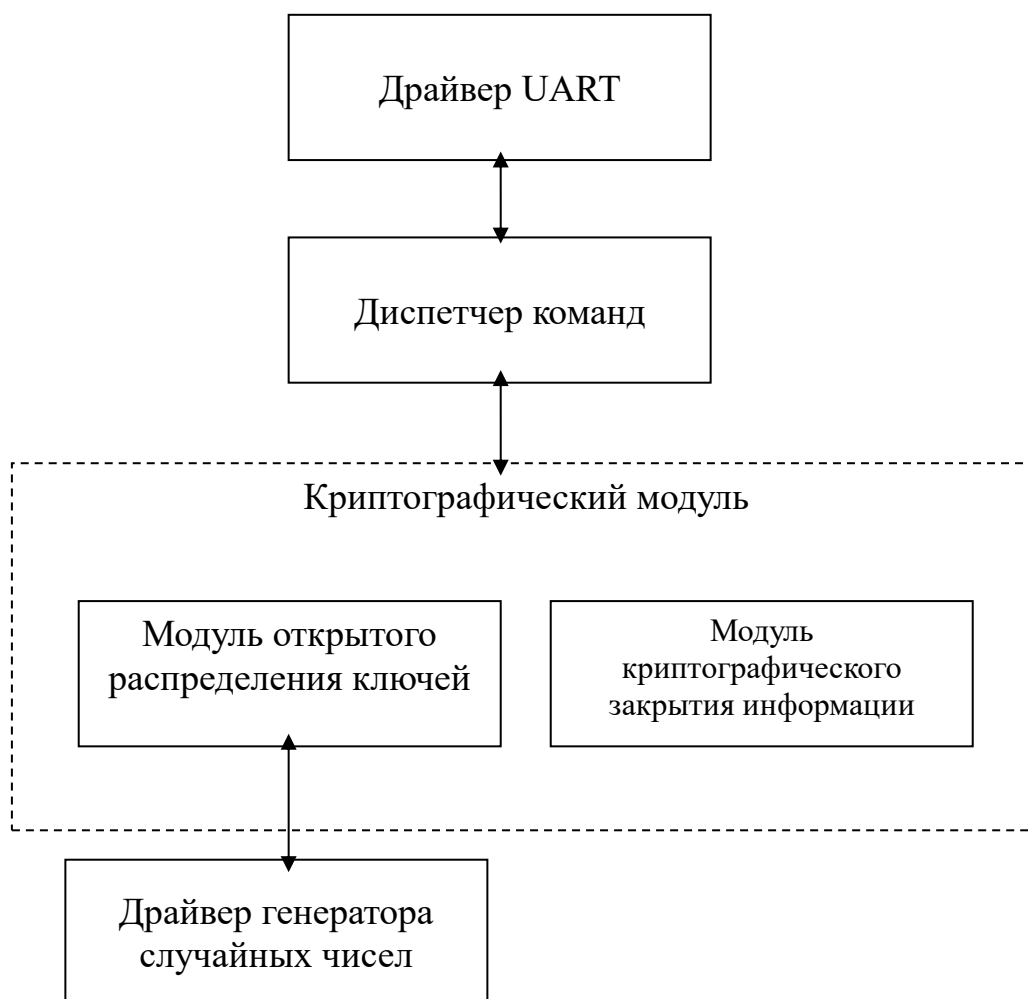


Рисунок 2.1 – Структура программного обеспечения специализированного вычислителя

Драйвер UART осуществляет прием команд через последовательный интерфейс UART с контролем целостности принятой команды.

При включении питания, встроенный загрузчик микропроцессора Cortex-M3 выполняет перезапись программного обеспечения специализированного вычислителя из внутреннего ПЗУ процессора в оперативную память. После окончания перезаписи ПО загрузчик передает управление по адресу точки старта программного обеспечения специализированного вычислителя в оперативной памяти. Программное обеспечение специализированного вычислителя предварительно записывается во внутреннее ПЗУ.

Криптографический модуль включает в себя два модуля:

- модуль открытого распределения ключей;
- модуль криптографического закрытия информации, который осуществляет засекречивание информации на основе ГОСТ 28147-89 с алгоритмами усложнения.

Драйвер генератора случайных чисел порождает последовательности, элементы которой независимы друг от друга и подчиняются закону распределения ДСЧ.

2.2 Семейство ARM Cortex

Семейство ARM Cortex представляет собой поколение процессоров, которые были стандартизированы и отвечают различным технологическим требованиям. В отличие от других процессоров, семейство ARM Cortex представляет собой полноценное процессорное ядро, которое объединяет архитектуру процессора и стандартные системы. Микроконтроллер основан на профиле STM32 Cortex-M3, специально разработанном для приложений, где необходимы системные ресурсы и, в то же время, существует потребность в низком энергопотреблении. Семейство ARM Cortex характеризуются низкой стоимостью, что могут конкурировать с традиционными 8 и 16-разрядными микроконтроллерами. 4-гигабайтное адресное пространство Cortex-M3 разделено на выделенную область программного кода, статическое ОЗУ,

устройство ввода-вывода и системный ресурс. Cortex-M3 изготовлен по Гарвардской архитектуре и, следовательно, имеет несколько шин, которые обеспечивают параллельную работу. Семейство Cortex имеет возможность работать с разделенными данными (unaligned data). Это обеспечивает максимальную эффективность с использованием встроенной статической памяти. Интерфейс Cortex-M3 представлен на рисунке 2.2.

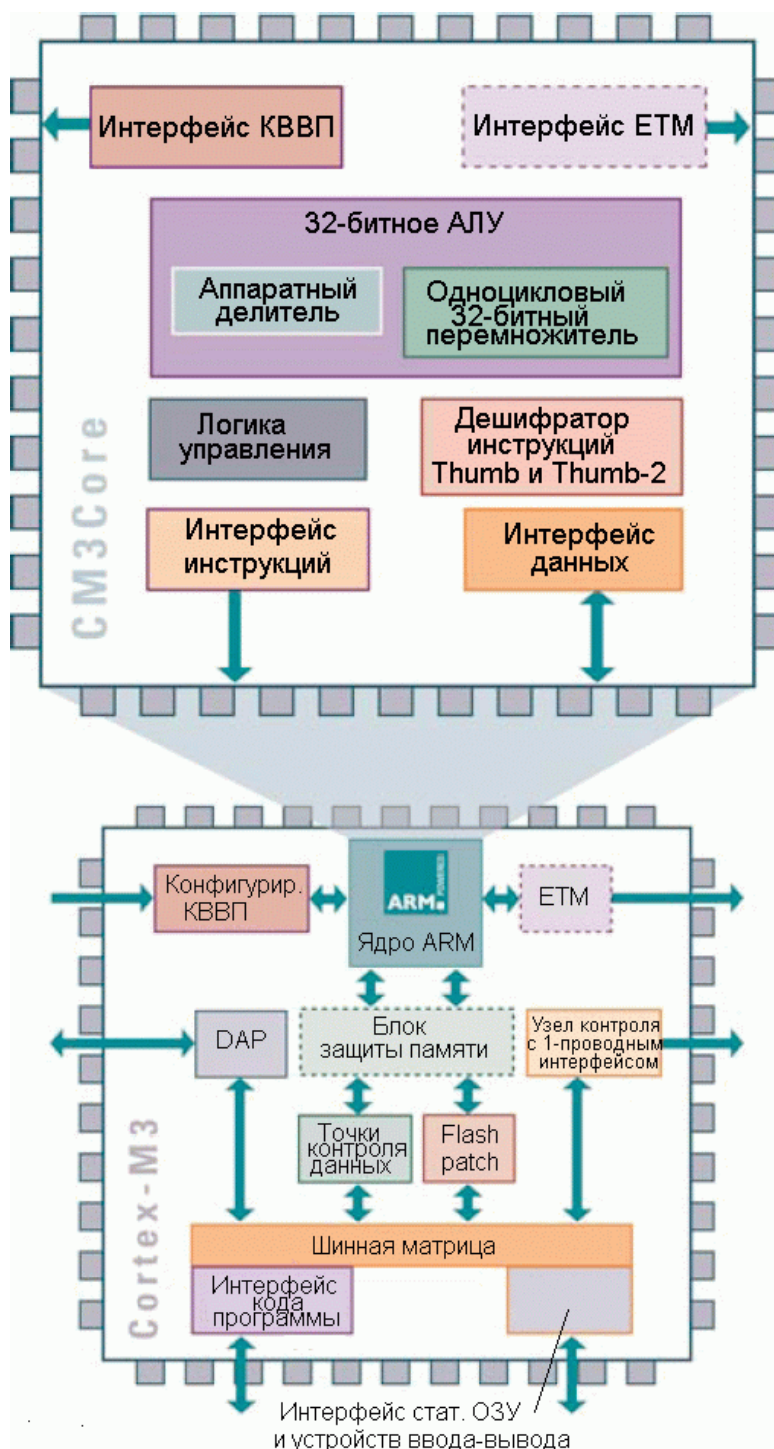


Рисунок 2.2 – Интерфейс Cortex-M3

Это стандартизованный микроконтроллер, который объединяет 32-битный процессор, блок вложенных прерываний, шинную структуру, систему отладки и предопределенную организацию памяти.

Другим ключевым компонентом ядра Cortex-M3 является контроллер вложенных векторных преобразований (КВВП).

2.3 Семейство микроконтроллеров STM32

Стратегия организации вычислительной системы RISC включает в себя использование упрощенного набора инструкций, то есть каждая операция отдельного процессора выполняет фиксированный объем работы, например, считывание данных из памяти в регистр, запись данных из регистра в память, операции с данными, содержащимися в регистрах (арифметические, логические и т. д.) и другие. Как правило, инструкции RISC-процессора имеют фиксированную длину и простой формат кодирования, что упрощает выбор и декодирование команды. Упрощение набора инструкций служит, чтобы повысить производительность системы. Cortex-M - это серия 32-разрядных ядер ARM-процессоров, которые в основном используются для работы в микроконтроллерах. В серию входят ядра Cortex-M0, Cortex-M0 +, Cortex-M1, Cortex-M3, Cortex-M4 (F), Cortex-M7 (F), Cortex-M23, Cortex-M33 (F). Префикс «F» указывает на наличие модуля плавающей точки (FPU - Floating Point Unit), однако в некоторых конфигурациях может и не быть данного модуля. Ядра Cortex-M0, Cortex M0 + и Cortex-M23 отличаются небольшими размерами и относительно низкой ценой. Ядра Cortex-M3, Cortex-M4 и Cortex-M33 поддерживают оптимальный баланс между производительностью и низким энергопотреблением. Интерфейс STM32 представлен на рисунке 2.3.

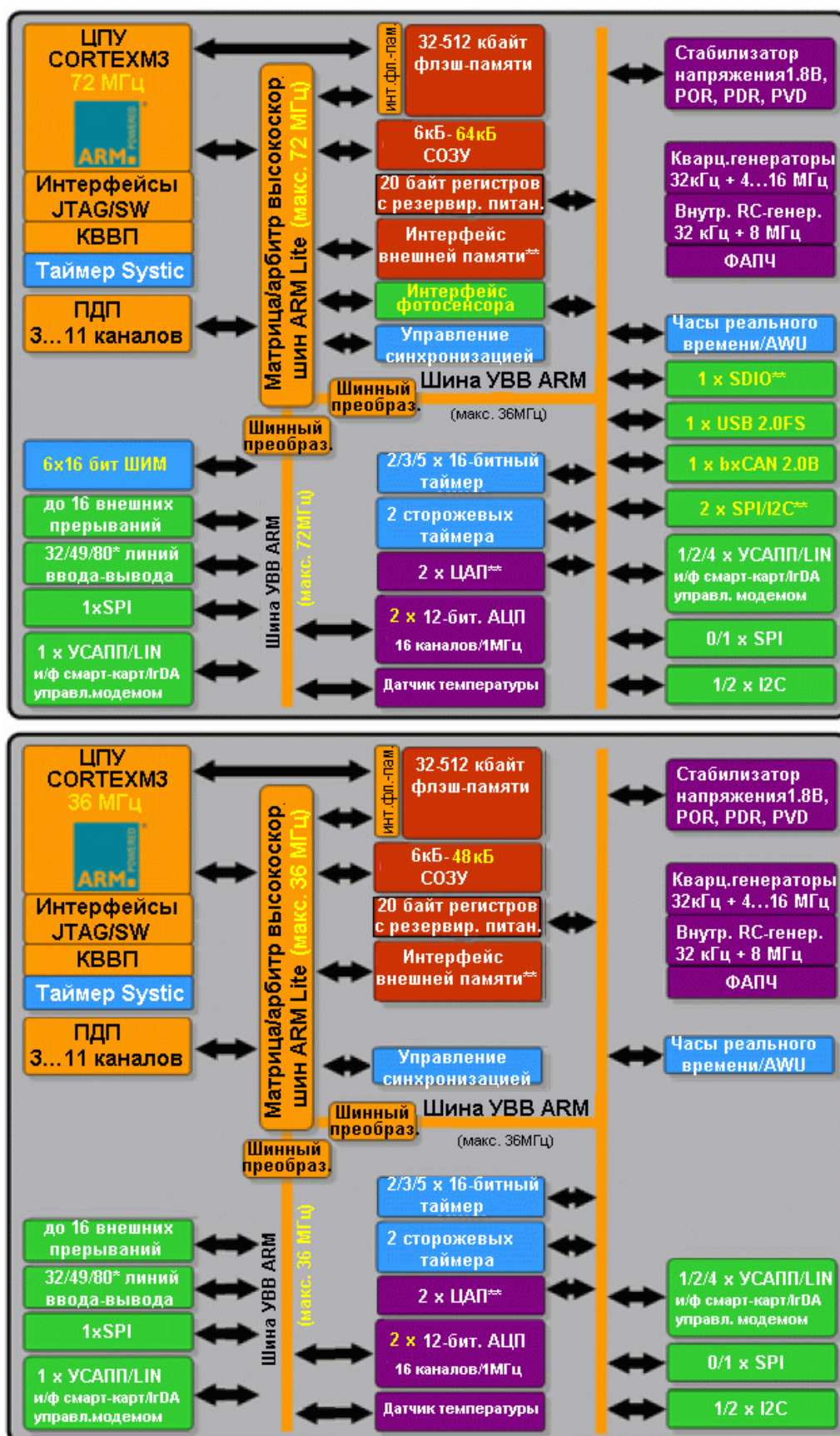


Рисунок 2.3 - Интерфейс STM32

Современные микроконтроллеры, помимо высокопроизводительных и многофункциональных УВВ, должны иметь инструменты, обеспечивающие безопасность. Это требование было учтено в STM32. Они объединяют ряд аппаратных блоков, отвечающих за безопасность микроконтроллера, в том числе диспетчер мощности, систему защиты синхронизации и два отдельных сторожевых таймера. Первый сторожевой таймер относится к оконному типу (windowed watchdog). Его необходимо обновлять только в течение периода времени, заданного пользователем. Если это делается слишком рано или слишком поздно, сторожевой таймер генерирует триггерный сигнал. Другой сторожевой таймер полностью не зависит от первого. Он синхронизируется от отдельного внутреннего генератора, который не подключен к основным системным часам. МК также использует систему защиты синхронизации, которая может обнаруживать переключение на работу от внутреннего RC-генератора с частотой 8 МГц.

Еще одним сложным требованием для современного развития является защита программного кода от кражи. В этом отношении флэш-память STM32 МК оснащена программируемой блокировкой чтения через порт отладки. После активации этой блокировки невозможно будет записать что-либо во флэш-память, что исключает возможность внесения изменений в таблицу векторов прерываний. В остальной части флэш-памяти можно активировать блокировку записи. STM32 также имеет часы реального времени и небольшую область энергонезависимой статической памяти, которая питается от отдельного источника резервной батареи. В этой области есть вход для ответа на внешние помехи.

С увеличением числа стран, занятых развитием информационных технологий, проблема информационной безопасности становится все более актуальной. Растет вероятность числа атакующих, который могут вносить изменения в передаваемые данные, что недавно доказал инцидент с вирусом Stuxnet, который нанес огромный ущерб ядерной программе Ирана. Таким образом, для снижения вероятности реализации угроз, направленных на критически важные объекты, требуется защита данных путем криптографических преобразований. Для работы в таких объектах многообещающими являются алгоритмы на эллиптических кривых в конечных полях (Elliptic Curve Cryptography) [4].

3 Разработка алгоритма модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа

В разделе 3 были разработаны алгоритмы модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа. Алгоритмы функций модуля приведены на рисунках 3.1 – 3.9.

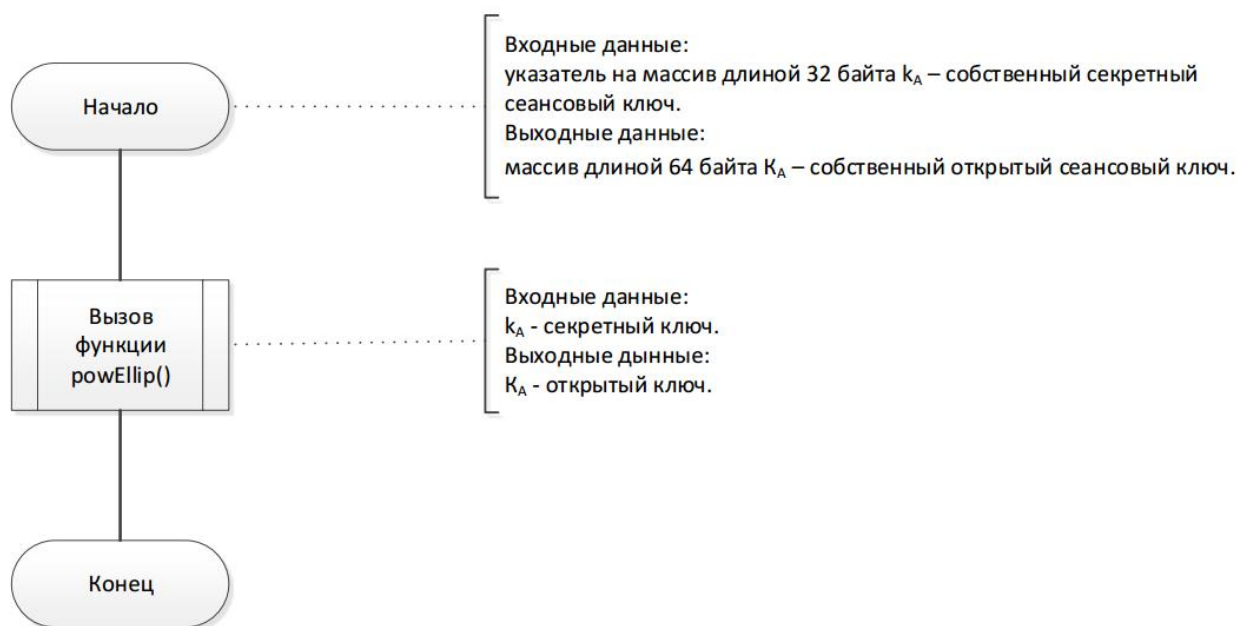


Рисунок 3.1 – Алгоритм функции `orkA()`

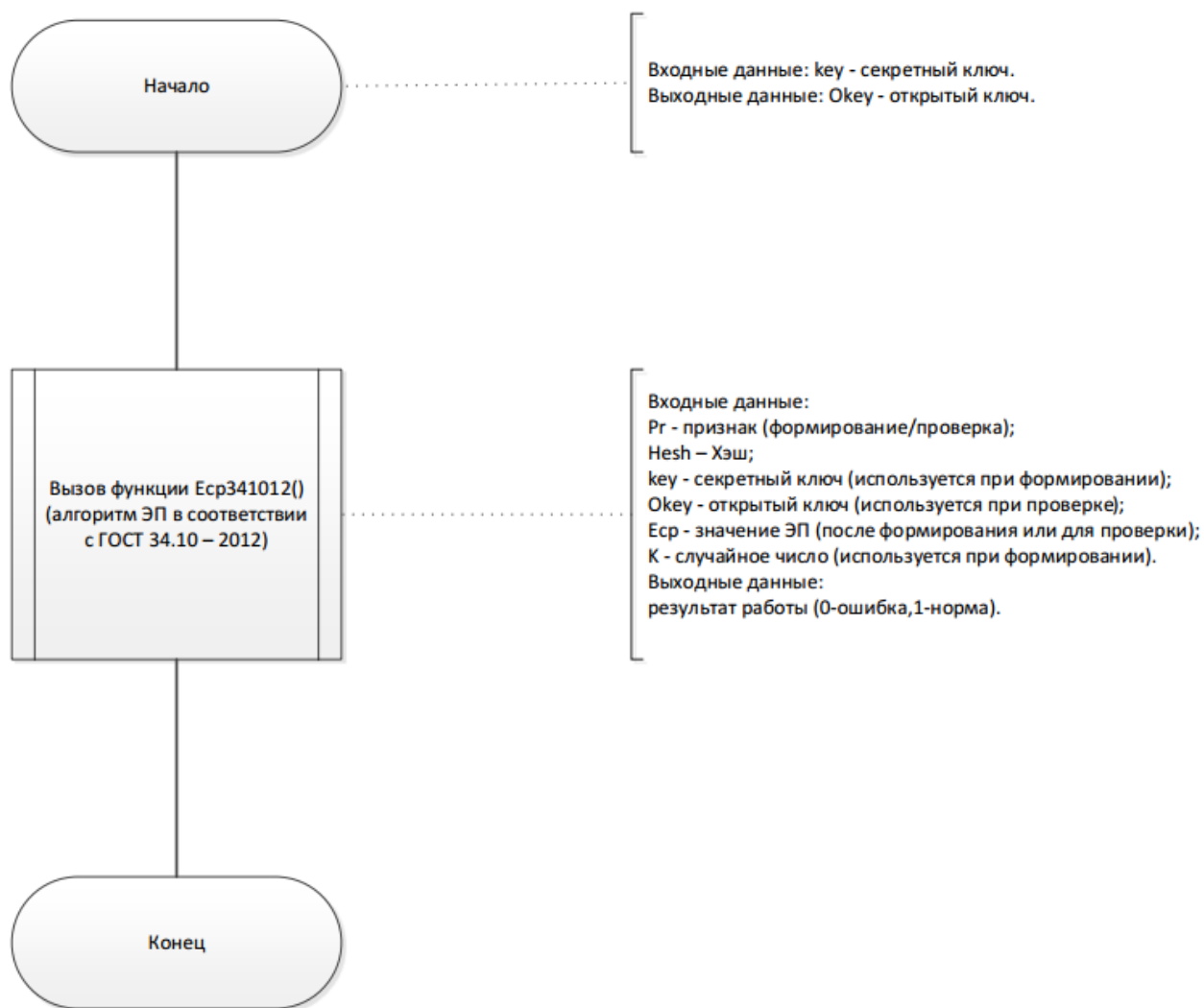


Рисунок 3.2 – Алгоритм функции rowEllip ()

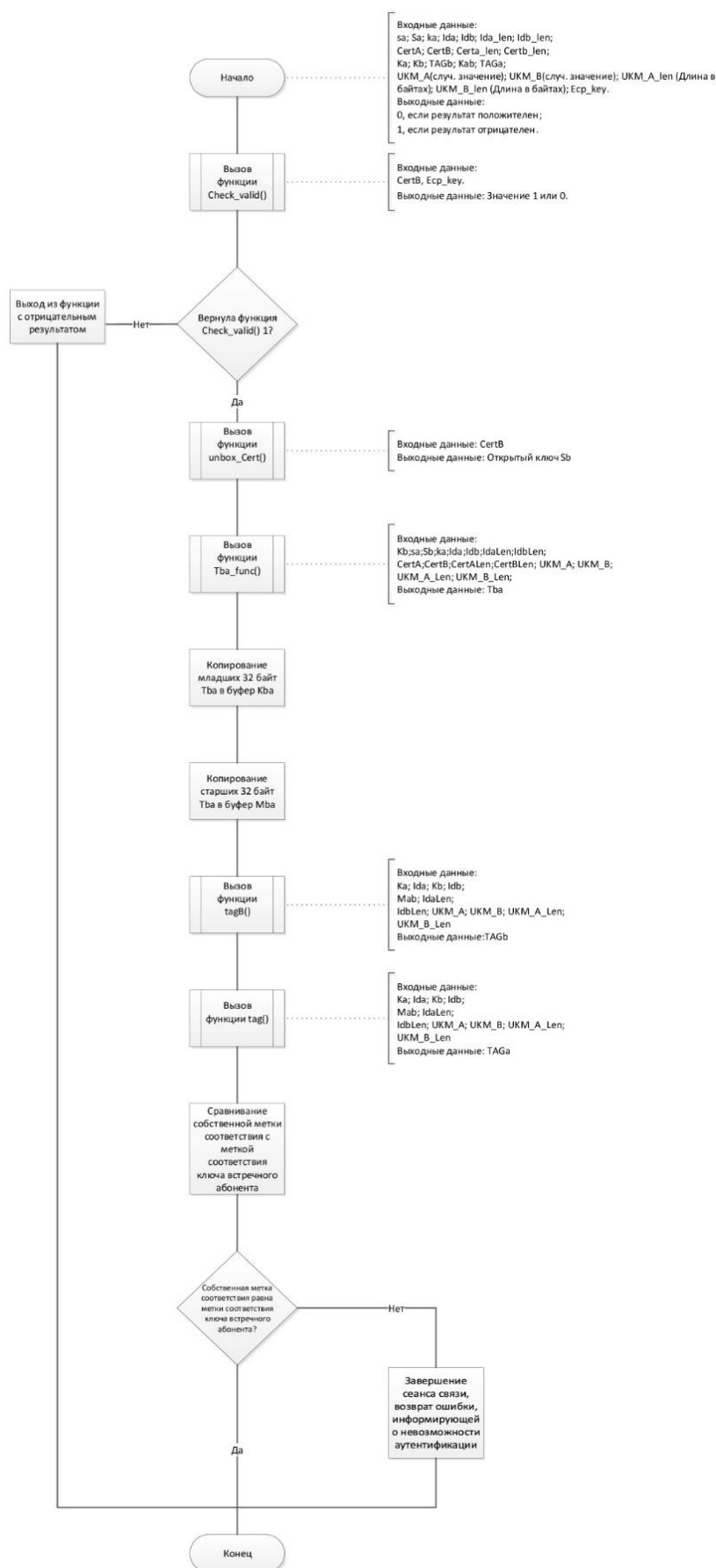


Рисунок 3.3 – Алгоритм функции orkB ()

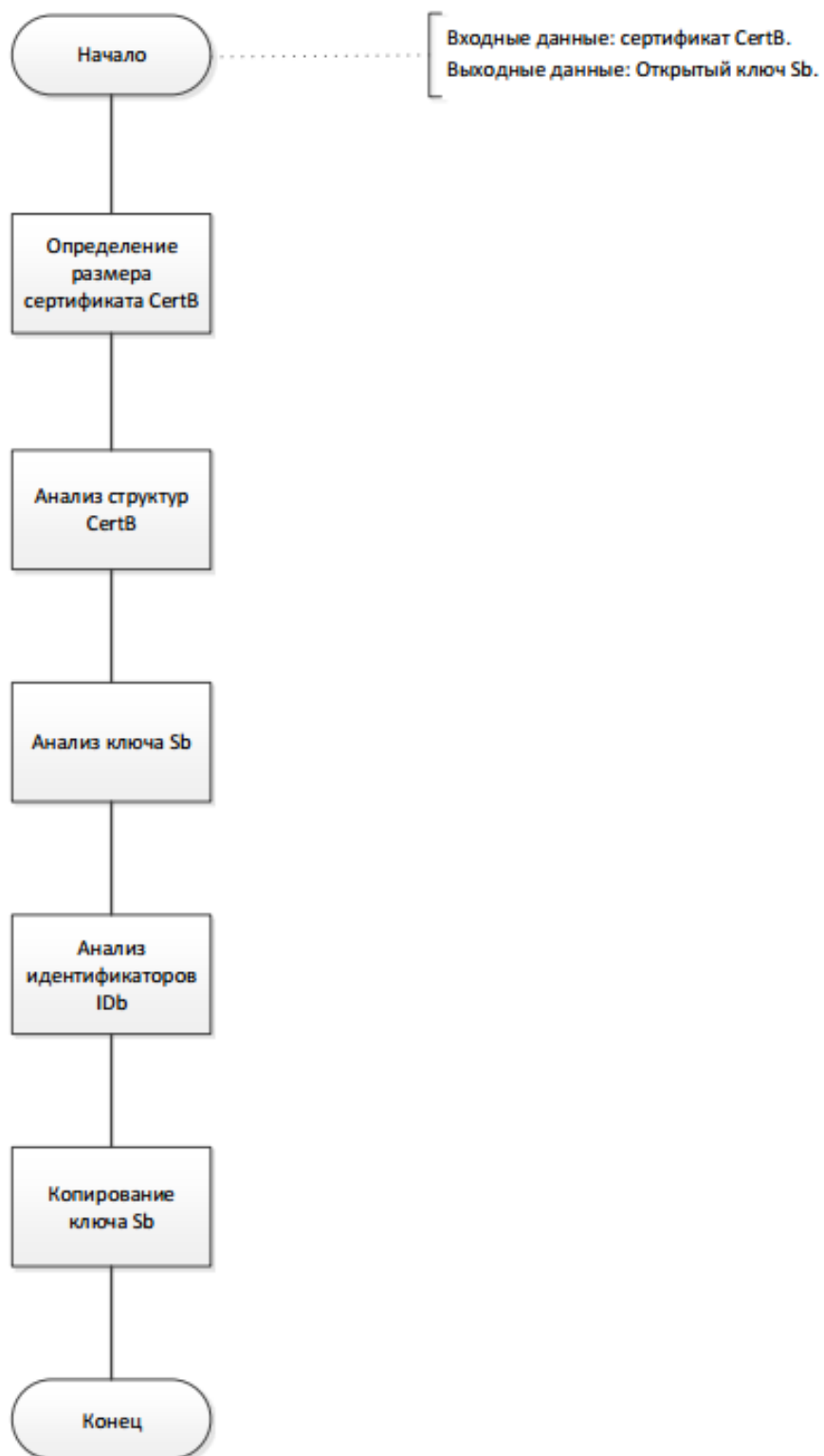


Рисунок 3.4 – Алгоритм функции `unbox_Cert()`

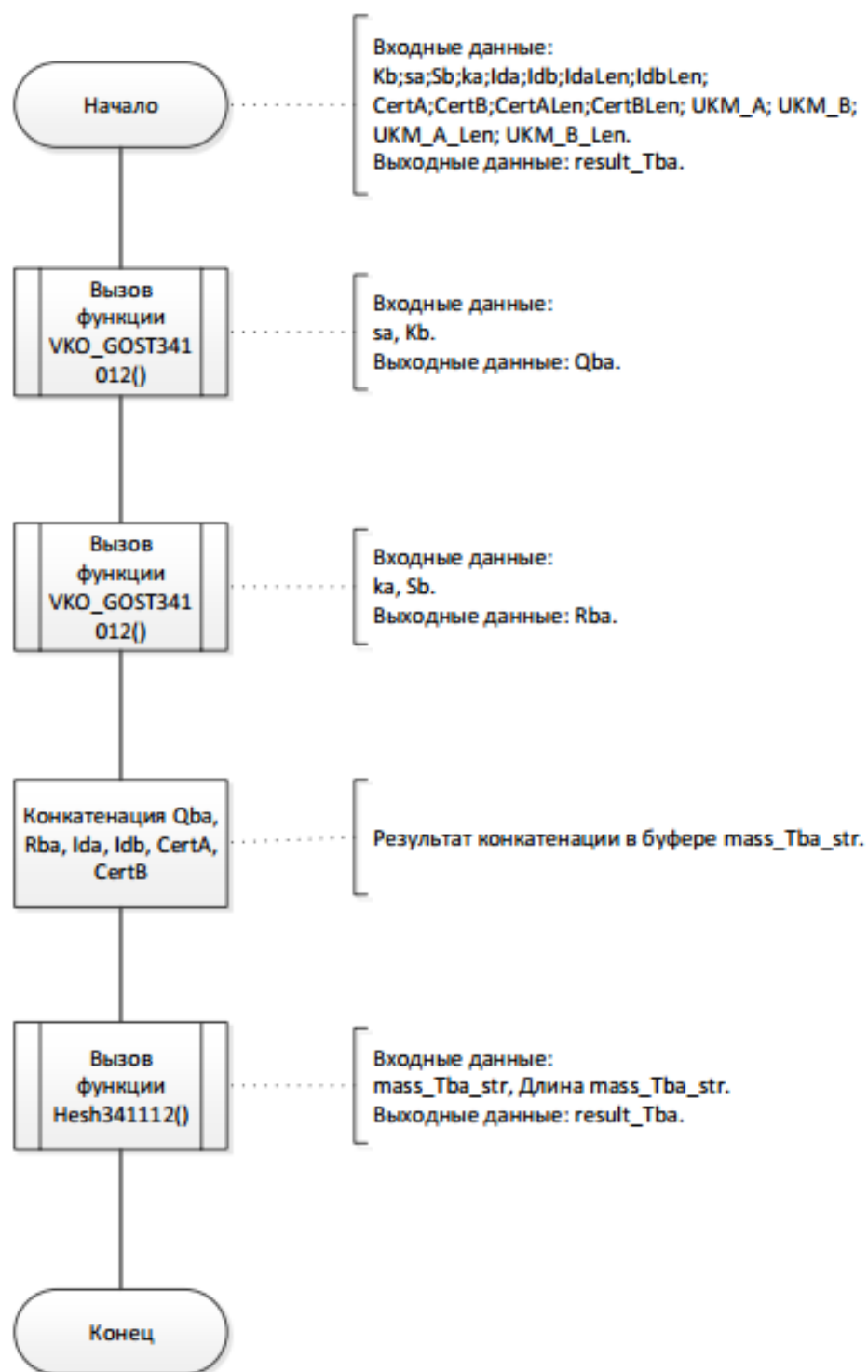


Рисунок 3.5 – Алгоритм функции TBA_func ()

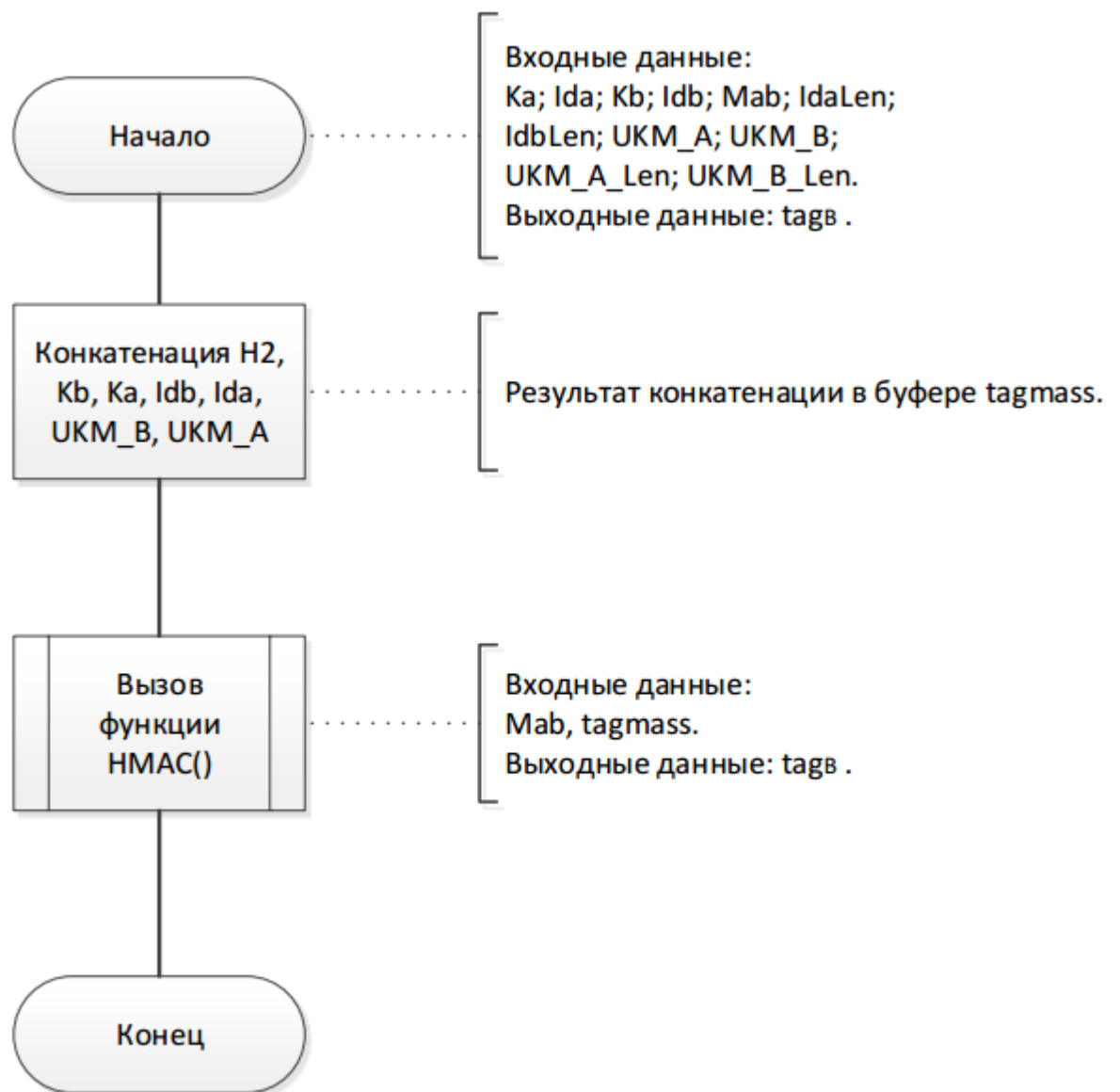


Рисунок 3.6 – Алгоритм функции tagB ()

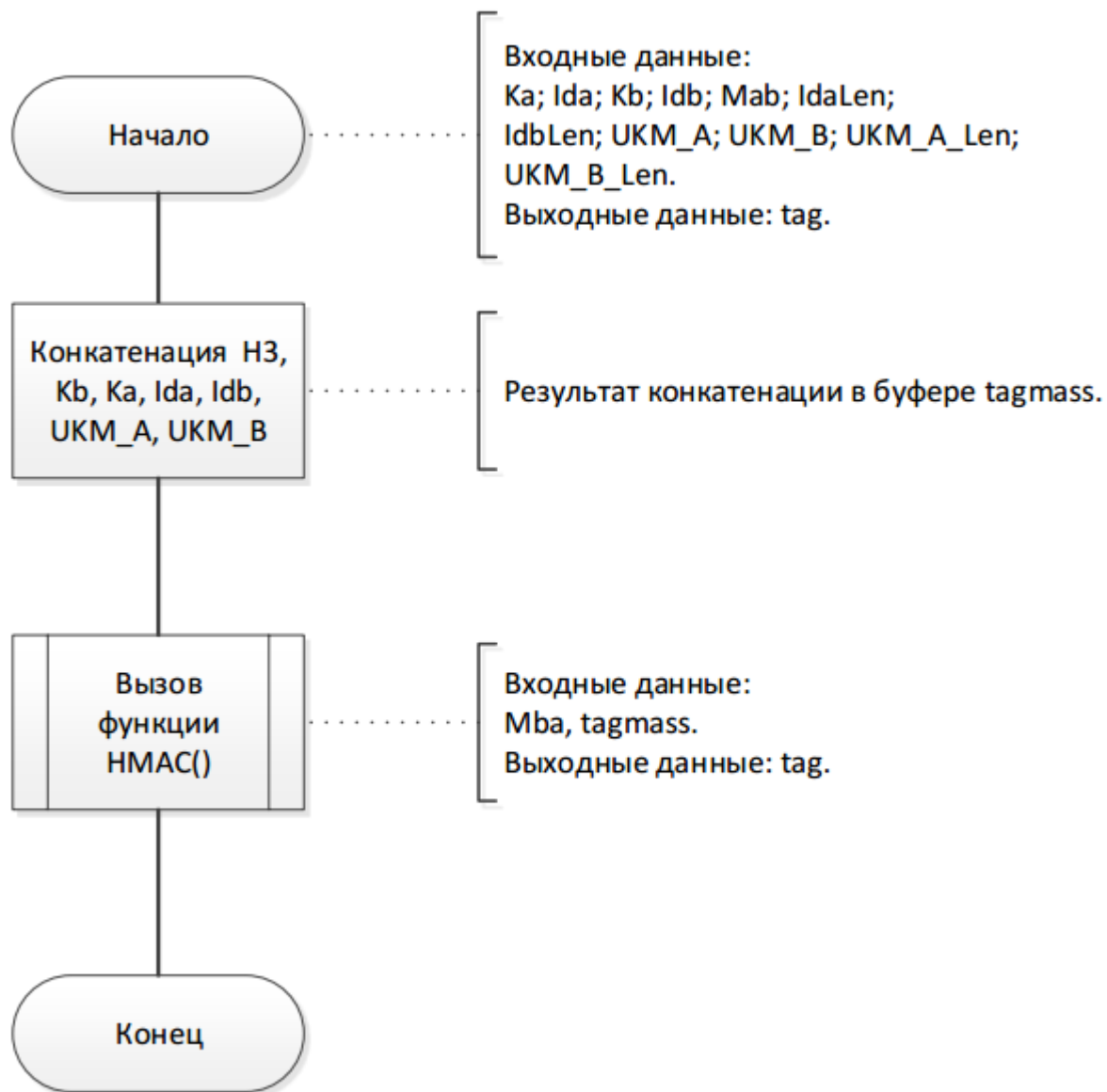


Рисунок 3.7 – Алгоритм функции tag ()

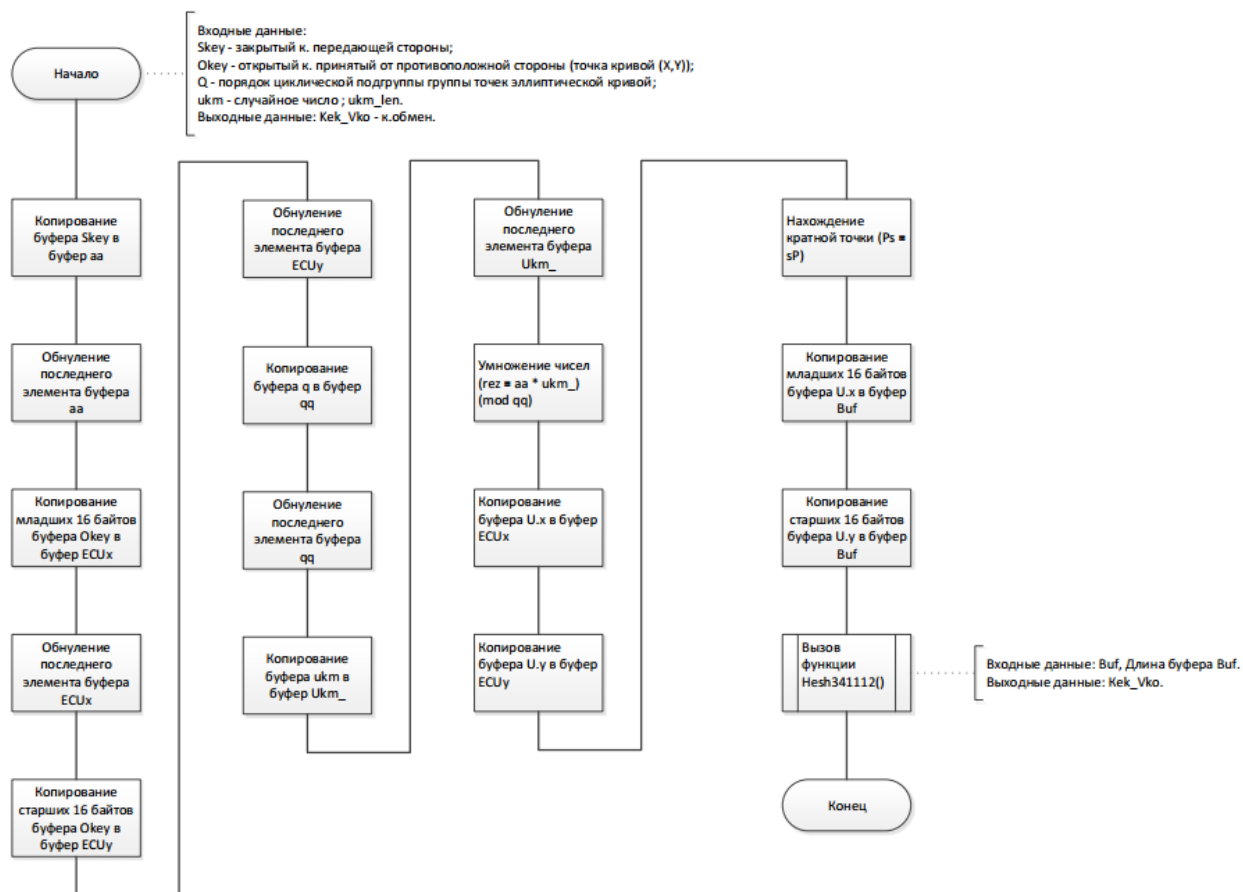


Рисунок 3.8 – Алгоритм функции VKO_GOST ()

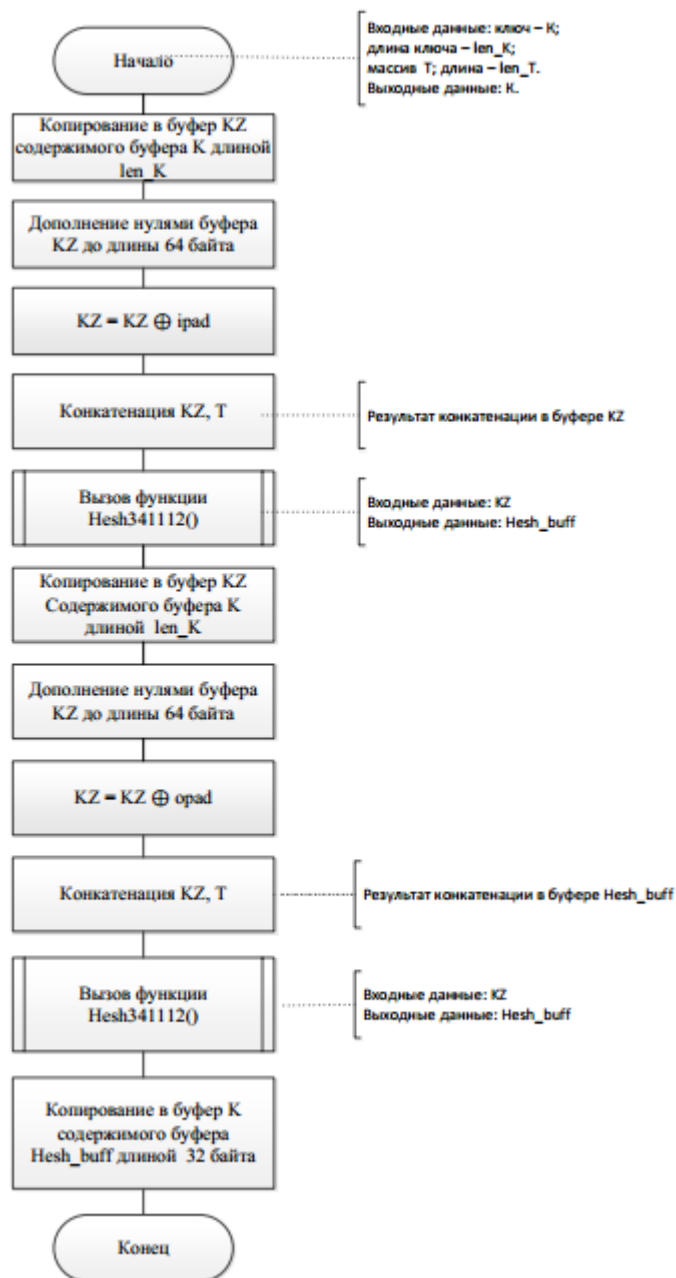


Рисунок 3.9 – Алгоритм функции HMAC ()

Алгоритм работы программного модуля выработки общего ключа с аутентификацией на основе открытого ключа представлен на рисунке 3.10.

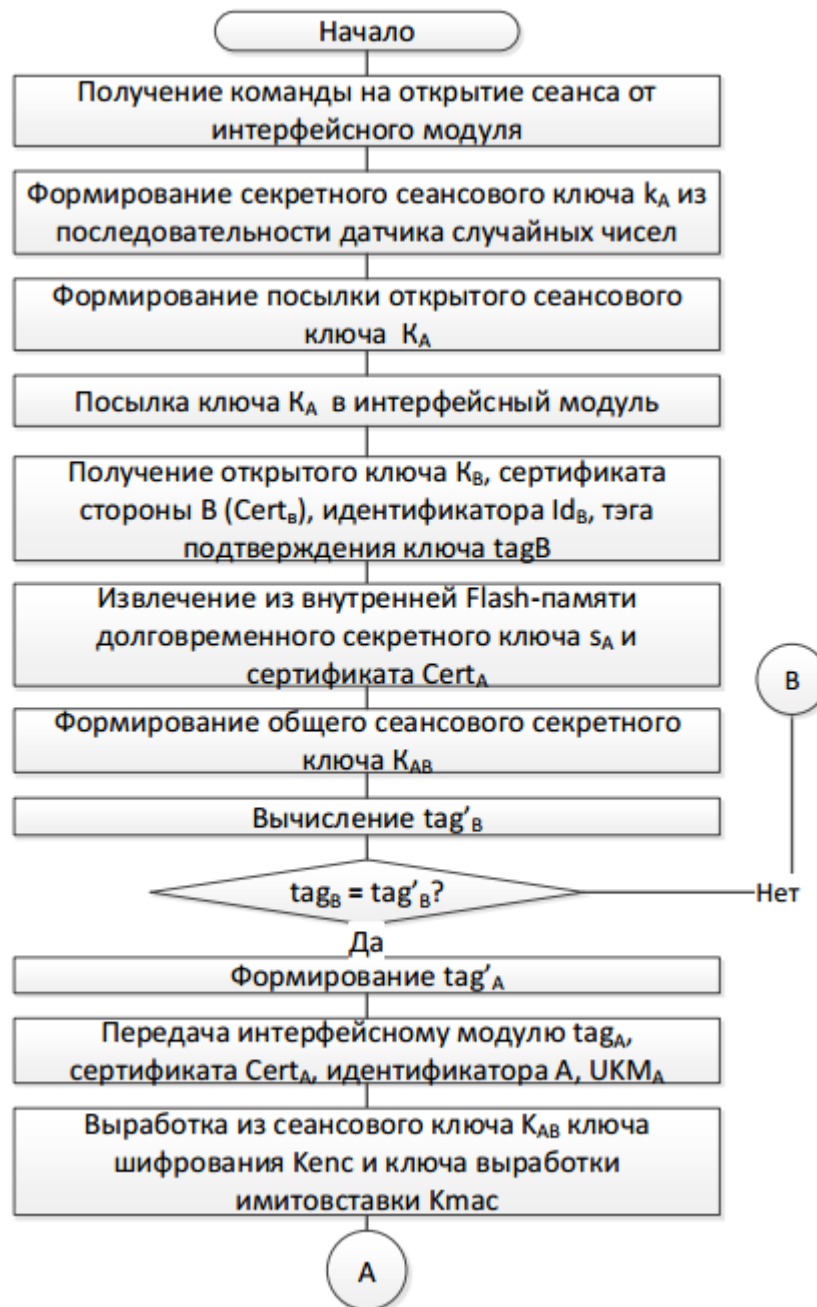


Рисунок 3.10, лист 1 – Алгоритм работы программного модуля выработки общего ключа с аутентификацией на основе открытого ключа

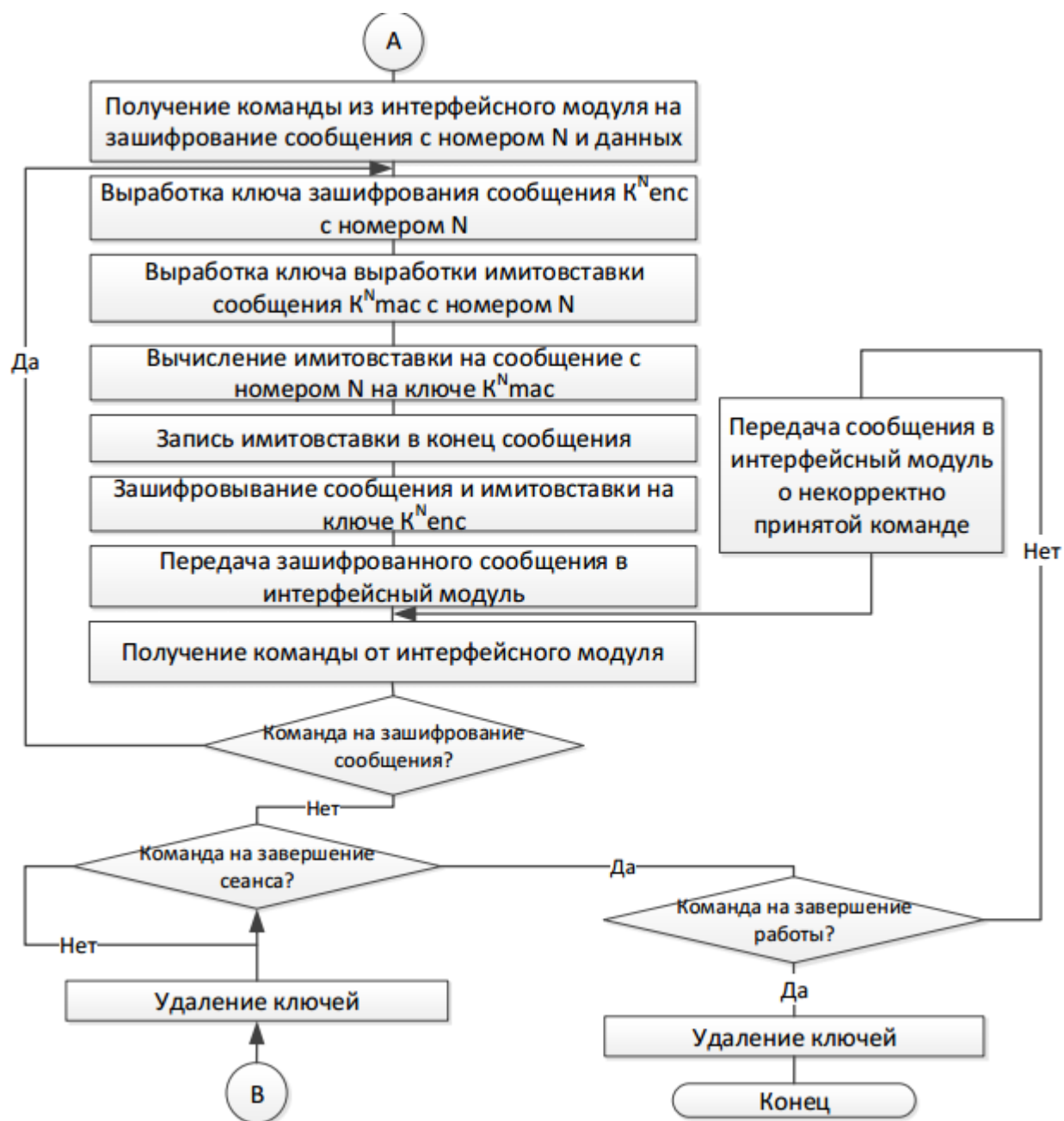


Рисунок 3.10, лист 2 – Алгоритм работы программного модуля выработки общего ключа с аутентификацией на основе открытого ключа

4 Программная реализация модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа

В процессе дипломного проектирования были реализованы следующие функции:

- orkA();
- orkB().

Функция orkA() служит для формирования посылки открытого ключа. Код функции orkA() приведен ниже:

```
void orkA(u8 * ka, u8 * Ka) // gener open key - > abon B
{
    powEllip(Ka, ka); //формирование открытого ключа сторона A
}
```

Функция orkA() вызывает функцию powEllip(), которая формирует открытый ключ, путем умножения точки эллиптической кривой на секретный ключ. Код функции powEllip() приведен ниже:

```
void powEllip(u8 * open_key, u8 * sec_key)
{
    EcpStruct pow_Ellip; //Основная структура
    pow_Ellip.Pr = ECP_FORM_OKEY; //признак
    (формирование/проверка)
    pow_Ellip.Skey = sec_key; /секретный ключ (используется при
    формировании)
    pow_Ellip.Okey = open_key; //открытый ключ (используется при
    проверке)
    pow_Ellip.K = 0; //случайное число (используется при
    формировании)
    pow_Ellip.Hesh = 0; //хэш
    pow_Ellip.Ecp = 0; //значение ЭЦП (после формирования или для
    проверки)
    Ecp341012(&pow_Ellip); // формирования открытого ключа ЭП
}
```

Функция powEllip() вызывает функцию Ecp341012(), которая реализует алгоритм формирования открытого ключа ЭП в соответствии с ГОСТ 34.10 - 2012. Функция Ecp341012() может работать в трех режимах (формирование ЭП, проверка ЭП, формирование открытого ключа проверки ЭП). Код функции Ecp341012() приведен ниже:

```
unsigned short Ecp341012(EcpStruct * Data)
{
```

```

unsigned short res = 0;

if(ECp0inv && ECq0inv)
{
    switch(Data->Pr)
    {
        //Формирование ЭП
        case ECP_FORM:
            if((Data->Hesh == NULL_) || (Data->Skey ==
NULL_) || (Data->Ecp == NULL_) || (Data->K == NULL_))
            {
                break;
            }
            res = EcpForm(Data);
            break;
        //Проверка ЭП
        case ECP_CONTROL:
            if((Data->Hesh == NULL_) || (Data->Okey ==
NULL_) || (Data->Ecp == NULL_))
            {
                break;
            }
            res = EcpControl(Data);
            break;
        //Формирование открытого ключа проверки ЭП
        case ECP_FORM_OKEY:
            if((Data->Skey == NULL_) || (Data->Okey ==
NULL_))
            {
                break;
            }
            res = EcpFormOkey(Data);
            break;
    }
}
return res;
}

```

Функция orkB() позволяет провести аутентификацию встречного абонента и сформировать общий секретный ключ.

Код функции orkB():

```

int orkB(orkB_str * p_orkB_str)
{
    u8 Tba[64]; //массив Tba
    u8 Mab[32]; //массив Mba
    Tba_str struct_Tba_str;
    tag_str tag_str_orkb;

    if (!Chech_valid(p_orkB_str->Certb, p_orkB_str-
>key_ECP_Cert)) return 0; //проверка валидности сертификата

```

```

    if (!unbox_Cert(p_orkB_str->Certb, buff_Sb)) return 0;
//извлечение ключа Sb из сертификата

    struct_Tba_str.Kb = p_orkB_str->Kb;
    struct_Tba_str.sa = p_orkB_str->sa;
    struct_Tba_str.Sb = buff_Sb;

    struct_Tba_str.ka = p_orkB_str->ka;

    struct_Tba_str.Ida = p_orkB_str->Ida;
    struct_Tba_str.Idb = p_orkB_str->Idb;
    struct_Tba_str.IdaLen = __strlen((char*)p_orkB_str->Ida);
    struct_Tba_str.IdbLen = __strlen((char*)p_orkB_str->Idb);
    struct_Tba_str.CertA = p_orkB_str->Certa;
    struct_Tba_str.CertB = p_orkB_str->Certb;

    struct_Tba_str.CertALen = p_orkB_str->Certa_len;
    struct_Tba_str.CertBLen = p_orkB_str->Certb_len;
    struct_Tba_str.UKM_A = p_orkB_str->UKM_A;
    struct_Tba_str.UKM_B = p_orkB_str->UKM_B;
    struct_Tba_str.UKM_A_Len = p_orkB_str->UKM_A_len;
    struct_Tba_str.UKM_B_Len = p_orkB_str->UKM_B_len;
    struct_Tba_str.result_Tba = Tba;

    Tba_func(&struct_Tba_str); // создание последовательности ТВА
    __memcpy(p_orkB_str->Kab, Tba, 32); //копирование первых 32-х
значений Тба в массив Kab
    __memcpy(Mab, &Tba[32], 32); //копирование последних 32-х
значений Тба в массив Mab

    tag_str_orkb.Ida = p_orkB_str->Ida;
    tag_str_orkb.IdaLen = __strlen((char*)p_orkB_str->Ida);
//расчет длины строки
    tag_str_orkb.Idb = p_orkB_str->Idb;
    tag_str_orkb.IdbLen = __strlen((char*)p_orkB_str->Idb);
//расчет длины строки
    tag_str_orkb.Ka = p_orkB_str->Ka;
    tag_str_orkb.Kb = p_orkB_str->Kb;
    tag_str_orkb.Mab = Mab;
    tag_str_orkb.tag = p_orkB_str->TAGb_res;
    tag_str_orkb.UKM_B = p_orkB_str->UKM_B;
    tag_str_orkb.UKM_B_Len = p_orkB_str->UKM_B_len;
    tag_str_orkb.UKM_A = p_orkB_str->UKM_A;
    tag_str_orkb.UKM_A_Len = p_orkB_str->UKM_A_len;
    tagB(&tag_str_orkb); // вычисление метки соответствия ключа
встречного абонента
    if(!__memcmp (TAGb_res, p_orkB_str->TAGb, 32)) return 1; );
//Сравнивание собственной метки соответствия с меткой соответствия
ключа встречного абонента

    __memcpy(Mab, &Tba[32], 32); //копирование последних 32-х
значений Тба в массив Mab

```

```

    tag_str_orkb.Ida = p_orkB_str->Ida;
    tag_str_orkb.IdaLen = __strlen((char*)p_orkB_str->Ida);
    //определение длины строки

    tag_str_orkb.Idb = p_orkB_str->Idb;
    tag_str_orkb.IdbLen = __strlen((char*)p_orkB_str->Idb);
    //определение длины строки

    tag_str_orkb.Ka = p_orkB_str->Ka;
    tag_str_orkb.Kb = p_orkB_str->Kb;
    tag_str_orkb.Mab = Mab;
    tag_str_orkb.tag = p_orkB_str->TAGa;
    tag_str_orkb.UKM_B = p_orkB_str->UKM_B;
    tag_str_orkb.UKM_B_Len = p_orkB_str->UKM_B_len;
    tag_str_orkb.UKM_A = p_orkB_str->UKM_A;
    tag_str_orkb.UKM_A_Len = p_orkB_str->UKM_A_len;

    tag(&tag_str_orkb); // вычисление собственной метки
    соответствия ключа

    return 1;
}

```

Функция `orkB()` вызывает следующие функции:

- `Check_valid()` - проверка валидности сертификата;
- `unbox_Cert()` - извлечение ключа S_B из сертификата;
- `Tba_func` - создание последовательности T_{BA} . T_{BA} состоит из общего сеансового ключа K_{BA} и последовательности M_{BA} , которая используется при подсчете метки соответствия;
- `tagB()` – вычисление метки соответствия ключа встречного абонента;
- `tag ()` – вычисление собственной метки соответствия ключа.

Код функции `Tba_func ()`:

```

void Tba_func(Tba_str * str_Tba)
{
    u8 mass_Tba_str[4096];
    VKOStruct VKO_str;
    HeshStruct Tba_hesh;
    u8 UKM_buf[32];

    __memset(UKM_buf, 0, 32); // обнуление буфера UKM_buf
    __memcpy (UKM_buf, str_Tba->UKM_A, str_Tba->UKM_A_Len);
    //копирование массива str_Tba->UKM_A в массив UKM_buf

    VKO_str.Skey = str_Tba->sa;
}

```

```

VKO_str.Okey = str_Tba->Kb;
VKO_str.q = q;
VKO_str.ukm = UKM_buf;
VKO_str.ukm_len = 32;
VKO_str.Kek_Vko = mass_Tba_str;
VKO_GOST341012(&VKO_str); // вычисление точки эл. кривой QBA

__memset(UKM_buf, 0, 32); // обнуление буфера UKM_buf
__memcpy (UKM_buf, str_Tba->UKM_B, str_Tba->UKM_B_Len);
//копирование содержимого массива str_Tba->UKM_B в массив UKM_buf
VKO_str.Skey = str_Tba->ka;
VKO_str.Okey = str_Tba->Sb;
VKO_str.q = q;
VKO_str.ukm = UKM_buf;
VKO_str.ukm_len = 32;
VKO_str.Kek_Vko = mass_Tba_str + 32;
VKO_GOST341012(&VKO_str); // // вычисление точки эл. кривой
Rba
conc(mass_Tba_str, 64, str_Tba->Ida, str_Tba->IdaLen); //conc
from ID_A

conc(mass_Tba_str, 64 + str_Tba->IdaLen, str_Tba->Idb,
str_Tba->IdbLen); //conc from ID_B

conc(mass_Tba_str, 64 + str_Tba->IdaLen + str_Tba->IdbLen,
str_Tba->CertA, str_Tba->CertALen); //conc from CertALen

conc(mass_Tba_str, 64 + str_Tba->IdaLen + str_Tba->IdbLen +
str_Tba->CertALen, str_Tba->CertB, str_Tba->CertBLen); //conc from
CertBLen
Tba_hesh.Data = mass_Tba_str;
Tba_hesh.Hesh = str_Tba->result_Tba;
Tba_hesh.DataLen = 64 + str_Tba->IdaLen + str_Tba->IdbLen +
str_Tba->CertALen + str_Tba->CertBLen;
Tba_hesh.STATE = 0;
Hesh341112(HESH_LEN_512, &Tba_hesh); //вычисление хэш-суммы в
соответствии с ГОСТ 34.10 - 2012
}

```

Код функции tagB():

```

void tagB(tag_str* tag_str_ptr)
{
    u32 length = 0;
    conc(tagmass, 0, H2, H2LEN); // конк. с h2
    conc(tagmass, H2LEN, tag_str_ptr->Kb, PILEN); // конк. с
PI(Kb)
    conc(tagmass, H2LEN + PILEN, tag_str_ptr->Ka, PILEN); // конк.
с PI(Ka)
    conc(tagmass, H2LEN + PILEN + PILEN, tag_str_ptr->Idb,
tag_str_ptr->IdbLen); // конк. с Idb
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen,
tag_str_ptr->Ida, tag_str_ptr->IdaLen); // конк. с Ida
}

```

```

        conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen, tag_str_ptr->UKM_B, tag_str_ptr->UKM_B_Len);
// конк. с UKM_B
        conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen + tag_str_ptr->UKM_B_Len, tag_str_ptr->UKM_A,
tag_str_ptr->UKM_A_Len); // конк. с UKM_A

        length = H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen + tag_str_ptr->UKM_B_Len + tag_str_ptr-
>UKM_A_Len; // расчет длины
        HMAC(tag_str_ptr->Mab, tagmass, 32, length, tag_str_ptr->Mab);
// вычисления кода аутентификации сообщения
        __memcpy (tag_str_ptr->tag, tag_str_ptr->Mab, 32);
//копирование 32-х значений tag_str_ptr->Mab в массив tag_str_ptr-
>tag
    }

```

Код функции tag ():

```

void tag(tag_str* tag_str_ptr)
{
    u32 length = 0;
    conc(tagmass, 0, H3, H3LEN); // конк. с h3
    conc(tagmass, H3LEN, tag_str_ptr->Ka, PILEN); // конк. с
PI(Ka)
    conc(tagmass, H3LEN + PILEN, tag_str_ptr->Kb, PILEN); // конк.
с PI(Kb)
    conc(tagmass, H3LEN + PILEN + PILEN, tag_str_ptr->Ida,
tag_str_ptr->IdaLen); // конк. с Ida
    conc(tagmass, H3LEN + PILEN + PILEN + tag_str_ptr->IdaLen,
tag_str_ptr->Idb, tag_str_ptr->IdbLen); // конк. с Idb
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen, tag_str_ptr->UKM_A, tag_str_ptr->UKM_A_Len);
// конк. с UKM_A
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen + tag_str_ptr->UKM_A_Len, tag_str_ptr->UKM_B,
tag_str_ptr->UKM_B_Len); // конк. с UKM_B

    length = H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen +
tag_str_ptr->IdaLen + tag_str_ptr->UKM_A_Len + tag_str_ptr-
>UKM_B_Len;
// расчет длины
    HMAC(tag_str_ptr->Mab, tagmass, 32, length , tag_str_ptr-
>Mab);
// вычисления кода аутентификации сообщения
    __memcpy (tag_str_ptr->tag, tag_str_ptr->Mab, 32);
//копирование 32-х значений tag_str_ptr->Mab в массив tag_str_ptr-
>tag
}

```

Все функции написаны на языке С. Код всех функций, реализованных в программном модуле, представлен в приложении А.

5 Тестирование разработанного программного модуля

5.1 Тестирование функции `orkA ()`

Тестирование отдельных функций программного модуля выработки общего ключа с аутентификацией на основе открытого ключа производилось путем сравнения результатов выполнения программного модуля и тестовых значений (контрольных примеров).

Программный модуль специализированного вычислителя был запущен в среде разработки IAR Embedded Workbench в режиме отладки на плате СВ.

Функция `orkA ()` служит для формирования посылки открытого ключа. Входными данными является последовательность: d2 b2 68 62 fe ef f0 bd 84 96 42 a6 65 fb 2f ca bf f1 74 df 9b 82 0f b1 74 6f 27 33 f0 54 5c e1 секретного ключа k_A , представленная на рисунке 5.1.

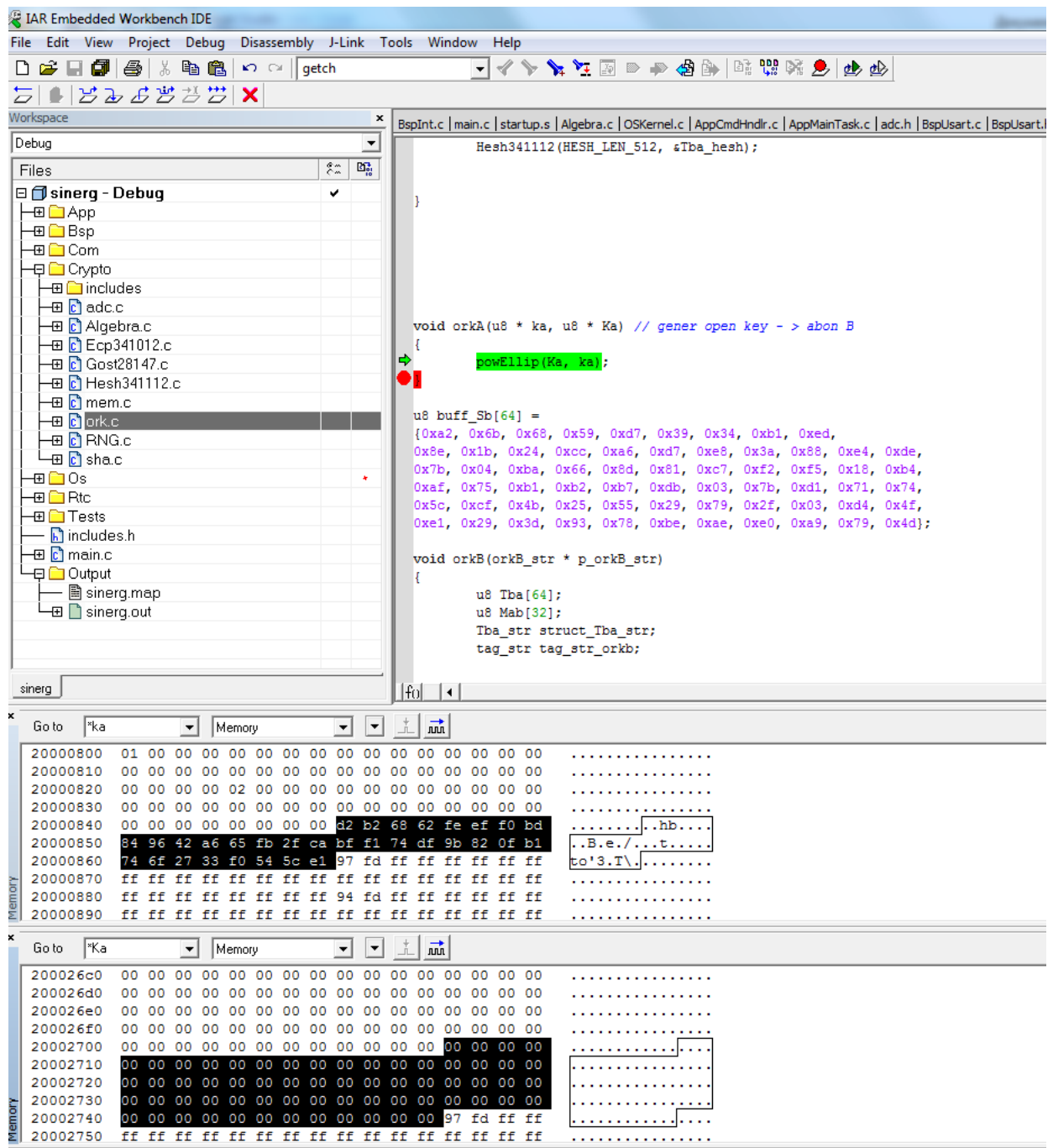


Рисунок 5.1 – Входные данные тестирования функции orkA ()

Ожидаемый результат: ea ff 4f 9c 9c 3f f0 b8 bd 01 1b 17 2f 6f ae ca 30 d1 f8 19 f7 e2 08 ec 6b ad 62 e3 aa c5 de 2f.

Фактический результат, представленный на рисунке 5.2, совпадает с ожидаемым результатом, взятом из тестового значения для выхода функции orkA(). Таким образом, можно сделать вывод, что функция orkA () реализована корректно.

d0 cf;

- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c 97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a 4f 9c bc 0f 29 60 19 3e 79 5c 5a 70 7c 91 5d 3a 55 a1 25 e9 a6 35 a6 85 7e 2a 32 94 1e 82 10 02;
- Значение UKM_A : 7c ec 6a f7 9e c4 ed 81;

Входные данные VKO_GOST () представлены на рисунке 5.3.

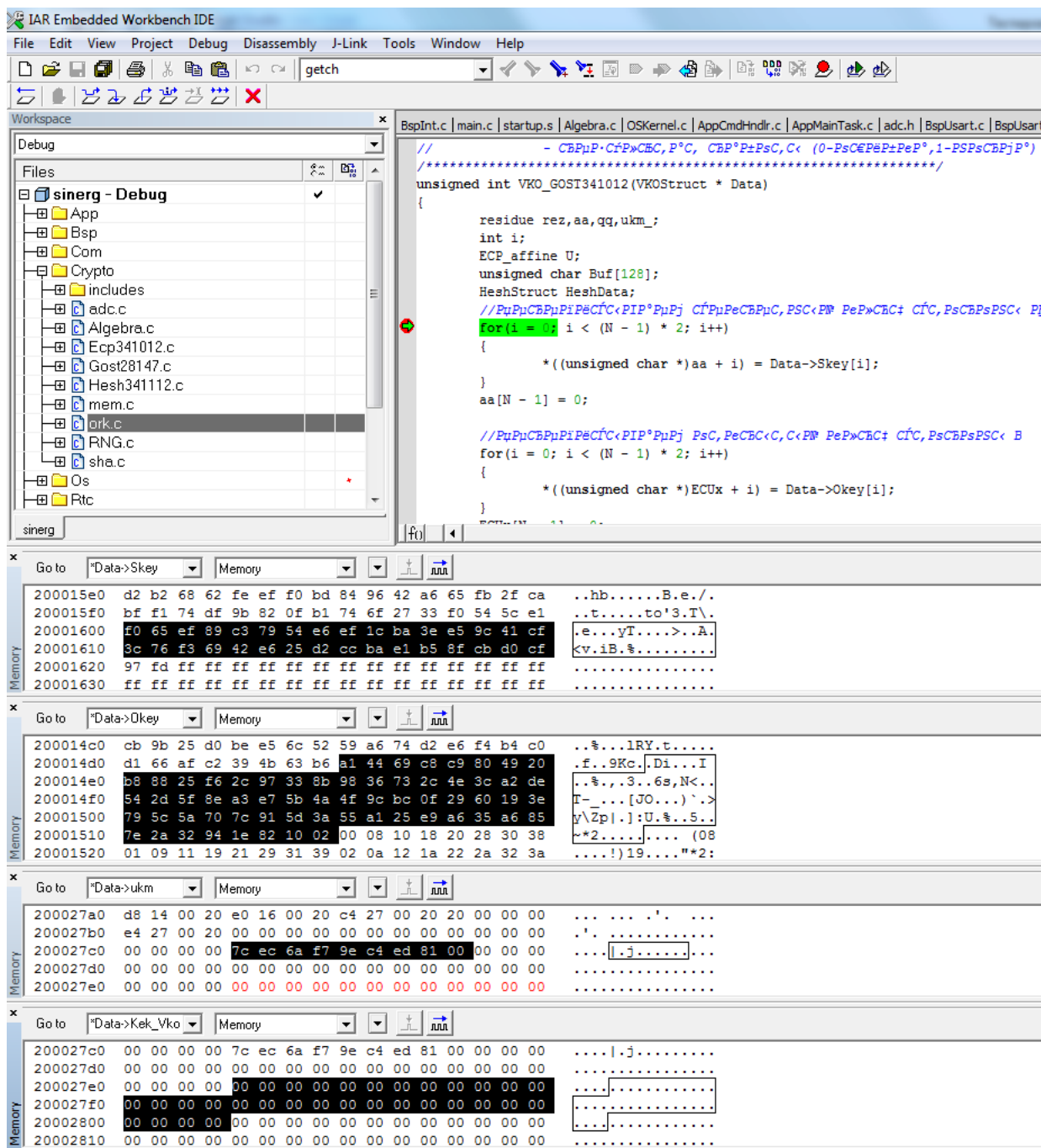


Рисунок 5.3 – Входные данные функции VKO_GOST ()

Ожидаемый результат: 25 97 fd 1f 61 26 6a 25 bf 5a 84 f0 cd 97 72 c0 d8 89 47 d3 e6 f6 a3 45 fc 0e 55 d8 6b d9 39 e4. Фактический результат функции VKO_GOST () представлен на рисунке 5.4.

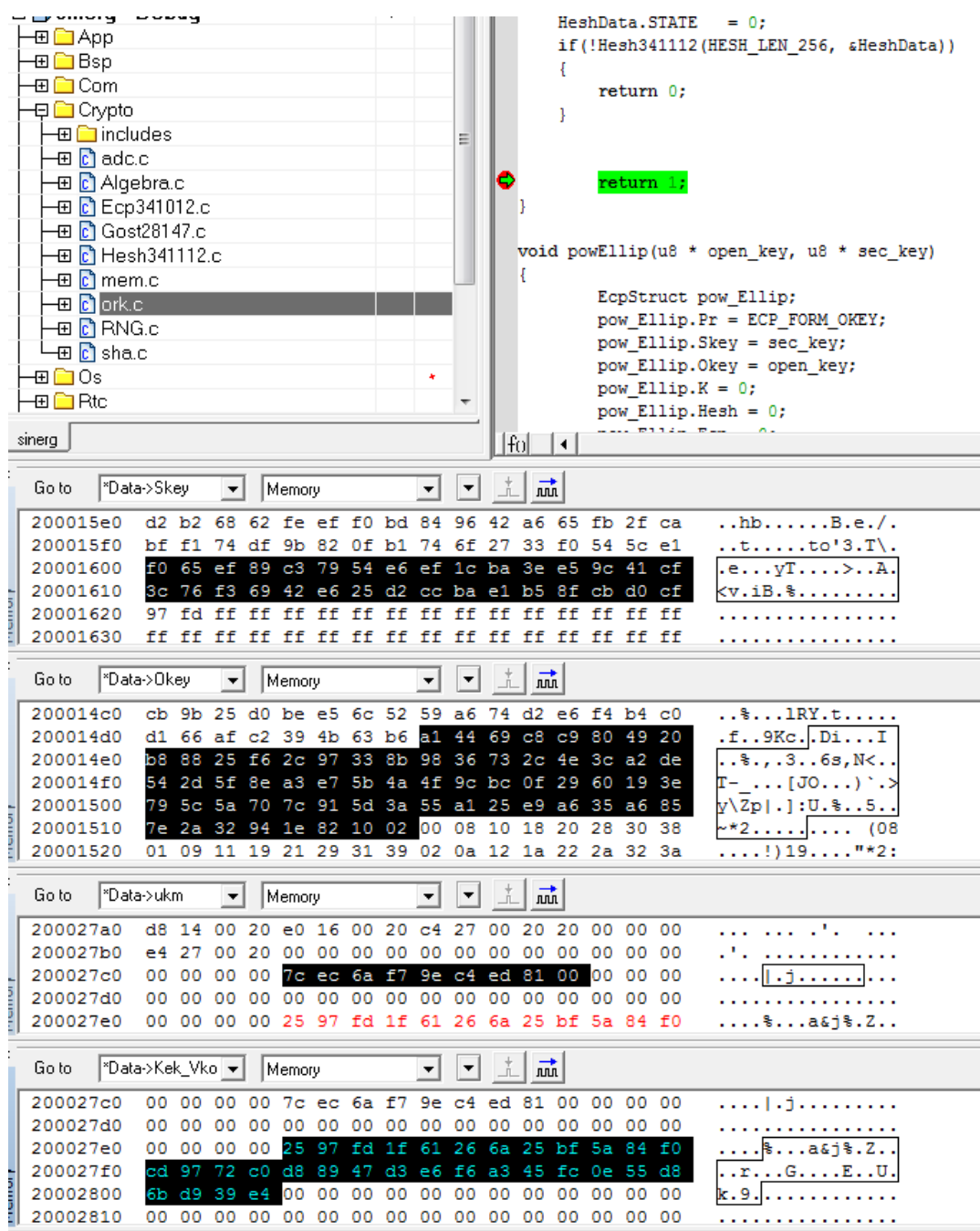


Рисунок 5.4 – Фактический результат функции VKO_GOST ()

Фактический результат, представленный на рисунке 5.4, совпадает с ожидаемым результатом, взятом из тестового значения для функции VKO_GOST(). Таким образом, можно сделать вывод, что функция VKO_GOST () реализована корректно. Далее было произведено модульное тестирование

VKO_GOST (), т.е. тестирование основных функций, вызываемых в функции VKO_GOST ().

5.2.2 Тестирование функции Tba ()

Было произведено интеграционное тестирование функции Tba (). Входными данными функции Tba () являются последовательности:

- Секретный долговременный ключ S_A : f0 65 ef 89 c3 79 54 e6 ef 1c ba 3e e5 9c 41 cf 3c 76 f3 69 42 e6 25 d2 cc ba e1 b5 8f cb d0 cf;
- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c 97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a 4f 9c bc 0f 29 60 19 3e 79 5c 5a 70 7c 91 5d 3a 55 a1 25 e9 a6 35 a6 85 7e 2a 32 94 1e 82 10 02;
- Открытый ключ S_B : a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7 e8 3a 88 e4 de 7b 04 ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db 03 7b d1 71 74 5c cf 4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be ae e0 a9 79 4d;
- Секретный ключ k_A : d2 b2 68 62 fe ef f0 bd 84 96 42 a6 65 fb 2f ca bf f1 74 df 9b 82 0f b1 74 6f 27 33 f0 54 5c e1;
- Идентификатор стороны A Id_A : 52 65 73 70 6f 6e 64 65 72;
- Идентификатор стороны B Id_B : 53 65 6e 64 65 72;
- Значение UKM_A : 7c ec 6a f7 9e c4 ed 81;
- Случайное число UKM_B : 3d 20 5c 6c 2b 76 3f ab;
- Сертификат $Cert_B$: 30 82 03 65 30 82 03 12 a0 03 02 01 02 02 11 00 c1 0a 67 5d 46 50 b9 80 e7 11 c5 fa 4a 56 03 eb 30 0a 06 08 2a 85 03 07 01 01 03 02 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 30 1e 17 0d 31 38 30 31 31 36 31 33 35 32 34 33 5a 17 0d 31 39 30 34 31 36 31 34 30 32 34 33 5a 30 11 31 0f 30 0d 06 03 55 04 03 0c 06 53 65 6e 64 65 72 30 66 30 1f 06 08 2a 85 03 07 01 01 01 01 30 13 06 07

2a 85 03 02 02 24 00 06 08 2a 85 03 07 01 01 02 02 03 43 00 04 40
 a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7 e8 3a 88 e4 de 7b 04
 ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db 03 7b d1 71 74 5c cf
 4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be ae e0 a9 79 4d a3 82
 02 02 30 82 01 fe 30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 03 a8
 30 1d 06 03 55 1d 0e 04 16 04 14 c4 02 97 8f 16 e2 3f 49 43 d6 6b
 e6 e7 af aa 38 fa ba 24 f4 30 1f 06 09 2b 06 01 04 01 82 37 15 07
 04 12 30 10 06 08 2a 85 03 02 02 2e 00 08 02 01 01 02 01 00 30 81
 83 06 03 55 1d 23 04 7c 30 7a 80 14 5e 94 aa dd 0f 73 aa 75 ba 76
 62 e2 80 b2 de 3d 49 06 77 3b a1 4f a4 4d 30 4b 31 18 30 16 06 05
 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a
 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37
 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f
 74 43 41 82 11 00 c1 0a 67 5d 46 50 b8 80 e7 11 41 df 43 c4 9f 41
 30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01 05 05 07 03 02 06
 08 2b 06 01 05 05 07 03 04 30 27 06 09 2b 06 01 04 01 82 37 15 0a
 04 1a 30 18 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0a 06 08 2b 06
 01 05 05 07 03 04 30 52 06 03 55 1d 1f 04 4b 30 49 30 47 a0 45 a0
 43 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 2f 63 64
 70 2f 35 65 39 34 61 61 64 64 30 66 37 33 61 61 37 35 62 61 37 36
 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30 36 37 37 33 62 2e 63
 72 6c 30 5d 06 08 2b 06 01 05 05 07 01 01 04 51 30 4f 30 4d 06 08
 2b 06 01 05 05 07 30 02 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c
 68 6f 73 74 2f 61 69 61 2f 35 65 39 34 61 61 64 64 30 66 37 33 61
 61 37 35 62 61 37 36 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30
 36 37 37 33 62 2e 63 72 74 30 2b 06 03 55 1d 10 04 24 30 22 80 0f
 32 30 31 38 30 31 31 36 31 33 35 32 34 32 5a 81 0f 32 30 31 39 30
 31 31 36 31 33 35 32 34 32 5a 30 0a 06 08 2a 85 03 07 01 01 03 02
 03 41 00 d9 0c 3c 27 93 c7 08 99 9e ef e3 a1 d7 01 49 13 29 38 99
 74 9a 70 a6 5c e4 c7 b2 4c 54 94 32 70 fa 1f 55 26 1a 9c 2a c4
 fe fb cf 44 07 b5 75 e1 25 a4 c0 5b 1f 37 73 72 0f 20 49 73 90 fb
 ad cb;

Входные данные представлены на рисунках 5.5 – 5.6.

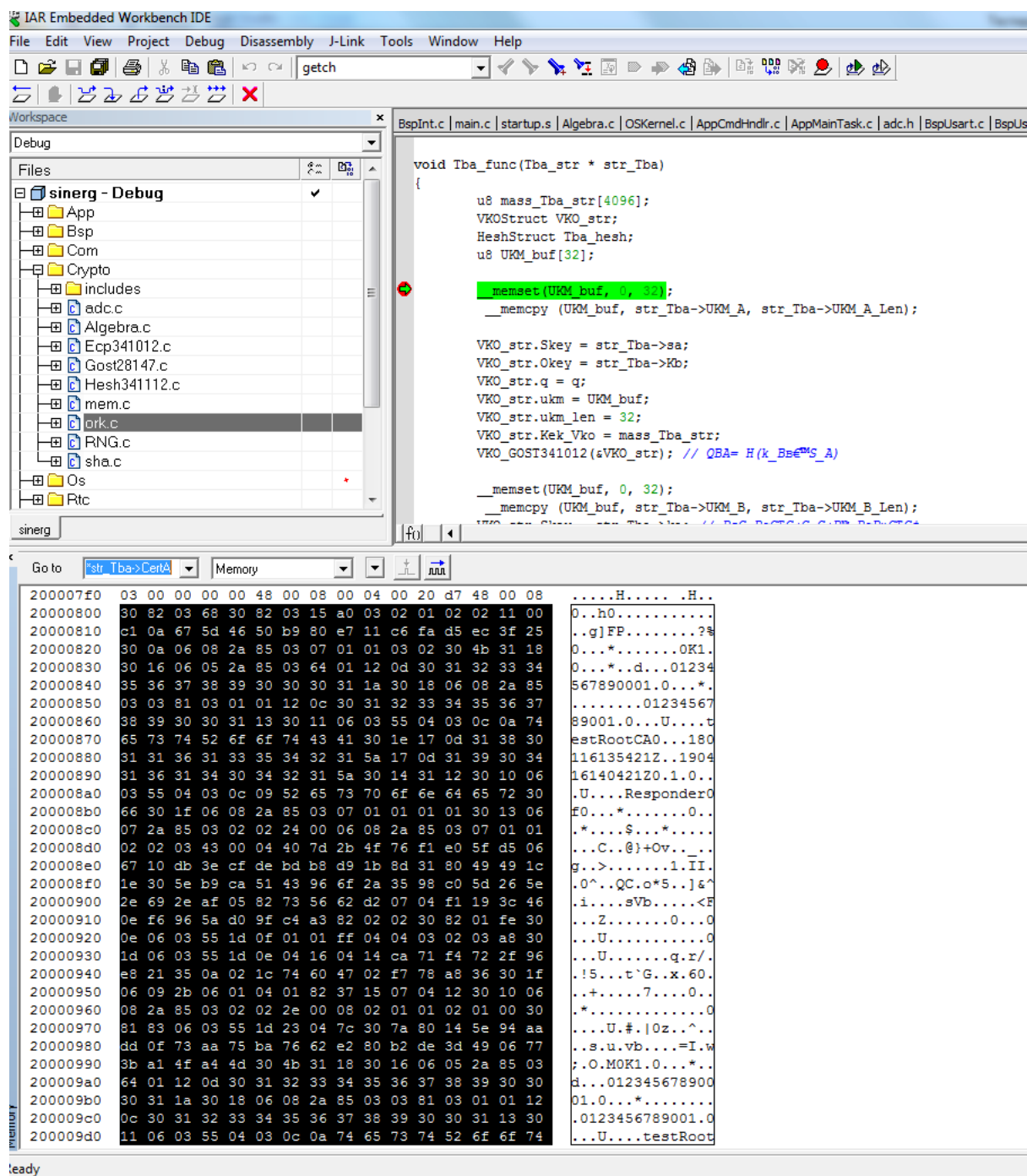


Рисунок 5.6 – Входные данные функции Tba ()

Ожидаемый результат: 6c 03 60 d9 0c f2 cb be b7 cf be 1a 4f 13 fe 68 d5 d5 c6 78 e3 e8 17 d4 3c 76 e7 56 11 30 6f aa 2d 35 d8 2a 8f 72 1f 37 77 ec 13 f3 7a 26 82 7f 82 80 55 c1 c5 ef e6 2e 29 2f 11 72 14 d0 cf 70. Фактический результат функции Tba () представлен на рисунке 5.7.

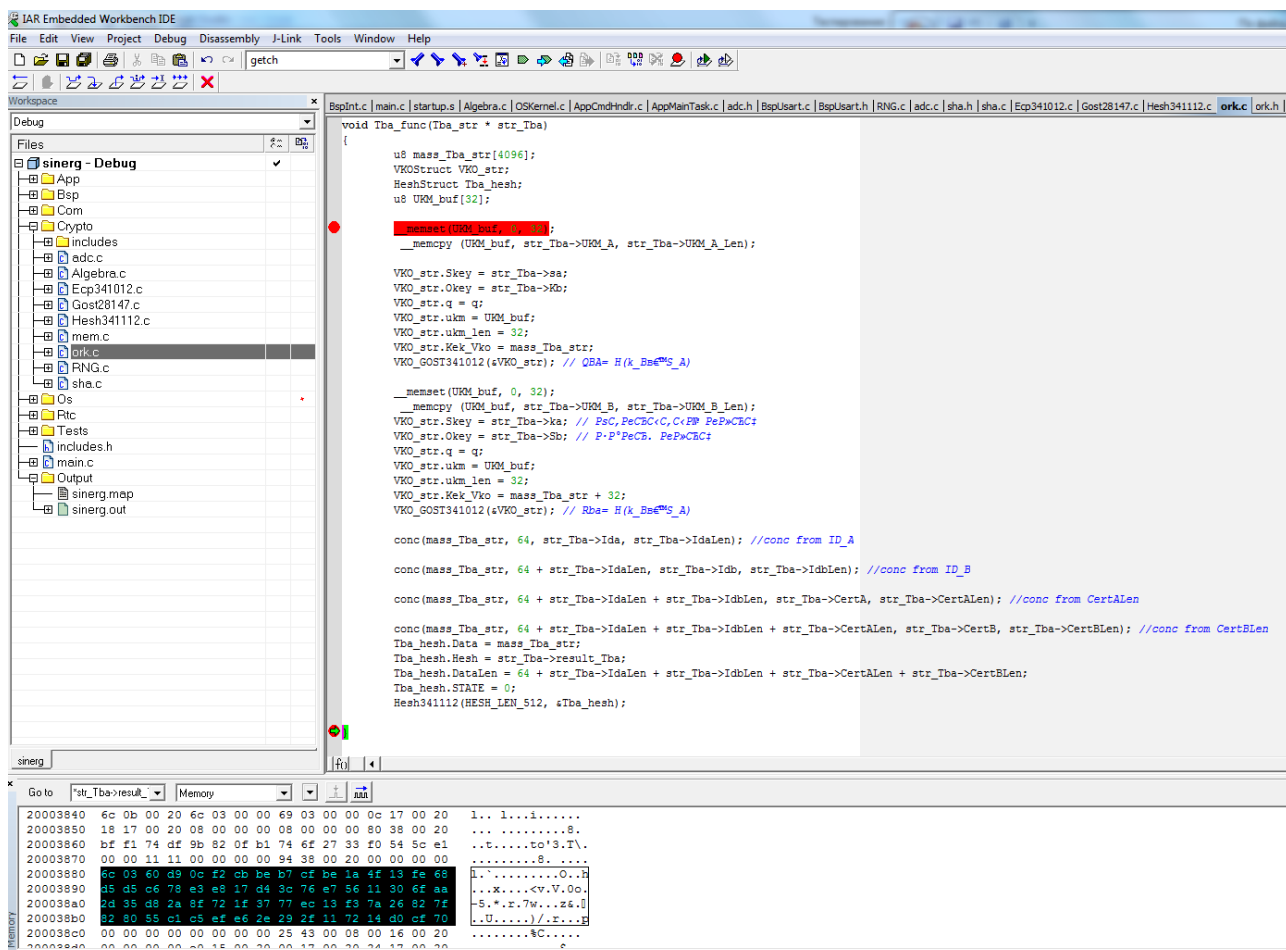


Рисунок 5.7 – Фактический результат функции Tab ()

Фактический результат, представленный на рисунке 5.7, совпадает с ожидаемым результатом, взятым из тестовых значений для функции Tab (). Таким образом, можно сделать вывод, что функция Tab () реализована корректно.

5.2.3 Тестирование функции tag ()

Было произведено интеграционное тестирование функции tag (). Входными данными являются последовательности:

- Секретный долговременный ключ K_A : ea ff 4f 9c 9c 3f f0 b8 bd 01 1b 17 2f 6f ae ca 30 d1 f8 19 f7 e2 08 ec 6b ad 62 e3 aa c5 de 2f 12 63 e4 e1 4f dd b2 b3 0e a0 12 8e 3b 76 24 f0 77 52 ee b8 8e 2d 28 59 5f 45 51 48 2e 7e 1f de;
- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c 97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a 4f 9c bc

0f 29 60 19 3e 79 5c 5a 70 7c 91 5d 3a 55 a1 25 e9 a6 35 a6 85 7e
2a 32 94 1e 82 10 02;

- Идентификатор стороны A Id_A : 52 65 73 70 6f 6e 64 65 72;
- Идентификатор стороны B Id_B : 53 65 6e 64 65 72;
- Значение UKM_A : 7c ec 6a f7 9e c4 ed 81;
- Случайное число UKM_B : 3d 20 5c 6c 2b 76 3f ab;
- Сеансовый ключ Ma_b : 2d 35 d8 2a 8f 72 1f 37 77 ec 13 f3 7a
26 82 7f 82 80 55 c1 c5 ef e6 2e 29 2f 11 72 14 d0 cf 70.

Входные данные функции tag () представлены на рисунке 5.8.

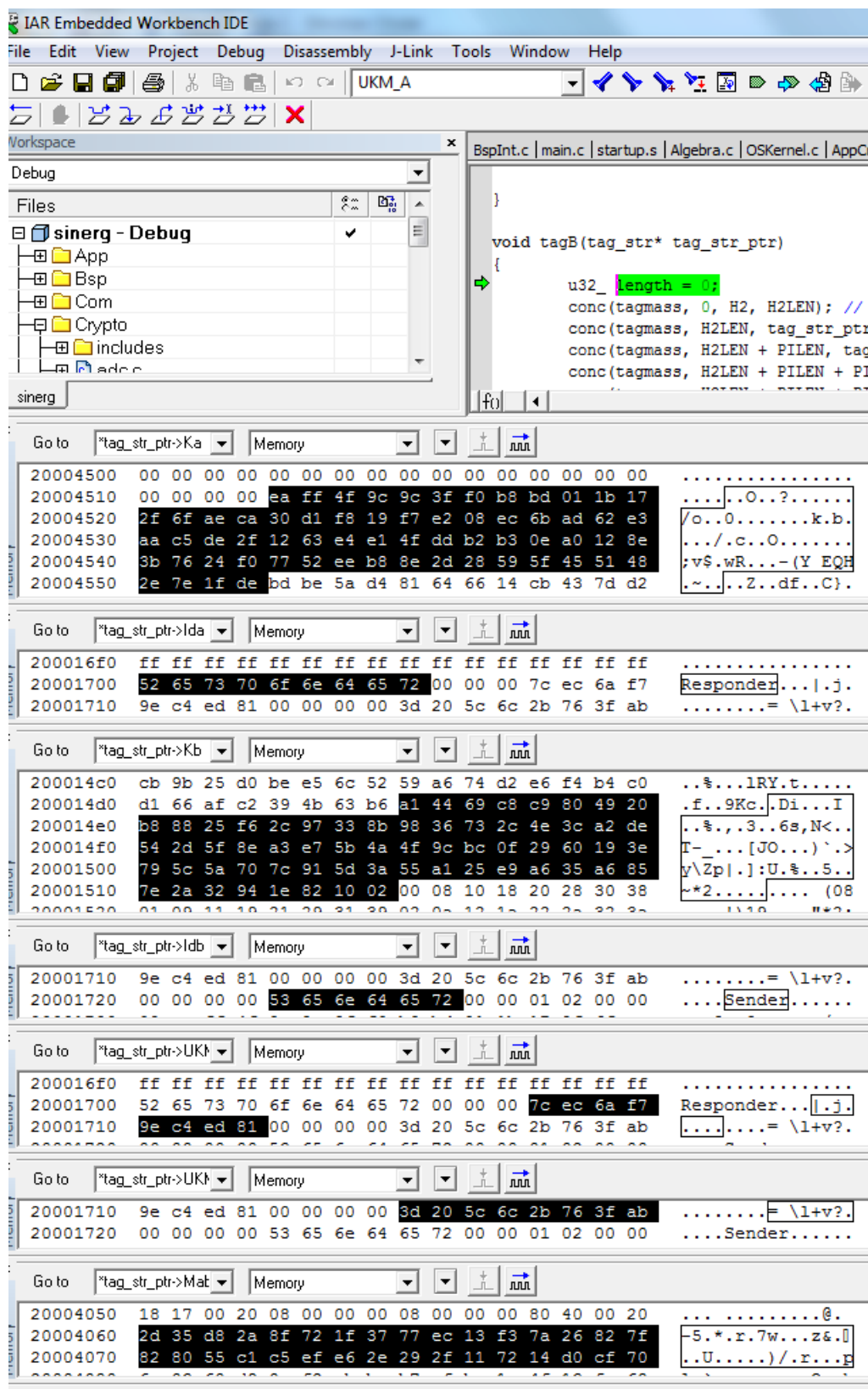


Рисунок 5.8 – Входные данные функции tag ()

Ожидаемый результат: 4f 6c a7 5e 59 e7 7d 2c 46 4b 32 58 46 96 56 59 9e 41 62 c4 5d 86 12 ae e0 72 a1 89 7a b4 77 22. Фактический результат функции tag () представлен на рисунке 5.9.

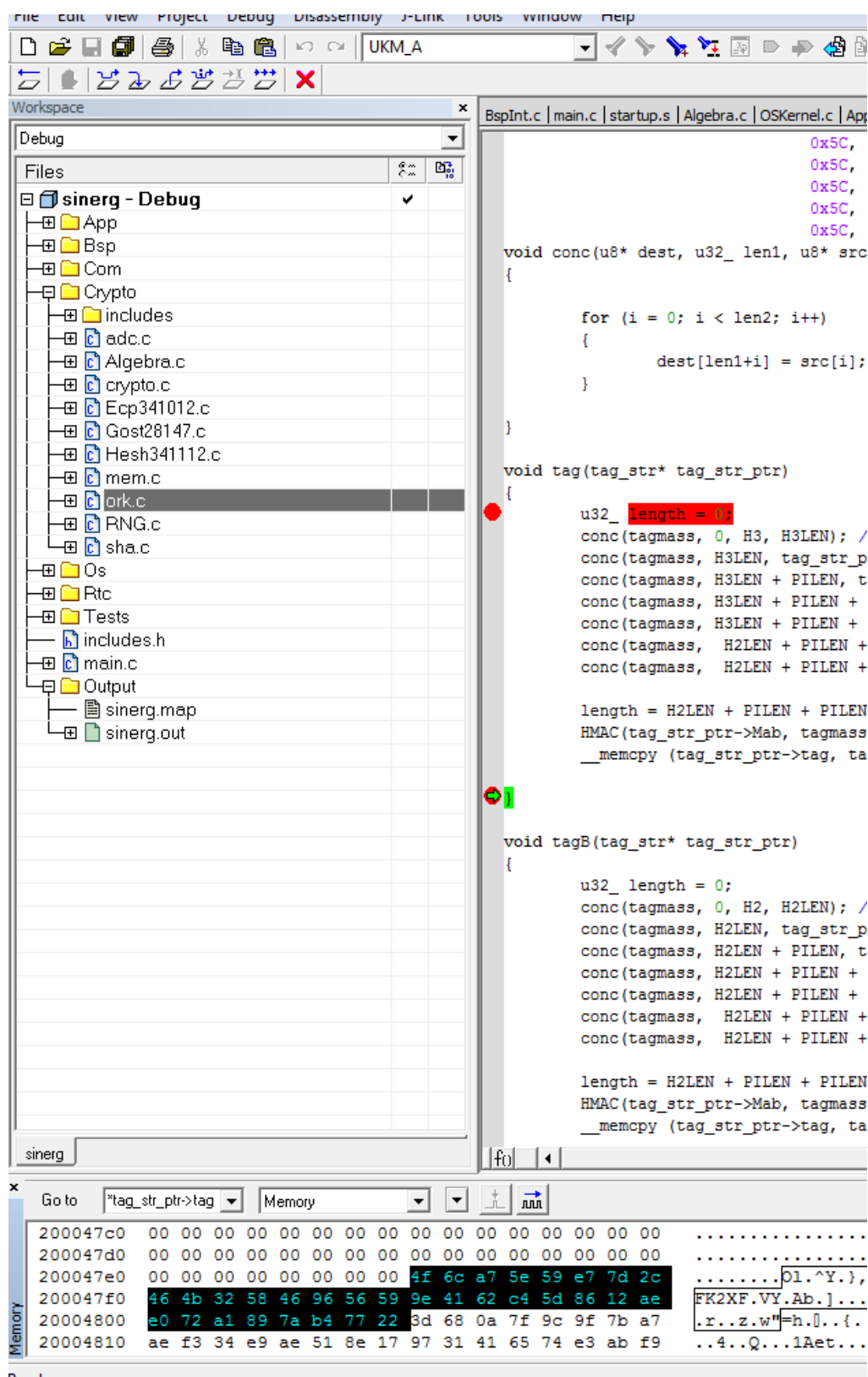


Рисунок 5.9 – Фактический результат функции tag ()

Фактический результат, представленный на рисунке 5.9, совпадает с ожидаемым результатом, взятом из тестовых значений для функции tag (). Таким образом, можно сделать вывод, что функция tag () реализована корректно.

5.2.4 Тестирование функции tagB ()

Было произведено интеграционное тестирование функции tagB ().
Входными данными являются последовательности:

- Секретный долговременный ключ K_A : ea ff 4f 9c 9c 3f f0 b8
bd 01 1b 17 2f 6f ae ca 30 d1 f8 19 f7 e2 08 ec 6b ad 62 e3 aa c5
de 2f 12 63 e4 e1 4f dd b2 b3 0e a0 12 8e 3b 76 24 f0 77 52 ee b8
8e 2d 28 59 5f 45 51 48 2e 7e 1f de;
- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c
97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a 4f 9c bc
0f 29 60 19 3e 79 5c 5a 70 7c 91 5d 3a 55 a1 25 e9 a6 35 a6 85 7e
2a 32 94 1e 82 10 02;
- Идентификатор стороны A Id_A : 52 65 73 70 6f 6e 64 65 72;
- Идентификатор стороны B Id_B : 53 65 6e 64 65 72;
- Значение UKM_A : 7c ec 6a f7 9e c4 ed 81;
- Случайное число UKM_B : 3d 20 5c 6c 2b 76 3f ab;
- Сеансовый ключ Mab : 2d 35 d8 2a 8f 72 1f 37 77 ec 13 f3 7a
26 82 7f 82 80 55 c1 c5 ef e6 2e 29 2f 11 72 14 d0 cf 70.

Входные данные функции tagB () представлены на рисунке 5.10.

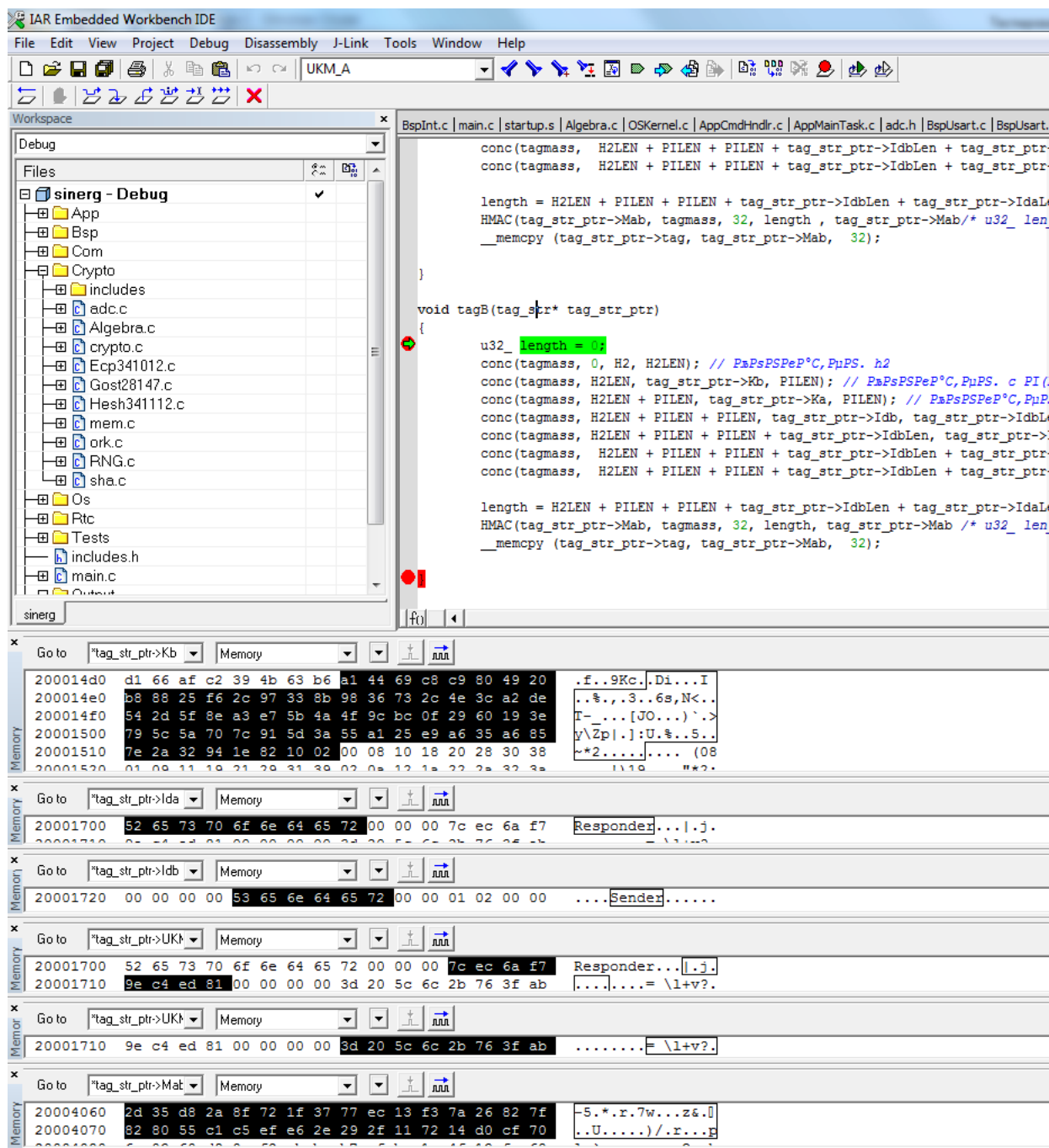


Рисунок 5.10 – Входные данные функции tagB ()

Ожидаемый результат: 3d 68 0a 7f 9c 9f 7b a7 ae f3 34 e9 ae 51 8e 17 97 31 41 65 74 e3 ab f9 1f 92 d8 70 dd 64 aa 55. Фактический результат функции tagB () представлен на рисунке 5.11.

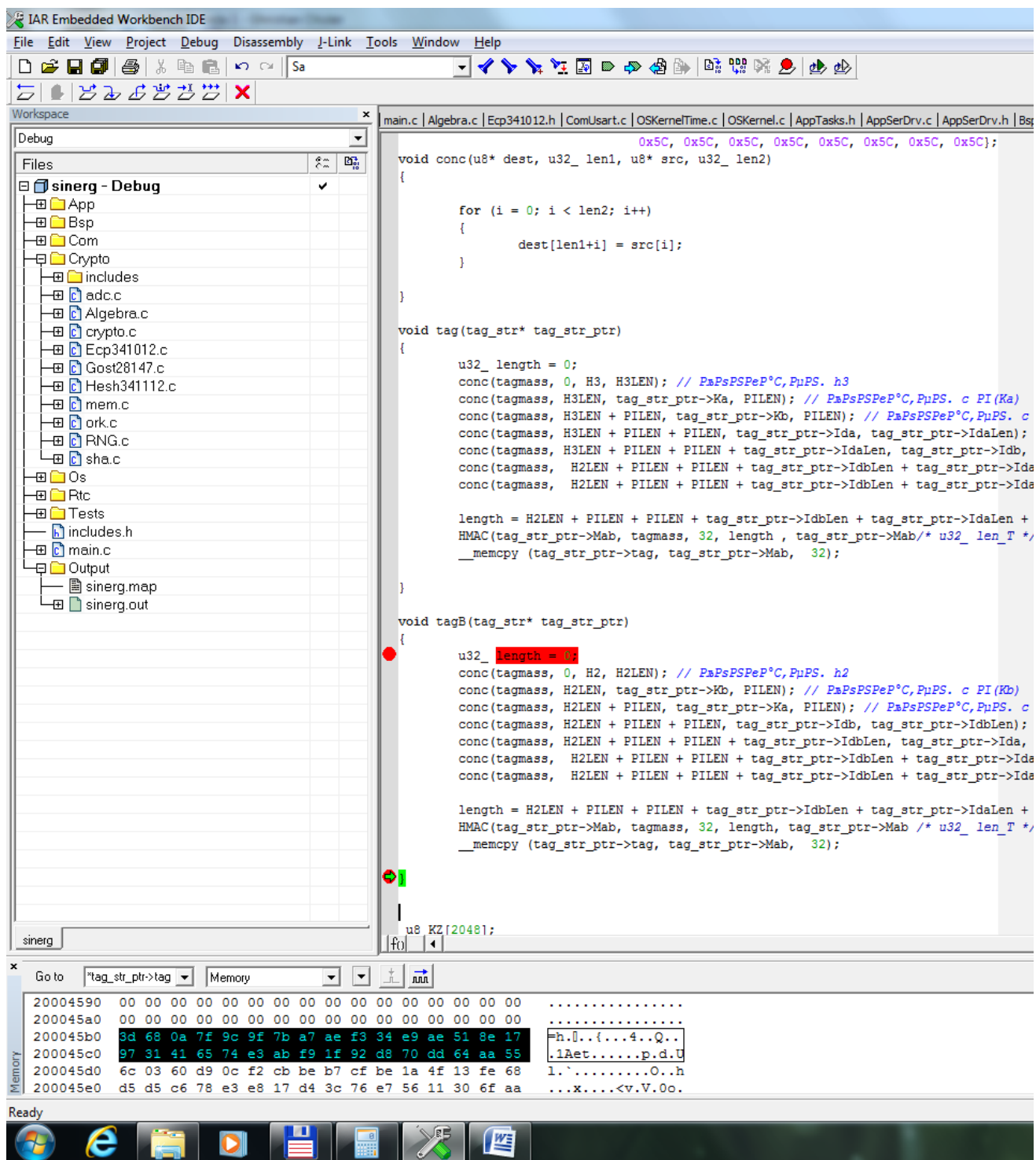


Рисунок 5.11 – Фактический результат функции tagB ()

Фактический результат, представленный на рисунке 5.11, совпадает с ожидаемым результатом, взятом из тестовых значений для функции tagB (). Таким образом, можно сделать вывод, что функция tagB () реализована корректно.

5.2.5 Тестирование функции HMAC ()

Было произведено интеграционное тестирование функции HMAC ().
Входными данными функции HMAC () являются последовательности:

- Сеансовый ключ M_{BA} : 2d 35 d8 2a 8f 72 1f 37 77 ec 13 f3 7a 26 82 7f 82 80 55 c1 c5 ef e6 2e 29 2f 11 72 14 d0 cf 70;
- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c 97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a.

Входные данные функции HMAC () представлены на рисунке 5.12.

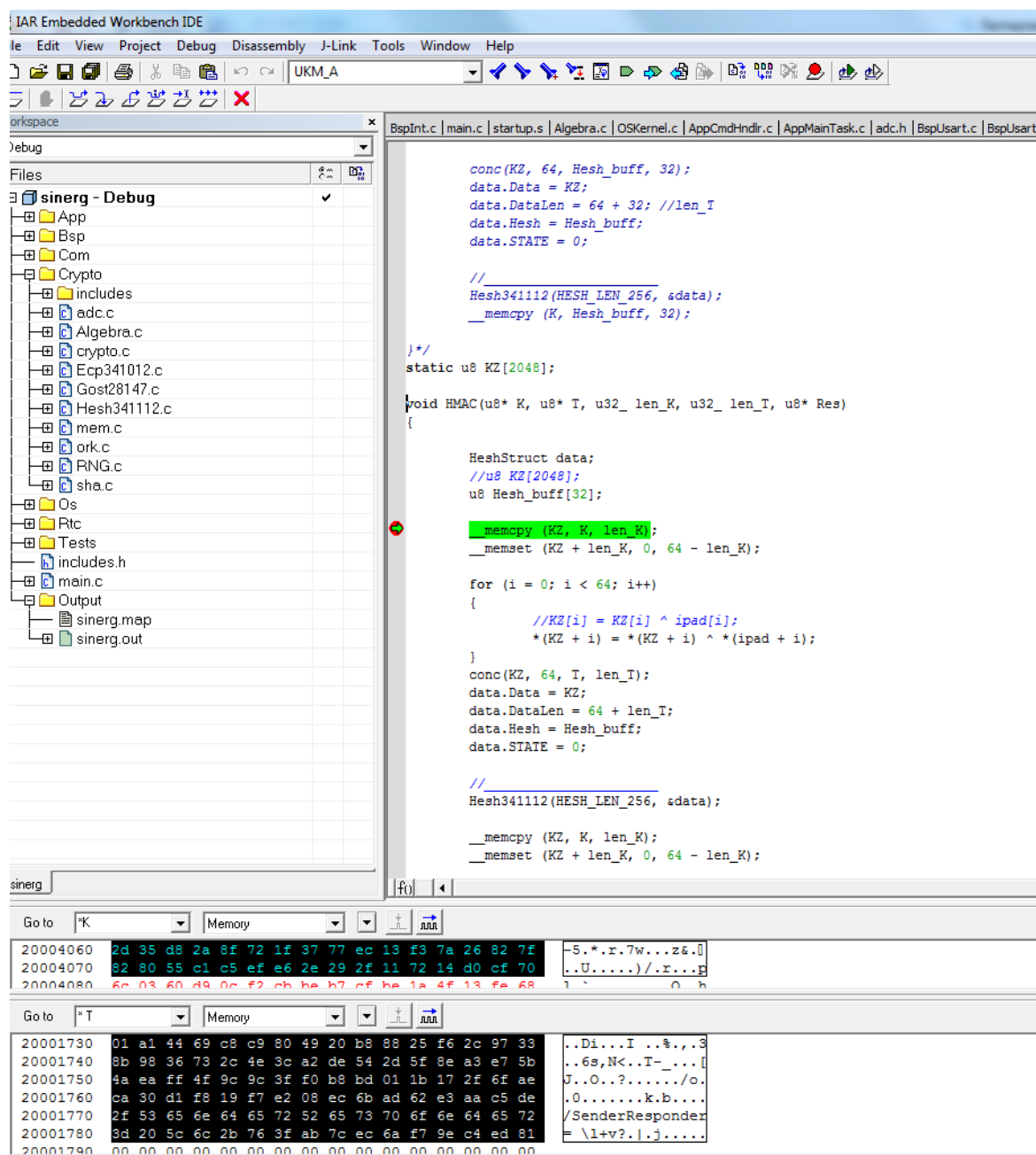


Рисунок 5.12 – Входные данные функции HMAC ()

Ожидаемый результат: 3d 68 0a 7f 9c 9f 7b a7 ae f3 34 e9 ae 51 8e 17 97 31 41 65 74 e3 ab f9 1f 92 d8 70 dd 64 aa 55. Фактический результат функции HMAC () представлен на рисунке 5.13.

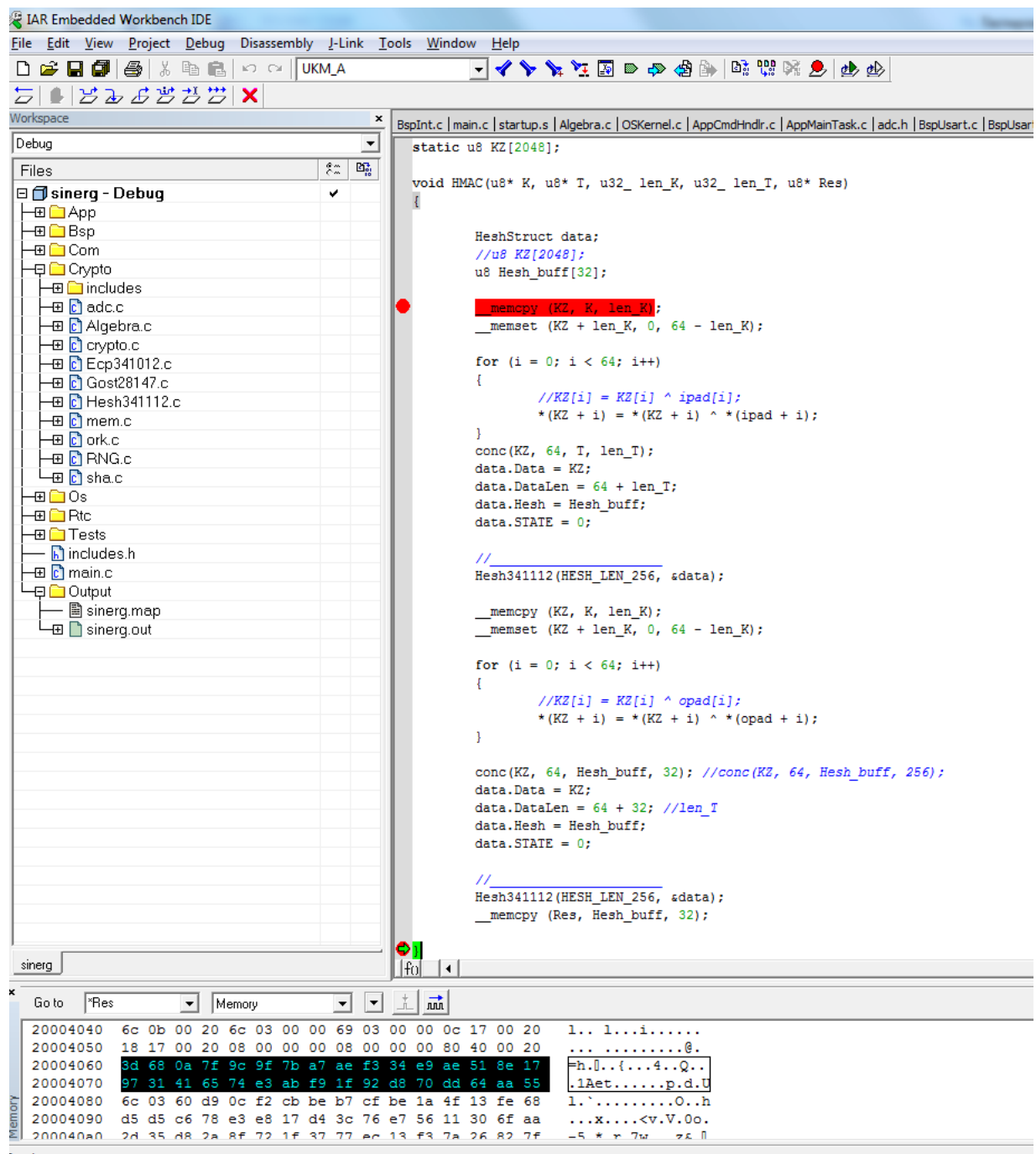


Рисунок 5.13 – Фактический результат функции HMAC ()

Фактический результат, представленный на рисунке 5.13, совпадает с ожидаемым результатом, взятом из тестовых значений для функции HMAC (). Таким образом, можно сделать вывод, что функция HMAC () реализована корректно.

5.2.6 Тестирование функции Chech_valid ()

Было произведено интеграционное тестирование функции Chech_valid ().

Входными данными функции Chech_valid () являются последовательности:

– Ключ проверки ЭП Escr_key: c0 2e 00 11 6b 8b 7e ad b8 3b 4e 8c 0e fc 18 b1 81 c8 41 98 52 bc 16 78 e6 a2 23 80 1b 1c 7c 50 ea d4 ac 22 09 c6 3b 6c 77 be 25 33 ff c3 3d b1 0a 93 83 52 38 5d 95 9d 60 94 41 f2 67 b2 2c 9e;

– Сертификат Cert_B: 30 82 03 65 30 82 03 12 a0 03 02 01 02 02 11 00 c1 0a 67 5d 46 50 b9 80 e7 11 c5 fa 4a 56 03 eb 30 0a 06 08 2a 85 03 07 01 01 03 02 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 30 1e 17 0d 31 38 30 31 31 36 31 33 35 32 34 33 5a 17 0d 31 39 30 34 31 36 31 34 30 32 34 33 5a 30 11 31 0f 30 0d 06 03 55 04 03 0c 06 53 65 6e 64 65 72 30 66 30 1f 06 08 2a 85 03 07 01 01 01 01 30 13 06 07 2a 85 03 02 02 24 00 06 08 2a 85 03 07 01 01 02 02 03 43 00 04 40 a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7 e8 3a 88 e4 de 7b 04 ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db 03 7b d1 71 74 5c cf 4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be ae e0 a9 79 4d a3 82 02 02 30 82 01 fe 30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 03 a8 30 1d 06 03 55 1d 0e 04 16 04 14 c4 02 97 8f 16 e2 3f 49 43 d6 6b e6 e7 af aa 38 fa ba 24 f4 30 1f 06 09 2b 06 01 04 01 82 37 15 07 04 12 30 10 06 08 2a 85 03 02 02 2e 00 08 02 01 01 02 01 00 30 81 83 06 03 55 1d 23 04 7c 30 7a 80 14 5e 94 aa dd 0f 73 aa 75 ba 76 62 e2 80 b2 de 3d 49 06 77 3b a1 4f a4 4d 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 82 11 00 c1 0a 67 5d 46 50 b8 80 e7 11 41 df 43 c4 9f 41 30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01 05 05 07 03 02 06 08 2b 06 01 05 05 07 03 04 30 27 06 09 2b 06 01 04 01 82 37 15 0a 04 1a 30 18 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0a 06 08 2b 06

01 05 05 07 03 04 30 52 06 03 55 1d 1f 04 4b 30 49 30 47 a0 45 a0
43 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 2f 63 64
70 2f 35 65 39 34 61 61 64 64 30 66 37 33 61 61 37 35 62 61 37 36
36 32 65 32 38 30 62 32 64 65 33 64 34 39 30 36 37 37 33 62 2e 63
72 6c 30 5d 06 08 2b 06 01 05 05 07 01 01 04 51 30 4f 30 4d 06 08
2b 06 01 05 05 07 30 02 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c
68 6f 73 74 2f 61 69 61 2f 35 65 39 34 61 61 64 64 30 66 37 33 61
61 37 35 62 61 37 36 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30
36 37 37 33 62 2e 63 72 74 30 2b 06 03 55 1d 10 04 24 30 22 80 0f
32 30 31 38 30 31 31 36 31 33 35 32 34 32 5a 81 0f 32 30 31 39 30
31 31 36 31 33 35 32 34 32 5a 30 0a 06 08 2a 85 03 07 01 01 03 02
03 41 00 d9 0c 3c 27 93 c7 08 99 9e ef e3 a1 d7 01 49 13 29 38 99
74 9a 70 a6 5c e4 c7 b2 4c 54 94 32 70 fa 1f 55 26 1a 9c 2a c4 fe
fb cf 44 07 b5 75 e1 25 a4 c0 5b 1f 37 73 72 0f 20 49 73 90 fb ad
cb.

Входные данные функции Chech_valid () представлены на рисунке 5.14.

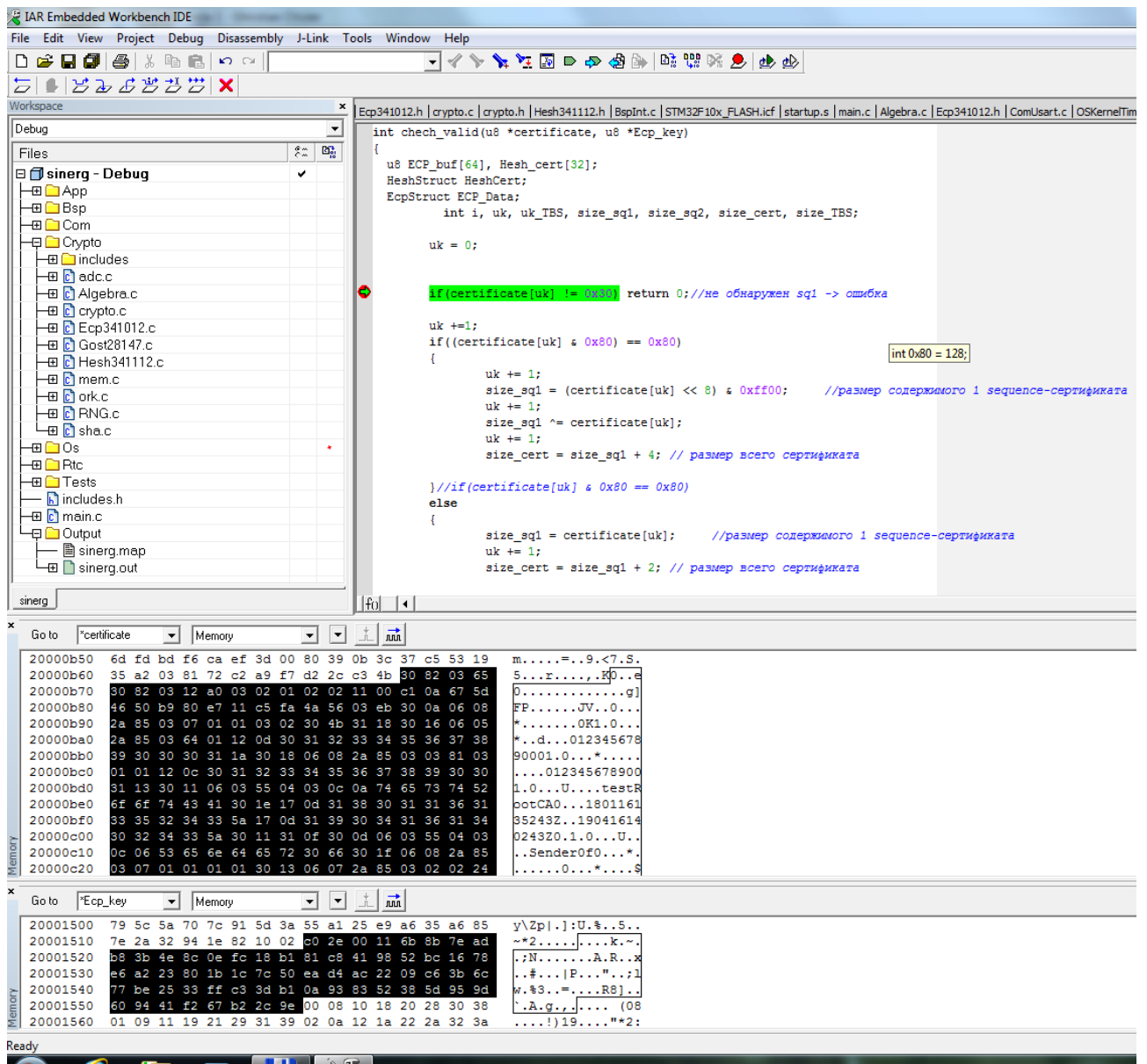


Рисунок 5.14 – Входные данные функции Chech_valid ()

Ожидаемый результат: функция должна вернуть 1 в случае успешной проверки сертификата. Фактический результат функции Chech_valid () представлен на рисунке 5.15.

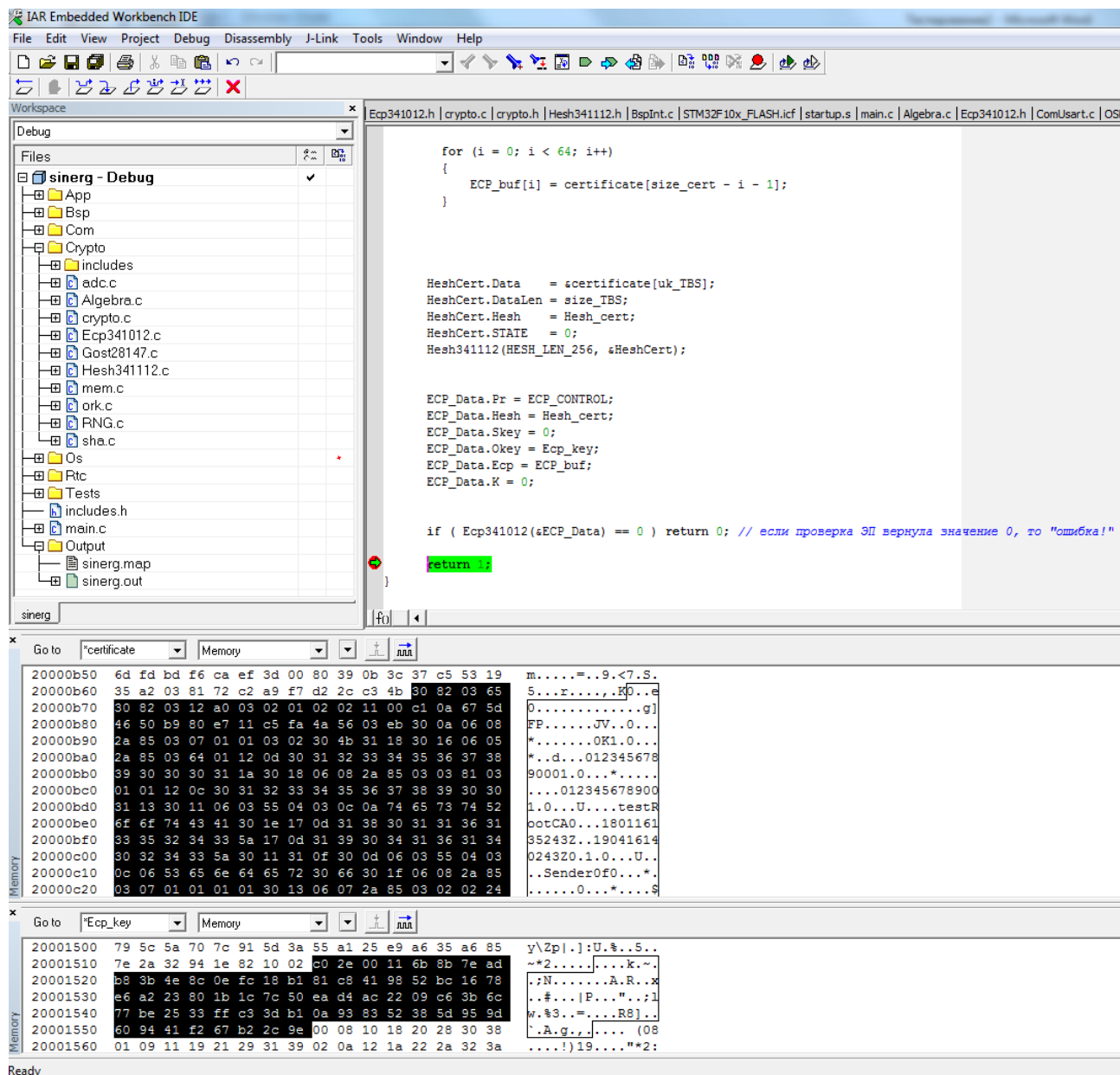


Рисунок 5.15 – Фактический результат функции `Chesh_valid ()` при позитивном тестировании

Фактический результат, представленный на рисунке 5.15, совпадает с ожидаемым результатом 1.

Далее сертификат был намеренно изменен. Ожидаемый результат: в случае неуспешной проверки сертификата функция `Chesh_valid ()` должна вернуть 0. Фактический результат функции `Chesh_valid ()` представлен на рисунке 5.16.

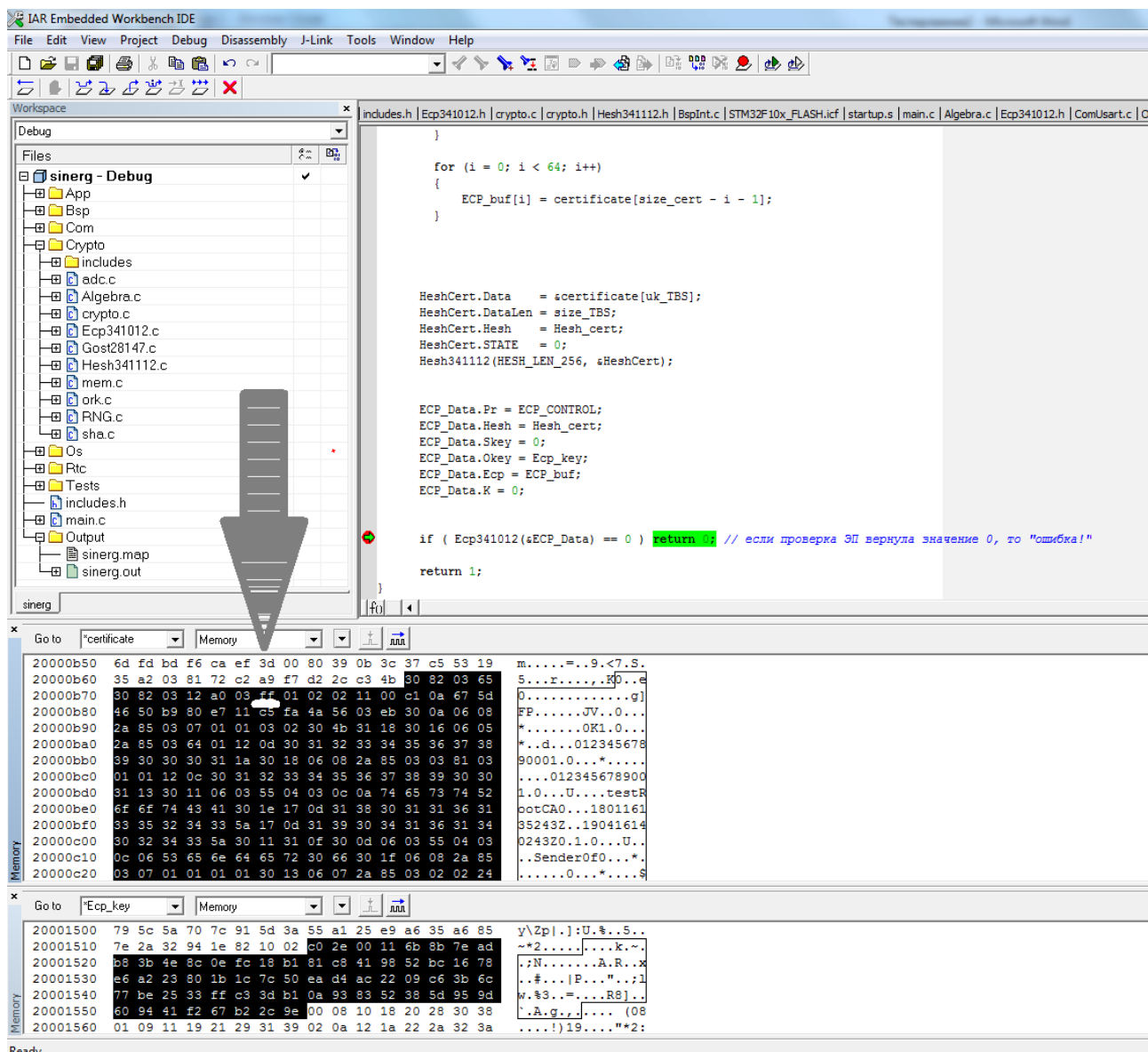


Рисунок 5.16 – Фактический результат функции `Chesh_valid()` при негативном тестировании

Фактический результат, представленный на рисунке 5.16, совпадает с ожидаемым результатом 0. Таким образом, можно сделать вывод, что функция `Chesh_valid()` реализована корректно.

5.2.7 Тестирование функции `unbox_Cert()`

Было произведено интеграционное тестирование функции `unbox_Cert()`. Входными данными функции `unbox_Cert()` являются последовательности:

- Ключ проверки ЭП `Ecp_key`: c0 2e 00 11 6b 8b 7e ad b8 3b 4e 8c 0e fc 18 b1 81 c8 41 98 52 bc 16 78 e6 a2 23 80 1b 1c 7c 50 ea

d4 ac 22 09 c6 3b 6c 77 be 25 33 ff c3 3d b1 0a 93 83 52 38 5d 95
9d 60 94 41 f2 67 b2 2c 9e;

– Сертификат Cert_B: 30 82 03 65 30 82 03 12 a0 03 02 01 02 02
11 00 c1 0a 67 5d 46 50 b9 80 e7 11 c5 fa 4a 56 03 eb 30 0a 06 08
2a 85 03 07 01 01 03 02 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12
0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85
03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13
30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 30 1e 17
0d 31 38 30 31 31 36 31 33 35 32 34 33 5a 17 0d 31 39 30 34 31 36
31 34 30 32 34 33 5a 30 11 31 0f 30 0d 06 03 55 04 03 0c 06 53 65
6e 64 65 72 30 66 30 1f 06 08 2a 85 03 07 01 01 01 01 30 13 06 07
2a 85 03 02 02 24 00 06 08 2a 85 03 07 01 01 02 02 03 43 00 04 40
a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7 e8 3a 88 e4 de 7b 04
ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db 03 7b d1 71 74 5c cf
4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be ae e0 a9 79 4d a3 82
02 02 30 82 01 fe 30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 03 a8
30 1d 06 03 55 1d 0e 04 16 04 14 c4 02 97 8f 16 e2 3f 49 43 d6 6b
e6 e7 af aa 38 fa ba 24 f4 30 1f 06 09 2b 06 01 04 01 82 37 15 07
04 12 30 10 06 08 2a 85 03 02 02 2e 00 08 02 01 01 02 01 00 30 81
83 06 03 55 1d 23 04 7c 30 7a 80 14 5e 94 aa dd 0f 73 aa 75 ba 76
62 e2 80 b2 de 3d 49 06 77 3b a1 4f a4 4d 30 4b 31 18 30 16 06 05
2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a
30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37
38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f
74 43 41 82 11 00 c1 0a 67 5d 46 50 b8 80 e7 11 41 df 43 c4 9f 41
30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01 05 05 07 03 02 06
08 2b 06 01 05 05 07 03 04 30 27 06 09 2b 06 01 04 01 82 37 15 0a
04 1a 30 18 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0a 06 08 2b 06
01 05 05 07 03 04 30 52 06 03 55 1d 1f 04 4b 30 49 30 47 a0 45 a0
43 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 2f 63 64
70 2f 35 65 39 34 61 61 64 64 30 66 37 33 61 61 37 35 62 61 37 36
36 32 65 32 38 30 62 32 64 65 33 64 34 39 30 36 37 37 33 62 2e 63
72 6c 30 5d 06 08 2b 06 01 05 05 07 01 01 04 51 30 4f 30 4d 06 08
2b 06 01 05 05 07 30 02 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c
68 6f 73 74 2f 61 69 61 2f 35 65 39 34 61 61 64 64 30 66 37 33 61

61 37 35 62 61 37 36 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30
 36 37 37 33 62 2e 63 72 74 30 2b 06 03 55 1d 10 04 24 30 22 80 0f
 32 30 31 38 30 31 31 36 31 33 35 32 34 32 5a 81 0f 32 30 31 39 30
 31 31 36 31 33 35 32 34 32 5a 30 0a 06 08 2a 85 03 07 01 01 03 02
 03 41 00 d9 0c 3c 27 93 c7 08 99 9e ef e3 a1 d7 01 49 13 29 38 99
 74 9a 70 a6 5c e4 c7 b2 4c 54 94 32 70 fa 1f 55 26 1a 9c 2a c4 fe
 fb cf 44 07 b5 75 e1 25 a4 c0 5b 1f 37 73 72 0f 20 49 73 90 fb ad
 cb.

Входные данные функции unbox_Cert () представлены на рисунке 5.17.

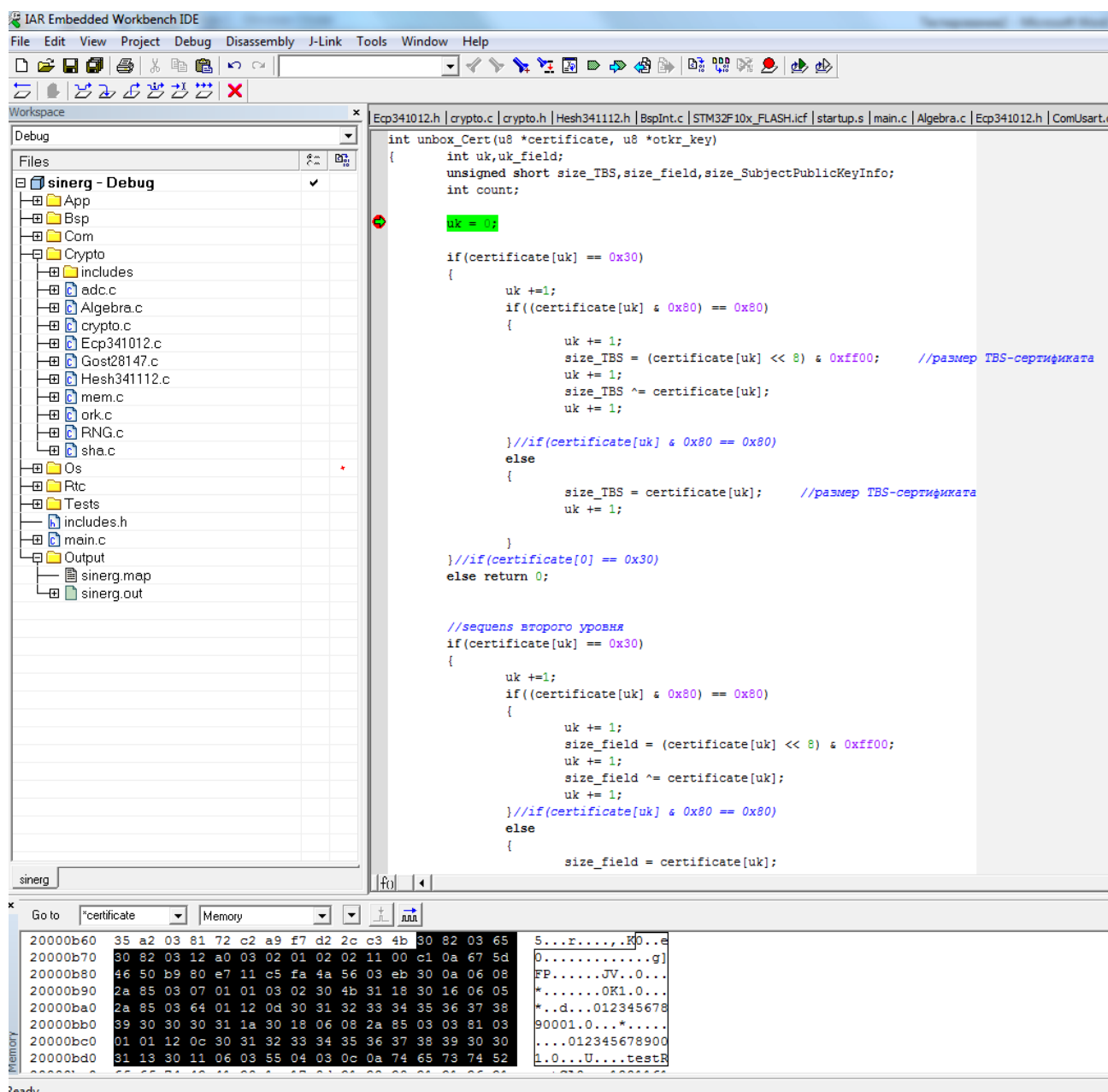


Рисунок 5.17 – Входные данные функции unbox_Cert ()

Ожидаемый результат: a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7

e8 3a 88 e4 de 7b 04 ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db
03 7b d1 71 74 5c cf 4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be
ae e0 a9 79 4d. Фактический результат функции `unbox_Cert()` представлен на
рисунке 5.18.

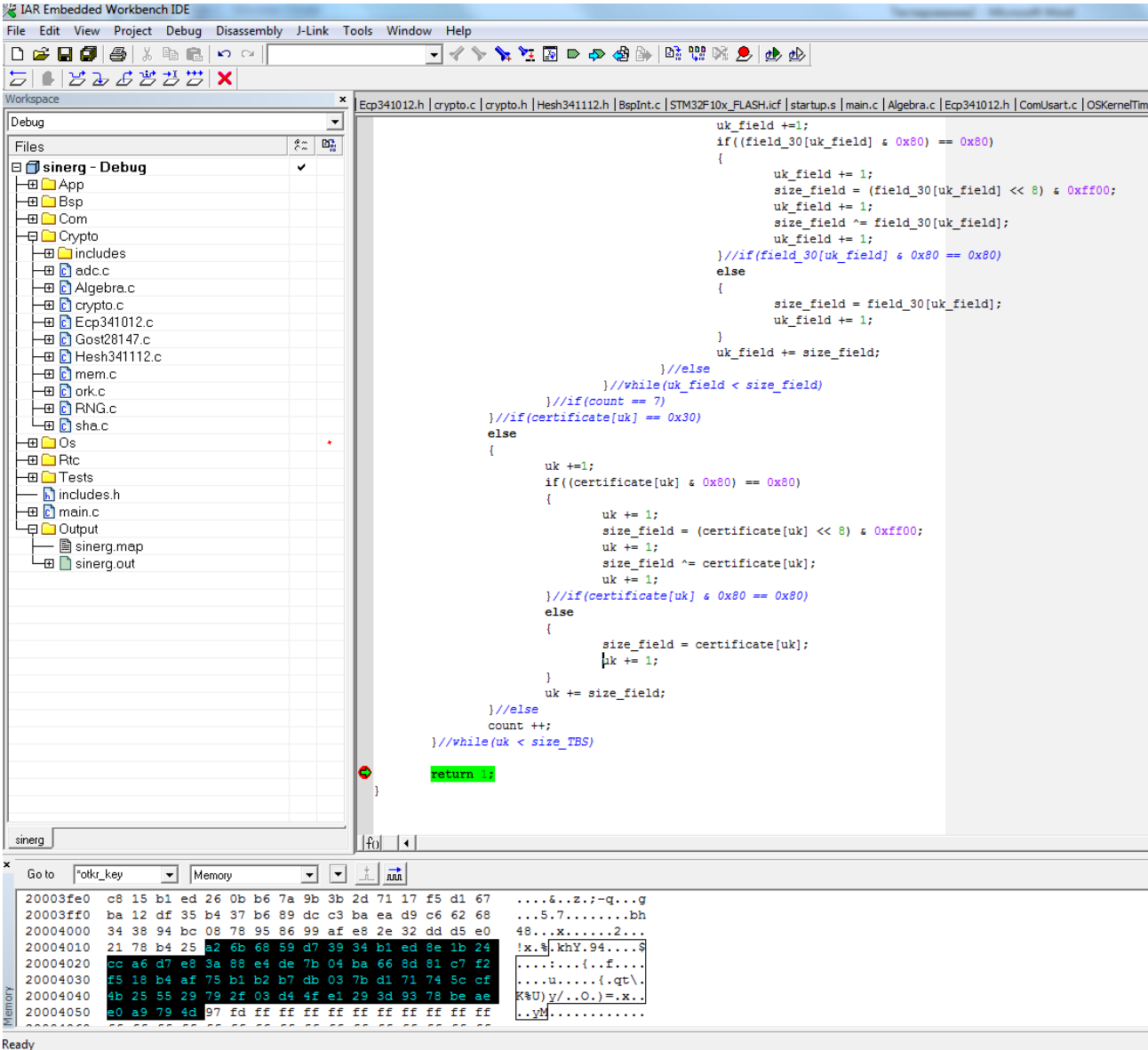


Рисунок 5.18 – Фактический результат функции `unbox Cert()`

Фактический результат, представленный на рисунке 5.18, совпадает с ожидаемым результатом, взятом из тестовых значений для функции `unbox_Cert()`. Таким образом, можно сделать вывод, что функция `unbox_Cert()` реализована корректно.

5.2.8 Тестирование функции Hesh341112 ()

Было произведено интеграционное тестирование функции Hesh341112 () с длиной хэш-кода 256 бит. Входными данными функции является последовательность: 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32.

Входные данные функции Hesh341112 () с длиной хэш-кода 256 бит представлены на рисунке 5.19.

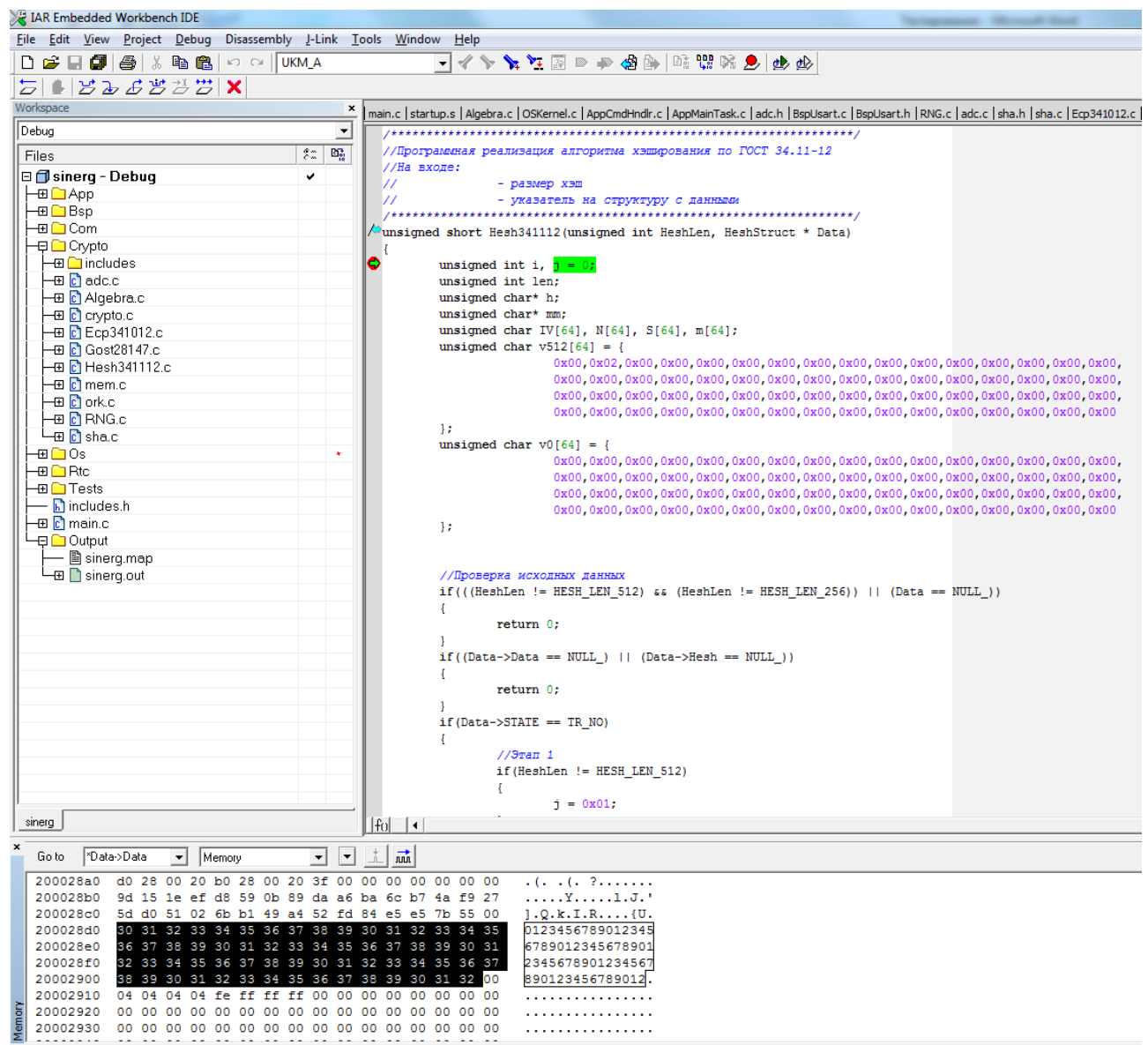


Рисунок 5.19 – Входные данные функции Hesh341112 () с длиной хэш-кода 256 бит

Ожидаемый результат: 9d 15 1e ef d8 59 0b 89 da a6 ba 6c b7 4a f9 27 5d d0 51 02 6b b1 49 a4 52 fd 84 e5 e5 7b 55 00. Фактический результат функции Hesh341112 () с длиной хэш-кода 256 бит представлен на рисунке 5.20.

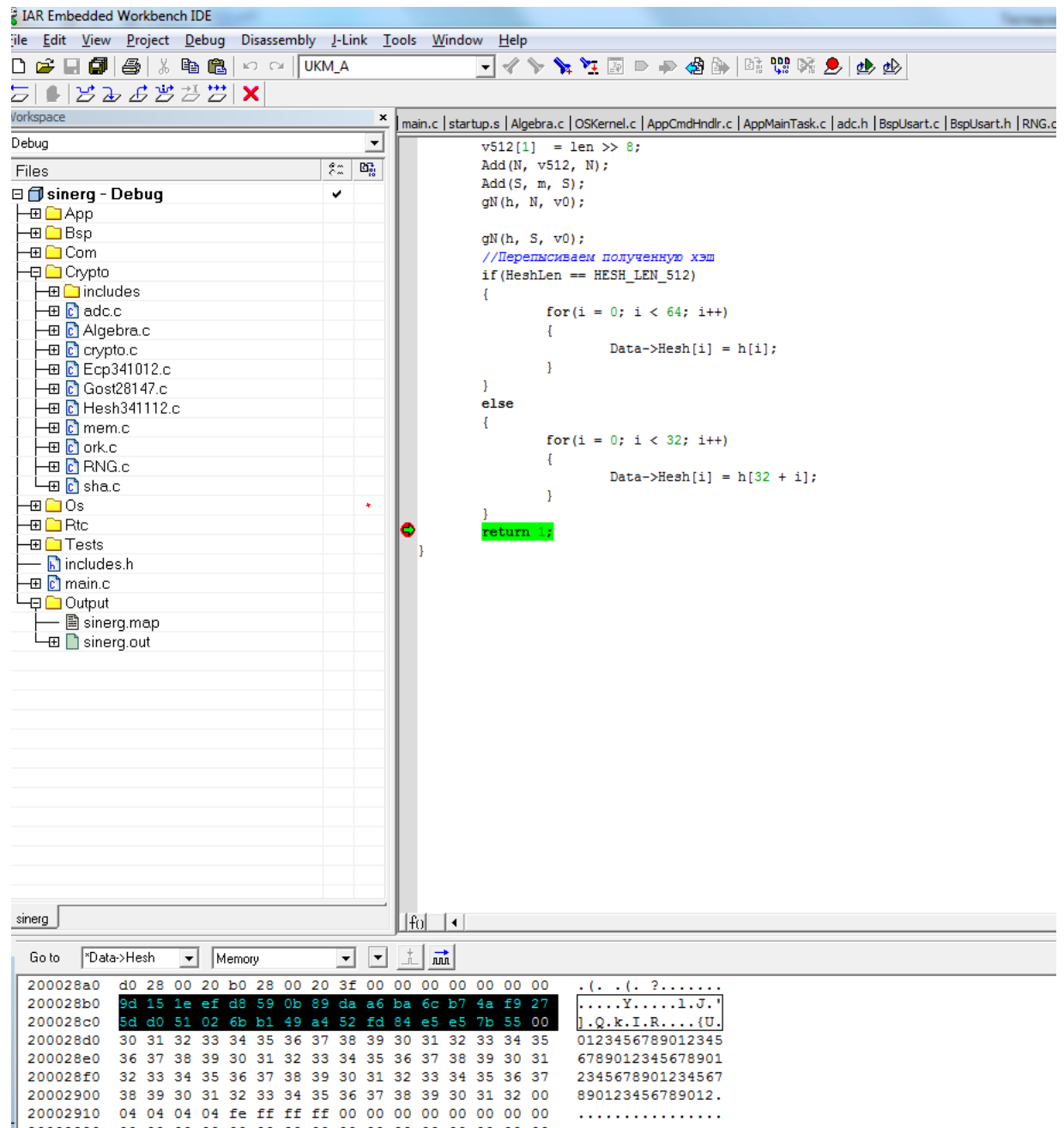


Рисунок 5.20 – Фактический результат функции Hesh341112 () с длиной хэш-кода 256 бит

Фактический результат, представленный на рисунке 5.20, совпадает с ожидаемым результатом, взятом из контрольных примеров для функции Hesh341112 () с длиной хэш-кода 256 бит.

Было произведено интеграционное тестирование функции Hesh341112 () с длиной хэш-кода 512 бит. Входными данными функции является последовательность: 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32.

Входные данные функции Hesh341112 () с длиной хэш-кода 512 бит представлены на рисунке 5.21.

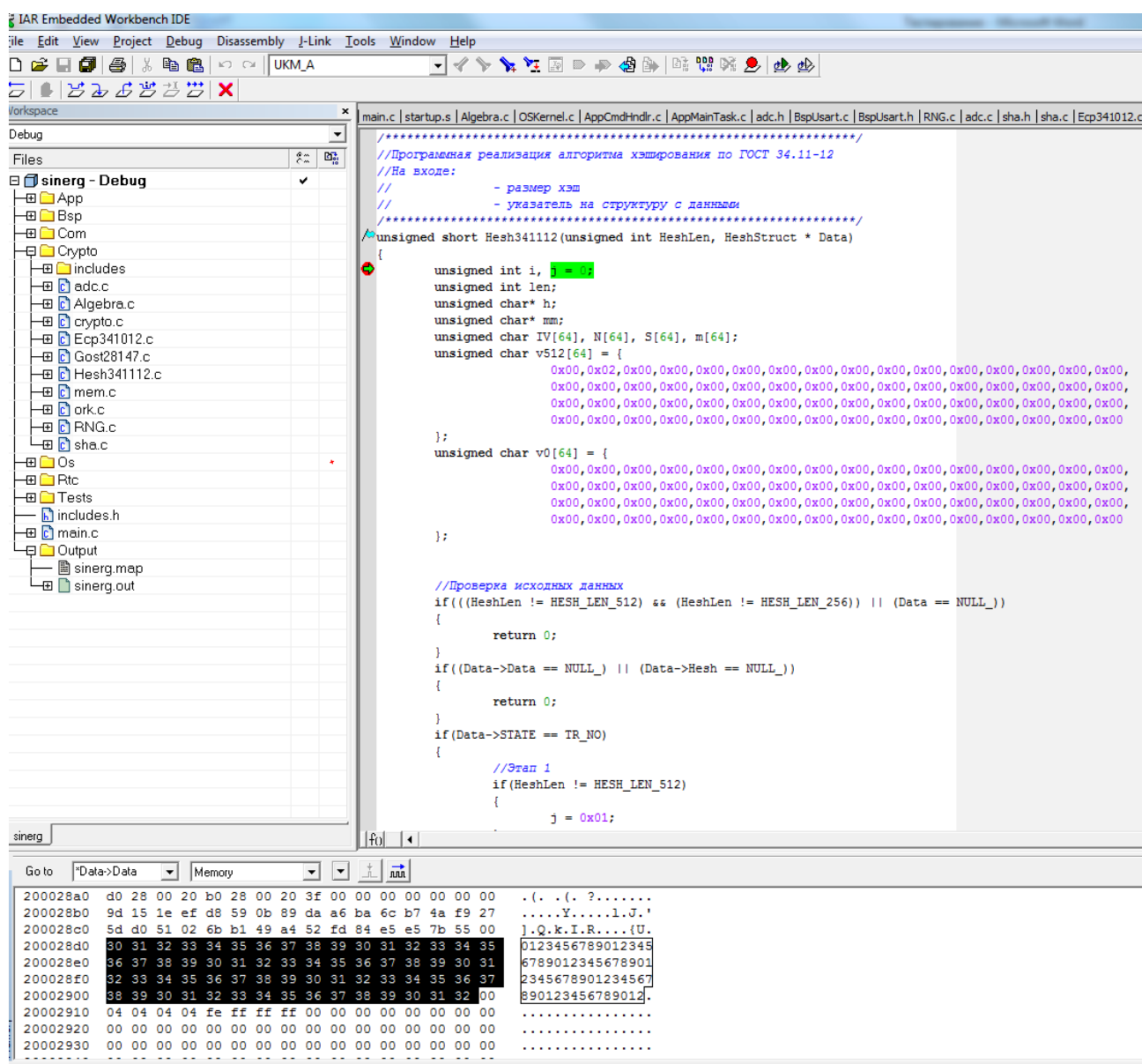


Рисунок 5.21 – Входные данные функции Hesh341112 () с длиной хэш-кода 512 бит

Ожидаемый результат: 1b 54 d0 1a 4a f5 b9 d5 cс 3d 86 d6 8d 28
 54 62 b1 9a bc 24 75 22 2f 35 c0 85 12 2b e4 ba 1f fa 00 ad 30 f8
 76 7b 3a 82 38 4c 65 74 f0 24 c3 11 e2 a4 81 33 2b 08 ef 7f 41 79
 78 91 c1 64 6f 48. Фактический результат функции Hesh341112 () с длиной
 хэш-кода 512 бит представлен на рисунке 5.22.

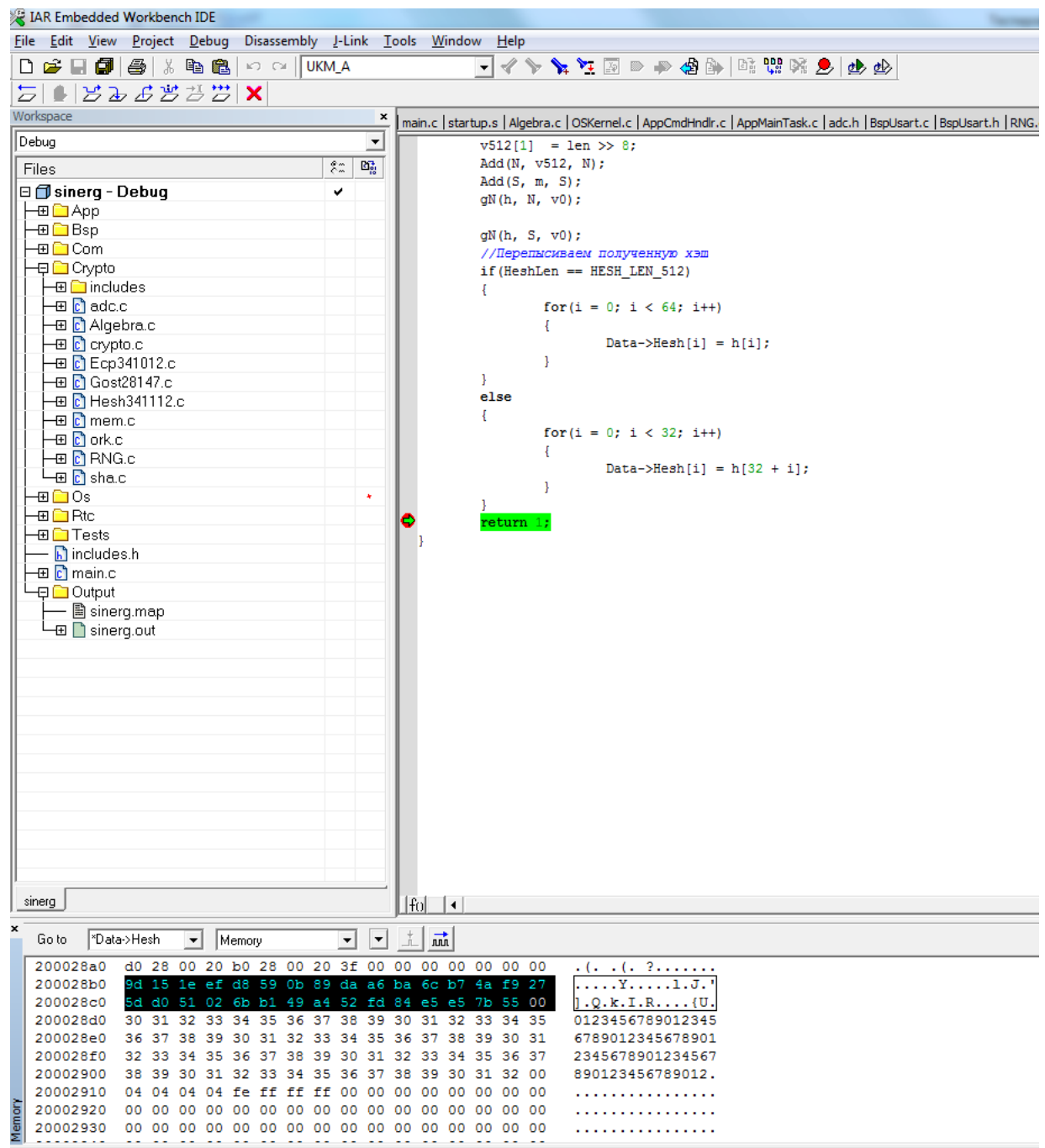


Рисунок 5.22 – Фактический результат функции Hesh341112 () с длиной хэш-кода 512 бит

Фактический результат, представленный на рисунке 5.22, совпадает с ожидаемым результатом, взятом из контрольных примеров для функции Hesh341112 () с длиной хэш-кода 512 бит.

Таким образом, можно сделать вывод, что функция Hesh341112 () реализована корректно.

5.3 Интеграционное тестирование функции orkB ()

Было произведено интеграционное тестирование функции orkB (). Входными данными функции функции orkB () являются последовательности:

- Секретный долговременный ключ S_A f0 65 ef 89 c3 79 54 e6 ef 1c ba 3e e5 9c 41 cf 3c 76 f3 69 42 e6 25 d2 cc ba e1 b5 8f cb d0 cf;
- Секретный ключ k_A d2 b2 68 62 fe ef f0 bd 84 96 42 a6 65 fb 2f ca bf f1 74 df 9b 82 0f b1 74 6f 27 33 f0 54 5c e1;
- Открытый ключ K_B : a1 44 69 c8 c9 80 49 20 b8 88 25 f6 2c 97 33 8b 98 36 73 2c 4e 3c a2 de 54 2d 5f 8e a3 e7 5b 4a 4f 9c bc 0f 29 60 19 3e 79 5c 5a 70 7c 91 5d 3a 55 a1 25 e9 a6 35 a6 85 7e 2a 32 94 1e 82 10 02;
- Значение UKM_A : 7c ec 6a f7 9e c4 ed 81;
- Случайное число UKM_B : 3d 20 5c 6c 2b 76 3f ab;
- Сертификат $Cert_A$: 30 82 03 68 30 82 03 15 a0 03 02 01 02 02 11 00 c1 0a 67 5d 46 50 b9 80 e7 11 c6 fa d5 ec 3f 25 30 0a 06 08 2a 85 03 07 01 01 03 02 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 30 1e 17 0d 31 38 30 31 31 36 31 33 35 34 32 31 5a 17 0d 31 39 30 34 31 36 31 34 30 34 32 31 5a 30 14 31 12 30 10 06 03 55 04 03 0c 09 52 65 73 70 6f 6e 64 65 72 30 66 30 1f 06 08 2a 85 03 07 01 01 01 01 30 13 06 07 2a 85 03 02 02 24 00 06 08 2a 85 03 07 01 01 02 02 03 43 00 04 40 7d 2b 4f 76 f1 e0 5f d5 06 67 10 db 3e cf de bd b8 d9 1b

8d 31 80 49 49 1c 1e 30 5e b9 ca 51 43 96 6f 2a 35 98 c0 5d 26 5e
2e 69 2e af 05 82 73 56 62 d2 07 04 f1 19 3c 46 0e f6 96 5a d0 9f
c4 a3 82 02 02 30 82 01 fe 300e 06 03 55 1d 0f 01 01 ff 04 04 03 02
03 a8 30 1d 06 03 55 1d 0e 04 16 04 14 ca 71 f4 72 2f 96 e8 21 35
0a 02 1c 74 60 47 02 f7 78 a8 36 30 1f 06 09 2b 06 01 04 01 82 37
15 07 04 12 30 10 06 08 2a 85 03 02 02 2e 00 08 02 01 01 02 01 00
30 81 83 06 03 55 1d 23 04 7c 30 7a 80 14 5e 94 aa dd 0f 73 aa 75
ba 76 62 e2 80 b2 de 3d 49 06 77 3b a1 4f a4 4d 30 4b 31 18 30 16
06 05 2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30
31 1a 30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35
36 37 38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52
6f 6f 74 43 41 82 11 00 c1 0a 67 5d 46 50 b8 80 e7 11 41 df 43 c4
9f 41 30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01 05 05 07 03
02 06 08 2b 06 01 05 05 07 03 04 30 27 06 09 2b 06 01 04 01 82 37
15 0a 04 1a 30 18 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0a 06 08
2b 06 01 05 05 07 03 04 30 52 06 03 55 1d 1f 04 4b 30 49 30 47 a0
45 a0 43 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 2f
63 64 70 2f 35 65 39 34 61 61 64 64 30 66 37 33 61 61 37 35 62 61
37 36 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30 36 37 37 33 62
2e 63 72 6c 30 5d 06 08 2b 06 01 05 05 07 01 01 04 51 30 4f 30 4d
06 08 2b 06 01 05 05 07 30 02 86 41 68 74 74 70 3a 2f 2f 6c 6f 63
61 6c 68 6f 73 74 2f 61 69 61 2f 35 65 39 34 61 61 64 64 30 66 37
33 61 61 37 35 62 61 37 36 36 32 65 32 38 30 62 32 64 65 33 64 34
39 30 36 37 37 33 62 2e 63 72 74 30 2b 06 03 55 1d 10 04 24 30 22
80 0f 32 30 31 38 30 31 31 36 31 33 35 34 32 30 5a 81 0f 32 30 31
39 30 31 31 36 31 33 35 34 32 30 5a 30 0a 06 08 2a 85 03 07 01 01
03 02 03 41 00 7a 08 bc d0 00 86 83 01 37 80 1a 79 2a 5f 15 94 43
a7 1b 54 0b 8e 09 b3 35 61 cf e5 17 f6 8c 70 ce f9 9d a3 6d fd bd
f6 ca ef 3d 00 80 39 0b 3c 37 c5 53 19 35 a2 03 81 72 c2 a9 f7 d2
2c c3 4b;

— Сертификат Cert_B: 30 82 03 65 30 82 03 12 a0 03 02 01 02 02
11 00 c1 0a 67 5d 46 50 b9 80 e7 11 c5 fa 4a 56 03 eb 30 0a 06 08
2a 85 03 07 01 01 03 02 30 4b 31 18 30 16 06 05 2a 85 03 64 01 12
0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a 30 18 06 08 2a 85
03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37 38 39 30 30 31 13

30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f 74 43 41 30 1e 17
0d 31 38 30 31 31 36 31 33 35 32 34 33 5a 17 0d 31 39 30 34 31 36
31 34 30 32 34 33 5a 30 11 31 0f 30 0d 06 03 55 04 03 0c 06 53 65
6e 64 65 72 30 66 30 1f 06 08 2a 85 03 07 01 01 01 01 30 13 06 07
2a 85 03 02 02 24 00 06 08 2a 85 03 07 01 01 02 02 03 43 00 04 40
a2 6b 68 59 d7 39 34 b1 ed 8e 1b 24 cc a6 d7 e8 3a 88 e4 de 7b 04
ba 66 8d 81 c7 f2 f5 18 b4 af 75 b1 b2 b7 db 03 7b d1 71 74 5c cf
4b 25 55 29 79 2f 03 d4 4f e1 29 3d 93 78 be ae e0 a9 79 4d a3 82
02 02 30 82 01 fe 30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 03 a8
30 1d 06 03 55 1d 0e 04 16 04 14 c4 02 97 8f 16 e2 3f 49 43 d6 6b
e6 e7 af aa 38 fa ba 24 f4 30 1f 06 09 2b 06 01 04 01 82 37 15 07
04 12 30 10 06 08 2a 85 03 02 02 2e 00 08 02 01 01 02 01 00 30 81
83 06 03 55 1d 23 04 7c 30 7a 80 14 5e 94 aa dd 0f 73 aa 75 ba 76
62 e2 80 b2 de 3d 49 06 77 3b a1 4f a4 4d 30 4b 31 18 30 16 06 05
2a 85 03 64 01 12 0d 30 31 32 33 34 35 36 37 38 39 30 30 30 31 1a
30 18 06 08 2a 85 03 03 81 03 01 01 12 0c 30 31 32 33 34 35 36 37
38 39 30 30 31 13 30 11 06 03 55 04 03 0c 0a 74 65 73 74 52 6f 6f
74 43 41 82 11 00 c1 0a 67 5d 46 50 b8 80 e7 11 41 df 43 c4 9f 41
30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01 05 05 07 03 02 06
08 2b 06 01 05 05 07 03 04 30 27 06 09 2b 06 01 04 01 82 37 15 0a
04 1a 30 18 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0a 06 08 2b 06
01 05 05 07 03 04 30 52 06 03 55 1d 1f 04 4b 30 49 30 47 a0 45 a0
43 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 2f 63 64
70 2f 35 65 39 34 61 61 64 64 30 66 37 33 61 61 37 35 62 61 37 36
36 32 65 32 38 30 62 32 64 65 33 64 34 39 30 36 37 37 33 62 2e 63
72 6c 30 5d 06 08 2b 06 01 05 05 07 01 01 04 51 30 4f 30 4d 06 08
2b 06 01 05 05 07 30 02 86 41 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c
68 6f 73 74 2f 61 69 61 2f 35 65 39 34 61 61 64 64 30 66 37 33 61
61 37 35 62 61 37 36 36 32 65 32 38 30 62 32 64 65 33 64 34 39 30
36 37 37 33 62 2e 63 72 74 30 2b 06 03 55 1d 10 04 24 30 22 80 0f
32 30 31 38 30 31 31 36 31 33 35 32 34 32 5a 81 0f 32 30 31 39 30
31 31 36 31 33 35 32 34 32 5a 30 0a 06 08 2a 85 03 07 01 01 03 02
03 41 00 d9 0c 3c 27 93 c7 08 99 9e ef e3 a1 d7 01 49 13 29 38 99
74 9a 70 a6 5c e4 c7 b2 4c 54 94 32 70 fa 1f 55 26 1a 9c 2a c4

fe fb cf 44 07 b5 75 e1 25 a4 c0 5b 1f 37 73 72 0f 20 49 73 90 fb
ad cb.

Входные данные функции orkB () представлены на рисунке 5.23.

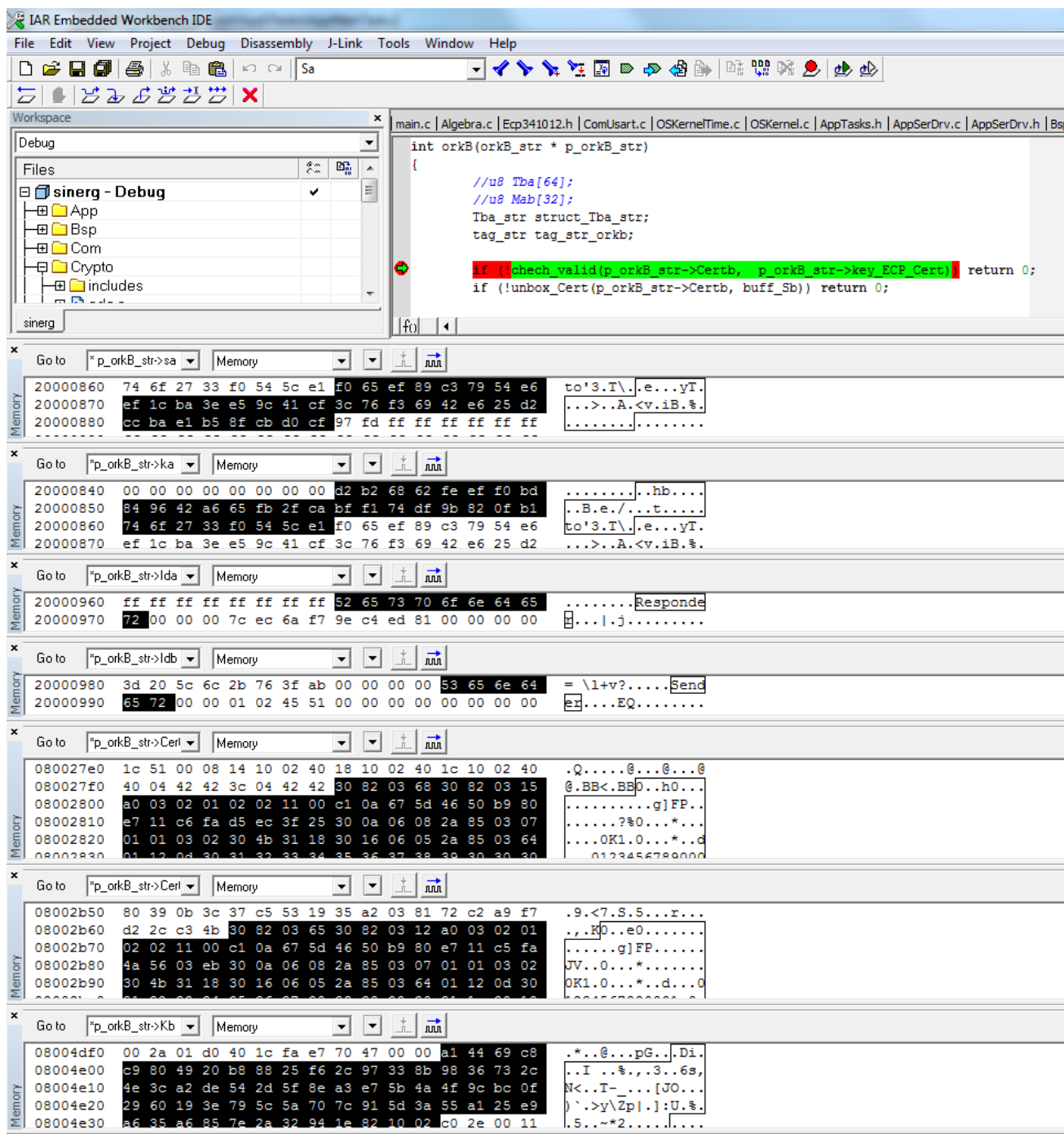


Рисунок 5.23 – Входные данные функции orkB ()

Ожидаемый результат параметров:

– Метка подтверждения ключа tag_A: 4f 6c a7 5e 59 e7 7d 2c 46 4b
32 58 46 96 56 59 9e 41 62 c4 5d 86 12 ae e0 72 a1 89 7a b4 77 22;

- Метка подтверждения ключа tag_B: 3d 68 0a 7f 9c 9f 7b a7 ae f3 34 e9 ae 51 8e 17 97 31 41 65 74 e3 ab f9 1f 92 d8 70 dd 64 aa 55;
- Сеансовый ключ K_{AB}: 6c 03 60 d9 0c f2 cb be b7 cf be 1a 4f 13 fe 68 d5 d5 c6 78 e3 e8 17 d4 3c 76 e7 56 11 30 6f aa.

Фактический результат функции orkB () представлен на рисунке 5.24.

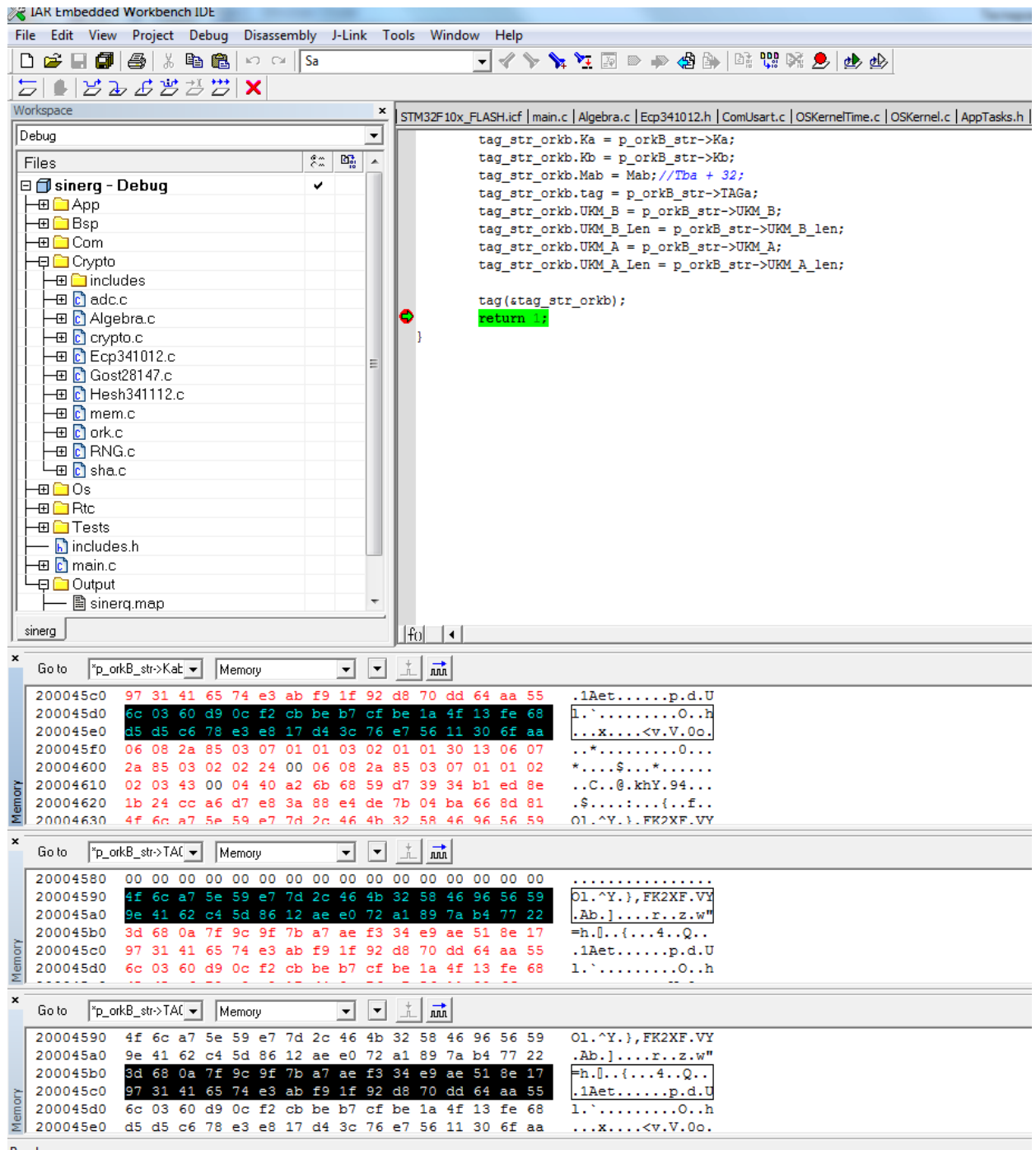


Рисунок 5.24 – Фактический результат функции orkB ()

Фактический результат, представленный на рисунке 5.24, совпадает с ожидаемым результатом, взятом из тестовых значений для функции `orkB ()`.

Таким образом, можно сделать вывод, что функция `orkB ()` реализована корректно.

6 Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

6.1 Анализ условий труда специалиста по защите информации, влияющих на сердечно-сосудистую систему специалиста по защите информации

6.2.1 Электромагнитные излучения

В разделе 6 рассмотрены следующих вопросы:

- анализ условий труда специалиста по защите информации, влияющих на сердечно-сосудистую систему специалиста по защите информации;
- определение методов и средств физической культуры, снижающие воздействие факторов, влияющих на сердечно-сосудистую систему специалиста по защите информации.

На рабочем месте специалиста по защите информации необходимо присутствие различных мер защиты от опасных и вредных факторов промышленности. Для того, чтобы влияние опасных и вредных факторов на специалиста по защите информации было устранено совсем, либо находилось в допустимых пределах необходимо руководствоваться правовыми, техническими и санитарно–техническими нормами.

Согласно статье 209 Трудового Кодекса Российской Федерации, рабочее место - место, где работник должен находиться или куда ему необходимо прибыть в связи с его работой и которое прямо или косвенно находится под контролем работодателя [5].

К тому же, это еще и часть пространства, в которой работник проводит большую часть своего трудового процесса.

Рабочее место, организованное надлежащим образом, дает высокую производительность труда с наименьшим психическим давлением и

физическими затратами.

Расположение рабочего места и относительное положение всех его компонентов обязаны соответствовать различным требованиям, а именно антропометрическим, физическим и психологическим. Крайне важен также характер работы. В частности, при организации рабочего места специалиста по защите информации должны быть соблюдены следующие условия:

- рациональное размещение оборудования, которое является частью рабочего места;
- отсутствие ограниченности рабочего пространства, позволяющее выполнять все необходимые движения;
- необходимо естественное и искусственное освещение;
- допустимые значения для акустического шума.

Главными элементами рабочего места являются компьютерные кресло и стол, так как в положении сидя специалист по защите информации проводит большую часть рабочего времени.

Положение работы в положении сидя вызывает минимальную усталость специалиста по защите информации. Рациональная компоновка рабочего места и фиксированное размещение предметов и документации обеспечивает четкое выполнение поставленных задач. Комфортные условия труда и надлежащая организация рабочих мест необходимы для облегчения работы сотрудников, что положительно влияет на производительность. Цвет стен и мебель должны помочь создать идеальную атмосферу для зрительного восприятия, хорошего настроения. В офисной комнате, где производится монотонная работа, происходит нервное напряжение и постоянная концентрация внимания, поэтому цвет должен быть оранжевого оттенка, так как именно этот оттенок является наиболее подходящим для пространств.

Во время использования персонального компьютера, происходит воздействие на живые организмы, поэтому для защиты от излучений должны использоваться мониторы с высоким разрешением и частотой обновления экрана не ниже 60 Гц.

При функционировании монитора электризуется не только его экран, но и воздух в помещении. Так как электростатическое притягивает частицы пыли, то при циркуляции кислорода в легкие человека попадают микроскопические частицы пыли, что ведет к неправильной работе органов дыхания, что несомненно приведёт к развитию сердечно – сосудистых заболеваний.

Для минимизации вредных факторов применяется следующие защитные меры:

- жидкокристаллические мониторы;
- выключение или перевод в спящий режим компьютера при неиспользовании;
- ионизаторы воздуха;
- регулярное проветривание комнаты.

6.2.2 Освещенность

Искусственное освещение в помещениях при эксплуатации мониторов и ПЭВМ должно осуществляться системой общего равномерного освещения. В производственных и административно – общественных помещениях, в случаях преимущественной работы с документами, допускается применение комбинированного освещения.

Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 лк. Допускается установка светильников местного освещения для подсветки документов. Местное освещение не должно создавать бликов поверхности экрана и увеличивать освещенность экрана более 300 лк.

Следует ограничивать прямую блескость от источников освещения, при этом яркость светящихся поверхностей (окна, светильники и др.), находящихся в поле зрения, доля быть не более 200 кд на квадратный метр.

Следует ограничивать отраженную блескость на рабочих поверхностях (экран, стол, клавиатура и др.) за счет правильного выбора типов светильников и расположения рабочих мест по отношению к источникам естественного и

искусственного освещения.

Следует ограничивать неравномерность распределения яркости в поле зрения пользователя монитора и ПЭВМ, при этом соотношение яркости между рабочими поверхностями не должно превышать от 3 к 1 до 5 к 1, а между рабочими поверхностями и поверхностями стен и оборудования – 10 к 1 [6].

В качестве источника света искусственные люминесцентные лампы должны быть в основном люминесцентными лампами ЛВ. При отраженном освещении допускается использование металлогалогенных ламп мощностью до 250 Вт. Допускается использование ламп накаливания в осветительных приборах.

Общее освещение должно выполняться в непрерывной линии светильников, расположенной сбоку рабочей станции, параллельно линии зрения пользователя.

Чтобы обеспечить нормализованные значения освещенности при использовании мониторов и ПК, необходимо чистить окна и осветительные приборы не реже двух раз в год и заменять перегоревшие лампы (диоды) в нужное время.

6.2.3 Шум

Ключевой особенностью шума является то, что с временем воздействия шума увеличивается и негативное влияние на нервную систему, что приводит к учащению пульса и увеличению кровяного давления.

Шум оказывает вредное влияние на нервную систему и снижает производительность труда. В частности, снижается скорость и точность сенсомоторных процессов, увеличивается число ошибок при решении интеллектуальных задач.

При увеличении уровня шума до 80 дБ и более шум оказывает серьезное физиологическое воздействие на организм — может возникнуть гипертония, язвенная болезнь, болезни сердца, неврозы, желудочно–кишечные и кожные заболевания. Шум уровня от 90 до 100дБ приводит к общему утомлению и

глухоте, притупляется острота зрения, появляются головные боли, повышается кровяное и внутричерепное давление, изменяется объем внутренних органов и т.д. Пребывание работающих в зонах с уровнем звукового давления более 135 дВ в любой октавной полосе запрещается.

Таким образом, в рабочих помещениях специалистов по защите информации допустимый уровень шума не должен превышать 60 дБ. Для защиты сотрудников от шума применяются следующие защитные меры:

- размещение рабочих мест и оборудования в зависимости от источников шума;
- проектирование шумоизоляционных пространств.

6.2.4 Микроклимат

Микроклиматические параметры производственной среды - сочетание температуры, относительной влажности и скорости воздуха. Микроклиматические параметры сильно влияют на функциональную активность, здоровье, и надежность работы компьютерной техники.

Большое влияние на микроклимат в помещении имеет собственный источник тепла - это осветительные приборы, обслуживающий персонал и даже солнечный свет. Кроме того, наибольшее количество тепловыделения из помещения оснащено компьютером, который обеспечивает в среднем 80% общего тепловыделения.

Что касается сердечно-сосудистой системы, а также работы оборудования, то относительная влажность воздуха оказывает большое влияние. При влажности воздуха до 40% магнитные ленты становятся хрупкими, износ магнитной головки ухудшается, проводная изоляция отключается, возникает статическое электричество, когда носитель информации перемещается на компьютер.

Для поддержания оптимального микроклимата предусмотрены организационные меры, профилактические и архитектурно-планировочные решения:

- системы кондиционирования воздуха;

- отопление помещения;
- правила рабочего времени (рабочие перерывы, сокращения рабочего дня, период отпуска, сокращение продолжительности обслуживания и т. д.);
- исключается местонахождение оборудования, нуждающегося в постоянном охлаждении, вблизи района, где находится источник тепла;
- надежная теплоизоляция от напольного покрытия солнечной радиацией;
- рабочее место специалиста по защите информации должно быть защищено от негативных источников тепла.

6.2.5 Электробезопасность

Компьютер представляет собой электрический прибор с трехфазным четырехпроводным напряжением питания 220 или 380В с заземленной нейтралью.

В случае поражения электрическим током может возникнуть проблема с сердечно-сосудистой системой. Во избежание проблемы, а также возникновения пожара и повреждения компьютера необходимо руководствоваться следующими мерами безопасности:

- не включать компьютер при отсутствии части корпуса ПК;
- не использовать компьютер с поврежденным шнуром питания;
- не использовать компьютер в помещении с высокой влажностью или загрязненным воздухом;
- во время работы необходимо принять меры для предотвращения удара падения компьютера;
- не оставлять компьютер без присмотра;
- не допускать проникновение или просочение объектов, жидкостей и твердых тел;
- не допускать перегибы и натяжение питающих кабелей;
- не устанавливать ПК рядом с источником тепла;

- не блокировать вентиляционные отверстия.

Для обеспечения безопасности электричества во время установки, ввода в эксплуатацию и эксплуатации сети необходимо уделять особое внимание созданию защитных мер от пользователя и персонала для предотвращения возможных проблем при работе с сетью.

6.2.6 Эргономика

Согласно исследованиям, 60% людей, которые постоянно находятся в контакте с аппаратурой ПЭВМ, страдают сердечно-сосудистыми заболеваниями. Эффект на сердце и кровеносные сосуды являются неотъемлемой частью взаимодействия пользователя с компьютером и прогрессирует со временем. Все начинается с неправильного положения тела и постоянного пребывания в сидячем положении. Это нарушает кровообращение, что ведет к слипанию эритроцитов. Сердечная мышца в этом состоянии испытывает кислородный голод. Снижение кровотока может привести к увеличению тромбоза, особенно у людей, которые имеют риск развития заболеваний сердечно-сосудистой системы.

Кроме того, с неправильным положением тела, страдает позвоночник, появляется остеохондроз. При хроническом сдавливании нервных отростков в грудном отделе позвоночника возникают боли в сердце из-за невралгий, что может вызвать головную боль и цереброваскулярные расстройства. Легкие и диафрагма при изогнутом положении тела не позволяют сердцу полнофункционально циркулировать кровь.

Электромагнитное излучение влияет на клетки крови, клетки. Они имеют отрицательные и положительные заряды, электрические потенциалы. Излучение может привести к разрушению или слипания в один комок эритроцитов, тромбоцитов, нарушения проходимости мембран.

Радиация может влиять на спинной мозг, селезенку, вызывать гематопозную депрессию. Гормональная система также страдает, производя большое количество адреналиновой дозы в ответ на хронический стресс из-за

вредного воздействия. Электромагнитные эффекты могут влиять на проводимость миокарда и вызывать аритмии.

Неутешительный факт заставил специалистов по информационной безопасности ограничить время, затрачиваемое на компьютеры, и узнать, как соблюдать правила его безопасного использования.

При творческой работе, требующей большого психического напряжения или высокой концентрации внимания, рекомендуется изолировать рабочие места друг от друга с ограничениями перегороджающими конструкциями с высотой от 1,5 до 2,0 м.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на поверхности используемого оборудования с учетом количества и особенностей конструкции, вида выполненной работы. В тоже время разрешено использовать различные настольные компьютеры (моноблоки), отвечающие современным требованиям эргономики. Поверхность рабочего стола должна иметь коэффициент отражения от 0,5 до 0,7.

Крайне желательно, чтобы высота рабочей поверхности стола могла бы быть отрегулирована в пределах от 680 до 800 мм, и, если такой возможности нет, она должна быть равна 725 мм. Следует учитывать модульные размеры рабочей поверхности рабочего стола компьютера на основе расчетных размеров: ширина 800, 1000, 1200, 1400 мм; глубина 800 и 1000 мм.

Конструкция рабочего сиденья должна обеспечивать следующие требования:

- ширина и глубина поверхности сиденья не менее 400 мм;
- поверхность сиденья с передними краями, имеющие закругления;
- регулировка высоты сиденья в диапазоне от 400 до 550 мм и угол переднего хода до 15 градусов и обратно до 5 градусов;
- высота спинки сиденья составляет 300 плюс минус 20 мм, ширина не менее 380 мм и радиус кривизны горизонтальной плоскости составляет 400 мм;
- задний угол в вертикальной плоскости 30 градусов;

- регулировка задней дистанции от переднего края сиденья в диапазоне от 260 до 400 мм;
- фиксированные или съемные ободья длиной не менее 250 мм и шириной от 50 до 70 мм;
- регулировка подлокотников по высоте диапазоне 230 плюс минус 30 мм и внутреннее расстояние между плечами рук в пределах от 350 до 500 мм.

Рабочее место пользователя ПК должно быть оборудовано подставкой для ног с минимальной шириной 300 мм, глубиной не менее 400 мм, путем регулировки высоты до 150 мм и угла наклона опорной поверхности стула до 20 градусов. Поверхность подставки должна быть наклонена и имеет передний край 10 мм.

Клавиатуру следует размещать на поверхности стола на расстоянии от 100 до 300 мм от края, обращенного к пользователю, или на высоко регулируемой рабочей поверхности, отделенной от основного стола.

Дизайн рабочего кресла (стула) должен обеспечивать рациональное рабочее положение при работе на ПК, позволяя вам изменить свою осанку, чтобы уменьшить статическое напряжение мышц шеи и спины, чтобы предотвратить развитие усталости. Тип сиденья (стула) следует выбирать с учетом роста пользователей, характера работы и продолжительности работы с ПК.

Стул сиденья должен быть поднят, отрегулирован по высоте и углу наклона сиденья и спинки. Регулировка должна иметь надежную фиксацию.

Поверхность сиденья, спинки и других элементов стула (кресла) должна быть мягкой, с нескользящим покрытием, обеспечивающим легкую очистку грязи.

Экран монитора должен быть от глаза пользователя на расстоянии от 600 до 700 мм, но не ближе 500 мм, с учетом размера шрифта и буквенно-цифровых символов. Угол между экраном и специалистом по защите информации должен не превышать 60° по горизонтальной линии, как показано на рисунке 6.1.

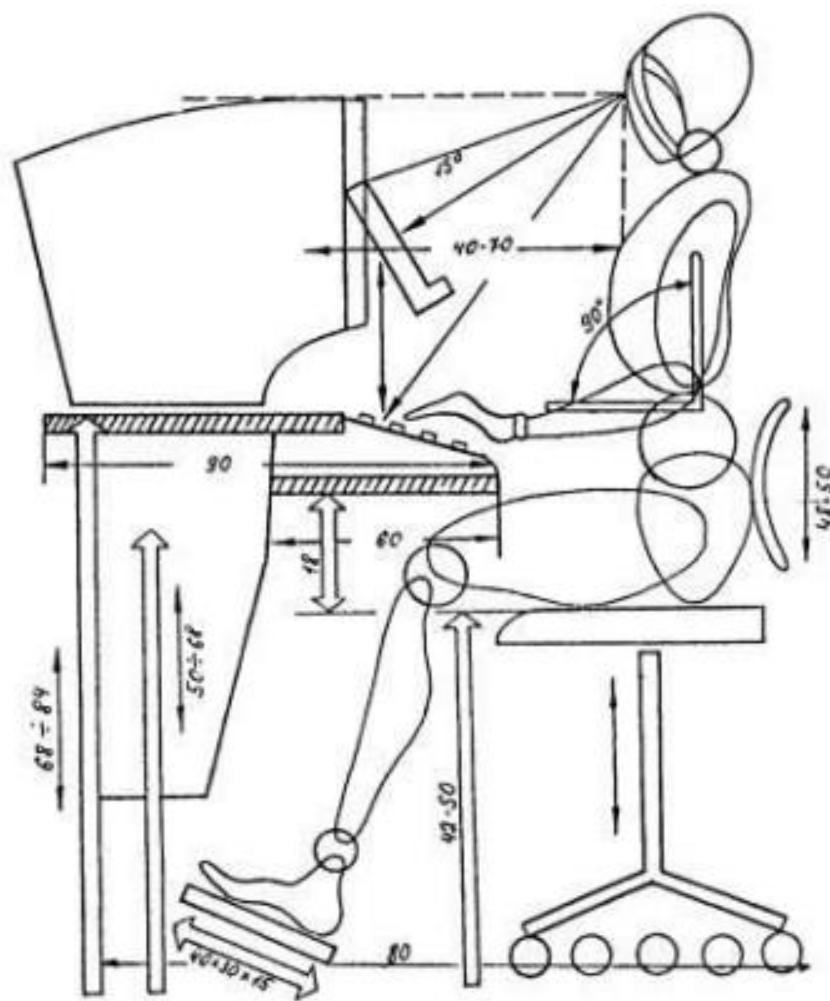


Рисунок 6.1 - Расположение дисплея на столе

Чтобы предотвратить преждевременную усталость пользователей ПК рекомендуется организовать рабочие смены с прерываниями на отдых и физическую нагрузку, особенно если работа происходит в положении сидя.

Если специалист по защите информации испытывает дискомфорт, несмотря на принятые меры, то рекомендуется применять индивидуальный подход с ограниченным временем работы с ПК.

В случаях, когда характер работы, требующий постоянного взаимодействия с видеодисплеем при напряжении внимания и фокусировки, не имеет учета возможности периодического переключения на другие виды работы, то рекомендуется, чтобы деятельность прерывалась в течение от 10 до 15 минут, каждые от 45 до 60 минут рабочего времени.

Продолжительность непрерывной работы с видеодисплеями не должна превышать 1 час [9].

6.2 Определение методов и средств физической культуры, снижающие воздействие факторов, влияющих на сердечно-сосудистую систему специалиста по защите информации

При определении методов и средств физической культуры, снижающие воздействие факторов, влияющих на сердечно-сосудистую систему специалиста по защите информации выбор тренировочной системы пал на программу круговой системы тренировок (далее – Программа). Программа основана на системе общей физической подготовки, которая состоит из чередующихся упражнений с собственным весом тела. Цель Программы направлена на физическое развитие специалиста по защите информации по таким направлениям как выносливость, сила, гибкость, скорость, координация, точность, а также улучшение работоспособности сердечно - сосудистой и дыхательной систем, быстрота адаптации к смене нагрузок и др. Программа подходит для людей различного уровня подготовки.

По подобной методике тренируются многие пожарные службы, силовые структуры, органы правопорядка. Однако не стоит предполагать, что Программа рассчитана лишь на людей с высоким уровнем подготовки: она рекомендуется всем, кто хочет быть в хорошей физической форме. Так как, для тренировок совершенно не обязательно ходить в зал, большую часть комплексов можно выполнять на улице или даже в помещении, что играет важную роль для деятельности специалиста по защите информации.

Так как первостепенной задачей было создать методику тренировок, которую специалист по защите информации может выполнять, не отходя от рабочего места, то выбор был сделан на несколько разноплановых упражнений с собственным весом тела. Упражнения выполняются циклично (по кругу). Количестве циклов – 2, однако при желании их можно увеличить. При возникновении болезненных ощущений выполнение упражнений следует прекратить. Таким образом, программа тренировок выглядит следующим

образом:

- наклоны в сторону (от 20 до 30 повторений);
- вращение рук (от 20 до 25 повторений);
- подъем на носки стоя (от 15 до 20 повторений);
- приседания (от 15 до 20 повторений);
- вращение шей (от 10 до 15 повторений).

Не возбраняется выполнить отжимания с широкой расстановкой рук в количестве от 15 до 20 повторений.

Для закрепления результата и прогрессии тренировок, программа должна выполняться в соответствие со следующими рекомендациями:

- тренировка должна выполняться ежедневно по 3 раза в день;
- упражнения должны выполняться без пауз, с разной степенью интенсивности.

6.3 Анализ раздела

В разделах 6.1 и 6.2 были изложены рекомендации специалисту по защите информации, позволяющие обеспечивать комфортную работу. Соблюдение условий, определяющих оптимальную организацию рабочего места специалиста по защите информации, позволит сохранить работоспособность в течение всего рабочего дня и повысить производительность труда специалиста по защите информации.

Заключение

В ходе работы были рассмотрены алгоритмы выработки общего ключа с аутентификацией на основе открытого ключа, структура программного обеспечения специализированного вычислителя. Были проанализированы процессы выработки общего секретного ключа с аутентификацией на основе открытого ключа, разработан общий алгоритм работы программного модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа, разработаны алгоритмы функций модуля функций для программной реализации модуля, выполнена программная реализация модуля выработки общего секретного ключа с аутентификацией на основе открытого ключа, выбраны инструменты для программной реализации и отладки, был разработан и протестирован программный модуль выработки общего ключа с аутентификацией на основе открытого ключа.

Изучены вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

Таким образом, все пункты технического задания на ВКР выполнены в полном объёме.

Список используемых источников

- 1 Центр защиты информации и специальной связи ФСБ России ГОСТ Р 34.11 — 2012 Информационная технология. Криптографическая защита информации. Функция хэширования — М.: Федеральное агентство по техническому регулированию и метрологии, 2012. — 38 с.
- 2 Технический комитет 026 Гребнев С.В. Схемы выработки общего ключа с аутентификацией на основе открытого ключа. — М.: Федеральное агентство по техническому регулированию и метрологии, 2016. — 17 с.
- 3 ТК 26 РСТ Р 50.1.113 — 2016 Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования — М.: Федеральное агентство по техническому регулированию и метрологии, 2016 — 28 с.
- 4 Справочник по электронным компонентам [Электронный ресурс]: Обзор семейства STM32 — Режим доступа: http://www.gaw.ru/html.cgi/txt/doc/micros/arm/cortex_arh/1_2.htm, свободный. — Загл. с экрана.
- 5 ТК РФ 2018 с Комментариями [Электронный ресурс]: Статья 209 ТК РФ. Основные понятия — Режим доступа: <http://www.tkodeksrf.ru/ch-3/rzd-10/gli-33/st-209-tk-rf>, свободный. — Загл. с экрана.
- 6 Российская газета [Электронный ресурс]: Постановление Главного государственного санитарного врача Российской Федерации от 13 июня 2003 г. N 118 г. Москва о введении в действие санитарно-эпидемиологических правил и нормативов санпин 2.2.2/2.4.1340-03 — Режим доступа: <https://rg.ru/2003/06/21/134.html>, свободный. — Загл. с экрана.
- 7 Википедия [Электронный ресурс]: Протокол Диффи — Хеллмана — Режим доступа: https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана, свободный. — Загл. с экрана.
- 8 Электронный фонд [Электронный ресурс]: ГОСТ Р 34.11-2012.

Информационная технология. Криптографическая защита информации. Функция хэширования — Режим доступа: <http://docs.cntd.ru/document/gost-r-34-11-2012>, свободный. — Загл. с экрана.

9 РефератКо [Электронный ресурс]: Анализ условий труда на рабочем месте инженера, при разработке проекта комплексной защиты информации — Режим доступа: <https://referat.co/ref/623261/read>, свободный. — Загл. с экрана.

ПРИЛОЖЕНИЕ А
(обязательное)
ИСХОДНЫЙ КОД ПРОГРАММЫ

```

1.cpp
#include <stdio.h>
#include "ork.h"
#include <conio.h>
#include "Esp341012.h"

//анализ сертификата
u8 certificate[1024]; //сертификат
u8 otkr_key[64]; //открытый ключ
u16 size_otkr; //размер ключа
u8 ID[100];

u8 oopen_key[64];
u8 ssec_key[32] = {0xd2, 0xb2, 0x68, 0x62, 0xfe, 0xef, 0xf0, 0xbd, 0x84, 0x96,
0x42, 0xa6, 0x65, 0xfb, 0x2f, 0xca,
0xbf, 0xf1, 0x74, 0xdf, 0x9b, 0x82, 0x0f, 0xb1, 0x74, 0x6f, 0x27, 0x33, 0xf0,
0x54, 0x5c, 0xe1};

u8 mass_tag[32];
tag_str tAstr;
u8 mass_Mab[32] = {0xe3, 0x85, 0x97, 0x66, 0x2e, 0xe1, 0x03, 0xde, 0x49, 0x1f,
0x35, 0x04, 0x1d, 0x52, 0xfe, 0x85, 0xcb, 0x14, 0x57, 0x38, 0xcd, 0xad, 0x07,
0xe3, 0xba, 0xc8, 0xfd, 0x6a, 0x3b, 0xcc, 0x58, 0x20};
u8 mass_kb[32] = {0x6f, 0x6b, 0x85, 0xd0, 0x5e, 0x26, 0x38, 0x01, 0x6b, 0x08,
0x2a, 0xf1, 0xe9, 0xa5, 0x0f, 0x31, 0x2c, 0x72, 0x07, 0xb1, 0x6b, 0x75, 0x1f,
0x45, 0x01, 0x15, 0x6f, 0x22, 0x48, 0xcc, 0x4d, 0x9d};
u8 mass_ka[32] = {0xcb, 0x6c, 0x5a, 0xc2, 0x44, 0x90, 0x75, 0x63, 0x95, 0x84,
0xf9, 0x9c, 0x4c, 0xb3, 0x53, 0x98, 0x8e, 0x77, 0x8b, 0xca, 0xfa, 0x1f, 0xfa,
0x36, 0xca, 0x97, 0x7a, 0xc7, 0x68, 0x4e, 0xf3, 0x5f};
u8 mass_IDb[6] = {0x53, 0x65, 0x6e, 0x64, 0x65, 0x72};
u8 mass_IDa[9] = {0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x64, 0x65, 0x72};

u8 k_A[32] = {0xd2, 0xb2, 0x68, 0x62, 0xfe, 0xef, 0xf0, 0xbd, 0x84, 0x96, 0x42,
0xa6, 0x65, 0xfb, 0x2f, 0xca, 0xbf, 0xf1, 0x74, 0xdf, 0x9b, 0x82, 0x0f, 0xb1,
0x74, 0x6f, 0x27, 0x33, 0xf0, 0x54, 0x5c, 0xe1};

u8 K_A[32]; // открытый ключ стороны А

u8 sa[32] = {0xf0, 0x65, 0xef, 0x89, 0xc3, 0x79, 0x54, 0xe6, 0xef, 0x1c, 0xba,
0x3e, 0xe5, 0x9c, 0x41, 0xcf, 0x3c, 0x76, 0xf3, 0x69, 0x42, 0xe6, 0x25, 0xd2,
0xcc, 0xba, 0xe1, 0xb5, 0x8f, 0xcb, 0xd0, 0xcf}; //долговременный ключ стороны А
u8 IDa[9] = {0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x64, 0x65, 0x72};
//идентификатор стороны А
u8 IDb[6] = {0x53, 0x65, 0x6e, 0x64, 0x65, 0x72}; // идентификатор стороны В
u8 TAGa[32];
u8 TAGb[32];
u8 Kab[32]; //общий секретный ключ
u8 UKM_A[9] = {0x7c, 0xec, 0x6a, 0xf7, 0x9e, 0xc4, 0xed, 0x81, 0}; //тестовое
значение UKM_A
u8 UKM_B[9] = {0x3d, 0x20, 0x5c, 0x6c, 0x2b, 0x76, 0x3f, 0xab, 0}; //тестовое
значение UKM_B
u8 Certa[876] =
{0x30, 0x82, 0x03, 0x68, 0x30, 0x82, 0x03, 0x15, 0xa0, 0x03, 0x02, 0x01, 0x02,
0x02, 0x11, 0x00,
0xc1, 0x0a, 0x67, 0x5d, 0x46, 0x50, 0xb9, 0x80, 0xe7, 0x11, 0xc6, 0xfa, 0xd5,
0xec, 0x3f, 0x25,
0x30, 0x0a, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x03, 0x02, 0x30,
0x4b, 0x31, 0x18,
0x30, 0x16, 0x06, 0x05, 0x2a, 0x85, 0x03, 0x64, 0x01, 0x12, 0x0d, 0x30, 0x31,
0x32, 0x33, 0x34,
0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x30, 0x30, 0x31, 0x1a, 0x30, 0x18, 0x06,
0x08, 0x2a, 0x85,

```

0x03, 0x03, 0x81, 0x03, 0x01, 0x01, 0x12, 0x0c, 0x30, 0x31, 0x32, 0x33, 0x34,
0x35, 0x36, 0x37,
0x38, 0x39, 0x30, 0x30, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03,
0x0c, 0x0a, 0x74,
0x65, 0x73, 0x74, 0x52, 0x6f, 0x6f, 0x74, 0x43, 0x41, 0x30, 0x1e, 0x17, 0x0d,
0x31, 0x38, 0x30,
0x31, 0x31, 0x36, 0x31, 0x33, 0x35, 0x34, 0x32, 0x31, 0x5a, 0x17, 0x0d, 0x31,
0x39, 0x30, 0x34,
0x31, 0x36, 0x31, 0x34, 0x30, 0x34, 0x32, 0x31, 0x5a, 0x30, 0x14, 0x31, 0x12,
0x30, 0x10, 0x06,
0x03, 0x55, 0x04, 0x03, 0x0c, 0x09, 0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x64,
0x65, 0x72, 0x30,
0x66, 0x30, 0x1f, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x01, 0x01,
0x30, 0x13, 0x06,
0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00, 0x06, 0x08, 0x2a, 0x85, 0x03,
0x07, 0x01, 0x01,
0x02, 0x02, 0x03, 0x43, 0x00, 0x04, 0x40, 0x7d, 0x2b, 0x4f, 0x76, 0xf1, 0xe0,
0x5f, 0xd5, 0x06,
0x67, 0x10, 0xdb, 0x3e, 0xcf, 0xde, 0xbd, 0xb8, 0xd9, 0x1b, 0x8d, 0x31, 0x80,
0x49, 0x49, 0x1c,
0x1e, 0x30, 0x5e, 0xb9, 0xca, 0x51, 0x43, 0x96, 0x6f, 0x2a, 0x35, 0x98, 0xc0,
0x5d, 0x26, 0x5e,
0x2e, 0x69, 0x2e, 0xaf, 0x05, 0x82, 0x73, 0x56, 0x62, 0xd2, 0x07, 0x04, 0xf1,
0x19, 0x3c, 0x46,
0x0e, 0xf6, 0x96, 0x5a, 0xd0, 0x9f, 0xc4, 0xa3, 0x82, 0x02, 0x02, 0x30, 0x82,
0x01, 0xfe, 0x30,
0x0e, 0x06, 0x03, 0x55, 0x1d, 0x0f, 0x01, 0x01, 0xff, 0x04, 0x04, 0x03, 0x02,
0x03, 0xa8, 0x30,
0x1d, 0x06, 0x03, 0x55, 0x1d, 0x0e, 0x04, 0x16, 0x04, 0x14, 0xca, 0x71, 0xf4,
0x72, 0x2f, 0x96,
0xe8, 0x21, 0x35, 0x0a, 0x02, 0x1c, 0x74, 0x60, 0x47, 0x02, 0xf7, 0x78, 0xa8,
0x36, 0x30, 0x1f,
0x06, 0x09, 0x2b, 0x06, 0x01, 0x04, 0x01, 0x82, 0x37, 0x15, 0x07, 0x04, 0x12,
0x30, 0x10, 0x06,
0x08, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x2e, 0x00, 0x08, 0x02, 0x01, 0x01, 0x02,
0x01, 0x00, 0x30,
0x81, 0x83, 0x06, 0x03, 0x55, 0x1d, 0x23, 0x04, 0x7c, 0x30, 0x7a, 0x80, 0x14,
0x5e, 0x94, 0xaa,
0xdd, 0x0f, 0x73, 0xaa, 0x75, 0xba, 0x76, 0x62, 0xe2, 0x80, 0xb2, 0xde, 0x3d,
0x49, 0x06, 0x77,
0x3b, 0xa1, 0x4f, 0xa4, 0x4d, 0x30, 0x4b, 0x31, 0x18, 0x30, 0x16, 0x06, 0x05,
0x2a, 0x85, 0x03,
0x64, 0x01, 0x12, 0x0d, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38,
0x39, 0x30, 0x30,
0x30, 0x31, 0x1a, 0x30, 0x18, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x03, 0x81, 0x03,
0x01, 0x01, 0x12,
0x0c, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x30,
0x31, 0x13, 0x30,
0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x0c, 0x0a, 0x74, 0x65, 0x73, 0x74, 0x52,
0x6f, 0x6f, 0x74,
0x43, 0x41, 0x82, 0x11, 0x00, 0xc1, 0x0a, 0x67, 0x5d, 0x46, 0x50, 0xb8, 0x80,
0xe7, 0x11, 0x41,
0xdf, 0x43, 0xc4, 0x9f, 0x41, 0x30, 0x1d, 0x06, 0x03, 0x55, 0x1d, 0x25, 0x04,
0x16, 0x30, 0x14,
0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x03, 0x02, 0x06, 0x08, 0x2b,
0x06, 0x01, 0x05,
0x05, 0x07, 0x03, 0x04, 0x30, 0x27, 0x06, 0x09, 0x2b, 0x06, 0x01, 0x04, 0x01,
0x82, 0x37, 0x15,
0x0a, 0x04, 0x1a, 0x30, 0x18, 0x30, 0x0a, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05,
0x05, 0x07, 0x03,
0x02, 0x30, 0x0a, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x03, 0x04,
0x30, 0x52, 0x06,

```

0x03, 0x55, 0x1d, 0x1f, 0x04, 0x4b, 0x30, 0x49, 0x30, 0x47, 0xa0, 0x45, 0xa0,
0x43, 0x86, 0x41,
0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x68,
0x6f, 0x73, 0x74,
0x2f, 0x63, 0x64, 0x70, 0x2f, 0x35, 0x65, 0x39, 0x34, 0x61, 0x61, 0x64, 0x64,
0x30, 0x66, 0x37,
0x33, 0x61, 0x61, 0x37, 0x35, 0x62, 0x61, 0x37, 0x36, 0x36, 0x32, 0x65, 0x32,
0x38, 0x30, 0x62,
0x32, 0x64, 0x65, 0x33, 0x64, 0x34, 0x39, 0x30, 0x36, 0x37, 0x37, 0x33, 0x62,
0x2e, 0x63, 0x72,
0x6c, 0x30, 0x5d, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x01, 0x01,
0x04, 0x51, 0x30,
0x4f, 0x30, 0x4d, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x30, 0x02,
0x86, 0x41, 0x68,
0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x68, 0x6f,
0x73, 0x74, 0x2f,
0x61, 0x69, 0x61, 0x2f, 0x35, 0x65, 0x39, 0x34, 0x61, 0x61, 0x64, 0x64, 0x30,
0x66, 0x37, 0x33,
0x61, 0x61, 0x37, 0x35, 0x62, 0x61, 0x37, 0x36, 0x36, 0x32, 0x65, 0x32, 0x38,
0x30, 0x62, 0x32,
0x64, 0x65, 0x33, 0x64, 0x34, 0x39, 0x30, 0x36, 0x37, 0x37, 0x33, 0x62, 0x2e,
0x63, 0x72, 0x74,
0x30, 0x2b, 0x06, 0x03, 0x55, 0x1d, 0x10, 0x04, 0x24, 0x30, 0x22, 0x80, 0x0f,
0x32, 0x30, 0x31,
0x38, 0x30, 0x31, 0x31, 0x36, 0x31, 0x33, 0x35, 0x34, 0x32, 0x30, 0x5a, 0x81,
0x0f, 0x32, 0x30,
0x31, 0x39, 0x30, 0x31, 0x31, 0x36, 0x31, 0x33, 0x35, 0x34, 0x32, 0x30, 0x5a,
0x30, 0x0a, 0x06,
0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x03, 0x02, 0x03, 0x41, 0x00, 0x7a,
0x08, 0xbc, 0xd0,
0x00, 0x86, 0x83, 0x01, 0x37, 0x80, 0x1a, 0x79, 0x2a, 0x5f, 0x15, 0x94, 0x43,
0xa7, 0x1b, 0x54,
0x0b, 0x8e, 0x09, 0xb3, 0x35, 0x61, 0xcf, 0xe5, 0x17, 0xf6, 0x8c, 0x70, 0xce,
0xf9, 0x9d, 0xa3,
0x6d, 0xfd, 0xbd, 0xf6, 0xca, 0xef, 0x3d, 0x00, 0x80, 0x39, 0x0b, 0x3c, 0x37,
0xc5, 0x53, 0x19,
0x35, 0xa2, 0x03, 0x81, 0x72, 0xc2, 0xa9, 0xf7, 0xd2, 0x2c, 0xc3, 0x4b};
//тестовое значение Certa

```

```

u8 Certb[873] = {
0x30, 0x82, 0x03, 0x65, 0x30, 0x82, 0x03, 0x12, 0xa0, 0x03, 0x02, 0x01, 0x02,
0x02, 0x11, 0x00,
0xc1, 0x0a, 0x67, 0x5d, 0x46, 0x50, 0xb9, 0x80, 0xe7, 0x11, 0xc5, 0xfa, 0x4a,
0x56, 0x03, 0xeb,
0x30, 0x0a, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x03, 0x02, 0x30,
0x4b, 0x31, 0x18,
0x30, 0x16, 0x06, 0x05, 0x2a, 0x85, 0x03, 0x64, 0x01, 0x12, 0x0d, 0x30, 0x31,
0x32, 0x33, 0x34,
0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x30, 0x30, 0x31, 0x1a, 0x30, 0x18, 0x06,
0x08, 0x2a, 0x85,
0x03, 0x03, 0x81, 0x03, 0x01, 0x01, 0x12, 0x0c, 0x30, 0x31, 0x32, 0x33, 0x34,
0x35, 0x36, 0x37,
0x38, 0x39, 0x30, 0x30, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03,
0x0c, 0x0a, 0x74,
0x65, 0x73, 0x74, 0x52, 0x6f, 0x6f, 0x74, 0x43, 0x41, 0x30, 0x1e, 0x17, 0x0d,
0x31, 0x38, 0x30,
0x31, 0x31, 0x36, 0x31, 0x33, 0x35, 0x32, 0x34, 0x33, 0x5a, 0x17, 0x0d, 0x31,
0x39, 0x30, 0x34,
0x31, 0x36, 0x31, 0x34, 0x30, 0x32, 0x34, 0x33, 0x5a, 0x30, 0x11, 0x31, 0x0f,
0x30, 0x0d, 0x06,
0x03, 0x55, 0x04, 0x03, 0x0c, 0x06, 0x53, 0x65, 0x6e, 0x64, 0x65, 0x72, 0x30,
0x66, 0x30, 0x1f,

```

0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x01, 0x01, 0x30, 0x13, 0x06,
0x07, 0x2a, 0x85,
0x03, 0x02, 0x02, 0x24, 0x00, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x02, 0x03,
0x43, 0x00, 0x04, 0x40, 0xa2, 0x6b, 0x68, 0x59, 0xd7, 0x39, 0x34, 0xb1, 0xed,
0x8e, 0x1b, 0x24,
0xcc, 0xa6, 0xd7, 0xe8, 0x3a, 0x88, 0xe4, 0xde, 0x7b, 0x04, 0xba, 0x66, 0x8d,
0x81, 0xc7, 0xf2,
0xf5, 0x18, 0xb4, 0xaf, 0x75, 0xb1, 0xb2, 0xb7, 0xdb, 0x03, 0x7b, 0xd1, 0x71,
0x74, 0x5c, 0xcf,
0x4b, 0x25, 0x55, 0x29, 0x79, 0x2f, 0x03, 0xd4, 0x4f, 0xe1, 0x29, 0x3d, 0x93,
0x78, 0xbe, 0xae,
0xe0, 0xa9, 0x79, 0x4d, 0xa3, 0x82, 0x02, 0x02, 0x30, 0x82, 0x01, 0xfe, 0x30,
0x0e, 0x06, 0x03,
0x55, 0x1d, 0x0f, 0x01, 0x01, 0xff, 0x04, 0x04, 0x03, 0x02, 0x03, 0xa8, 0x30,
0x1d, 0x06, 0x03,
0x55, 0x1d, 0x0e, 0x04, 0x16, 0x04, 0x14, 0xc4, 0x02, 0x97, 0x8f, 0x16, 0xe2,
0x3f, 0x49, 0x43,
0xd6, 0x6b, 0xe6, 0xe7, 0xaf, 0xaa, 0x38, 0xfa, 0xba, 0x24, 0xf4, 0x30, 0x1f,
0x06, 0x09, 0x2b,
0x06, 0x01, 0x04, 0x01, 0x82, 0x37, 0x15, 0x07, 0x04, 0x12, 0x30, 0x10, 0x06,
0x08, 0x2a, 0x85,
0x03, 0x02, 0x02, 0x2e, 0x00, 0x08, 0x02, 0x01, 0x01, 0x02, 0x01, 0x00, 0x30,
0x81, 0x83, 0x06,
0x03, 0x55, 0x1d, 0x23, 0x04, 0x7c, 0x30, 0x7a, 0x80, 0x14, 0x5e, 0x94, 0xaa,
0xdd, 0x0f, 0x73,
0xaa, 0x75, 0xba, 0x76, 0x62, 0xe2, 0x80, 0xb2, 0xde, 0x3d, 0x49, 0x06, 0x77,
0x3b, 0xa1, 0x4f,
0xa4, 0x4d, 0x30, 0x4b, 0x31, 0x18, 0x30, 0x16, 0x06, 0x05, 0x2a, 0x85, 0x03,
0x64, 0x01, 0x12,
0x0d, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x30,
0x30, 0x31, 0x1a,
0x30, 0x18, 0x06, 0x08, 0x2a, 0x85, 0x03, 0x03, 0x81, 0x03, 0x01, 0x01, 0x12,
0x0c, 0x30, 0x31,
0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x30, 0x31, 0x13, 0x30,
0x11, 0x06, 0x03,
0x55, 0x04, 0x03, 0x0c, 0x0a, 0x74, 0x65, 0x73, 0x74, 0x52, 0x6f, 0x6f, 0x74,
0x43, 0x41, 0x82,
0x11, 0x00, 0xc1, 0x0a, 0x67, 0x5d, 0x46, 0x50, 0xb8, 0x80, 0xe7, 0x11, 0x41,
0xdf, 0x43, 0xc4,
0x9f, 0x41, 0x30, 0x1d, 0x06, 0x03, 0x55, 0x1d, 0x25, 0x04, 0x16, 0x30, 0x14,
0x06, 0x08, 0x2b,
0x06, 0x01, 0x05, 0x05, 0x07, 0x03, 0x02, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05,
0x05, 0x07, 0x03,
0x04, 0x30, 0x27, 0x06, 0x09, 0x2b, 0x06, 0x01, 0x04, 0x01, 0x82, 0x37, 0x15,
0x0a, 0x04, 0x1a,
0x30, 0x18, 0x30, 0x0a, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x03,
0x02, 0x30, 0x0a,
0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x03, 0x04, 0x30, 0x52, 0x06,
0x03, 0x55, 0x1d,
0x1f, 0x04, 0x4b, 0x30, 0x49, 0x30, 0x47, 0xa0, 0x45, 0xa0, 0x43, 0x86, 0x41,
0x68, 0x74, 0x74,
0x70, 0x3a, 0x2f, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x68, 0x6f, 0x73, 0x74,
0x2f, 0x63, 0x64,
0x70, 0x2f, 0x35, 0x65, 0x39, 0x34, 0x61, 0x61, 0x64, 0x64, 0x30, 0x66, 0x37,
0x33, 0x61, 0x61,
0x37, 0x35, 0x62, 0x61, 0x37, 0x36, 0x36, 0x32, 0x65, 0x32, 0x38, 0x30, 0x62,
0x32, 0x64, 0x65,
0x33, 0x64, 0x34, 0x39, 0x30, 0x36, 0x37, 0x37, 0x33, 0x62, 0x2e, 0x63, 0x72,
0x6c, 0x30, 0x5d,
0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x01, 0x01, 0x04, 0x51, 0x30,
0x4f, 0x30, 0x4d,

```

0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x30, 0x02, 0x86, 0x41, 0x68,
0x74, 0x74, 0x70,
0x3a, 0x2f, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x68, 0x6f, 0x73, 0x74, 0x2f,
0x61, 0x69, 0x61,
0x2f, 0x35, 0x65, 0x39, 0x34, 0x61, 0x61, 0x64, 0x64, 0x30, 0x66, 0x37, 0x33,
0x61, 0x61, 0x37,
0x35, 0x62, 0x61, 0x37, 0x36, 0x36, 0x32, 0x65, 0x32, 0x38, 0x30, 0x62, 0x32,
0x64, 0x65, 0x33,
0x64, 0x34, 0x39, 0x30, 0x36, 0x37, 0x37, 0x33, 0x62, 0x2e, 0x63, 0x72, 0x74,
0x30, 0x2b, 0x06,
0x03, 0x55, 0x1d, 0x10, 0x04, 0x24, 0x30, 0x22, 0x80, 0x0f, 0x32, 0x30, 0x31,
0x38, 0x30, 0x31,
0x31, 0x36, 0x31, 0x33, 0x35, 0x32, 0x34, 0x32, 0x5a, 0x81, 0x0f, 0x32, 0x30,
0x31, 0x39, 0x30,
0x31, 0x31, 0x36, 0x31, 0x33, 0x35, 0x32, 0x34, 0x32, 0x5a, 0x30, 0x0a, 0x06,
0x08, 0x2a, 0x85,
0x03, 0x07, 0x01, 0x01, 0x03, 0x02, 0x03, 0x41, 0x00, 0xd9, 0x0c, 0x3c, 0x27,
0x93, 0xc7, 0x08,
0x99, 0x9e, 0xef, 0xe3, 0xa1, 0xd7, 0x01, 0x49, 0x13, 0x29, 0x38, 0x99, 0x74,
0x9a, 0x70, 0xa6,
0x5c, 0xe4, 0xc7, 0xb2, 0x4c, 0x54, 0x94, 0x32, 0x70, 0xfa, 0x1f, 0x55, 0x26,
0x1a, 0x9c, 0x2a,
0xc4, 0xfe, 0xfb, 0xcf, 0x44, 0x07, 0xb5, 0x75, 0xe1, 0x25, 0xa4, 0xc0, 0x5b,
0x1f, 0x37, 0x73,
0x72, 0x0f, 0x20, 0x49, 0x73, 0x90, 0xfb, 0xad, 0xcb}; //тестовое значение Certb

```

```

u8 Kb [64] =
{0xa1, 0x44, 0x69, 0xc8, 0xc9, 0x80, 0x49, 0x20, 0xb8, 0x88, 0x25, 0xf6, 0x2c,
0x97, 0x33, 0x8b,
0x98, 0x36, 0x73, 0x2c, 0x4e, 0x3c, 0xa2, 0xde, 0x54, 0x2d, 0x5f, 0x8e, 0xa3,
0xe7, 0x5b, 0x4a,
0x4f, 0x9c, 0xbc, 0x0f, 0x29, 0x60, 0x19, 0x3e, 0x79, 0x5c, 0x5a, 0x70, 0x7c,
0x91, 0x5d, 0x3a,
0x55, 0xa1, 0x25, 0xe9, 0xa6, 0x35, 0xa6, 0x85, 0x7e, 0x2a, 0x32, 0x94, 0x1e,
0x82, 0x10, 0x02}; //тестовое значение Kb

```

```

extern unsigned char p[SIZE];
extern unsigned char a[SIZE];
extern unsigned char b[SIZE];
extern unsigned char q[SIZE];
extern unsigned char x[SIZE];
extern unsigned char y[SIZE];

```

```

void main()
{
    InitEcpStruct Data;
    orkB_str p_orkB_str;
    FILE* file1;
    // инициализация значения элементов структуры (из теста)
    //tag(&tAstr); // на вход функции передается указатель на структуру

    Data.A = a;
    Data.B = b;
    Data.P = p;
    Data.Px = x;
    Data.Py = y;
    Data.Q = q;
    InitEcp341012(&Data);

    orkA(k_A, K_A); // gener open key - > abon B

    p_orkB_str.sa = sa;
    p_orkB_str.Sa = 0;
}

```

```

    p_orkB_str.ka = k_A;
    p_orkB_str.Ida = IDa;
    p_orkB_str.Idb = IDb;
    p_orkB_str.Certa = Certa;
    p_orkB_str.Certb = Certb;
    p_orkB_str.Certa_len = sizeof(Certa);
    p_orkB_str.Certb_len = sizeof(Certb);
    p_orkB_str.Ka = K_A;
    p_orkB_str.Kb = Kb;
    p_orkB_str.TAGb = TAGb;
    p_orkB_str.Kab = Kab;
    p_orkB_str.TAGa = TAGa;
    p_orkB_str.UKM_A = UKM_A;
    p_orkB_str.UKM_B = UKM_B;
    p_orkB_str.UKM_A_len = 8;
    p_orkB_str.UKM_B_len = 8;

    getch();
}
Algebra.cpp
#include "Algebra.h"

//-----
//Обмен координатами (если sm = 1)
//-----
static void ECPswap(ECP_projective *P0, ECP_projective *P1, int sm)
{
    int tmp0, tmp1, i, *ecp[2], mask, smmask, mask1;

    ecp[0] = (int*)P0;
    ecp[1] = (int*)P1;
    mask = 0;
    smmask = sm & 1;
    for(i = 0; i < sizeof(ECP_projective) / sizeof(int); i++)
    {
        mask1 = ecp[0][i] & 1;
        mask ^= mask1;
        smmask ^= mask1;
        tmp0 = ecp[mask][i];
        tmp1 = ecp[mask ^ 1][i];
        ecp[smmask][i] = tmp0;
        ecp[smmask ^ 1][i] = tmp1;
    }
}

//-----
//Удвоение точки (P0 = 2P0)
//-----
static void ECPsqr(ECP_projective *P0)
{
    residue T, S, U;

    Rmul(T, P0->Z, P0->Z, ECP); //T=Z1*Z1
    Rmul(S, ECae, T, ECP);
    Rmul(S, S, T, ECP); //S=a*T*T
    Rmul(T, P0->X, P0->X, ECP);
    Radd(U, T, T, ECP);
    Radd(U, U, T, ECP);
    Radd(U, U, S, ECP); //U=3(X1*X1)+S
    Rmul(T, P0->Y, P0->Y, ECP);
    Radd(T, T, T, ECP); //T=2(Y1*Y1)
    Rmul(S, P0->X, T, ECP);

```

```

    Radd(S, S, S, ECp); //S=2 (X1*T)
    Rmul(P0->Z, P0->Z, P0->Y, ECp);
    Radd(P0->Z, P0->Z, P0->Z, ECp); //Z2=2 (Z1*Y1)
    Rmul(P0->X, U, U, ECp);
    Rsub(P0->X, P0->X, S, ECp);
    Rsub(P0->X, P0->X, S, ECp); //X2=U*U-2S
    Rmul(T, T, T, ECp); //T=T*T
    Rsub(S, S, P0->X, ECp);
    Rmul(P0->Y, U, S, ECp);
    Rsub(P0->Y, P0->Y, T, ECp);
    Rsub(P0->Y, P0->Y, T, ECp); //Y2=U*(S-X2)-2T
}

//-----
//Нахождение кратной точки (Ps = sP)
//-----
void ECPpow(ECP_affine *Ps, ECP_affine *P, residue s)
{
    ECP_projective P0, P1;
    int m, sm;

    affine2projective(&P0, P);
    for(m = n - 1; ((s[m >> 4] >> (m & 15)) == 0) && (m > 0); m--);
    for(m--; m >= 0; m--)
    {
        sm = (s[m >> 4] >> (m & 15)) & 1;
        ECPsqr(&P0);
        ECPmul(&P1, &P0, P);
        ECPswap(&P0, &P1, sm);
    }
    projective2affine(Ps, &P0);
}

//-----
//Определение проективных координат по аффинным
//-----
void affine2projective(ECP_projective *Pp, ECP_affine *Pa)
{
    GF2MF(Pp->X, Pa->x, ECe2_p, ECp);
    GF2MF(Pp->Y, Pa->y, ECe2_p, ECp);
    Rcopy(Pp->Z, ECe2_p);
}

//-----
//Определение аффинных координат по проективным
//-----
void projective2affine(ECP_affine *Pa, ECP_projective *Pp)
{
    residue Zinv;

    Rinv(Zinv, Pp->Z, ECp);
    Rmul(Pa->x, Pp->X, Zinv, ECp);
    Rmul(Pa->y, Pp->Y, Zinv, ECp);
    MF2GF(Pa->x, Pa->x, ECp);
    Rmul(Pa->y, Pp->Y, Zinv, ECp);
    Rmul(Pa->y, Pa->y, Zinv, ECp);
    Rmul(Pa->y, Pa->y, Zinv, ECp);
    MF2GF(Pa->y, Pa->y, ECp);
}

//-----
//Сумма точек (P3 = P1 + P2)
//-----

```

```

void ECPmul(ECP_projective *P3, ECP_projective *P1, ECP_affine *P2)
{
    residue T, S, TT, A, B, Z12, B12, B13;
    residue x2, y2;

    GF2MF(x2, P2->x, ECe2_p, ECp);
    GF2MF(y2, P2->y, ECe2_p, ECp);

    Rmul(Z12, P1->Z, P1->Z, ECp);           //Z1*Z1
    Rmul(T, Z12, P1->Z, ECp);               //Z1*Z1*Z1
    Rmul(S, T, y2, ECp);                   //Z1*Z1*Z1*y2
    Rsub(A, S, P1->Y, ECp);                 //A = Z1*Z1*Z1*y2-Y1

    Rmul(T, Z12, x2, ECp);                 //Z1*Z1*x2
    Rsub(B, T, P1->X, ECp);                 //B = Z1*Z1*x2-X1

    Rmul(B12, B, B, ECp);                  //B*B
    Rmul(TT, P1->X, B12, ECp);              //X1*B*B
    Rmul(S, A, A, ECp);                    //A*A
    Rmul(B13, B12, B, ECp);                //B*B*B
    Rsub(T, S, B13, ECp);                  //A*A-B*B*B
    Rsub(S, T, TT, ECp);                   //X3 = A*A-B*B*B-
2*X1*B*B
    Rsub(T, TT, P3->X, ECp);                //X1*B*B-X3
    Rmul(S, A, T, ECp);                    //A*(X1*B*B-X3)
    Rmul(T, P1->Y, B13, ECp);              //Y1*B*B*B
    Rsub(P3->Y, S, T, ECp);                 //Y3 = A*(X1*B*B-X3)-
Y1*B*B*B

    Rmul(P3->Z, P1->Z, B, ECp);             //Z3 = Z1*B
}

//-----
//Восстановление координаты Y
//-----
void ECPy(ECP_affine *P, residue x)
{
    residue g, t;

    GF2MF(t, x, ECe2_p, ECp);
    Rmul(g, t, t, ECp);
    Radd(g, g, ECae, ECp);
    Rmul(g, g, t, ECp);
    Radd(g, g, ECbe, ECp);
    Rsqrt(t, g);
    MF2GF(P->y, t, ECp);
}

//-----
//Копирование
//-----
void Rcopy(residue dest, residue src)
{
    int i;

    for(i = 0; i < N; i++)
    {
        dest[i] = src[i];
    }
}

```

```

//-----
//Сложение чисел (S = A + B) (mod P)
//-----
void Radd(residue S, residue A, residue B, residue P)
{
    int i;
    long r;

    r = 0;
    for(i = 0; i < N; i++)
    {
        r = r + A[i] + B[i];
        S[i] = (short int)r;
        r >>= 16;
    }
    Mod(S, S, P);
}

//-----
//Вычитание чисел (S = A - B) (mod P)
//-----
void Rsub(residue S, residue A, residue B, residue P)
{
    int i;
    long r;

    r = 0;
    for(i = 0; i < N; i++)
    {
        r = r + A[i] - B[i];
        S[i] = (short int)r;
        r >>= 16;
    }
    if(r < 0)
    {
        r=0;
        for(i = 0; i < N; i++)
        {
            r = r + S[i] + P[i];
            S[i] = (short int)r;
            r >>= 16;
        }
    }
}

//-----
//Умножение чисел (S = A * B) (mod P) - умножение Монтгомери
//-----
void Rmul(residue S, residue A, residue B, residue P)
{
    int i, j;
    unsigned short int u, b, z;
    unsigned long int r0, r1;
    residue S0;

    if(P == ECp)
    {
        z = ECp0inv;
    }
    else
    {
        z = ECq0inv;
    }
}

```

```

    for(i = 0; i < N; i++)
    {
        S0[i] = 0;
    }
    for(i = 0; i < N; i++)
    {
        b = B[i];
        r0 = A[0] * (unsigned long)b + S0[0];
        u = ((short int)r0) * z;
        r1 = P[0] * (unsigned long)u;
        r0 += (unsigned short int)r1;
        r1 >>= 16;
        r0 >>= 16;
        for(j = 1; j < N; j++)
        {
            r1 += P[j] * (unsigned long)u + S0[j];
            r0 += A[j] * (unsigned long)b + ((unsigned short int)r1);
            S0[j - 1] = (unsigned short int)r0;
            r1 >>= 16;
            r0 >>= 16;
        }
        S0[N - 1] = (unsigned short int)(r1 + r0);
    }
    Rcopy(S, S0);
    Mod(S, S, P);
}

//-----
//Возведение в степень (S = A ^ (P - 3)/4) (mod p)
//-----
void Rpow_pdiv4(residue S, residue A, residue P)
{
    int m;

    Rcopy(S, A);
    for(m = n - 1; ((P[m >> 4] >> (m & 15)) & 1) == 0; m--);
    for(m--; m >= 2; m--)
    {
        Rmul(S, S, S, ECp);
        if(((P[m >> 4] >> (m & 15)) & 1) == 1)
        {
            Rmul(S, S, A, ECp);
        }
    }
}

//-----
//Возведение в степень (S = A^B) (mod P)
//-----
void Rstep(residue S, residue A, residue B, residue P)
{
    int m;
    unsigned short *E2;

    Rcopy(S, A);
    for(m = n - 1; ((B[m >> 4] >> (m & 15)) & 1) == 0; m--);
    if(m < 0)
    {
        if(P == ECp)
        {
            E2 = ECe2_p;
        }
        else

```

```

        {
            E2 = ECe2_q;
        }
        GF2MF(S, ECone, E2, P);
    }
    else
    {
        for(m--; m >= 0; m--)
        {
            Rmul(S, S, S, P);
            if(((B[m >> 4] >> (m & 15)) & 1) == 1)
            {
                Rmul(S, S, A, P);
            }
        }
    }
}

//-----
//Нахождение обратного ( $S = A^{-1} \pmod P$ )
//-----
void Rinv(residue S, residue A, residue P)
{
    unsigned short *PP;

    if(P == ECp)
    {
        PP = ECp_2;
    }
    else
    {
        PP = ECq_2;
    }
    Rstep(S, A, PP, P);
}

//-----
//Деление на 2 в GF ( $S = A/2$ )
//-----
static void Div2(residue S, residue A)
{
    int i;

    for(i = 0; i < N - 1; i++)
    {
        S[i] = A[i] >> 1;
        S[i] |= (A[i + 1] & 0x01) << 15;
    }
    S[N - 1] = 0;
}

//-----
//Сравнение двух длинных чисел ( $S \neq A(1), S == A(0)$ )
//-----
static int Rcmp(residue S, residue A)
{
    int i, res = 0;

    for(i = 0; i < N - 1; i++)
    {
        if(S[i] != A[i])
        {
            res = 1;

```

```

        break;
    }
}
return res;
}

//-----
//Нахождение квадратичного невычета ( $B^{(p-1)/2} = -1 \pmod{p}$ )
//-----
static void Findb(residue S, residue START, residue P_12)
{
    int trying = 100;
    residue B, T, TT;

    Rcopy(TT, START);
    while(trying)
    {
        GF2MF(B, TT, ECe2_p, ECp);
        Rstep(T, B, P_12, ECp);
        MF2GF(B, T, ECp);
        Radd(T, B, ECone, ECp);
        if(!Rcmp(T, ECzr))
        {
            break;
        }
        Rsub(TT, TT, ECone, ECp);
    }
    Rcopy(S, TT);
}

//-----
//Извлечение квадратного корня
//-----
void Rsqrt(residue S, residue A)
{
    residue B, R1, R2, Ss, Sss, Z, T, P_1;

    if((ECp[0] & 0x03) == 0x03)
    {
        //p = 3 (mod 4) -> x = a^((p+1)/4) (mod p)
        Rpow_pdiv4(S, A, ECp);
        Rmul(S, S, A, ECp);
    }
    else
    {
        //Произвольный простой модуль
        Rsub(P_1, ECp, ECone, ECp);
        Div2(Ss, P_1);
        Findb(B, P_1, Ss);
        Rcopy(Sss, Ss);
        Rstep(R1, A, Sss, ECp);
        GF2MF(R2, ECone, ECe2_p, ECp);
        Rcopy(Z, ECzr);
        while(1)
        {
            Rmul(T, R1, R2, ECp);
            MF2GF(T, T, ECp);
            if(Rcmp(T, ECone))
            {
                if(!Rcmp(T, P_1))
                {
                    Radd(Z, Z, Sss, ECp);
                }
            }
        }
    }
}

```

```

    }
    Div2(Z, Z);
    if (Ss[0] & 0x01)
    {
        break;
    }
    Div2(Ss, Ss);
    Rstep(R1, A, Ss, ECp);
    Rstep(R2, B, Z, ECp);
}
Radd(Ss, Ss, ECone, ECp);
Div2(Ss, Ss);
Rstep(R1, A, Ss, ECp);
Rstep(R2, B, Z, ECp);
Rmul(S, R1, R2, ECp);
}
}

```

```

//-----
//Определение  $z = -P_0^{-1} \pmod{b}$  (используется в умножении Монтгомери)
//-----

```

```

unsigned short Findz(unsigned short a)
{
    int i;
    unsigned short d;

    if((a & 1) == 0)
    {
        return 0;
    }
    for(i=0, d=a; i < 16 - 2; i++)
    {
        d *= d;
        d *= a;
    }
    return (0 - d);
}

```

```

//-----
//Определение  $e = (2^{16})^N \pmod{P}$  (проективная координата Z)
//-----

```

```

void Finde(residue RES, residue P)
{
    int i;

    for(i = 0; i < (N - 1); i++)
    {
        RES[i] = 0;
    }
    RES[i] = 0x0001;
    Mod(RES, RES, P);
    // * 2^16
    while(i)
    {
        RES[i] = RES[i - 1];
        i--;
    }
    RES[0] = 0x0000;
    Mod(RES, RES, P);
}

```

```

}

//-----
//Нахождение B = A (mod P)
//-----
void Mod(residue B, residue A, residue P)
{
    Rsub(A, A, P, P);
    while(A[N - 1] || (A[N - 2] > P[N - 2]))
    {
        Rsub(A, A, P, P);
    }
    Rcopy(B, A);
}

//-----
//Сравнение длинных чисел размерностью 2 * N
//Выход:
// - 0 - A > B
// - 1 - иначе
//-----
static unsigned int BigCmp(residue A, residue B)
{
    int i;

    for(i = 2 * N - 1; i >= 0; i--)
    {
        if(A[i] > B[i])
        {
            return 0;
        }
        if(A[i] < B[i])
        {
            return 1;
        }
    }
    return 1;
}

//-----
//Сдвиг числа размерностью на S слов вправо
//-----
static void ModShift(residue A, residue B, int S)
{
    int i;

    for(i = 0; i < N - 1; i++)
    {
        A[N - 2 - i + S] = B[N - 2 - i];
    }
    for(i = 0; i < S; i++)
    {
        A[i] = 0;
    }
    for(i = N - 1 + S; i < 2 * N; i++)
    {
        A[i] = 0;
    }
}

//-----

```

```

//Вычитание чисел размерность 2 * N по модулю P
//-----
static void BigSub(residue A, residue B, residue P)
{
    long i, r;

    r = 0;
    while(r >= 0)
    {
        for(i = 0; i < 2 * N; i++)
        {
            r = r + A[i] - B[i];
            A[i] = (short int)r;
            r >>= 16;
        }
    }
    r = 0;
    for(i = 0; i < 2 * N; i++)
    {
        r = r + A[i] + P[i];
        A[i] = (short int)r;
        r >>= 16;
    }
}

//-----
//Умножение чисел (S = A * B) (mod p)
//-----
void Mul(residue S, residue A, residue B, residue P)
{
    int i, j, k;
    unsigned short int d;
    unsigned long int T;
    unsigned short int TMP[2 * N];
    unsigned short int MOD[2 * N];
    unsigned short int MODs[2 * N];

    for(i = 0; i < N; i++)
    {
        TMP[i] = 0;
        MOD[i] = P[i];
    }
    for(i = N; i < 2 * N; i++)
    {
        MOD[i] = 0;
    }
    for(i = 0; i < N; i++)
    {
        d = 0;
        for(j = 0; j < N; j++)
        {
            T = (unsigned long int)A[i] * B[j] + TMP[i + j] + d;
            TMP[i + j] = (unsigned short)T;
            d = (unsigned short)(T >> 16);
        }
        TMP[i + j] = d;
    }
    //Вычисление модуля
    //Определение номера старшего слова числа
    for(j = 2 * N; (TMP[j - 1] == 0) && (j != 0); j--);
    if(j < (N - 1))
    {
        Rcopy(S, TMP);
    }
}

```

```

        return;
    }
    //Определение сдвига модуля
    k = j - N + 1;
    while(k)
    {
        //Сдвиг модуля
        ModShift(MODs, MOD, k);
        //Сравнение и вычитание
        if(!BigCmp(TMP, MODs))
        {
            BigSub(TMP, MODs, MODs);
        }
        k--;
    }
    BigSub(TMP, MOD, MOD);
    Rcopy(S, TMP);
}
//-----
//Умножение чисел (S = A * B)
//-----
void Mul_(residue S, residue A, residue B)
{
    int i, j;
    unsigned short int d;
    unsigned long int T;
    unsigned short int TMP[2 * N];

    for(i = 0; i < N; i++)
    {
        TMP[i] = 0;
    }

    for(i = 0; i < N; i++)
    {
        d = 0;
        for(j = 0; j < N; j++)
        {
            T = (unsigned long int)A[i] * B[j] + TMP[i + j] + d;
            TMP[i + j] = (unsigned short)T;
            d = (unsigned short)(T >> 16);
        }
        TMP[i + j] = d;
    }

    Rcopy(S, TMP);
    return;
}

```

Еср341012.cpp

```

//#include <stdio.h>
#include "Еср341012.h"
#include "Неш341112.h"
/*****
*****/
//Исходные данные
/*****
*****/
//p (модуль эллиптической кривой)
residue ЕСр;

residue ЕСр_2;

```

```

//a (коэффициент эллиптической кривой в поле Монтгомери)
residue ECae;

//b (коэффициент эллиптической кривой в поле Монтгомери)
residue ECbe;

//q (порядок циклической подгруппы группы точек эллиптической кривой)
residue ECq;

residue ECq_2;

//P0x (коэффициент точки эллиптической кривой)
residue ECUX;

//P0y (коэффициент точки эллиптической кривой)
residue ECUY;
residue ECe_p;
residue ECe_q;
residue ECe2_p;
residue ECe2_q;
residue ECone = {0x0001, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,
0x0000
};

residue Const2 = { 0x0002, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,
0x0000
};

residue ECzr = { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,
0x0000
};

//Точка эллиптической кривой (x - координата)
extern unsigned char x[SIZE];
//Точка эллиптической кривой (y - координата)
extern unsigned char y[SIZE];

unsigned short int ECp0inv = 0;
unsigned short int ECq0inv = 0;
/*****
*****/

HeshStruct HeshData;
//unsigned char Hesh[64];
unsigned short IsNull(unsigned char *digit);
unsigned int VKO_GOST341012(VKOStruct * Data);

//-----
//Проверка длинного числа на 0
//На входе:
//      - длинное число
//На выходе:
//      - результат (0-не равно,1-равно)

```

```

//-----
unsigned short IsNull(unsigned char *digit)
{
    int i;

    for(i = 0; i < SIZE; i++)
    {
        if(digit[i])
        {
            return 0;
        }
    }
    return 1;
}

//-----
//Проверка неравенства (0 < digit < q)
//На входе:
//      - длинное число
//На выходе:
//      - результат (0-не удовлетворяет,1-удовлетворяет)
//-----
static unsigned short IsCmp(unsigned char *digit)
{
    int i;

    if(IsNull(digit))
    {
        return 0;
    }
    i = SIZE;
    while(i)
    {
        if(*((unsigned char *)ECq + i - 1) > digit[i - 1])
        {
            return 1;
        }
        else if(*((unsigned char *)ECq + i - 1) < digit[i - 1])
        {
            return 0;
        }
        i--; //!!!!!!!!!!!!
    }
    return 0;
}

//-----
//Формирование ЭЦП
//На входе:
//      - структура с данными
//На выходе:
//      - результат работы (0-ошибка-некорректное K,1-норма)
//-----
static unsigned short EcpForm(EcpStruct * Data)
{
    int i;
    residue h, e, r, d, k, s;
    ECP_affine U;

    //Переписываем ХЭШ, k и d
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)h + i) = Data->Hesh[i];
    }
}

```

```

        *((unsigned char *)k + i) = Data->K[i];
        *((unsigned char *)d + i) = Data->Skey[i];
    }
    h[N - 1] = 0;
    k[N - 1] = 0;
    d[N - 1] = 0;
    //ШАГ 2 (e = a(modq))
    Mod(e, h, ECq);
    //Проверка на 0
    if(IsNull((unsigned char *)e))
    {
        //e = 1
        e[0] = 0x0001;
    }
    //ШАГ 3 (0 < k < q)
    if(!IsCmp(Data->K))
    {
        return 0;
    }
    //ШАГ 4 (C = kP)
    Rcopy(U.x, ECUx);
    Rcopy(U.y, ECUy);
    ECPpow(&U, &U, k);
    //r = Xc(modq)
    Mod(r, U.x, ECq);
    //Проверка на 0
    if(IsNull((unsigned char *)r))
    {
        return 0;
    }
    //ШАГ 5 (s = (rd + ke) (modq))
    GF2MF(s, r, ECe2_q, ECq);
    GF2MF(d, d, ECe2_q, ECq);
    GF2MF(k, k, ECe2_q, ECq);
    GF2MF(e, e, ECe2_q, ECq);
    Rmul(h, s, d, ECq);
    Rmul(e, k, e, ECq);
    Radd(s, h, e, ECq);
    MF2GF(s, s, ECq);
    Mod(s, s, ECq);
    //Переписываем результат r || s
    for(i = 0; i < SIZE; i++)
    {
        Data->Ecp[i] = *((unsigned char *)r + i);
        Data->Ecp[i + SIZE] = *((unsigned char *)s + i);
    }
    return 1;
}

//-----
//Проверка ЭЦП (спомогательная функция)
//На входе:
//      - данные
//На входе:
//      - результат работы (0-ошибка-ЭЦП не верна,1-норма)
//-----
static unsigned short EcpCheck(residue e, residue r, residue s, residue x,
residue y)
{
    ECP_affine U, V;
    ECP_projective C;
    residue v, z1, z2;
    residue ee, ss, rr;

```

```

int i;

GF2MF(ee, e, ECe2_q, ECq);
GF2MF(ss, s, ECe2_q, ECq);
GF2MF(rr, r, ECe2_q, ECq);
GF2MF(ECzr, ECzr, ECe2_q, ECq);
//ШАГ 4 ( $v = e^{-1} \pmod{q}$ )
Rinv(v, ee, ECq);
//ШАГ 5 ( $z1 = sv \pmod{q}$ ,  $z2 = -rv \pmod{q}$ )
Rmul(z1, ss, v, ECq);
Rmul(z2, rr, v, ECq);
Rsub(z2, ECzr, z2, ECq);
MF2GF(z1, z1, ECq);
MF2GF(z2, z2, ECq);
//ШАГ 6 ( $C = z1P + z2Q$ )
Rcopy(U.x, ECUx);
Rcopy(U.y, ECUy);
ECPpow(&U, &U, z1);
Rcopy(V.x, x);
Rcopy(V.y, y);
ECPpow(&V, &V, z2);
affine2projective(&C, &U);
ECPmul(&C, &C, &V);
projective2affine(&U, &C);
//R = Xc(modq)
Mod(v, U.x, ECq);
//ШАГ 7 сравниваем подписи (v и r)
for(i = 0; i < (N - 1); i++)
{
    if(v[i] != r[i])
    {
        return 0;
    }
}
return 1;
}

//-----
//Проверка ЭЦП
//На входе:
//      - структура с данными
//На входе:
//      - результат работы (0-ошибка-ЭЦП не верна, 1-норма)
//-----
static unsigned short EcpControl(EcpStruct * Data)
{
    unsigned int i;
    ECP_affine U;
    residue h, e, r, s, x, y;

    //ШАГ 1 (r и s) ШАГ 2 (h)
    if((IsNull(Data->Ecp)) || (IsNull(Data->Ecp + SIZE)))
    {
        return 0;
    }
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)r + i) = Data->Ecp[i];
        *((unsigned char *)s + i) = Data->Ecp[i + SIZE];
        *((unsigned char *)h + i) = Data->Hesh[i];
        *((unsigned char *)x + i) = Data->Okey[i];
    }
    r[N - 1] = 0;

```

```

s[N - 1] = 0;
h[N - 1] = 0;
x[N - 1] = 0;
//ШАГ 3 (e = a(modq))
Mod(e, h, ECq);
//Проверка на 0
if(IsNull((unsigned char *)e))
{
    //e = 1
    e[0] = 0x0001;
}
//Восстановление y-координаты
ECPy(&U, x);
Rcopy(y, U.y);
if(!EcpCheck(e, r, s, x, y))
{
    //Еще раз восстанавливаем y-координату
    Rsub(y, ECp, y, ECp);
    return EcpCheck(e, r, s, x, y);
}
return 1;
}

//-----
//Формирование открытого ключа проверки ЭЦП
//На входе:
//      - структура с данными
//На входе:
//      - результат работы (0-ошибка,1-норма)
//-----
static unsigned short EcpFormOkey(EcpStruct * Data)
{
    int i;
    residue d;
    ECP_affine U;

    //(0 < d < q)
    if(!IsCmp(Data->Skey))
    {
        return 0;
    }
    //Переписываем секретный ключ
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)d + i) = Data->Skey[i];
    }
    d[N - 1] = 0;

    //Запись параметров
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)ECUx + i) = x[i]; //координата X
        *((unsigned char *)ECUy + i) = y[i]; //координата Y
    }
    //Q = dP
    Rcopy(U.x, ECUx);
    Rcopy(U.y, ECUy);
    ECPpow(&U, &U, d);
    //Проверка на 0
    if(IsNull((unsigned char *) (U.x)))
    {

```

```

        return 0;
    }
    //Проверка на 0
    if(IsNull((unsigned char *) (U.y)))
    {
        return 0;
    }
    //Переписываем результат x
    for(i = 0; i < SIZE; i++)
    {
        Data->Okey[i] = *((unsigned char *) (U.x) + i);
    }
    //Переписываем результат y
    for(i = 0; i < SIZE; i++)
    {
        Data->Okey[i + SIZE] = *((unsigned char *) (U.y) + i);
    }
    return 1;
}

/*****
//Инициализация параметров ЭЦП
//На входе:
//      - указатель на структуру с данными
//На выходе:
//      - результат работы (0-ошибка, 1-норма)
*****/
unsigned short InitEcp341012(InitEcpStruct * Data)
{
    unsigned int i;
    residue Tmp, Tmp1, A, B;

    //Запись параметров
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *) Ecp + i) = Data->P[i];           //модуль
        //эллиптической кривой
        *((unsigned char *) ECq + i) = Data->Q[i];           //порядок
        //циклической подгруппы группы точек эллиптической кривой
        *((unsigned char *) A + i) = Data->A[i];
        //коэффициент A эллиптической кривой
        *((unsigned char *) B + i) = Data->B[i];
        //коэффициент B эллиптической кривой
        *((unsigned char *) ECUx + i) = Data->Px[i];
        //координата X точки эллиптической кривой
        *((unsigned char *) ECUy + i) = Data->Py[i];
        //координата Y точки эллиптической кривой
    }
    Ecp[N - 1] = 0;
    ECq[N - 1] = 0;
    A[N - 1] = 0;
    B[N - 1] = 0;
    ECUx[N - 1] = 0;
    ECUy[N - 1] = 0;
    //Определение  $z = -P_0^{-1} \pmod{b}$  (используется в умножении Монтгомери)
    Ecp0inv = Findz(Ecp[0]);
    ECq0inv = Findz(ECq[0]);
    if((!Ecp0inv) || (!ECq0inv))
    {
        Ecp0inv = 0;
        ECq0inv = 0;
        return 0;
    }
}

```

```

//Определение  $e = (2^{16})^N \pmod P$  (проективная координата Z)
Finde(ECe_p, ECp);
Finde(ECe_q, ECq);
//Проверка
Rmul(Tmp, ECe_p, ECe_p, ECp);
Rmul(Tmp1, ECe_q, ECe_q, ECq);
for(i = 0; i < (N - 1); i++)
{
    if((Tmp[i] != ECe_p[i]) || (Tmp1[i] != ECe_q[i]))
    {
        ECp0inv = 0;
        ECq0inv = 0;
        return 0;
    }
}
//Определение  $e^2 = e^2 \pmod P$ 
Mul(ECe2_p, ECe_p, ECe_p, ECp);
Mul(ECe2_q, ECe_q, ECe_q, ECq);
GF2MF(Ecae, A, ECe2_p, ECp);
GF2MF(ECbe, B, ECe2_q, ECq);
//Нахождение P-2 и Q-2 (для вычисления обратного элемента)
Rsub(ECp_2, ECp, Const2, ECp);
Rsub(ECq_2, ECq, Const2, ECq);
return 1;
}

/*****/
//Программная реализация алгоритма ЭЦП по ГОСТ 34.10-2012
//На входе:
//      - указатель на структуру с данными
//На выходе:
//      - результат работы (0-ошибка, 1-норма)
/*****/
unsigned short Ecp341012(EcpStruct * Data)
{
    unsigned short res = 0;

    if(ECp0inv && ECq0inv)
    {
        switch(Data->Pr)
        {
            //Формирование ЭЦП
            case ECP_FORM:
                if((Data->Hesh == NULL_) || (Data->Skey == NULL_) ||
(Data->Ecp == NULL_) || (Data->K == NULL_))
                {
                    break;
                }
                res = EcpForm(Data);
                break;
            //Проверка ЭЦП
            case ECP_CONTROL:
                if((Data->Hesh == NULL_) || (Data->Okey == NULL_) ||
(Data->Ecp == NULL_))
                {
                    break;
                }
                res = EcpControl(Data);
                break;
            //Формирование открытого ключа проверки ЭЦП
            case ECP_FORM_OKEY:
                if((Data->Skey == NULL_) || (Data->Okey == NULL_))

```

```

        {
            break;
        }
        res = EcpFormOkey(Data);
        break;
    }
}
return res;
}

```

Hesh341112.cpp

```

#define _CRT_SECURE_NO_WARNINGS
//#include <stdio.h>
#include "Hesh341112.h"

#define TR_NO 0x00 //не использовать
//промежуточное состояние
#define TR_YES 0x01 //использовать
//промежуточное состояние
unsigned short Hesh341112(unsigned int HeshLen, HeshStruct * Data);

//Значения подстановки
static unsigned char pi[] = {
    252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218,
35, 197, 4, 77,
    233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187,
20, 205, 95, 193,
    249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66,
139, 1, 142, 79,
    5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152,
127, 212, 211, 31,
    235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162,
253, 58, 206, 204,
    181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71,
156, 183, 93, 135,
    21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111,
157, 158, 178, 177,
    50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128,
195, 189, 13, 87,
    223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246,
124, 34, 185, 3,
    224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80,
78, 51, 10, 74,
    167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130,
100, 159, 38, 65,
    173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125,
105, 213, 149, 59,
    7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228,
136, 217, 231, 137,
    225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144,
202, 216, 133, 97,
    32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208,
190, 229, 108, 82,
    89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194,
57, 75, 99, 182
};

//Значения перестановки
static unsigned char t[] = {
    0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25,
33, 41, 49, 57,
    2, 10, 18, 26, 34, 42, 50, 58, 3, 11, 19, 27,
35, 43, 51, 59,

```

```

        4, 12, 20, 28, 36, 44, 52, 60, 5, 13, 21, 29,
37, 45, 53, 61,
        6, 14, 22, 30, 38, 46, 54, 62, 7, 15, 23, 31,
39, 47, 55, 63
};

```

//Матрица A

```

static unsigned long int A[] = {
    0x2BA0B470, 0x8E20FAA7, 0x9B505A38, 0x47107DDD, 0xC3282D1C,
0xAD08B0E0, 0xEF14980E, 0xD8045870,
    0xF90A4C07, 0x6C022C38, 0xF205268D, 0x3601161C, 0x798C13C8,
0x1B8E0B0E, 0xB2468764, 0x83478B07,
    0x818E8F40, 0xA011D380, 0xCE47C920, 0x5086E740, 0x67ADEA10,
0x2843FD20, 0xBDD87508, 0x14AFF010,
    0xD06CB404, 0x0AD97808, 0x68365A02, 0x05E23C04, 0x341B2D01,
0x8C711E02, 0x1A83988E, 0x46B60F01,
    0x387AE76F, 0x90DAB52A, 0x1C3DFDB9, 0x486DD415, 0x0E90F0D2,
0x24B86A84, 0x07487869, 0x125C3542,
    0x8D243CBA, 0x092E9421, 0xC8121E5D, 0x8A174A9E, 0x64090FA0,
0x4585254F, 0x328A8950, 0xACCC9CA9,
    0x5F661451, 0x9D4DF05D, 0xA1330AA6, 0xC0A878A0, 0xDE970553,
0x60543C50, 0x6FC58CA7, 0x302A1E28,
    0xB9EC46DD, 0x18150F14, 0xD27623E0, 0x0C84890A, 0x693B9F70,
0x0642CA05, 0xBA93C138, 0x0321658C,
    0x9CE8AAA8, 0x86275DF0, 0x4E745554, 0x439DA078, 0x273AA42A,
0xAFC0503C, 0x9D1D5215, 0xD960281E,
    0xC0802984, 0xE230140F, 0x60409A42, 0x71180A89, 0x30204D21,
0xB60C05CA, 0x1810A89E, 0x5B068C65,
    0x7A3805B9, 0x456C3488, 0x3D1C8CD2, 0xAC361A44, 0x900E4669,
0x561B0D22, 0x480723BA, 0x2B838811,
    0x248D9F5D, 0x9BCF4486, 0x12C8C1A0, 0xC3E92243, 0x0964EE50,
0xEFFA11AF, 0x8A327728, 0xF97D86D9,
    0xA80B329C, 0xE4FA2054, 0x548B194E, 0x727D102A, 0x2ACB8227,
0x39B00815, 0x15EB419D, 0x92580484,
    0x84FBAEC0, 0x492C0242, 0x42F35760, 0xAA160121, 0x21F7A530,
0x550B8E9E, 0x9EF5DC18, 0xA48B474F,
    0x2440598E, 0x70A6A56E, 0x1220A247, 0x3853DC37, 0x091051AD,
0x1CA76E95, 0x8A08A6D8, 0x0EDD37C4,
    0x4504536C, 0x07E09562, 0xAC02A736, 0x8D70C431, 0x5601DD1B,
0xC8386296, 0x2B8EE083, 0x641C314B
};

```

//Итерационные константы

```

static unsigned char C[12][64] = {
    {
        0x07, 0x45, 0xA6, 0xF2, 0x59, 0x65, 0x80, 0xDD, 0x23, 0x4D,
0x74, 0xCC, 0x36, 0x74, 0x76, 0x05,
        0x15, 0xD3, 0x60, 0xA4, 0x08, 0x2A, 0x42, 0xA2, 0x01, 0x69,
0x67, 0x92, 0x91, 0xE0, 0x7C, 0x4B,
        0xFC, 0xC4, 0x85, 0x75, 0x8D, 0xB8, 0x4E, 0x71, 0x16, 0xD0,
0x45, 0x2E, 0x43, 0x76, 0x6A, 0x2F,
        0x1F, 0x7C, 0x65, 0xC0, 0x81, 0x2F, 0xCB, 0xEB, 0xE9, 0xDA,
0xCA, 0x1E, 0xDA, 0x5B, 0x08, 0xB1
    },
    {
        0xB7, 0x9B, 0xB1, 0x21, 0x70, 0x04, 0x79, 0xE6, 0x56, 0xCD,
0xCB, 0xD7, 0x1B, 0xA2, 0xDD, 0x55,
        0xCA, 0xA7, 0x0A, 0xDB, 0xC2, 0x61, 0xB5, 0x5C, 0x58, 0x99,
0xD6, 0x12, 0x6B, 0x17, 0xB5, 0x9A,
        0x31, 0x01, 0xB5, 0x16, 0x0F, 0x5E, 0xD5, 0x61, 0x98, 0x2B,
0x23, 0x0A, 0x72, 0xEA, 0xFE, 0xF3,
        0xD7, 0xB5, 0x70, 0x0F, 0x46, 0x9D, 0xE3, 0x4F, 0x1A, 0x2F,
0x9D, 0xA9, 0x8A, 0xB5, 0xA3, 0x6F
    }
};

```

```

    },
    {
        0xB2, 0x0A, 0xBA, 0x0A, 0xF5, 0x96, 0x1E, 0x99, 0x31, 0xDB,
0x7A, 0x86, 0x43, 0xF4, 0xB6, 0xC2,
        0x09, 0xDB, 0x62, 0x60, 0x37, 0x3A, 0xC9, 0xC1, 0xB1, 0x9E,
0x35, 0x90, 0xE4, 0x0F, 0xE2, 0xD3,
        0x7B, 0x7B, 0x29, 0xB1, 0x14, 0x75, 0xEA, 0xF2, 0x8B, 0x1F,
0x9C, 0x52, 0x5F, 0x5E, 0xF1, 0x06,
        0x35, 0x84, 0x3D, 0x6A, 0x28, 0xFC, 0x39, 0x0A, 0xC7, 0x2F,
0xCE, 0x2B, 0xAC, 0xDC, 0x74, 0xF5
    },
    {
        0x2E, 0xD1, 0xE3, 0x84, 0xBC, 0xBE, 0x0C, 0x22, 0xF1, 0x37,
0xE8, 0x93, 0xA1, 0xEA, 0x53, 0x34,
        0xBE, 0x03, 0x52, 0x93, 0x33, 0x13, 0xB7, 0xD8, 0x75, 0xD6,
0x03, 0xED, 0x82, 0x2C, 0xD7, 0xA9,
        0x3F, 0x35, 0x5E, 0x68, 0xAD, 0x1C, 0x72, 0x9D, 0x7D, 0x3C,
0x5C, 0x33, 0x7E, 0x85, 0x8E, 0x48,
        0xDD, 0xE4, 0x71, 0x5D, 0xA0, 0xE1, 0x48, 0xF9, 0xD2, 0x66,
0x15, 0xE8, 0xB3, 0xDF, 0x1F, 0xEF
    },
    {
        0x57, 0xFE, 0x6C, 0x7C, 0xFD, 0x58, 0x17, 0x60, 0xF5, 0x63,
0xEA, 0xA9, 0x7E, 0xA2, 0x56, 0x7A,
        0x16, 0x1A, 0x27, 0x23, 0xB7, 0x00, 0xFF, 0xDF, 0xA3, 0xF5,
0x3A, 0x25, 0x47, 0x17, 0xCD, 0xBF,
        0xBD, 0xFF, 0x0F, 0x80, 0xD7, 0x35, 0x9E, 0x35, 0x4A, 0x10,
0x86, 0x16, 0x1F, 0x1C, 0x15, 0x7F,
        0x63, 0x23, 0xA9, 0x6C, 0x0C, 0x41, 0x3F, 0x9A, 0x99, 0x47,
0x47, 0xAD, 0xAC, 0x6B, 0xEA, 0x4B
    },
    {
        0x6E, 0x7D, 0x64, 0x46, 0x7A, 0x40, 0x68, 0xFA, 0x35, 0x4F,
0x90, 0x36, 0x72, 0xC5, 0x71, 0xBF,
        0xB6, 0xC6, 0xBE, 0xC2, 0x66, 0x1F, 0xF2, 0x0A, 0xB4, 0xB7,
0x9A, 0x1C, 0xB7, 0xA6, 0xFA, 0xCF,
        0xC6, 0x8E, 0xF0, 0x9A, 0xB4, 0x9A, 0x7F, 0x18, 0x6C, 0xA4,
0x42, 0x51, 0xF9, 0xC4, 0x66, 0x2D,
        0xC0, 0x39, 0x30, 0x7A, 0x3B, 0xC3, 0xA4, 0x6F, 0xD9, 0xD3,
0x3A, 0x1D, 0xAE, 0xAE, 0x4F, 0xAE
    },
    {
        0x93, 0xD4, 0x14, 0x3A, 0x4D, 0x56, 0x86, 0x88, 0xF3, 0x4A,
0x3C, 0xA2, 0x4C, 0x45, 0x17, 0x35,
        0x04, 0x05, 0x4A, 0x28, 0x83, 0x69, 0x47, 0x06, 0x37, 0x2C,
0x82, 0x2D, 0xC5, 0xAB, 0x92, 0x09,
        0xC9, 0x93, 0x7A, 0x19, 0x33, 0x3E, 0x47, 0xD3, 0xC9, 0x87,
0xBF, 0xE6, 0xC7, 0xC6, 0x9E, 0x39,
        0x54, 0x09, 0x24, 0xBF, 0xFE, 0x86, 0xAC, 0x51, 0xEC, 0xC5,
0xAA, 0xEE, 0x16, 0x0E, 0xC7, 0xF4
    },
    {
        0x1E, 0xE7, 0x02, 0xBF, 0xD4, 0x0D, 0x7F, 0xA4, 0xD9, 0xA8,
0x51, 0x59, 0x35, 0xC2, 0xAC, 0x36,
        0x2F, 0xC4, 0xA5, 0xD1, 0x2B, 0x8D, 0xD1, 0x69, 0x90, 0x06,
0x9B, 0x92, 0xCB, 0x2B, 0x89, 0xF4,
        0x9A, 0xC4, 0xDB, 0x4D, 0x3B, 0x44, 0xB4, 0x89, 0x1E, 0xDE,
0x36, 0x9C, 0x71, 0xF8, 0xB7, 0x4E,
        0x41, 0x41, 0x6E, 0x0C, 0x02, 0xAA, 0xE7, 0x03, 0xA7, 0xC9,
0x93, 0x4D, 0x42, 0x5B, 0x1F, 0x9B
    },
    {

```

```

        0xDB, 0x5A, 0x23, 0x83, 0x51, 0x44, 0x61, 0x72, 0x60, 0x2A,
0x1F, 0xCB, 0x92, 0xDC, 0x38, 0x0E,
        0x54, 0x9C, 0x07, 0xA6, 0x9A, 0x8A, 0x2B, 0x7B, 0xB1, 0xCE,
0xB2, 0xDB, 0x0B, 0x44, 0x0A, 0x80,
        0x84, 0x09, 0x0D, 0xE0, 0xB7, 0x55, 0xD9, 0x3C, 0x24, 0x42,
0x89, 0x25, 0x1B, 0x3A, 0x7D, 0x3A,
        0xDE, 0x5F, 0x16, 0xEC, 0xD8, 0x9A, 0x4C, 0x94, 0x9B, 0x22,
0x31, 0x16, 0x54, 0x5A, 0x8F, 0x37
    },
    {
        0xED, 0x9C, 0x45, 0x98, 0xFB, 0xC7, 0xB4, 0x74, 0xC3, 0xB6,
0x3B, 0x15, 0xD1, 0xFA, 0x98, 0x36,
        0xF4, 0x52, 0x76, 0x3B, 0x30, 0x6C, 0x1E, 0x7A, 0x4B, 0x33,
0x69, 0xAF, 0x02, 0x67, 0xE7, 0x9F,
        0x03, 0x61, 0x33, 0x1B, 0x8A, 0xE1, 0xFF, 0x1F, 0xDB, 0x78,
0x8A, 0xFF, 0x1C, 0xE7, 0x41, 0x89,
        0xF3, 0xF3, 0xE4, 0xB2, 0x48, 0xE5, 0x2A, 0x38, 0x52, 0x6F,
0x05, 0x80, 0xA6, 0xDE, 0xBE, 0xAB
    },
    {
        0x1B, 0x2D, 0xF3, 0x81, 0xCD, 0xA4, 0xCA, 0x6B, 0x5D, 0xD8,
0x6F, 0xC0, 0x4A, 0x59, 0xA2, 0xDE,
        0x98, 0x6E, 0x47, 0x7D, 0x1D, 0xCD, 0xBA, 0xEF, 0xCA, 0xB9,
0x48, 0xEA, 0xEF, 0x71, 0x1D, 0x8A,
        0x79, 0x66, 0x84, 0x14, 0x21, 0x80, 0x01, 0x20, 0x61, 0x07,
0xAB, 0xEB, 0xBB, 0x6B, 0xFA, 0xD8,
        0x94, 0xFE, 0x5A, 0x63, 0xCD, 0xC6, 0x02, 0x30, 0xFB, 0x89,
0xC8, 0xEF, 0xD0, 0x9E, 0xCD, 0x7B
    },
    {
        0x20, 0xD7, 0x1B, 0xF1, 0x4A, 0x92, 0xBC, 0x48, 0x99, 0x1B,
0xB2, 0xD9, 0xD5, 0x17, 0xF4, 0xFA,
        0x52, 0x28, 0xE1, 0x88, 0xAA, 0xA4, 0x1D, 0xE7, 0x86, 0xCC,
0x91, 0x18, 0x9D, 0xEF, 0x80, 0x5D,
        0x9B, 0x9F, 0x21, 0x30, 0xD4, 0x12, 0x20, 0xF8, 0x77, 0x1D,
0xDF, 0xBC, 0x32, 0x3C, 0xA4, 0xCD,
        0x7A, 0xB1, 0x49, 0x04, 0xB0, 0x80, 0x13, 0xD2, 0xBA, 0x31,
0x16, 0xF1, 0x67, 0xE7, 0x8E, 0x37
    }
};

//Промежуточные результаты
static unsigned char IVp[64], Np[64], Sp[64];

//-----
//Сложение по модулю 2
//-----
static void AddMod2(unsigned char *sum, unsigned char *s1, unsigned char *s2)
{
    int i;

    for(i = 0; i < 64; i++)
    {
        sum[i] = s1[i] ^ s2[i];
    }
}

//-----
//Преобразования L, P, S
//-----
static void LPS(unsigned char *K)

```

```

{
    int i, ii, j;
    unsigned char TMP[64];
    unsigned long int temp[2] = {0, 0};

    //Преобразование S
    for(i = 0; i < 64; i++)
    {
        K[i] = pi[K[i]];
    }
    //Преобразование P
    for(i = 0; i < 64; i++)
    {
        TMP[i] = K[t[i]];
    }
    //Преобразование L
    for(i = 0; i < 8; i++)
    {
        for(ii = 0; ii < 8; ii++)
        {
            for(j = 0; j < 8; j++)
            {
                if((TMP[i * 8 + ii] >> j) & 1)
                {
                    temp[0] ^= A[2 * (63 - (ii * 8 + j))];
                    temp[1] ^= A[2 * (63 - (ii * 8 + j)) + 1];
                }
            }
        }

        //__memcpy(K + (i*8),temp, 8);
        K[i * 8] = *((unsigned char *)&temp[0]);
        K[i * 8 + 1] = *((unsigned char *)&temp[0] + 1);
        K[i * 8 + 2] = *((unsigned char *)&temp[0] + 2);
        K[i * 8 + 3] = *((unsigned char *)&temp[0] + 3);
        K[i * 8 + 4] = *((unsigned char *)&temp[1]);
        K[i * 8 + 5] = *((unsigned char *)&temp[1] + 1);
        K[i * 8 + 6] = *((unsigned char *)&temp[1] + 2);
        K[i * 8 + 7] = *((unsigned char *)&temp[1] + 3);
        temp[0] = 0;
        temp[1] = 0;
    }
}

//-----
//Преобразование E
//-----
static void E(unsigned char *tt, unsigned char *K, unsigned char *m)
{
    int i;

    AddMod2(tt, K, m);
    //Итерации
    for(i = 0; i < 12; i++)
    {
        LPS(tt);
        //Вычисление K
        AddMod2(K, C[i], K);
        LPS(K);
        AddMod2(tt, K, tt);
    }
}

```

```

//-----
//Функция сжатия
//-----
static void gN(unsigned char *h, unsigned char *m, unsigned char *N)
{
    unsigned char tt[64], K[64];

    AddMod2(K, h, N);
    LPS(K);
    E(tt, K, m);
    AddMod2(tt, h, tt);
    AddMod2(h, m, tt);
}

//-----
//Сложение (C = A + B)
//-----
static void Add(unsigned char *C, unsigned char *A, unsigned char *B)
{
    int i, t = 0;

    for(i = 0; i < 64; i++)
    {
        t = A[i] + B[i] + (t >> 8);
        C[i] = t & 0xFF;
    }
}

/*****
//Программная реализация алгоритма хэширования по ГОСТ 34.11-12
//На входе:
//      - размер хэш
//      - указатель на структуру с данными
*****/
unsigned short Hesh341112(unsigned int HeshLen, HeshStruct * Data)
{
    unsigned int i, j = 0;
    unsigned int len;
    unsigned char* h;
    unsigned char* mm;
    unsigned char IV[64], N[64], S[64], m[64];
    unsigned char v512[64] = {

        0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00,

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00,

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00,

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00
    };
    unsigned char v0[64] = {

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00,
    }
}

```

```

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00
};

//Проверка исходных данных
if((HeshLen != HESH_LEN_512) && (HeshLen != HESH_LEN_256)) || (Data ==
NULL_)
{
    return 0;
}
if((Data->Data == NULL_) || (Data->Hesh == NULL_))
{
    return 0;
}
if(Data->STATE == TR_NO)
{
    //Этап 1
    if(HeshLen != HESH_LEN_512)
    {
        j = 0x01;
    }
    for(i = 0; i < 64; i++)
    {
        IV[i] = j;
        N[i] = 0;
        S[i] = 0;
    }
}
else
{
    //Восстановление промежуточного состояния
    for(i = 0; i < 64; i++)
    {
        IV[i] = IVp[i];
        N[i] = Np[i];
        S[i] = Sp[i];
    }
}
h = IV;
//Этап 2
len = Data->DataLen;
while(len >= 64)
{
    mm = Data->Data + Data->DataLen - len;
    gN(h, mm, N);
    Add(N, v512, N);
    Add(S, mm, S);
    len -= 64;
}
//Запись промежуточного состояния
for(i = 0; i < 64; i++)
{
    IVp[i] = IV[i];
    Np[i] = N[i];
    Sp[i] = S[i];
}

```

```

//Этап 3
for(i = 0; i < len; i++)
{
    m[i] = Data->Data[Data->DataLen - len + i];
}
m[i++] = 1;
for(;i < 64; i++)
{
    m[i] = 0;
}
gN(h, m, N);

//Переводим в биты
len *= 8;
v512[0] = len & 0xFF;
v512[1] = len >> 8;
Add(N, v512, N);
Add(S, m, S);
gN(h, N, v0);

gN(h, S, v0);
//Перепысываем полученную хэш
if(HeshLen == HESH_LEN_512)
{
    for(i = 0; i < 64; i++)
    {
        Data->Hesh[i] = h[i];
    }
}
else
{
    for(i = 0; i < 32; i++)
    {
        Data->Hesh[i] = h[32 + i];
    }
}
return 1;
}

```

mem.cpp

```
#include "mem.h"
```

```

void __memcpy (void *Dst, void *Src, int size)
{
    char *dst = (char *)Dst, *src = (char *)Src;
    int i;
    for (i = 0; i < size; dst[i] = src[i], i++);
}

int __memcmp (void *Dst, void *Src, int size)
{
    char *dst = (char *)Dst, *src = (char *)Src;
    int i;

    for (i = 0; i < size; i++)
    {
        if (dst[i] != src[i])
            break;
    }
    if(i == size) return 1;
    else return 0;
}

```

```

void __memset (void *Dst, int val, int size)
{
    char *dst = (char *)Dst;
    int i;
    for (i = 0; i < size; dst[i++] = val);
}

int __strlen(char *buf)
{
    int size=0;
    //определение длины строки
    while (*(buf+size)!=0)
        size++;
    return(size);
}

void __strcpy(char *Dst, char* Src)
{
    int i = 0;
    do{
        Dst[i] = Src[i];
        i++;
    }
    while(Dst[i - 1]);
}

void __strcat(char *Dst, char* Src)
{
    int i, j;
    for(i = 0; Dst[i] != 0; i++);
    j = 0;
    do{
        Dst[i + j] = Src[j];
        j++;
    }
    while(Dst[i + j - 1]);
}

int __strcmp(char *Fst, char* Scnd)
{
    int i = 0;
    while(Fst[i] == Scnd[i]){
        if(Fst[i] == 0) return 0;
        i++;
    }
    return 1;
}

//////////Функции второго
ядра//////////
void __memcpy2 (void *Dst, void *Src, int size)
{
    char *dst = (char *)Dst, *src = (char *)Src;
    int i;
    for (i = 0; i < size; dst[i] = src[i], i++);
}

int __memcmp2 (void *Dst, void *Src, int size)

```

```

{
    char *dst = (char *)Dst, *src = (char *)Src;
    int i;

    for (i = 0; i < size; i++)
    {
        if (dst[i] != src[i])
            break;
    }
    if(i == size) return 1;
    else return 0;
}

void __memset2 (void *Dst, int val, int size)
{
    char *dst = (char *)Dst;
    int i;
    for (i = 0; i < size; dst[i++] = val);
}

int __strlen2(char *buf)
{
    int size=0;
    //определение длины строки
    while(*(buf+size)!=0)
        size++;
    return(size);
}

void __strcpy2(char *Dst, char* Src)
{
    int i = 0;
    do{
        Dst[i] = Src[i];
        i++;
    }
    while(Dst[i - 1]);
}

void __strcat2(char *Dst, char* Src)
{
    int i, j;
    for(i = 0; Dst[i] != 0; i++);
    j = 0;
    do{
        Dst[i + j] = Src[j];
        j++;
    }
    while(Dst[i + j - 1]);
}

int __strcmp2(char *Fst, char* Scnd)
{
    int i = 0;
    while(Fst[i] == Scnd[i]){
        if(Fst[i] == 0) return 0;
        i++;
    }
    return 1;
}

```

```

}

//CP1251toISO866
long WinToDos(unsigned char * text){
    long textSize;
    long i;

    unsigned char symb;
    //char mas[100];

    textSize = __strlen((char*)text);
    if(textSize < 0 || textSize > 100) return 1;

    for( i = 0; i < textSize; ++i){
        symb = *(text + i);
        if((* (text + i) > 191) && (* (text + i) < 224)){ //А - Я
            *(text + i) = symb - 64;
        }
        else if((* (text + i) > 223) && (* (text + i) < 240)){ //а - я
            *(text + i) = symb - 64;
        }
        else if(* (text + i) > 239){ //р - я
            *(text + i) = symb - 16;
        }
        else if(* (text + i) == 168){ //Ё
            *(text + i) = 240;
        }
        else if(* (text + i) == 184){ //ё
            *(text + i) = 241;
        }
        else if(* (text + i) == 185){ //ѐ
            *(text + i) = 252;
        }
    }
    return 0;
}

void toWin(u8 *pText, int numb_symb)
{
    int i;
    for (i = 0; i < numb_symb; i++)
    {
        if((pText[i] > 127) && (pText[i] < 160)){ //А - Я
            pText[i] = pText[i] + 64;
        }
        else if((pText[i] > 159) && (pText[i] < 176)){ //а - я
            pText[i] = pText[i] + 64;
        }
        else if((pText[i] > 223) && (pText[i] < 240)){ //р - я
            pText[i] = pText[i] + 16;
        }
        else if(pText[i] == 240){ //Ё
            pText[i] = 168;
        }
        else if(pText[i] == 241){ //ё
            pText[i] = 184;
        }
        else if(pText[i] == 252){ //ѐ
            pText[i] = 185;
        }
    }
}

```

```

void toDos(u8 *pText, int numb_symb)
{
    int i;
    for (i = 0; i < numb_symb; i++)
    {
        if((pText[i] > 191) && (pText[i] < 224)){ //А - Я
            pText[i] = pText[i] - 64;
        }
        else if((pText[i] > 223) && (pText[i] < 240)){ //а - я
            pText[i] = pText[i] - 64;
        }
        else if(pText[i] > 239){ //р - я
            pText[i] = pText[i] - 16;
        }
        else if(pText[i] == 168){ //Ё
            pText[i] = 240;
        }
        else if(pText[i] == 184){ //ё
            pText[i] = 241;
        }
        else if(pText[i] == 185){ //№
            pText[i] = 252;
        }
        else if(pText[i] > 0 && pText[i] < 128) {
            //pText[i] = pText[i];
        }
        else{
            pText[i] = 254;
        }
    }
}

```

ork.cpp

```

//#include <stdio.h>
//#include <conio.h>
//#include <string.h>
#include "ork.h"
#include "Hesh341112.h"
#include "mem.h"
#include "Algebra.h"
#include "ECP341012.h"
#include "types.h"

#define H2LEN 1
#define H3LEN 1
#define PILEN 32
u8 H2[H2LEN] = {1};
u8 H3[H3LEN] = {2};
u8 tagmass[2048];
u8 tagA[64];

int i;

//---- GostR3410-2001-CryptoPro-A-ParamSet)
static unsigned char q[SIZE] = {0x93, 0xB8, 0x61, 0xB7, 0x09, 0x1B, 0x84,
0x45, 0x00, 0xD1, 0x5A, 0x99, 0x70, 0x10, 0x61, 0x6C,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

const u8 ipad[64] =
{0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,

```

```

0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36,
0x36,0x36,0x36,0x36,0x36,0x36,0x36,0x36};

const u8 opad[64] =
    {0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C,
    0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C, 0x5C};

void conc(u8* dest, u32 len1, u8* src, u32 len2)
{
    for (i = 0; i < len2; i++)
    {
        dest[len1+i] = src[i];
    }
}

void tag(tag_str* tag_str_ptr)
{
    u32 length = 0;
    conc(tagmass, 0, H3, H3LEN); // конк. с h3
    conc(tagmass, H3LEN, tag_str_ptr->Ka, PILEN); // конк. с PI(Ka)
    conc(tagmass, H3LEN + PILEN, tag_str_ptr->Kb, PILEN); // конк. с PI(Kb)
    conc(tagmass, H3LEN + PILEN + PILEN, tag_str_ptr->Ida, tag_str_ptr->
>IdaLen); // конк. с Ida
    conc(tagmass, H3LEN + PILEN + PILEN + tag_str_ptr->IdaLen, tag_str_ptr->
>Idb, tag_str_ptr->IdbLen); // конк. с Idb
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr->
>IdaLen, tag_str_ptr->UKM_A, tag_str_ptr->UKM_A_Len);
    // конк. с UKM_A
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr->
>IdaLen + tag_str_ptr->UKM_A_Len, tag_str_ptr->UKM_B, tag_str_ptr->UKM_B_Len);
    // конк. с UKM_B

    length = H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr->IdaLen
+ tag_str_ptr->UKM_A_Len + tag_str_ptr->UKM_B_Len;
    // расчет длины
    HMAC(tag_str_ptr->Mab, tagmass, 32, length, tag_str_ptr->Mab);
    // вычисления кода аутентификации сообщения
    __memcpy (tag_str_ptr->tag, tag_str_ptr->Mab, 32); //копирование 32-х
значений tag_str_ptr->Mab в массив tag_str_ptr->tag
}

void tagB(tag_str* tag_str_ptr)
{
    u32 length = 0;
    conc(tagmass, 0, H2, H2LEN); // конк. с h2
    conc(tagmass, H2LEN, tag_str_ptr->Kb, PILEN); // конк. с PI(Kb)
    conc(tagmass, H2LEN + PILEN, tag_str_ptr->Ka, PILEN); // конк. с PI(Ka)
    conc(tagmass, H2LEN + PILEN + PILEN, tag_str_ptr->Idb, tag_str_ptr->
>IdbLen); // конк. с Idb
    conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen, tag_str_ptr->
>Ida, tag_str_ptr->IdaLen); // конк. с Ida

```

```

        conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr-
>IdaLen, tag_str_ptr->UKM_B, tag_str_ptr->UKM_B_Len); // конк. с UKM_B
        conc(tagmass, H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr-
>IdaLen + tag_str_ptr->UKM_B_Len, tag_str_ptr->UKM_A, tag_str_ptr->UKM_A_Len);
// конк. с UKM_A

        length = H2LEN + PILEN + PILEN + tag_str_ptr->IdbLen + tag_str_ptr->IdaLen
+ tag_str_ptr->UKM_B_Len + tag_str_ptr->UKM_A_Len; // расчет длины
        HMAC(tag_str_ptr->Mab, tagmass, 32, length, tag_str_ptr->Mab);
// вычисления кода аутентификации сообщения
        __memcpy (tag_str_ptr->tag, tag_str_ptr->Mab, 32); //копирование 32-х
значений tag_str_ptr->Mab в массив tag_str_ptr->tag
    }
    static u8 KZ[2048];
    void HMAC(u8* K, u8* T, u32 len_K, u32 len_T, u8* Res)
    {
        HeshStruct data;
        //u8 KZ[2048];
        u8 Hesh_buff[32];

        __memcpy (KZ, K, len_K);
        __memset (KZ + len_K, 0, 64 - len_K);

        for (i = 0; i < 64; i++)
        {
            //KZ[i] = KZ[i] ^ ipad[i];
            *(KZ + i) = *(KZ + i) ^ *(ipad + i);
        }
        conc(KZ, 64, T, len_T);
        data.Data = KZ;
        data.DataLen = 64 + len_T;
        data.Hesh = Hesh_buff;
        data.STATE = 0;

        //_____
        Hesh341112(HESH_LEN_256, &data);

        __memcpy (KZ, K, len_K);
        __memset (KZ + len_K, 0, 64 - len_K);

        for (i = 0; i < 64; i++)
        {
            //KZ[i] = KZ[i] ^ opad[i];
            *(KZ + i) = *(KZ + i) ^ *(opad + i);
        }

        conc(KZ, 64, Hesh_buff, 32); //conc(KZ, 64, Hesh_buff, 256);
        data.Data = KZ;
        data.DataLen = 64 + 32; //len_T
        data.Hesh = Hesh_buff;
        data.STATE = 0;

        //_____
        Hesh341112(HESH_LEN_256, &data);
        __memcpy (Res, Hesh_buff, 32);
    }

    unsigned int VKO_GOST341012(VKOStruct * Data)
    {
        residue rez,aa,qq,ukm_;
        int i;

```

```

ECP_affine U;
unsigned char Buf[128];
HeshStruct HeshData;
    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)aa + i) = Data->Skey[i];
    }
aa[N - 1] = 0;

//PµPµCßPµPµPµCíC<PIP°PµPj PsC,PeCßC<C,C<PM PeP»CßC‡ CíC,PsCßPsPSC< B
for(i = 0; i < (N - 1) * 2; i++)
{
    *((unsigned char *)ECUx + i) = Data->Okey[i];
}
ECUx[N - 1] = 0;
for(i = 0; i < (N - 1) * 2; i++)
{
    *((unsigned char *)ECUy + i) = Data->Okey[i + ((N - 1) * 2)];
}
ECUy[N - 1] = 0;
for(i = 0; i < (N - 1) * 2; i++)
{
    *((unsigned char *)qq + i) = Data->q[i];
}
qq[N - 1] = 0;

    for(i = 0; i < (N - 1) * 2; i++)
    {
        *((unsigned char *)ukm_ + i) = Data->ukm[i];
    }
ukm_[N - 1] = 0;

Mul(rez, aa, ukm_, qq); //(UKM*CKeyA) mod q
Rcopy(U.x, ECUx);
Rcopy(U.y, ECUy);
ECPpow(&U, &U, rez);
for(i = 0; i < (N - 1) * 2; i++)
{
    Buf[i] = *((unsigned char *)U.x + i);
}
for(i = 0; i < (N - 1) * 2; i++)
{
    Buf[i + ((N - 1) * 2)] = *((unsigned char *)U.y + i);
}

HeshData.Data = Buf;
HeshData.DataLen = (N - 1) * 4;
HeshData.Hesh = Data->Kek_Vko;
HeshData.STATE = 0;
if(!Hesh341112(HESH_LEN_256, &HeshData))
{
    return 0;
}

return 1;
}

void powEllip(u8 * open_key, u8 * sec_key)
{
    EcpStruct pow_Ellip; //Основная структура
    pow_Ellip.Pr = ECP_FORM_OKEY; //признак (формирование/проверка)

```

```

    pow_Ellip.Skey = sec_key; //секретный ключ (используется при формировании)
    pow_Ellip.Okey = open_key; //открытый ключ (используется при проверке)
    pow_Ellip.K = 0; //случайное число (используется при формировании)
    pow_Ellip.Hesh = 0; //хэш
    pow_Ellip.Ecp = 0; //значение ЭЦП (после формирования или для проверки)
    Ecp341012(&pow_Ellip); // формирования открытого ключа ЭП
}
//u8 mass_Tba_str[4096];
void Tba_func(Tba_str * str_Tba)
{
    u8 mass_Tba_str[4096];
    VKOStruct VKO_str;
    HeshStruct Tba_hesh;
    u8 UKM_buf[32];

    __memset(UKM_buf, 0, 32); // обнуление буфера UKM_buf
    __memcpy(UKM_buf, str_Tba->UKM_A, str_Tba->UKM_A_Len); //копирование
    массива str_Tba->UKM_A в массив UKM_buf
    VKO_str.Skey = str_Tba->sa;
    VKO_str.Okey = str_Tba->Kb;
    VKO_str.q = q;
    VKO_str.ukm = UKM_buf;
    VKO_str.ukm_len = 32;
    VKO_str.Kek_Vko = mass_Tba_str;
    VKO_GOST341012(&VKO_str); // вычисление точки эл. кривой QBA

    __memset(UKM_buf, 0, 32); // обнуление буфера UKM_buf
    __memcpy(UKM_buf, str_Tba->UKM_B, str_Tba->UKM_B_Len); //копирование
    содержимого массива str_Tba->UKM_B в массив UKM_buf
    VKO_str.Skey = str_Tba->ka;
    VKO_str.Okey = str_Tba->Sb;
    VKO_str.q = q;
    VKO_str.ukm = UKM_buf;
    VKO_str.ukm_len = 32;
    VKO_str.Kek_Vko = mass_Tba_str + 32;
    VKO_GOST341012(&VKO_str); // // вычисление точки эл. кривой Rba
    conc(mass_Tba_str, 64, str_Tba->Ida, str_Tba->IdaLen); //conc from ID_A

    conc(mass_Tba_str, 64 + str_Tba->IdaLen, str_Tba->Idb, str_Tba->IdbLen);
    //conc from ID_B

    conc(mass_Tba_str, 64 + str_Tba->IdaLen + str_Tba->IdbLen, str_Tba->CertA,
    str_Tba->CertALen); //conc from CertALen

    conc(mass_Tba_str, 64 + str_Tba->IdaLen + str_Tba->IdbLen + str_Tba-
    >CertALen, str_Tba->CertB, str_Tba->CertBLen); //conc from CertBLen
    Tba_hesh.Data = mass_Tba_str;
    Tba_hesh.Hesh = str_Tba->result_Tba;
    Tba_hesh.DataLen = 64 + str_Tba->IdaLen + str_Tba->IdbLen + str_Tba-
    >CertALen + str_Tba->CertBLen;
    Tba_hesh.STATE = 0;
    Hesh341112(HESH_LEN_512, &Tba_hesh); //вычисление хэш-суммы в соответствии
    с ГОСТ 34.10 - 2012
}

u8 field_30[30];

int chech_valid(u8 *certificate, u8 *Ecp_key)
{
    u8 ECP_buf[64], Hesh_cert[32];
    HeshStruct HeshCert;
    EcpStruct ECP_Data;

```

```

    int i, uk, uk_TBS, size_sql, size_sq2, size_cert, size_TBS;
    //unsigned short size_TBS, size_field, size_SubjectPublicKeyInfo;
    //int count;

    uk = 0;

    if(certificate[uk] != 0x30) return 0; //не обнаружен sql -> ошибка

    uk += 1;
    if((certificate[uk] & 0x80) == 0x80)
    {
        uk += 1;
        size_sql = (certificate[uk] << 8) & 0xff00; //размер
        содержимого 1 sequence-сертификата
        uk += 1;
        size_sql ^= certificate[uk];
        uk += 1;
        size_cert = size_sql + 4; // размер всего сертификата

    } //if(certificate[uk] & 0x80 == 0x80)
    else
    {
        size_sql = certificate[uk]; //размер содержимого 1 sequence-
        сертификата
        uk += 1;
        size_cert = size_sql + 2; // размер всего сертификата

    }

    if(certificate[uk] != 0x30) return 0; //не обнаружен sq2 -> ошибка
    uk_TBS = uk;
    uk += 1;
    if((certificate[uk] & 0x80) == 0x80)
    {
        uk += 1;
        size_sq2 = (certificate[uk] << 8) & 0xff00; //размер
        содержимого 2 sequence-сертификата
        uk += 1;
        size_sq2 ^= certificate[uk];
        uk += 1;
        size_TBS = size_sq2 + 4; // размер sq2

    } //if(certificate[uk] & 0x80 == 0x80)
    else
    {
        size_sq2 = certificate[uk]; //размер содержимого 2 sequence-
        сертификата
        uk += 1;
        size_TBS = size_sq2 + 2; // размер sq2

    }

    for (i = 0; i < 64; i++)
    {
        ECP_buf[i] = certificate[size_cert - i - 1];
    }

    HeshCert.Data = &certificate[uk_TBS];
    HeshCert.DataLen = size_TBS;
    HeshCert.Hesh = Hesh_cert;
    HeshCert.STATE = 0;
    Hesh341112(HESH_LEN_256, &HeshCert);

```

```

ECP_Data.Pr = ECP_CONTROL;
ECP_Data.Hesh = Hesh_cert;
ECP_Data.Skey = 0;
ECP_Data.Okey = Ecp_key;
ECP_Data.Ecp = ECP_buf;
ECP_Data.K = 0;

    if ( Ecp341012(&ECP_Data) == 0 ) return 0; // если проверка ЭП вернула
значение 0, то "ошибка!"

    return 1;
}

int unbox_Cert(u8 *certificate, u8 *otkr_key)
{
    int uk, uk_field;
    unsigned short size_TBS, size_field, size_SubjectPublicKeyInfo;
    int count;

    uk = 0;

    if(certificate[uk] == 0x30)
    {
        uk +=1;
        if((certificate[uk] & 0x80) == 0x80)
        {
            uk += 1;
            size_TBS = (certificate[uk] << 8) & 0xff00; //размер TBS-
сертификата
            uk += 1;
            size_TBS ^= certificate[uk];
            uk += 1;

        }//if(certificate[uk] & 0x80 == 0x80)
        else
        {
            size_TBS = certificate[uk]; //размер TBS-сертификата
            uk += 1;

        }
    }//if(certificate[0] == 0x30)
    else return 0;

    //sequens второго уровня
    if(certificate[uk] == 0x30)
    {
        uk +=1;
        if((certificate[uk] & 0x80) == 0x80)
        {
            uk += 1;
            size_field = (certificate[uk] << 8) & 0xff00;
            uk += 1;
            size_field ^= certificate[uk];
            uk += 1;
        }//if(certificate[uk] & 0x80 == 0x80)
        else
        {
            size_field = certificate[uk];
            uk += 1;

        }
    }//if(certificate[0] == 0x30)

```

```

count = 1;
while(uk < size_TBS)
{
    if(certificate[uk] == 0x30)
    {
        uk +=1;
        if((certificate[uk] & 0x80) == 0x80)
        {
            uk += 1;
            size_field = (certificate[uk] << 8) & 0xff00;
            uk += 1;
            size_field ^= certificate[uk];
            uk += 1;
        }//if(certificate[uk] & 0x80 == 0x80)
        else
        {
            size_field = certificate[uk];
            uk += 1;
        }
        __memcpy(field_30, certificate + uk, size_field);
        uk += size_field;

        //поле Subject
        if(count == 6)
        {
            uk_field = 0;
            // поиск OBJECT IDENTIFIER
            while(field_30[uk_field] != 0x06)
            {
                size_field = field_30[uk_field + 1];
                __memcpy(field_30, field_30 + uk_field + 2,

size_field);

            }//while(field_30[uk_field] != 0x0c)

            uk_field = 0;
            size_SubjectPublicKeyInfo = size_field;
            while(uk_field < size_SubjectPublicKeyInfo)
            {
                //поиск строковых данных
                if(field_30[uk_field] == 0x0c)
                {
                    uk_field +=1;
                    if((field_30[uk_field] & 0x80) == 0x80)
                    {
                        uk_field += 1;
                        size_field = (field_30[uk_field] << 8)

& 0xff00;

                        uk_field += 1;
                        size_field ^= field_30[uk_field];
                        uk_field += 1;
                    }//if(field_30[uk_field] & 0x80 == 0x80)
                    else
                    {
                        size_field = field_30[uk_field];
                        uk_field += 1;
                    }
                    //__memcpy(ID, field_30 + uk_field,

size_field);

                    break;

                }//if(field_30[uk_field] == 0x0c)

```

```

else
{
    uk_field +=1;
    if((field_30[uk_field] & 0x80) == 0x80)
    {
        uk_field += 1;
        size_field = (field_30[uk_field] << 8)

& 0xff00;

        uk_field += 1;
        size_field ^= field_30[uk_field];
        uk_field += 1;
    }//if(field_30[uk_field] & 0x80 == 0x80)
    else
    {
        size_field = field_30[uk_field];
        uk_field += 1;
    }
    uk_field += size_field;
} //else
} //while(uk_field < size_field)
}

//поле subjectPublicKeyInfo SubjectPublicKeyInfo в
TBSCertificate (RFC 3280)
if(count == 7)
{
    uk_field = 0;
    size_SubjectPublicKeyInfo = size_field;
    while(uk_field < size_SubjectPublicKeyInfo)
    {
        //поиск OCTET_STRING
        if(field_30[uk_field] == 0x04)
        {
            uk_field +=1;
            if((field_30[uk_field] & 0x80) == 0x80)
            {
                uk_field += 1;
                size_field = (field_30[uk_field] << 8)

& 0xff00;

                uk_field += 1;
                size_field ^= field_30[uk_field];
                uk_field += 1;
            }//if(field_30[uk_field] & 0x80 == 0x80)
            else
            {
                size_field = field_30[uk_field];
                uk_field += 1;
            }
            __memcpy(otkr_key, field_30 + uk_field,

size_field);

            break;

        } //if(field_30[uk_field] == 0x04)
        else if(field_30[uk_field] == 0x03) //поиск BIT
        STRING
        {
            uk_field +=1;
            if((field_30[uk_field] & 0x80) == 0x80)
            {
                uk_field += 1;
                size_field = (field_30[uk_field] << 8)

& 0xff00;

                uk_field += 1;

```

```

        size_field ^= field_30[uk_field];
        uk_field += 1;
    } //if(field_30[uk_field] & 0x80 == 0x80)
    else
    {
        size_field = field_30[uk_field];
        uk_field += 1;
    }
    if(field_30[uk_field] == 0) uk_field += 1;

    //поиск OCTET_STRING
    if(field_30[uk_field] == 0x04)
    {
        uk_field +=1;
        if((field_30[uk_field] & 0x80) == 0x80)
        {
            uk_field += 1;
            size_field = (field_30[uk_field]

<< 8) & 0xff00;

            uk_field += 1;
            size_field ^= field_30[uk_field];
            uk_field += 1;
        } //if(field_30[uk_field] & 0x80 ==

0x80)

        else
        {
            size_field = field_30[uk_field];

            uk_field += 1;
        }
        __memcpy(otkr_key, field_30 + uk_field,

size_field);

        break;
    } //if(field_30[uk_field] == 0x04)
    } //else if(field_30[uk_field] == 0x03)
    else
    {
        uk_field +=1;
        if((field_30[uk_field] & 0x80) == 0x80)
        {
            uk_field += 1;
            size_field = (field_30[uk_field] << 8)

& 0xff00;

            uk_field += 1;
            size_field ^= field_30[uk_field];
            uk_field += 1;
        } //if(field_30[uk_field] & 0x80 == 0x80)
        else
        {
            size_field = field_30[uk_field];
            uk_field += 1;
        }
        uk_field += size_field;
    } //else
    } //while(uk_field < size_field)
    } //if(count == 7)
    } //if(certificate[uk] == 0x30)
    else
    {
        uk +=1;
        if((certificate[uk] & 0x80) == 0x80)
        {
            uk += 1;

```

```

        size_field = (certificate[uk] << 8) & 0xff00;
        uk += 1;
        size_field ^= certificate[uk];
        uk += 1;
    } //if(certificate[uk] & 0x80 == 0x80)
    else
    {
        size_field = certificate[uk];
        uk += 1;
    }
    uk += size_field;
} //else
count ++;
} //while(uk < size_TBS)

return 1;
}

void orkA(u8 * ka, u8 * Ka) // gener open key - > abon B
{
    powEllip(Ka, ka); //формирование открытого ключа сторона A
}

static u8 Tba[64];
static u8 Mab[32];
static u8 TAGb_res[32];
int orkB(orkB_str * p_orkB_str)
{
    u8 Tba[64]; //массив Tba
    u8 Mab[32]; //массив Mba
    Tba_str struct_Tba_str;
    tag_str tag_str_orkb;

    if (!Chech_valid(p_orkB_str->Certb, p_orkB_str->key_ECP_Cert)) return 0;
    //проверка валидности сертификата
    if (!unbox_Cert(p_orkB_str->Certb, buff_Sb)) return 0; //извлечение ключа
    Sb из сертификата

    struct_Tba_str.Kb = p_orkB_str->Kb;
    struct_Tba_str.sa = p_orkB_str->sa;
    struct_Tba_str.Sb = buff_Sb;

    struct_Tba_str.ka = p_orkB_str->ka;

    struct_Tba_str.Ida = p_orkB_str->Ida;
    struct_Tba_str.Idb = p_orkB_str->Idb;
    struct_Tba_str.IdaLen = __strlen((char*)p_orkB_str->Ida);
    struct_Tba_str.IdbLen = __strlen((char*)p_orkB_str->Idb);
    struct_Tba_str.CertA = p_orkB_str->Certa;
    struct_Tba_str.CertB = p_orkB_str->Certb;

    struct_Tba_str.CertALen = p_orkB_str->Certa_len;
    struct_Tba_str.CertBLen = p_orkB_str->Certb_len;
    struct_Tba_str.UKM_A = p_orkB_str->UKM_A;
    struct_Tba_str.UKM_B = p_orkB_str->UKM_B;
    struct_Tba_str.UKM_A_Len = p_orkB_str->UKM_A_len;
    struct_Tba_str.UKM_B_Len = p_orkB_str->UKM_B_len;
    struct_Tba_str.result_Tba = Tba;

    Tba_func(&struct_Tba_str); // создание последовательности ТВА
    __memcpy(p_orkB_str->Kab, Tba, 32); //копирование первых 32-х значений Tba
    в массив Kab

```

```

__memcpy(Mab, &Tba[32], 32); //копирование последних 32-х значений Tba в
массив Mab

tag_str_orkb.Ida = p_orkB_str->Ida;
tag_str_orkb.IdaLen = __strlen((char*)p_orkB_str->Ida); //расчет длины
строки
tag_str_orkb.Idb = p_orkB_str->Idb;
tag_str_orkb.IdbLen = __strlen((char*)p_orkB_str->Idb); //расчет длины
строки
tag_str_orkb.Ka = p_orkB_str->Ka;
tag_str_orkb.Kb = p_orkB_str->Kb;
tag_str_orkb.Mab = Mab;
tag_str_orkb.tag = p_orkB_str->TAGb_res;
tag_str_orkb.UKM_B = p_orkB_str->UKM_B;
tag_str_orkb.UKM_B_Len = p_orkB_str->UKM_B_len;
tag_str_orkb.UKM_A = p_orkB_str->UKM_A;
tag_str_orkb.UKM_A_Len = p_orkB_str->UKM_A_len;
tagB(&tag_str_orkb); // вычисление метки соответствия ключа встречного
абонента
if(!__memcmp (TAGb_res, p_orkB_str->TAGb, 32)) return 1; ); //Сравнивание
собственной метки соответствия с меткой соответствия ключа встречного абонента

__memcpy(Mab, &Tba[32], 32); //копирование последних 32-х значений Tba в
массив Mab

tag_str_orkb.Ida = p_orkB_str->Ida;
tag_str_orkb.IdaLen = __strlen((char*)p_orkB_str->Ida); //определение
длины строки

tag_str_orkb.Idb = p_orkB_str->Idb;
tag_str_orkb.IdbLen = __strlen((char*)p_orkB_str->Idb); //определение
длины строки

tag_str_orkb.Ka = p_orkB_str->Ka;
tag_str_orkb.Kb = p_orkB_str->Kb;
tag_str_orkb.Mab = Mab;
tag_str_orkb.tag = p_orkB_str->TAGa;
tag_str_orkb.UKM_B = p_orkB_str->UKM_B;
tag_str_orkb.UKM_B_Len = p_orkB_str->UKM_B_len;
tag_str_orkb.UKM_A = p_orkB_str->UKM_A;
tag_str_orkb.UKM_A_Len = p_orkB_str->UKM_A_len;

tag(&tag_str_orkb); // вычисление собственной метки соответствия ключа

return 1;
}

```

Algebra.h

```

#ifndef _ALGEBRA_H_
#define _ALGEBRA_H_

/*****
//Переменные и массивы
*****/

/*****
//Константы
*****/
#define n          256
#define N          17
#define SIZE      (2 * (N - 1))           //размерность в байтах

```

```

/*****
//Типы данных
*****/
typedef unsigned short int residue[N];
typedef struct {residue x,y;} ECP_affine;
typedef struct {residue X,Y,Z;} ECP_projective;

/*****
//Внешние ссылки
*****/
extern residue ECp;
extern residue ECp_2;
extern residue ECae;
extern residue ECbe;
extern residue ECq;
extern residue ECq_2;
extern residue ECUx;
extern residue ECUy;
extern residue ECe_p;
extern residue ECe_q;
extern residue ECe2_p;
extern residue ECe2_q;
extern residue ECone;
extern residue ECzr;
extern unsigned short int ECp0inv;
extern unsigned short int ECq0inv;

/*****
//Прототипы функций
*****/
void ECPpow(ECP_affine *Ps, ECP_affine *P, residue s);
void affine2projective(ECP_projective *Pp, ECP_affine *Pa);
void projective2affine(ECP_affine *Pa, ECP_projective *Pp);
void ECPmul(ECP_projective *P3, ECP_projective *P1, ECP_affine *P2);
void ECPy(ECP_affine *P, residue x);

void Rcopy(residue dest, residue src);
void Radd(residue S, residue A, residue B, residue P);
void Rsub(residue S, residue A, residue B, residue P);
void Rmul(residue S, residue A, residue B, residue P);
void Rpow_pdiv4(residue S, residue A, residue P);
void Rstep(residue S, residue A, residue B, residue P);
void Rinv(residue S, residue A, residue P);
void Rsqrt(residue S, residue A);
unsigned short Findz(unsigned short a);
void Finde(residue RES, residue P);
void Mod(residue B, residue A, residue P);
void Mul(residue S, residue A, residue B, residue P);
void Mul_(residue S, residue A, residue B);
/*****
//Макросы
*****/
//Переход из поля Галуа в поле Монтгомери
#define GF2MF(S, A, E, P) Rmul(S, A, E, P)
//Переход из поля Монтгомери в поле Галуа
#define MF2GF(S, A, P) Rmul(S, A, ECone, P)
#endif

Ecp341012.h
#ifdef _ECP341012_H_
#define _ECP341012_H_

#include "Algebra.h"

```

```

/*****/
//Переменные и массивы
/*****/

/*****/
//Константы
/*****/
#define ECP_FORM          0                //формирование ЭЦП
#define ECP_CONTROL      1                //проверка ЭЦП
#define ECP_FORM_OKEY    2                //формирование
открытого ключа проверки ЭЦП

//#define      NULL  0
/*****/
//Структура параметров ЭЦП
/*****/
typedef struct
{
    unsigned char *P;                //модуль
    эллиптической кривой
    unsigned char *A;                //коэффициент
    эллиптической кривой
    unsigned char *B;                //коэффициент
    эллиптической кривой
    unsigned char *Q;                //порядок циклической
    подгруппы группы точек эллиптической кривой
    unsigned char *Px;                //x -
    координата точки эллиптической кривой
    unsigned char *Py;                //y -
    координата точки эллиптической кривой
} InitEcpStruct;

/*****/
//Основная структура
/*****/
typedef struct
{
    unsigned short Pr;                //признак
    (формирование/проверка)
    unsigned char *Hesh;                //хэш
    unsigned char *Skey;                //секретный ключ
    (используется при формировании)
    unsigned char *Okey;                //открытый ключ
    (используется при проверке)
    unsigned char *Ecp;                //значение ЭЦП
    (после формирования или для проверки)
    unsigned char *K;                //случайное число
    (используется при формировании)
} EcpStruct;

typedef struct
{
    unsigned char *Skey;                //закрытый к.
    передающей стороны
    unsigned char *Okey;                //открытый к.
    принятый от противоположной стороны (точка кривой (X,Y))
    unsigned char *q;                //порядок циклической
    подгруппы группы точек эллиптической кривой
    unsigned char *ukm;                //случайное
    число
    unsigned int   ukm_len;
    unsigned char *Kek_Vko;                //к.обмена

```

```

} VKOStruct;
/*****
//Прототипы функций
*****/
extern unsigned short InitEcp341012(InitEcpStruct * Data);
extern unsigned short Ecp341012(EcpStruct * Data);
extern unsigned short IsNull(unsigned char *digit);
extern unsigned int VKO_GOST341012(VKOStruct * Data);
#endif
Hesh341112.h
#ifndef _HESH341112_H_
#define _HESH341112_H_

/*****
//Переменные и массивы
*****/
#define NULL_ 0
/*****
//Константы
*****/

#define HESH_LEN_512 0x00 //размер хэш 512 бит
#define HESH_LEN_256 0x01 //размер хэш 256 бит

/*****
//Основная структура
*****/
typedef struct
{
    unsigned char * Data; //исходные данные
    unsigned char * Hesh; //хэш на исходные данные
    unsigned long DataLen; //размер исходных данных в
байтах
    unsigned short STATE; //работа с промежуточным
состоянием
} HeshStruct;

extern unsigned short Hesh341112(unsigned int HeshLen, HeshStruct * Data);
#endif

mem.h
#ifndef _MEM_H
#define _MEM_H

#include "types.h"

void __memcpy (void *Dst, void *Src, int size);

int __memcmp (void *Dst, void *Src, int size);

void __memset (void *Dst, int val, int size);

int __strlen(char *buf);

void __strcpy(char *Dst, char* Src);

void __strcat(char *Dst, char* Src);

int __strcmp(char *Fst, char* Scnd);

```

```

void __memcpy2 (void *Dst, void *Src, int size);
int __memcmp2 (void *Dst, void *Src, int size);
void __memset2 (void *Dst, int val, int size);
int __strlen2(char *buf);
void __strcpy2(char *Dst, char* Src);
void __strcat2(char *Dst, char* Src);
int __strcmp2(char *Fst, char* Scnd);

long WinToDos(unsigned char * text);

void toWin(u8 *pText, int numb_symb);
void toDos(u8 *pText, int numb_symb);

#endif
ork.h
#include "types.h"

typedef struct{
    u8* Ka;
    u8* Ida;
    u8* Kb;
    u8* Idb;
    u8* Mab;
    u8* tag;
    u32 IdaLen;
    u32 IdbLen;
    u8* UKM_A;
    u8* UKM_B;
    u32 UKM_A_Len;
    u32 UKM_B_Len;
} tag_str;

typedef struct{
    u8* Kb;
    u8* sa;
    u8* Sb;
    u8* ka;
    //
    u8* Ida;
    u8* Idb;
    u32 IdaLen;
    u32 IdbLen;
    u8* CertA;
    u8* CertB;
    u32 CertALen;
    u32 CertBLen;
    u8* UKM_A;
    u8* UKM_B;
    u32 UKM_A_Len;
    u32 UKM_B_Len;
    u8 * result_Tba;
} Tba_str;
typedef struct{

    u8* sa;
    u8* Sa;
    u8* ka;

```

```

    u8* Ida;
    u8* Idb;
    u32 Ida_len;
    u32 Idb_len;
    u8* Certa;
    u8* Certb;
    u32 Certa_len;
    u32 Certb_len;
    u8* Ka;
    u8* Kb;
    u8* TAGb;
    u8* Kab;
    u8* TAGa;
    u8* UKM_A;
    u8* UKM_B;
    u32 UKM_A_len;
    u32 UKM_B_len;
    u8* key_ECP_Cert;
} orkB_str;

void orkA(u8 * k_A, u8 * K_A);
int orkB(orkB_str * p_orkB_str);

void conc(u8* mass1, u32 len1, u8* mass2, u32 len2);
void tag(tag_str* tag_str_ptr);
void HMAC(u8* K, u8* T, u32 len_K, u32 len_T, u8* Res);

void tagB(tag_str* tag_str_ptr);
void powEllip(u8 * open_key, u8 * sec_key);

int Analysis_certificate(u8 *certificate, u8 *otkr_key, u16 *size_otkr, u8 *ID);

types.h
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;

```