Задача

В документе представлено описание Системы, работающей по протоколу OIDC (см. раздел 1), обеспечивающему делегированную аутентификацию и идентификацию пользователей Клиентами посредством Сервера авторизации, и реализация ее компонент (см. раздел 2).

В рамках решения задачи нужно проанализировать указанный протокол на предмет обеспечения им необходимых свойств безопасности и его реализацию в компонентах, в том числе, в части корректности вызовов криптографических функций, реализуемых используемыми СКЗИ.

По результатам анализа необходимо подготовить отчет, включающий в себя в том числе:

- перечень замечаний к спецификации протокола OIDC и его реализации. Необходимо привести описание уязвимости в протоколе и описать некорректную реализацию механизма или вызова / отсутствие необходимой проверки, приводящее к наличию уязвимости;
- соответствующее обоснование замечания. Желательно привести описание возможной атаки на протокол, указав при этом реализуемую угрозу и последовательность действий, проводимых нарушителем (плюсом будет сопровождение картинками);
- соответствующие рекомендации по устранению обнаруженной уязвимости.

Важно! При анализе необходимо опираться на предположения, приведенные в разделе 1.2, а также модель нарушителя, указанную в разделе 0.

В случае отсутствия замечаний и возможных атак, это необходимо обосновать: например, привести описание атаки при отсутствии какого-либо из существующих в протоколе параметров/механизмов и пояснить, как этот параметр/механизм защищает от нее.

Краткий пример анализа протокола (не относится к заданию и протоколу OIDC).
Описание протокола: Клиент аутентифицируется на Сервере с помощью электронной подписи, вычисляемой от идентификатора Клиента и challenge, полученного от Сервера. Предварительно на Сервере зарегистрировано соответствие идентификатора и ключа проверки электронной подписи.
Модель нарушителя. Возможности: нарушитель может являться как Клиентом, так и Сервером; нарушитель в канале отсутствует (т.е не может читать и изменять данные в канале). Актуальной является угроза ложной аутентификации Клиента на Сервере.
Описание атаки: нарушитель параллельно устанавливает два соединения: с атакуемым Клиентом и Сервером. Нарушитель сначала получает challenge от Сервера, направляет его Клиенту и получает от Клиента электронную подпись, вычисленную от его идентификатора и данного challenge. Нарушитель предъявляет данное значение на Сервер и таким образом аутентифицируется от имени легитимного Клиента (реализует угрозу ложной аутентификации).
Мера защиты: необходимо в набор данных, от которых выисляется электронная подпись, включать также и идентификатор Сервера, который известен Клиенту в начале взаимодействия. Т.к. Клиент знает, с каким Сервером он изначально устанавливает соединение, в ЭП будет включен идентификатор нарушителя, и такую ЭП нарушитель не сможет предъявить на другой легитимный Сервер.

Дополнительные вопросы:

1) Приведите примеры того, как на практике может осуществляться навязывание ссылки пользователю.
2) Поясните актуальность угрозы «Некорректная идентификация пользователя». Достаточно привести практический пример использования реализации данной угрозы, поясняющий какую выгоду может получить нарушитель.
3) Какими способами еще может аутентифицироваться Клиент на Сервере авторизации? Необходимо также пояснить какие при этом данные должны быть зарегистрированы на Сервере авторизации.
4) Какие можно дать рекомендации к формированию ответа 11) Клиентом (см. раздел 1.1)?

1 Описание Системы

Объектом анализа является протокол OIDC, который построен на базе семейства протоколов авторизации OAuth 2.0/2.1[1] и семейства протоколов аутентификации OpenID Connect[2]. Исследуемый протокол OIDC используется в Системе для аутентификации пользователей Клиентами посредством Сервера авторизации.

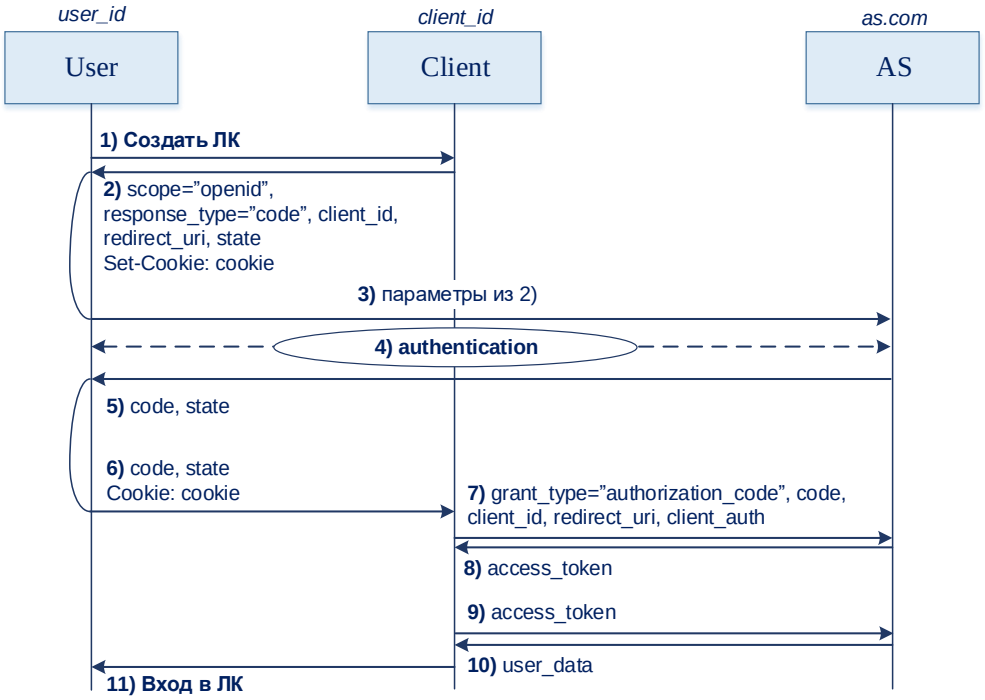
В Системе определены следующие роли участников протокола:

- Сервер авторизации (Authorization Server, AS, Сервер) – аутентифицирует пользователя и предоставляет его идентификационные данные Клиенту. В протоколе OAuth также выделяют роль Сервера ресурсов, который хранит данные пользователей. В данном случае Сервер авторизации включает в себя функции Сервера ресурсов.
- Клиент (Client) – web-приложение, целью которого является делегированная идентификация и аутентификация пользователя с помощью Сервера авторизации. На основе полученных идентификационных данных Клиент создает личный кабинет пользователя.
- Пользователь (User) – имеет учетную запись и идентификационные данные на Сервере авторизации. В результате выполнения протокола получает доступ в личный кабинет на сайте Клиента. Пользователь осуществляет взаимодействие с Клиентом и Сервером авторизации посредством браузера.

Система предполагает наличие единственного Сервера авторизации, многих Клиентов и многих пользователей.

1.1 Описание протокола OIDC

На рисунке ниже представлено схематичное описание протокола OIDC.



1) Пользователь заходит на сайт Клиента с выбирает услугу «создать личный кабинет».
2) Клиент перенаправляет браузер пользователя на сайт Сервера авторизации с параметрами:

[1] https://datatracker.ietf.org/doc/html/rfc6749, https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-01

[2] https://openid.net/developers/specs/

a. scope = “openid” – означает, что пользователь будет проходить аутентификацию на Сервере авторизации с целью выполнения протокола OIDC.

b. response_type=”code” – означает, что используется «*authorization code flow*», т.е доступ Клиента к данным пользователя осуществляется в обмен на code. Данный параметр носит технический характер. Сервер авторизации поддерживает только этот тип.

c. client_id – уникальный идентификатор Клиента, зарегистрированный на Сервере авторизации.

d. redirect_uri – адрес Клиента, на который Сервер авторизации должен перенаправить пользователя. Данный адрес также регистрируется Клиентом на Сервере авторизации.

e. state – идентификатор запроса.

При этом Клиент также устанавливает cookie браузеру пользователя.

3) Пользователь направляется на Сервер авторизации.

4) Пользователь аутентифицируется на Сервере авторизации посредством ввода логина и пароля.

5) В случае успешной аутентификации Сервер авторизации генерирует code и перенаправляет пользователя на Клиента с code и state, полученным в запросе 3).

6) Пользователь направляется на Клиент. При этом браузер автоматически подставляет установленные Клиентом cookie.

7) Клиент направляет запрос на Сервер авторизации с параметрами:

a. grant_type=”authorization_code” – тип доступа, является техническим параметром и означает, что доступ к данным осуществляется в обмен на code. Сервер авторизации поддерживает только данный тип.

b. code – код авторизации, полученный от пользователя.

c. client_id – см. описание из запроса 2).

d. redirect_uri – см. описание из запроса 2).

e. client_auth – параметр, с помощью которого Клиент аутентифицируется на Сервере авторизации. Данный параметр является подписанным JWT (json web token). В JWT подписываются следующие данные: {iss, aud}, где iss – издатель JWT (указывается client_id), aud – получатель JWT (указывается AS). Соответствующий ключу ЭП сертификат ключа проверки ЭП регистрируется Клиентом на AS.

8) Сервер авторизации возвращает токен доступа access_token, который является подписанным JWT (json web token). В JWT подписываются следующие данные: {iss, aud, sub}, где iss – издатель JWT (указывается AS), aud – получатель JWT (указывается client_id), sub – в отношении кого выпущен JWT (указывается user_id).

9) Клиент направляет access_token на Сервер авторизации.

10) Клиент получает в обмен на access_token данные пользователя user_data.

11) Клиент создает личный кабинет на основе данных user_data и предоставляет к нему доступ пользователю в ответ на запрос 6).

1.2 Исходные предположения для анализа

Предположение 1. Защита каналов

Все каналы связи участников взаимодействия защищены с использованием протокола TLS 1.2[3] на базе алгоритмов ГОСТ[4] с аутентификацией сервера (далее – протокол ГОСТ-TLS): в канале «Пользователь-Клиент» аутентифицируется Клиент, в каналах «Пользователь-AS» и «Клиент-AS» аутентифицируется AS. Необходимо проводить анализ в предположении стойкости протокола ГОСТ-TLS.

Предположение 2. Протоколы регистрации

Протокол OIDC предполагает предварительную регистрацию данных участников взаимодействия. Необходимо проводить анализ в предположении стойкости протоколов регистрации. Т.е. нарушитель не может повлиять на перечень регистрируемых данных, он всегда сформирован корректно. В таблице ниже приведен перечень регистрируемых данных.

Таблица. Исходное состояние участников перед выполнением протокола взаимодействия.

Участник	**Зарегистрированные параметры**
Клиент	Параметры хранятся на Клиенте (например, локально в конфигурационном файле). ▪ authorization_endpoint (ac) – адрес AS, на который Клиент перенаправляет пользователя для аутентификации; ▪ token_endpoint (te) – адрес AS, на который Клиент обращается при обмене code на access_token; ▪ userinfo_endpoint (ui) – адрес AS, на который Клиент обращается для получения дополнительной информации о пользователе; ▪ cert_AS – сертификат ключа проверки ЭП AS, предназначенный для проверки ЭП токена доступа access_token.
Сервер авторизации	Данные пользователей хранятся в БД Сервера авторизации (далее – DB_U): ▪ login, pwd – аутентификационные данные; ▪ user_data – идентификационные данные (персональные данные); ▪ user_id – уникальный идентификатор пользователя. Данные Клиентов хранятся в БД Сервера авторизации (далее – DB_C): ▪ redirect_uri – адрес Клиента, на который AS перенаправляет пользователя с code; ▪ client_id – уникальный идентификатор Клиента; ▪ cert_C – сертификат ключа проверки ЭП Клиента, предназначенный для проверки аутентифицированных запросов Клиента с code.

Предположение 3. Протокол аутентификации пользователя

В ходе протокола аутентификации пользователь предъявляет login и pwd на Сервер авторизации. Необходимо проводить анализ в предположении стойкости протокола аутентификации (только в рамках взаимодействия 4) согласно схеме). Сам протокол аутентификации не описывается, происходит некоторое взаимодействие, в ходе которого пользователь предъявляет логин и пароль. При этом необходимо доверять этому взаимодействию и нельзя предполагать, что в ходе аутентификации нарушитель каким-то образом сможет получить логин/пароль другого пользователя.

При этом необходимо предполагать, что пользователь не вводит логин и пароль при каждом обращении к Серверу авторизации, т.е. может иметь «активную» сессию в течение некоторого времени

[3] https://datatracker.ietf.org/doc/html/rfc5246

[4] https://tc26.ru/standarts/rekomendatsii-po-standartizatsii/r-1323565-1-020-2020-informatsionnaya-tekhnologiya-kriptograficheskaya-zashchita-informatsii-ispolzovanie-kriptograficheskikh-algoritmov-v-protokole-bezopasnosti-transportnogo-urovnya-tls-1-2-.html

(например, за счет установки аутентификационных cookie). В случае «активной» сессии взаимодействие пользователя с Сервером авторизации происходит «прозрачно». Браузер автоматически подставляет аутентификационные cookie, при этом от пользователя не требуется никаких действий и для него данное взаимодействие происходит незаметно.

Предположение 4. Функции СКЗИ

Выполнение всех криптографических операций на Клиенте и Сервере авторизации обеспечивается средствами криптографической защиты информации (СКЗИ), которые предоставляют соответствующий API. Необходимо проводить анализ в предположении стойкости криптографических функций и корректности их реализации в СКЗИ.

В частности, API СКЗИ содержит следующие методы:

```
1) sgn = Sign(sk, m) – функция формирования ЭП принимает на вход ключ ЭП sk и подписываемые данные m, возвращает значение электронной подписи sgn.

2) 0/1 = VerifyCertificate(cert) – функция проверки сертификата cert (осуществляет все необходимые проверки: срока действия, назначения, цепочки сертификатов, статуса отозванности по CRL), возвращает результат проверки – 0 (сертификат не действителен) или 1 (сертификат действителен).

3) 0/1 = Verify(cert, m, sgn) – функция проверки ЭП принимает на вход сертификат ключа проверки ЭП cert, подписанное сообщение m и подпись sgn. Возвращает результат проверки – 0 (ЭП неверна) или 1 (ЭП верна).

4) GenRandom(data) – функция генерации псевдослучайной последовательности с использованием СКЗИ (гарантируется достаточная степень близости распределения чисел в последовательности к равномерному), которая записывается в переменную data.
```

1.3 Модель нарушителя

1.3.1 Возможности нарушителя

Предположения о возможностях нарушителя представляются пунктами ниже, описывающими исследуемую Систему и порядок взаимодействия нарушителя с ней. Под честными участниками понимаются участники, которые действуют в соответствии со спецификацией протокола.

1) Все взаимодействия между участниками протокола происходят с использованием HTTP протокола в рамках защищенного ГОСТ-TLS соединения. Нарушитель не может компрометировать и/или модифицировать данные в каналах связи между честными участниками в силу стойкости протокола ГОСТ-TLS (см. Предположение 1 раздела 1.2).
2) Нарушитель не может влиять на процесс регистрации данных или изменять уже зарегистрированные данные (см. Предположение 2 раздела 1.2).
3) Нарушитель не может получить аутентификационные данные пользователей или навязывать их в процессе взаимодействия пользователя с AS в рамках шага 4) (см. Предположение 3 раздела 1.2).
4) Нарушитель не может повлиять на результат выполнения криптографической операции (см. Предположение 4 раздела 1.2).
5) Нарушитель может блокировать и/или задерживать передачу любых HTTP-запросов и ответов любых участников.
6) Нарушитель может являться пользователем и/или Клиентом, зарегистрированном в Системе, то есть обладать своим набором данных, зарегистрированных на Сервере авторизации.
7) Нарушитель не может выступать в роли Сервера авторизации.

8) Нарушитель имеет возможность делать HTTP-запросы к честным Клиентам и Серверу авторизации и получать соответствующие HTTP-ответы. Предполагается, что адреса, на которые делаются запросы, являются известными и константными.

9) Нарушитель имеет возможность навязывать браузеру честного пользователя сформированные HTTP-запросы. При этом нарушитель может навязать только адрес и параметры запроса (передаваемые как часть url). Но у нарушителя нет доступа к HTTP-ответу в силу стойкости протокола ГОСТ-TLS.

Например. Нарушитель может навязать пользователю ссылку: https://srv.com?product_id=123
Но соответствующий ответ будет отправлен браузеру сервером srv.com в рамках TLS-соединения, поэтому параметры ответа недоступны нарушителю.

10) Нарушитель не может навязывать и/или компрометировать Cookie, устанавливаемые между честными пользователями и Клиентом/Сервером авторизации, поскольку они устанавливаются браузером автоматически.

Пояснение. Браузер пользователя осуществит запрос https://srv.com?product_id=123 из примера выше, при этом в ссылке невозможно передать и установить значение заголовка Cookie. Значение Cookie либо подставится браузером автоматически при наличии активной сессии с сайтом srv.com (т.е. ранее Cookie уже были установлены), либо значение Cookie будет получено в ответе в заголовке Set-Cookie, который также недоступен нарушителю.

1.3.2 Угрозы

Ложная аутентификация пользователя

Целью нарушителя является ложная аутентификация на честном Клиенте от имени честного пользователя, аутентификационными данными (на Сервере авторизации) которого он не обладает.

Пример. Реализация угрозы: нарушитель с user_id2 формирует личный кабинет на честном Клиенте client_id1, в котором будут указаны данные другого честного пользователя user_id1 – user_data1. При этом нарушитель не знает аутентификационных данных user_id1 на Сервере авторизации. При необходимости для реализации угрозы нарушитель может также действовать как Клиент client_id2.

Некорректная идентификация пользователя

Рассматривается угроза, при которой честный пользователь успешно идентифицируется честным Клиентом как другой пользователь (например, честный Пользователь после аутентификации попадает в личный кабинет нарушителя).

Пример. Реализация угрозы: нарушитель с user_id2 взаимодействует с Системой таким образом, что честный пользователь user_id1 формирует личный кабинет на честном Клиенте client_id1, в котором будут указаны данные пользователя user_id2 – user_data2. Пользователь user_id2 может являться нарушителем. При необходимости для реализации угрозы нарушитель может также действовать как Клиент client_id2.

Ложная авторизация

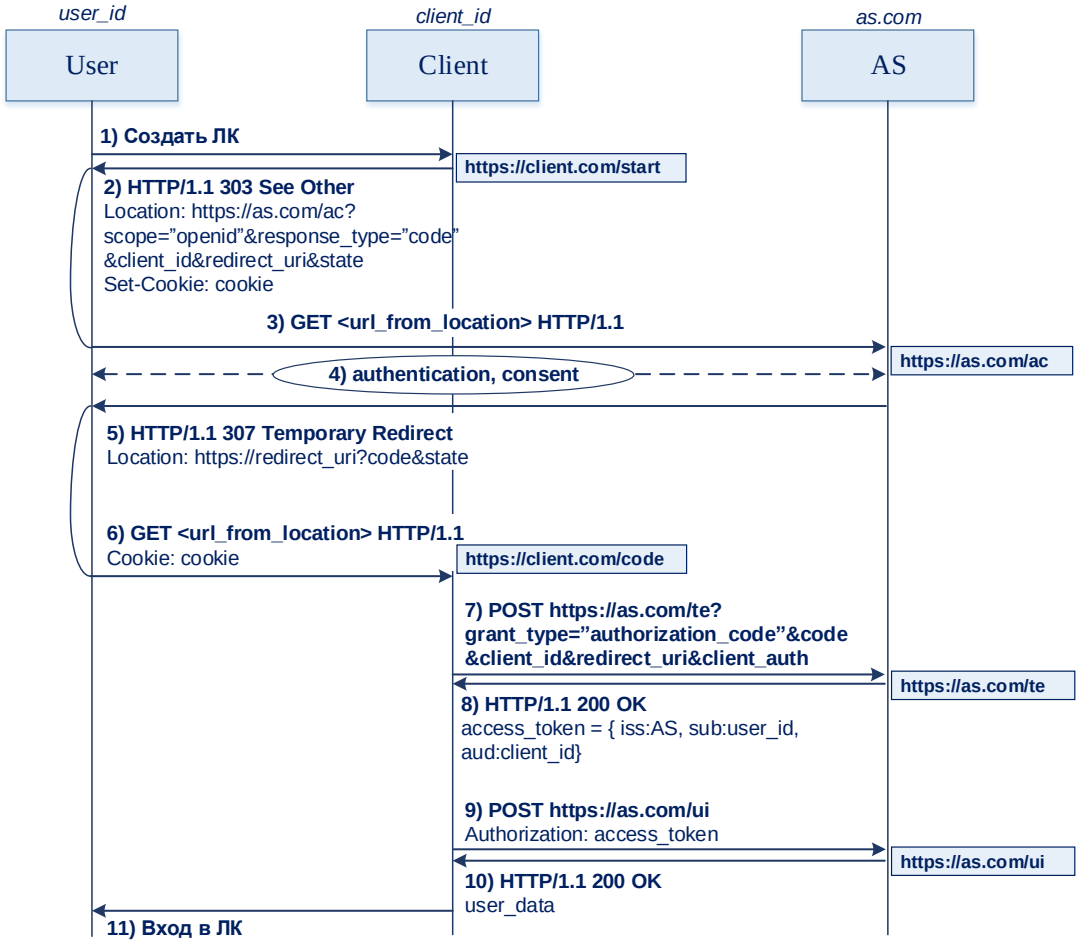
Целью нарушителя является получение токена доступа и данных пользователя без разрешения пользователя.

Пример. Реализация угрозы: нарушитель, являясь Клиентом client_id, получает доступ к access_token, а следовательно, и к user_data честного пользователя user_id. При условии, что user_id не давал согласия на Сервере авторизации при аутентификации на передачу данных client_id.

2 Реализация компонент Системы

В настоящем разделе представлена реализация на псевдокоде Клиента и Сервера авторизации.

Детали реализации HTTP-протокола приведены на рисунке ниже (без указания значений некоторых параметров).



При анализе реализации необходимо предполагать, что функции, связанные с «парсингом» данных запроса или формированием данных в определенном формате работают корректно. К таким функциям относятся следующие:

```
SendResponse // отправка HTTP-ответа с заданными параметрами
GetRequest // парсинг полученного HTTP-запроса на нужные параметры
SendRequest // отправлка HTTP-запроса с заданными параметрами
SaveToDB // создание записи и сохранение данных в базу
GetCookieFromDB // получение данных из базы
GetCertFromDB // получение данных из базы
GetUserIdFromDB // получение данных из базы
GetUserDataFromDB // получение данных из базы
Delete // удаление записи в базе данных
MakeJWT // создание JWT на основе переданных данных
ParseJWT // парсинг JWT на поля данных и подписи
```

2.1 Реализация Клиента

На Клиенте хранятся следующие данные:

```
// client_id_1 – идентификатор Клиента, зарегистрированный на AS
// redirect_uri_1 = "https://client.com/code" - адрес перенаправления, зарегистрированный на AS
// ac = "https://as.com/ac" – адрес AS, на который необходимо перенаправить пользователя
// te = "https://as.com/te" - адрес AS для получения access_token
// ui = "https://as.com/ui" – адрес AS для получения user_data
// sk_1 – ключ ЭП Клиента, которому соответствует cert_1
// cert_1 – сертификат ключа проверки ЭП клиента, зарегистрированный на AS
```

```
// cert_AS – сертификат ключа проверки ЭП AS, предназначенный для проверки ЭП токенов, полученных от AS
```

Реализация обработки запросов адреса https://client.com/start:

```
GenRandom(state_value1);

cookie_value1 = “b978b854-6844-4ad4-8e7a-3a6b85b328ed”;

loc_value = ‘ac?scope=”openid”&response_type=”code”
       &client_id=client_id_1&redirect_uri=redirect_uri_1&state=state_value1’;

SendResponse(303, Location: loc_value, Set-Cookie: cookie_value1);

SaveToDB(DB_temp, state_value1, cookie_value1);
```

Реализация обработки запросов адреса https://client.com/code:

```
 (cookie_value2, state_value2, code_value) = GetRequest(http_req);

cookie_value1 = GetCookieFromDB(DB_temp, state_value2);
if (cookie_value1 != cookie_value2)
       SendResponse(error); // возвращает ошибку в браузер пользователя и прекращает выполнение протокола

auth_data = {iss: client_id_1, aud: ac.com };

sgn = Sign(sk_1, auth_data);

client_auth_value = MakeJWT(auth_data, sgn);

req_value  = ‘te?grant_type=”authorization_code”&code=code_value&client_id=client_id_1
       &redirect_uri=redirect_uri_1&client_auth=client_auth_value’;

access_token_value = SendRequest(POST, req_value);

user_data_value = SendRequest(POST, ui, Authorization: access_token_value);

Delete(DB_temp);

// далее происходит формирование ЛК на основе user_data и ответ пользователю, которые не является
объектом анализа
```

2.2 Реализация Сервера авторизации

На Сервере авторизации хранятся следующие данные:

```
// DB_C – база данных Клиентов, в которой хранится соответствие значений (client_id, redirect_uri,
cert_C)
// DB_U – база данных пользователей, в которой хранится соответствие значений (user_id, (login, pwd),
user_data)
// sk_AS – ключ ЭП AS, которому соответствует cert_AS
// cert_AS – сертификат ключа проверки ЭП AS, предназначенный для проверки ЭП доступа токенов
```

Реализация обработки запросов адреса https://as.com/ac:

```
// scope и response_type опущены
(client_id_value, redirect_uri_value, state_value) = GetRequest(http_req);

// работа протокола аутентификации, который не является объектом анализа
// в результате определяется user_id пользователя
user_id_value = authentication();

GenRandom(code_value);

SaveToDB(DB_temp, client_id_value, user_id_value, code_value);
```

```
loc_value = ‘redirect_uri_value?code=code_value&state=state_value’;

SendResponse(307, Location: loc_value);
```

Реализация обработки запросов адреса https://as.com/te:

```
// grant_type опущен
(client_id_value, redirect_uri_value, code_value, client_auth_value) = GetRequest(http_req);

(auth_data, sgn) = ParseJWT(client_auth_value);

cert_C = GetCertFromDB(DB_C, client_id_value);

if (!Verify(cert_C, auth_data, sgn))
       SendResponse(error); // возвращает ошибку Клиенту и прекращает выполнение протокола

user_id_value = GetUserIdFromDB(DB_temp, code_value);

access_token_data = {iss: as.com, aud: client_id_value, sub: user_id_value };

sgn = Sign(sk_AS, access_token_data);

access_token_value = MakeJWT(access_token_data, sgn);

SendResponse(200, access_token_value);
```

Реализация обработки запросов адреса https://as.com/ui:

```
access_token_value = GetRequest(http_req);

(at_data, at_sgn) = ParseJWT(access_token_value);

VerifyCertificate(cert_AS);
Verify(cert_AS, at_data, at_sgn);

(user_id, client_id, as_id) = ParseJWT(at_data);

user_data = GetUserDataFromDB(DB_U, user_id);

SendResponse(200, user_data);

Delete(DB_temp);
```