ОБЩИЕ УКАЗАНИЯ

Работа с настоящим пособием предполагается последовательно, от работы к работе. После изучения теоретической части раздела студент переходит к выполнению практической части, которая включает в себя разработку алгоритмов и их программной реализации.

Выбор языка и среды программирования осуществляется студентом самостоятельно. *Конечным результатом выполнения практической работы должно явиться законченное и отлаженное приложение, содержащее графический интерфейс пользователя*. Приложение может быть настольным, мобильным или *web*-приложением (студент самостоятельно принимает решение). Каждая работа выполняется студентом индивидуально, для этого он получает начальные условия к заданию по указанному преподавателем варианту (либо персонально).

После выполнения работы студент должен предъявить файлы с исходными кодами разработанного приложения, реализующее поставленные в работе задачи, а также работоспособную программу, которая отражает выполнение поставленных задач. Работа считается выполненной, если она соответствует варианту (либо персональному заданию преподавателя) и в полной мере реализует поставленную задачу.

Лабораторная работа считается выполненной после её защиты. Для защиты работы необходимо представить отчёт, оформленный в соответствии с действующим стандартом университета по построению, изложению и оформлению документов учебной деятельности. На защите работы преподаватель может задать ряд вопросов, касающихся как самой работы, так и ее тематики.

Лабораторная работа №4
«Обеспечение безопасности *web*-приложений»

Цель работы: ознакомиться с базовыми механизмами обеспечения информационной безопасности в *web*-приложениях.

Задачи:

– ознакомиться с основами механизмами аутентификации в *web*-приложениях;

– ознакомиться с базовыми моделями доступа в задачах защиты от несанкционированного доступа;

– изучить криптографический протокол Диффи-Хеллмана и особенности его применения в задачах разработки механизмов аутентификации;

– ознакомиться с областью применения хеш-функций в задачах обеспечения информационной безопасности web-приложений;

– получить навыки разработки *web*-приложений с учетом аспектов их информационной безопасности.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Практически все современные *web*-приложения требуют определённого разграничения доступа пользователей к хранимой и/или обрабатываемой в них информации. Для реализации механизмов управления доступом в приложении оно должно каким-то образом понимать, какой именно пользователь сейчас пытается с ним работать. Для этих целей в приложении реализуются механизмы **аутентификации** и **авторизации** пользователей.

Аутентификация – это процесс подтверждения подлинности субъекта приложения (например, пользователя), успешным результатом которого является установление конкретного пользователя приложения. Другими словами, аутентификация проверяет, является ли субъект тем, кем себя заявляет. Обычно это реализуется путем проверки учётных данных пользователя, токена безопасности или иным способом.

Авторизация – это процесс предоставления определенного набора прав пользователю для работы с функционалом приложения. Авторизация выполняется только после успешной аутентификации

Аутентификация и авторизация являются одним из основных элементов защиты от несанкционированного доступа любого *web*-приложения, так как посредством них пользователь получает определенный набор функций *web*-приложения.

От того, насколько надёжно реализован механизм аутентификации,

зависит сложность получения злоумышленником несанкционированного доступа к конфиденциальной информации и выполнения от лица легального пользователя действий, влекущих за собой юридические последствия. В этой связи выбор способа аутентификации должен быть обоснован, аргументирован, и его стойкость должна соответствовать реальным угрозам безопасности, характерным для конкретной *Web*-ориентированной ИС.

1.1. Аутентификация, авторизация и управление доступом

Все методы аутентификации в общем случае можно разделить на четыре категории:

– аутентификация по наличию у пользователя уникального объекта заданного типа. В качестве такого объекта могут выступать: смарт-карта, токен, электронный ключ и т.п.

– аутентификация, основанная на некоторой конфиденциальной информации, известной пользователю. Наиболее распространённым методом данной категории является парольная защита.

– аутентификация, основанная на использовании пользовательских биометрических характеристик человека. В качестве таких признаков могут использоваться: отпечатки пальцев, рисунок сетчатки или радужной оболочки глаза, тепловой рисунок кисти руки, фотография или тепловой рисунок лица, почерк (роспись), голос или комбинации этих признаков.

– аутентификация, основанная на использовании некоторой информации, ассоциированной с пользователем. В данном контексте может имеет интерес информация о местоположении пользователя (*GPS* или *GeoIP*-локация).

Общая схема базового механизма аутентификации представлена на рис. 4.1.

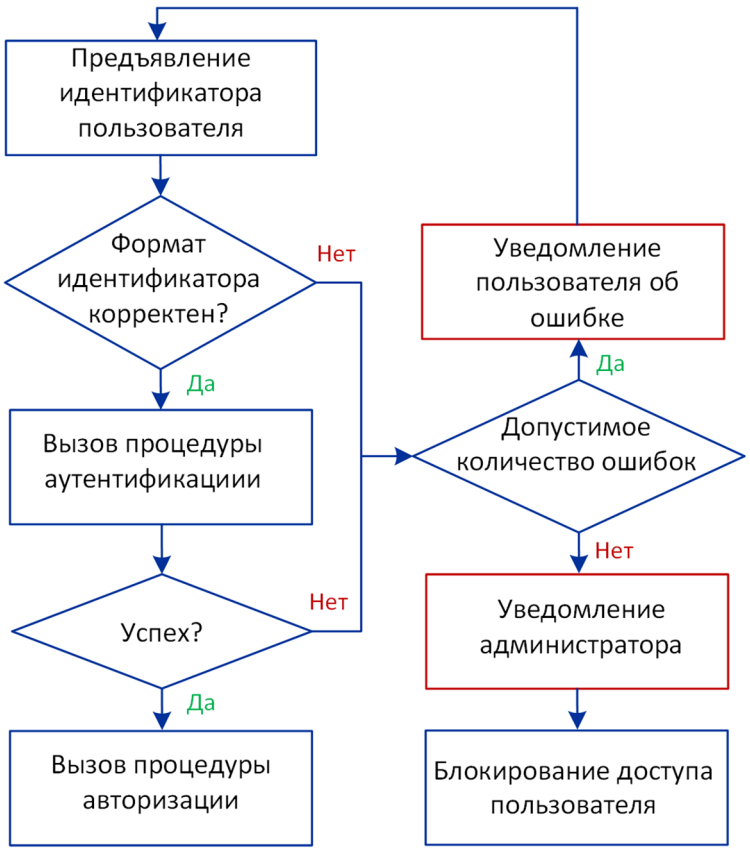

Рис. 4.1. Общая схема аутентификации

Кроме того, методы аутентификации можно разделять и по количеству сторон, участвующих в данной процедуре. В данном контексте методы аутентификации можно разделить на:

– односторонние, когда только одна сторона доказывает свою подлинность;

– двусторонние (взаимная аутентификация), когда обе стороны доказывают друг другу свою подлинность. Необходимо отметить, что взаимная аутентификация часто требует участия третьего участника – арбитра.

Сама по себе процедура аутентификации не самодостаточна в задачах защиты от несанкционированного доступа, а является лишь средством однозначной идентификации прав того или иного пользователя (группы пользователей) в приложении. При этом сами права и полномочия пользователя определяются, как уже было отмечено ранее, процедурой авторизации и в дальнейшей работе с приложением обеспечиваются конкретной моделью доступа.

К базовым моделям управления доступом обычно относят четыре: дискреционная, мандатная, ролевая и атрибутная модели.

Дискреционная модель управления доступом *(*модель Харрисона-Руззо-Ульмана, матричная модель) формализует понятие матрицы доступа – таблицы, описывающей права доступа субъектов приложения к объектам. Набор возможных прав определяется на стадии проектирования приложения. В дискреционной

модели у любого объекта есть владелец, который и устанавливает права доступа.

	Объект 1	Объект 2	…	Объект *N*
Субъект 1	Чтение	Чтение, запись	…	Нет прав
Субъект 2	Чтение, запись	Чтение	…	Чтение
…	…	...	…	…
Субъект *M*	Чтение	Чтение, запись	…	Нет прав

Мандатная модель управления доступом (модель Белла-ЛаПадулы) предусматривает разделение всех субъектов и объектов информационной системы по уровням доступа (секретности) путём назначения любому объекту метки конфиденциальности, а любому субъекту – уровня допуска. Мандатная модель управления доступом запрещает пользователю или процессу, обладающему определенным уровнем доверия, получать доступ к информации, процессам или устройствам более защищенного уровня. Мандатная модель управления доступом была разработана для автоматизированных систем, ориентированных на работу с информацией, составляющей государственную тайну.

Ролевая модель управления доступом (*Role-Based Access Control*, *RBAC*) предполагает назначение любому объекту прав доступа в соответствии с определенной ролью пользователя. В данной модели понятие субъект заменяется двумя новыми понятиями: «Пользователь» – человек, работающий в системе и «Роль» - активно действующая в системе абстрактная сущность, с которой связан ограниченный и логически непротиворечивый набор полномочий, необходимых для осуществления тех или иных действий в системе. При этом для каждой роли указывается набор полномочий, представляющий собой набор прав доступа к объектам приложения, а каждому пользователю назначается список доступных ему ролей в приложении (ролей может быть несколько).

Как правило, выделяют два базовых подхода к распределению ролей пользователей:

- создание иерархических ролей, полностью копирующих корпоративную иерархию и сохраняющих отношения между ролями, существующие в реальном мире (см. рис. 4.2);
- использование взаимоисключающих ролей, позволяющих эффективно реализовать разделение обязанностей.

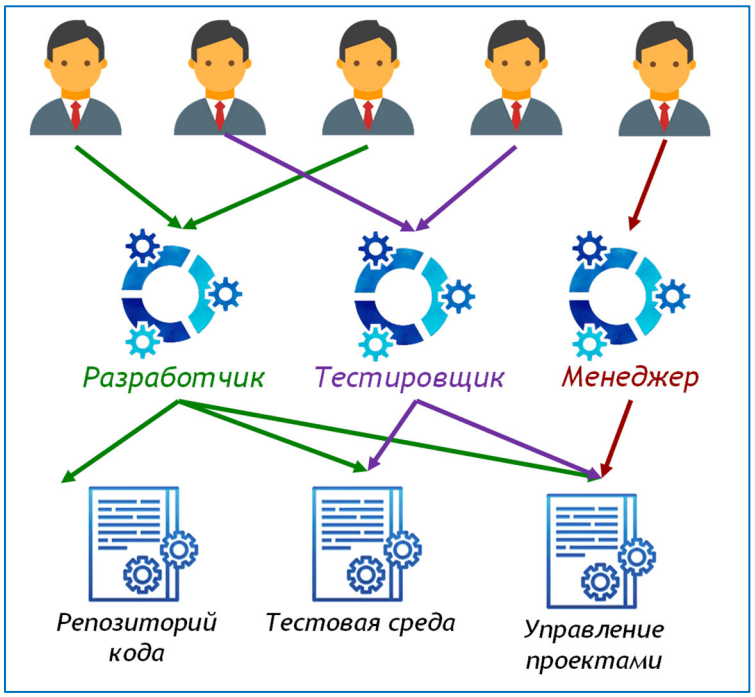

Рис. 4.2. Пример реализации ролевого управления доступом

Атрибутная модель управления доступом (*Attribute-Based Access Control*, *ABAC*) – модель контроля доступа к объектам, основанная на анализе правил, которые учитывают атрибуты объектов или субъектов, возможные операции с ними и окружение (среда выполнения), соответствующего запроса. Общая архитектура атрибутной модели управления доступом представлена на рис. 4.3.

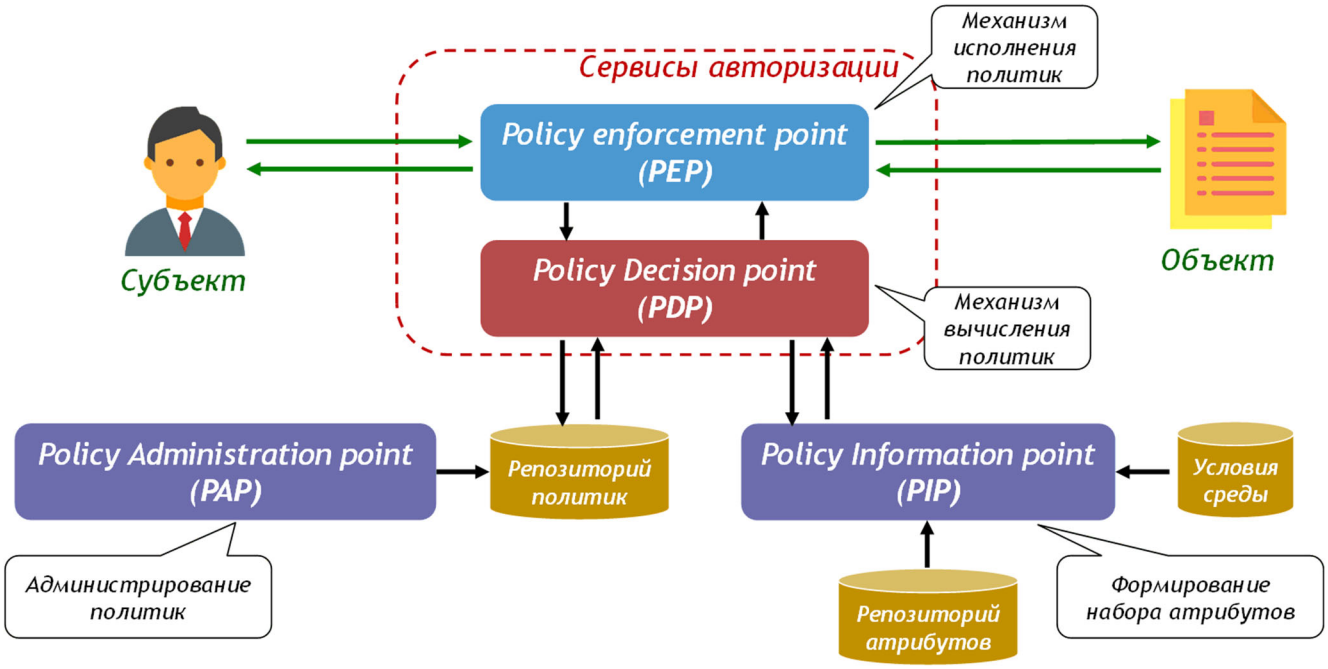

Рис. 4.3. Общая архитектура атрибутной модели управления доступом

Архитектура ABAC состоит из нескольких логических компонентов, взаимодействующих для обеспечения комплексного контроля доступа:

– точка принятия решений по политикам (*Policy Decision Point*, *PDP*) – ядро системы, где происходит исполнение политик безопасности и принятие решений о доступе субъектов к объектам;

– точка применения политик (*Policy Enforcement Point*, *PEP*) – также ключевой элемент модели, который обрабатывает все запросы на предоставление доступа от субъектов к объектам и обеспечивающий исполнение решений PDP;

– точка информации о политиках (*Policy Information Point*, *PIP*) – источник атрибутов, необходимых для исполнения политик;

– точка администрирования политик (*Policy Administration Point*, *PAP*) – интерфейс администратора, посредством которого создаются и управляются политики доступа.

Своей концепцией атрибутная модель управления доступом предлагает подход к управлению доступом, основываясь на контексте запроса, а не на предопределенных ролях или списках доступа. Это обеспечивает практически неограниченную гибкость и точечность настройки доступа.

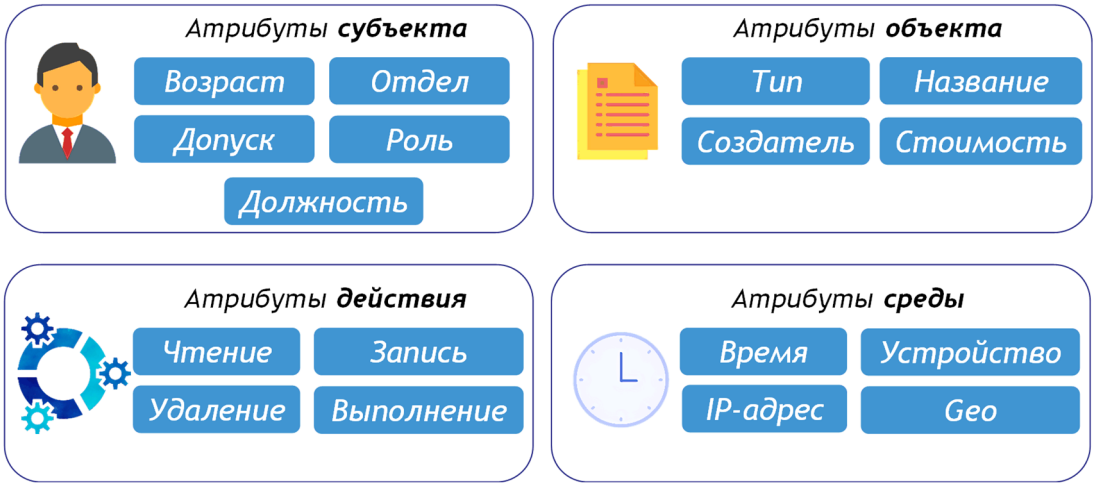

Рис. 4.4. Примеры атрибутов

1.1.1. Аутентификация в *web*-приложениях

Web-приложения отличаются от настольных приложений своей определённой спецификой, а именно:

– относятся к клиент-серверным решениям;

– являются распределёнными (в общем случае – в планетарном масштабе);

– являются массовыми (в общем случае);

– содержат конфиденциальные данные (в общем случае), например, персональные данные пользователей.

Кроме того, многие *web*-приложения предусматривают денежные отношения, между участниками таких систем и/или владельцами последних.

Эта специфика *web*-приложений в большинстве случаев предусматривает разграничение доступа пользователей к информации, а значит и процедуру аутентификации пользователей. Однако далеко не все методы аутентификации применимы в *web*-приложениях, в силу ряда очевидных причин. Давайте рассмотрим наиболее распространённые механизмы аутентификации, применимые для большинства ИС (исключая электронную коммерцию и некоторые иные особые ИС).

Длинный *URL*. Наиболее просто реализуемый метод аутентификации – сделать некоторый путь *URL* того или иного ресурса длинным и бессмысленным. Например, для получения определённого уровня доступа к приложению необходимо зайти по данному адресу:

```
https://mydomain.ru/nb4nz2yi679yztx4twkybkpjcio9x5.aspx
```

При этом дополнительных механизмов аутентификации не применять. Если на самом *web*-ресурсе не будет ссылок на этот адрес и это адрес будет держаться в секрете, то подбор такого *URL* для злоумышленника будет являться весьма сложной вычислительной задачей. В принципе такой *URL* сам по себе и является своеобразным паролем на доступ. Можно считать, что пароль содержится в самом *URL*–адресе. Однако очевидны и серьёзные недостатки этого метода, например неудобство применения.

Аутентификации на стороне клиента. В данном случае аутентификация целиком проводится на стороне клиента, а серверу лишь посылается ее результат, например, на перенаправление по некоторому длинному *URL*, описанному выше. В качестве примера рассмотрим *HTML*-страницу, содержащую простой *java*-скрипт:

```
      <html>
         <title>Проверка доступа к ИС</title>
         <head>
           <script type="text/javascript"> function authentification()
            {
              p = prompt('Введите пароль доступа');
              if(p == 'mypass')
              {
                document.location.href= nb4nz2yi679yztx4twkybkpjcio9x5.aspx
';
                exit;
              }
              else alert('Пароль неверный');
            }
```

```
          </script>
        </head>
        <body>
          <input type="button" value="Войти в систему" onclick=" authentifi-
cation ()">
        </body>
      </html>
```

Очевидно, что такая схема аутентификации также имеет серьёзные недостатки.

Аутентификация с использованием пары логин/пароль. Данный метод является одним из наиболее распространенных, в силу своей простоты реализации и относительно высокой надёжности (при правильной реализации). Суть метода заключается в том, что пользователь вводит в определённую форму свои учётные данные, затем они пересылаются на сервер, который осуществляет аутентификацию и авторизацию. Результатом успешной аутентификации и авторизации будет возвращаемый пользователю уникальный идентификатор его текущего сеанса, который на какое-то время будет его «пропуском» в *web*-приложение.

Необходимо отметить, что при всех своих достоинствах этот метод аутентификации подвержен угрозам безопасности, поскольку учётные данные могут быть перехвачены злоумышленником тем или иным способом. Можно выделить следующие потенциальные уязвимости парольной аутентификации:

- возможность подбора слабого пароля простым перебором;
- возможность автоматизированного подбора пароля по словарю;
- ошибки конфигурирования и администрирования (возможность использовать слабый пароль, повторно использовать пароль, длинные сроки эксплуатации пароля и т.п.);
- возможность кражи или перехвата пароля;
- фишинговые атаки;
- применение злоумышленником различных методов социальной инженерии;
- кража учётных данных посредством вредоносного программного обеспечения;

Поэтому надёжность такого метода аутентификации сильно зависит как от конкретной технологической реализации, так и от ответственности пользователей и администраторов.

Аутентификации с использованием одноразовых паролей. Данный метод подразумевает использование некоторого уникального пароля для каждой

процедуры аутентификации. Такая схема аутентификации может быть достаточно безопасной, но только при условии безопасной передачи одноразового пароля пользователю. На практике одноразовые ключи могут быть получены пользователем следующими механизмами:

- с использованием бумажного носителя, в этом случае перечень одноразовых паролей печатается на бумаге и передаётся на дальнейшее хранение пользователю, который использует одноразовые пароли по мере необходимости (например, при использовании интернет-банка);
- с использованием *SMS*-сервиса или мессенджера, в этом случае одноразовый пароль (например, для подтверждения транзакции) приходит пользователю на телефон в виде *SMS* или сообщения в мессенджере;
- с использованием генератора одноразовых паролей, в этом случае пользователю выдаётся генератор одноразовых паролей (небольшое устройство или программа для смартфона), синхронизированный с сервером. Для доступа к информационной системе пользователь с помощью устройства генерирует новый пароль и отправляет его на сервер. Аутентификация с использованием одноразовых паролей обычно применяется в качестве дополнительного барьера защиты к аутентификации с использованием пары логин/пароль.

Взаимная аутентификация с использованием модели «запрос-ответ». Данный класс методов аутентификации, как правило, используется в информационных системах, которые позволяют осуществлять юридически значимые операции (интернет-банкинг, государственные информационные системы и т.д.). Они предполагают использование криптографических механизмов и специальных средств криптографической защиты информации как на стороне клиента (пользователя), так и на стороне сервера. Основная идея такой аутентификации состоит в том, что в процессе обмена некоторой информацией одна из сторон (либо обе одновременно) доказывает другой стороне свою подлинность. Доказательство подлинности в таких протоколах основывается на подтверждении обладания некоторым определённым общим секретом:

- в системах с симметричной криптографией обычно доказывается обладание секретным ключом, который должен быть известен обеим сторонам;
- в системах, использующих инфраструктуру открытых ключей (*PKI*), каждая сторона доказывает обладание ключом электронной подписи (ЭП), который используется для аутентификации.

Очевидно, что такие методы аутентификации требуют использование надёжных криптографических алгоритмов, а также безопасное хранение и использование средств криптографической защиты и ключевой информации.

Как уже было отмечено, методы взаимной аутентификации по модели «запрос-ответ» предполагают использование криптографических методов как на стороне сервера, так и клиента. Остановимся на механизмах использования криптографии на стороне клиента, как на наиболее уязвимой. Основу криптографических методов составляют непосредственно алгоритмы и их реализации, которые в нашем случае могут быть либо программными (с использованием криптопровайдеров), либо аппаратными.

Криптопровайдер (англ. *Cryptography Service Provider*, *CSP*) – это независимый модуль, позволяющий осуществлять криптографические операции в операционных системах. Приложения не работают напрямую с криптопровайдером, вместо этого они вызывают специальные функции из системных библиотек операционной системы, которая обрабатывает вызовы этих функций и вызывает соответствующие им методы криптопровайдера. Для хранения секретных (закрытых) ключей криптопровайдер с программной реализацией криптоалгоритмов использует ключевой контейнер, который представляет собой зашифрованный файл, доступ к которому защищён *PIN*-кодом. Данный файл может храниться в любом месте на диске компьютера или на съёмном носителе. Однако в случае кражи файла с ключевым контейнером, злоумышленник может подобрать пароль к нему и тем самым получить доступ к закрытому ключу, а, следовательно, и к информационной системе.

В связи с этим, хранение ключевого контейнера на компьютере или незащищённом носителе не является безопасным. Более безопасной альтернативой является хранение ключевого контейнера на *USB*-накопителе или специальном устройстве – *eToken*. Такой способ хранения ключевого контейнера более безопасен, так как:

– *eToken* обеспечивает защиту контейнера от несанкционированного копирования. Защита обеспечивается тем, что контейнер помещается в защищённую область *eToken*, доступ к которой предоставляется только при предъявлении *PIN*-кода.

– *eToken* блокирует доступ к защищённой области, в которой хранится контейнер, при превышении заданного числа последовательных неправильных попыток ввода *PIN*-кода. Это позволяет защитить контейнер от несанкционированного доступа к нему в случае кражи *eToken* и попыток подбора *PIN*-кода.

Заметим, что аутентификация с использованием *eToken* является двухфакторной, поскольку при доступе к защищённой системе пользователь:

– подтверждает, что он обладает *eToken*,

– подтверждает, что он знает *PIN*-код этого *eToken*.

Двухфакторная аутентификация является более безопасной, так как ключевой контейнер не может быть скопирован и украден без знания *PIN*-кода. А в случае, если пользователь обнаружит кражу самого *eToken*, он сможет известить об этом администратора информационной системы, чтобы заблокировать его. Таким образом, *eToken* позволяют обеспечить относительно безопасный способ хранения ключевого контейнера криптопровайдера. Однако заметим, что в случае программного криптопровайдера существует (и вполне реализуема) угроза перехвата данных криптопровайдера из оперативной памяти вредоносным ПО, получившим доступ к выполняемым в оперативной памяти ПК процессам. В силу чего криптоконтейнер и/или хранимые в нём ключи и/или его *PIN* могут быть скомпрометированы ещё на стадии генерации. Данного недостатка лишены механизмы аутентификации с использованием аппаратных ключевых контейнеров, – *eToken* с аппаратно реализованными криптоалгоритмами и неизвлекаемыми ключами. Такое решение более безопасно, т.к. секретные ключи генерируются внутри самого устройства и никогда не покидают его, а значит не могут быть перехвачены в оперативной памяти компьютера.

1.1.2. Односторонняя аутентификация в *web*-приложениях. Токены аутентификации

Большинство *web*-приложений вполне ограничиваются только аутентификацией пользователя, т.к. её реализация сравнительно проста и не требует применения дополнительных аппаратных средств.

Аутентификация пользователя – это проверка подлинности предъявленного им идентификатором. Результатом аутентификации является успешная идентификация его в приложении, другими словами, теперь приложение понимает с каким именно пользователем оно взаимодействует. В зависимости от важности ресурса для доступа к нему могут применяться различные методы аутентификации, некоторые из которых были описаны в предыдущем подразделе. Рассмотрим среди них наиболее распространенные при реализации web-приложений.

Базовая аутентификация (нативная аутентификация *web*-сервера). Её суть заключается в следующем. Пользователь пытается зайти в защищённую область приложения, например, следующем запросом:

```
GET /site/private HTTP/1.0
```

В ответ на такой запрос сервер предлагает клиенту web-приложения пройти процедуру аутентификации, для чего посылает ему ответ со соответствующими заголовками:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="private"
```

После получения ответа, клиентский *web*-браузер отображает пользователю соответствующее окно (см. рис. 4.5). После ввода пользователем своих учётных данных, *web*-браузер отправляет новый запрос на сервер, который содержит в себе заголовок для аутентификации:

```
GET /site/private HTTP/1.0 Authorization: Basic YWRtaW46MTIzNDU=
```

Заметим, что учётные данные не передаются в явном виде, они кодируются с использованием кодировки *base64* (что, в принципе, для злоумышленника не является помехой, т.к. он одной строчкой кода может извлечь данный в явном виде). Так, строчка `YWRtaW46MTIzNDU=` есть ни что иное, как комбинация `admin:12345`.

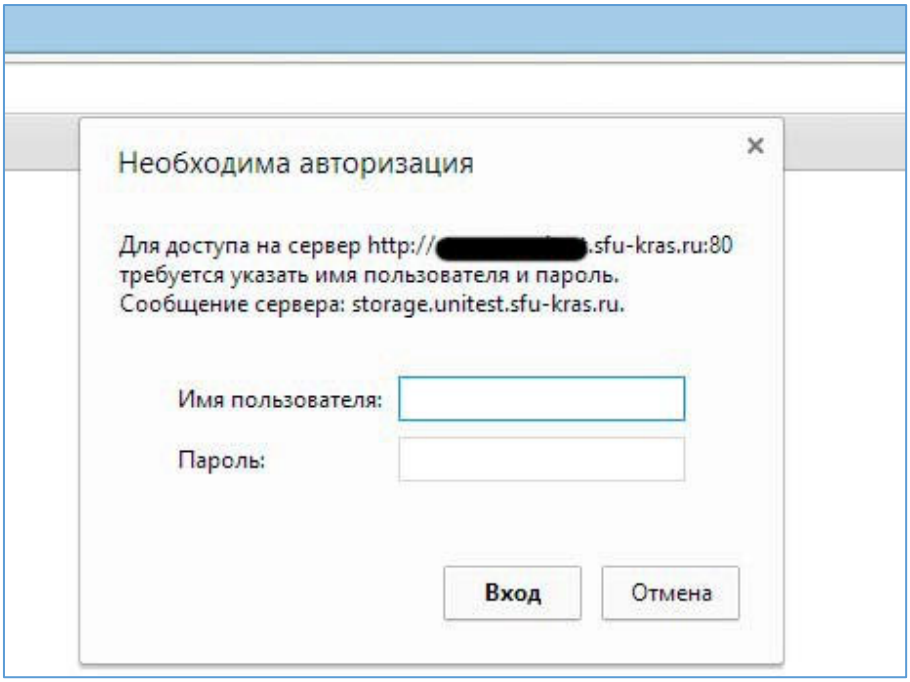

Рис. 4.5. Запрос *web*-браузером учётных данных пользователя

В зависимости от корректности введённых данных сервер разрешит доступ к запрашиваемому ресурсу, либо запретит.

Дайджест аутентификация. Её суть аналогична базовой аутентификации, за исключением того, что учётные данные не передаются в открытом виде, а предварительно обрабатываются. Логин и пароль пользователя предварительно подвергается хешированию и только потом передаются на *web*-сервер, на котором осуществляется аутентификация.

***Cookies*-аутентификация**. Данный механизм аутентификации реализовывается в виде соответствующей бизнес-логики приложения и является чрезвычайно популярным. Механизм включает в себя следующие этапы:

1. пользователь вводит свои учётные данные в соответствующие поля специально сформированной страницы входа в приложение и отправляет их на сервер в виде запроса;
2. *web*-сервер получает учётные данные пользователя, выполняется аутентификация и, в случае успеха, осуществляется переадресацию на страницу успешного входа (или иную), прикрепив в ответе *cookies* приложения с некоторым идентификатором (идентификаторами) организованной сессии. Эти *cookies* могут быть действительны только для текущей сессии *web*-браузера или могут быть настроены на иной период хранения (в том числе на формально- неограниченное);
3. При последующей работе с *web*-приложением для каждого запроса пользователя *web*-браузер автоматически отправляет соответствующей *cookies* с идентификатором, полученным на шаге 2;
4. Сервер проверяет идентификатор в своей базе идентификаторов и, при наличии в базе такого идентификатора, «узнает» пользователя и допускает его к запрошенному ресурсу (в случае наличия у него соответствующего права доступа) без повторной аутентификации.

Такой механизм аутентификации является очень удобным как для пользователя ИС (нет необходимости постоянно проходить аутентификацию), так и для сервера (минимизация вычислительных издержек). Однако данный метод аутентификации требует грамотной проработки, т.к. *cookies* могут быть украдены (тем или иным способом), а пользователь скомпрометирован.

Аутентификация по сертификатам. Механизмы аутентификации с применением цифровых сертификатов базируются на методах криптографии с открытым ключом. Кроме того, такой механизм аутентификации использует определённый протокол с запросом и ответом, и может включать в себя дополнительного участника процесса аутентификации (арбитра). Упрощённую (для принципиального понимания) схему такой аутентификации можно описать следующим образом (см. рис. 4.6). Сервер аутентификации отправляет клиенту последовательность символов (запрос Q), ответом выступает этот же запрос сервера, но зашифрованный с помощью закрытого ключа пользователя - $E(Q,k^s)$. Сервер расшифровывает его с помощью соответствующего открытого ключа и в случае, если $D(Q,k^o)=Q$ клиент успешно авторизовывается.

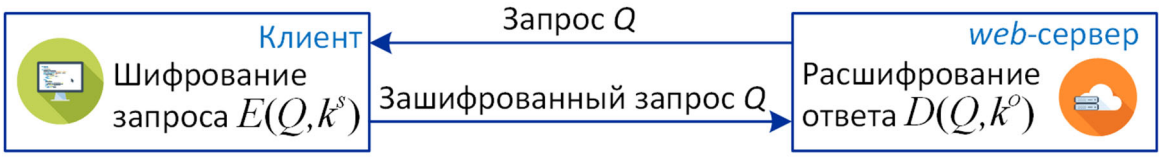

Рис. 4.6. Упрощённая схема аутентификации по сертификатам

Аутентификация с открытым ключом используется как защищенный механизм аутентификации в таких протоколах как *SSL*. Она также может использоваться как один из методов аутентификации в рамках протоколов *Kerberos* и *Radius*.

Открытая децентрализованная аутентификация. Данная схема аутентификации является весьма популярной. Суть ее заключается в том, чтобы осуществлять аутентификацию пользователей без их предварительной регистрации в каждом отдельном приложении. Для этих целей прибегают к некоторой третьей стороне, которой доверяют все участники процесса. В упрощённом виде этот механизм можно описать так: система *A* запрашивает у системы *B* проверку подлинности пользователя, в случае успешной проверки система *A* предполагает, что пользователь является известным (система *B* может передавать необходимые данные о пользователе системе *A*). Эта концепция получила название *Federated Identity*, а сама технология аутентификации называется **аутентификацией с единым входом** (***Single Sign-On***, ***SSO***).

Достоинство такого подхода заключается в том, что пользователю не нужно осуществлять регистрацию в приложении, а он может предоставить свои данные посредством сервиса, в котором он уже был зарегистрирован. Кроме того, разработчикам приложения не нужно самостоятельно реализовывать процедуру аутентификации пользователей, вместо этого можно делегировать эту задачу доверенной стороне. Для пользователя такой подход также является очень удобным, он может использовать одну учётную запись для доступа ко многим ресурсам (заметим, этот факт является и недостатком в контексте безопасности).

На сегодняшний день активно используются два протокола аутентификации с единым входом. **Протокол *OpenID*** – это открытая децентрализованная система аутентификации, которая позволяет пользователю использовать одну учетную запись для аутентификации в различных системах. При этом эти системы не обязательно должны быть связаны друг с другом. Начала свое развитие в 2005 году, одним из создателей *LiveJournal*. Протокол *OpenID* содержит три базовых понятия:

– *User* - пользователь, который делает запрос на аутентификацию

– *Provider* (провайдер) - система, которая хранит данные пользователя, и обрабатывает запросы *OpenID*

– *RelyingParty* - зависимая сторона, которая хочет проверить подлинность пользователя.

Процесс аутентификации по *OpenID* в общем виде состоит из следующих этапов:

1. пользователь пытается получить доступ к закрытому ресурсу *web*-приложения (зависимой стороны, *Relying Party*);
2. приложение запрашивает у пользователя учётные данные;
3. пользователь вместо учётных данных отправляет свой *URL OpenID* (уникальный для всех пользователей *OpenID*);
4. приложение перенаправляет пользователя к провайдеру *OpenID* для проверки подлинности;
5. пользователь вводит свои учётные данные *OpenID* для входа;
6. провайдер *OpenID* аутентифицирует пользователя;
7. в случае успешной аутентификации, провайдер перенаправляет пользователя обратно к приложению, включая идентификационную информацию (токен безопасности);
8. приложения проверяет токен от провайдера и допускает пользователя к системе, если он действующий.

Токен (*token*) – условный объект некоторой структуры, содержащий идентификационную информацию от провайдера для получения доступа к приложению. Важно отметить, что срок действия токена, как правило, ограничен временем – в этом случае токен называется временным. Регистрация временных штампов для токенов выполняется на сервере провайдера, таким образом, исключается вероятность подмена срока действия токена на стороне клиента (как это может произойти в случае с аутентификацией по *cookies*).

Проводя аналогию с аппаратно-реализованными токенами, система работает схожим образом, когда через определенные интервалы времени токеном и сервером генерируется новое значение кода доступа, действительное только определенный небольшой интервал времени, по истечении которого, процесс повторяется. Заметим, что токен может быть промежуточным, то есть доступ еще не получен, а сам токен просто хранит данные для следующих этапов аутентификации.

При построении токена в том или ином виде выполняются следующие шаги:

– генерируются внутренние начальные значения (*seed values*) (как в случае, например, с созданием ключей для алгоритма *RSA* – два больших

простых числа). Эта задача решается на стороне провайдера программный или аппаратными средствами;

– генерируется временной штамп (время периода) создания пароля;

– фиксируются предшествующие события (использованные пароли) – этот пункт используется в системах, когда провайдер принимает пароли только в определенной последовательности, однако сами значения паролей не поддаются статистическому или другому анализу, что исключает возможность предугадать следующий, ожидаемый системой ключ;

– генерируются собственно секретные ключи на основе принятого к использованию в конкретном токене алгоритма;

– результат аутентификации передаётся зависимой стороне (*web*-приложению).

Протокол *OAuth* (*Open Authorization, OAuth* 2.0) – протокол авторизации, позволяющий приложениям получать ограниченный доступ к ресурсам пользователя на сервисах без передачи учётных данных.

В реализации протокола *OAuth* 2.0 участвуют четыре ключевых роли: владелец ресурса, клиент, авторизационный сервер и ресурсный сервер.

Владелец ресурса – это пользователь, который предоставляет *web*-приложению доступ к своим данным. Например, пользователь Иван Иванов станет владельцем ресурса, когда разрешит приложению доступ к своим данным в облачном хранилище (например, фотографиям).

Клиент – это *web*-приложение, которое запрашивает доступ к данным владельца ресурса. Поскольку клиент не имеет прямого доступа к данным, он использует токен авторизации, выданный владельцем ресурса.

Авторизационный сервис – это сервис, который проверяет личность владельца ресурса, выдаёт токены доступа и управляет разрешениями, предоставленными клиенту. Так, сервис удостоверяется, что Иванов Иван действительно является владельцем ресурса. После проверки он выдаёт временный токен или отказывает в доступе, если учётные данные не совпадают.

Ресурсный сервис – это сервис, который хранит данные владельца ресурса и предоставляет к ним доступ на основе временных токенов. Например, облачное хранилище, где Иванов Иван хранит свои фотографии. Ресурсный сервер принимает токен от приложения, проверяет его и открывает доступ к фотографиям. Он также может разрешать различные действия, такие как просмотр, загрузка или удаление файлов. Конкретный список разрешённых действий зависит от прав, предоставленных владельцем ресурса.

В общем виде процесс аутентификации по протоколу *OAuth* 2.0 можно представить следующими этапами:

– пользователь (владелец ресурса) хочет авторизоваться в приложении, которое ему, в том числе предлагает это сделать через иной сервис (например, google);

– приложение перенаправляет владельца ресурса на сервис авторизации через *URL* с уникальными параметрами запроса:

- *client_id* – идентификатор приложения;
- *redirect_uri – URI* для перенаправления после авторизации;
- *scope* – набор права доступа, которые запрашивает приложение;

– авторизационный сервис распознаёт приложение и определяет, доступ к каким данным оно запрашивает;

– пользователь вводит свои учётные данные в сервисе авторизации. После успешной проверки сервис авторизации сообщит пользователю доступ к каким именно данным запрашивает приложение и запросит разрешение у пользователя предоставить доступ к ним;

– после согласия сервис авторизации выдаёт код авторизации и перенаправляет пользователя обратно в приложение, передавая код через *URL*. Этот код имеет ограниченный срок действия и может быть использован только один раз.

– приложение отправляет полученный код на сервер авторизации и обменивает его на токен доступа. Сервер проверяет код и выдаёт токен, который используется для запросов к ресурсному сервису.

– Приложение использует токен для запроса данных у ресурсного сервиса. В ответ сервис предоставляет доступ к запрашиваемым данным.

4.1.3. Протокол Диффи-Хеллмана для выработки общего секретного ключа

Протокол Диффи-Хеллмана (англ. *Diffie-Hellman*, *DH*) – криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный открытый канал связи. Полученный ключ в дальнейшем может использоваться обеими сторонами для шифрования дальнейшего информационного обмена с образованием защищённого канала связи посредством алгоритмов симметричного шифрования, а также для реализации методов аутентификации.

Кратко рассмотрим этапы выработки секретного ключа данным протоколом. Предположим, существует два пользователя – Алиса и Боб, которые хотят

организовать криптографически защищённый канал связи, для чего им необходимо выработать один общий секретный ключ в условиях, когда они взаимодействуют по открытому (незащищённому) каналу связи. Алису и Бобу нужно выполнить следующие шаги:

1. Каждый из пользователей выбирает некоторые два числа g и p, которые не являются секретными и могут быть также известны и другим заинтересованным лицам. При этом число p является *большим*, *случайным* и *простым*, а g является *первообразным корнем по модулю* p, т.е. $g^{\varphi(p)} \equiv 1 \pmod{p}$, где $\varphi(p)$ – функция Эйлера.
2. Для того, чтобы создать не известный более никому секретный ключ, оба пользователя генерируют большие случайные числа: Алиса – число ***a***, Боб – число ***b***, которые являются натуральными и большими.
3. Алиса вычисляет свой публичный ключ $A = g^a \bmod p$ и пересылает его Бобу, который, в свою очередь, вычисляет свой публичный ключ $B = g^b \bmod p$ и отсылает Алисе. Алгоритм предполагает, что злоумышленник может узнать числа A и B, но поменять их не может (справедливости ради заметим, что в некоторых случаях злоумышленник может их изменить).
4. Алиса, на основе имеющегося у нее значения a и полученного от Боба B вычисляет общий секретный ключ $K = B^a \bmod p$, в свою очередь Боб на основании имеющегося у него b и полученного A также вычисляет общий секретный ключ $K = A^b \bmod p$.
5. В силу того, что Боб и Алиса получили один и тот же секретный ключ, они могут использовать его в качестве ключа для симметричного алгоритма шифрования и организовать защищённый канал связи. Злоумышленник, же в свою очередь, встретится с практически неразрешимой (за разумное время) проблемой вычисления $B^a \bmod p$ и $A^b \bmod p$, если числа p, a и b выбраны достаточно большими.

1.1.3. Использование хеш-функций в процедурах аутентификации

Хеш-функция (функция свёртки) – функция, преобразующая массив входных данных произвольного размера в выходную битовую строку (хеш, хеш-значение) определённого (установленного) размера в соответствии с определённым алгоритмом. Выделяют два ключевых свойства хеш-функций:

– хеш-функция способна преобразовать последовательность любой длины в последовательность строго определенной длины.

– преобразование является математически необратимым.

Существует множество алгоритмов хеширования с различными свойствами (разрядность, вычислительная сложность, криптостойкость и т. п.). Выбор той или иной хеш-функции, как правило, определяется спецификой решаемой задачи. Одним из самых простых примеров хеш-функций может служить контрольная сумма файла.

Область применения хеш-функций огромна. Однако, в рамках данной работы, нас интересует прежде всего свойства необратимости преобразований, которое может использоваться для предварительной обработки паролей, с целью последующего их хранения (в защищённом виде).

В силу своих свойств, почти все однонаправленные функции, используемые для получения хеш-значений, обладают недостатком (проявляемый в разной степени в зависимости от хеш-функции) – они подвержены коллизиям. **Коллизией хеш-функции** называются такие два входных блока данных, которые дают одинаковые хеш-значения. В качестве наглядного примера рассмотрим следующий. Пусть дана функция $H(x) = x \bmod 4$, которая определена на множестве целых чисел. Её область значений состоит из 4 элементов (кольца вычетов по модулю 4), а область определения – бесконечна. Так как множество прообразов заведомо больше множества значений, коллизии обязаны существовать и, как можно заметить, существуют. Так, для $x = 5$ и для $x = 8$ значение хеш-значения будет одинаковым, $H(x) = 1$. Таким образом, пара входных значений (5 и 8) образуют коллизию хеш-функции. Конечно, хеш-функции, используемые в реальных условиях намного более сложные и существенно менее подвержены коллизиям, но всё же подвержены.

ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ

Разработайте безопасное *web*-приложение с любой бизнес-логикой (функциональностью), нативно реализующее механизмы аутентификации, авторизации и управления доступом (посредством ролевой модели управления доступом) согласно схеме взаимодействия клиент-сервер, представленной на рис. 4.7. Безопасность приложения заключается в защитных мерах, реализованных непосредственно в приложении, способных нивелировать угрозы со стороны злоумышленника.

Рис. 4.7. Схема взаимодействия *web*-приложения и потенциал злоумышленника

Особенности и требования к web-приложению следующие:

– приложение взаимодействует с пользователем посредством браузера и открытого канала связи, основанному на протоколе *http*;

– на стороне *web*-сервера допускается использование любых языков программирования. На стороне клиента допускается применение скриптов, в том числе фреймворков;

– *не допускается* как на стороне сервера, так и на стороне клиента использование готовых фреймворков, библиотек и/или алгоритмов, непосредственно реализующих элементы задания практической работы (за исключением библиотек с криптографическими функциями);

– *web*-приложение может функционировать под управлением любой операционной системы, а также может использовать любую систему управления базами данных;

– потенциал нарушителя позволяет ему свободно слушать открытый канал связи между клиентом и сервером приложения, а также осуществлять взаимодействие с сервером (отправлять ему любые запросы). Кроме того, потенциал нарушителя позволяет ему украсть *cookies* пользователя (не важно каким именно способом);

После выполнения работы и проверки адекватности полученных результатов, необходимо подготовить отчёт, включающий в себя:

– титульный лист;

– задание (согласно вашему варианту или индивидуальному заданию, выданному преподавателем);

– подробное описание разработанного *web*-приложения, в том числе описание всех разработанных мер по защите информации;

– контрольные примеры работы *web*-приложения в режиме аутентификации, авторизации и непосредственно функционала. Для демонстрации запросов клиентов и ответов сервера можно воспользоваться специализированным программным обеспечением (например, Burp Suite) или специальным режимом браузера (*web*-разработка), например, скриншотами со схожей информацией (рис. 4.8)

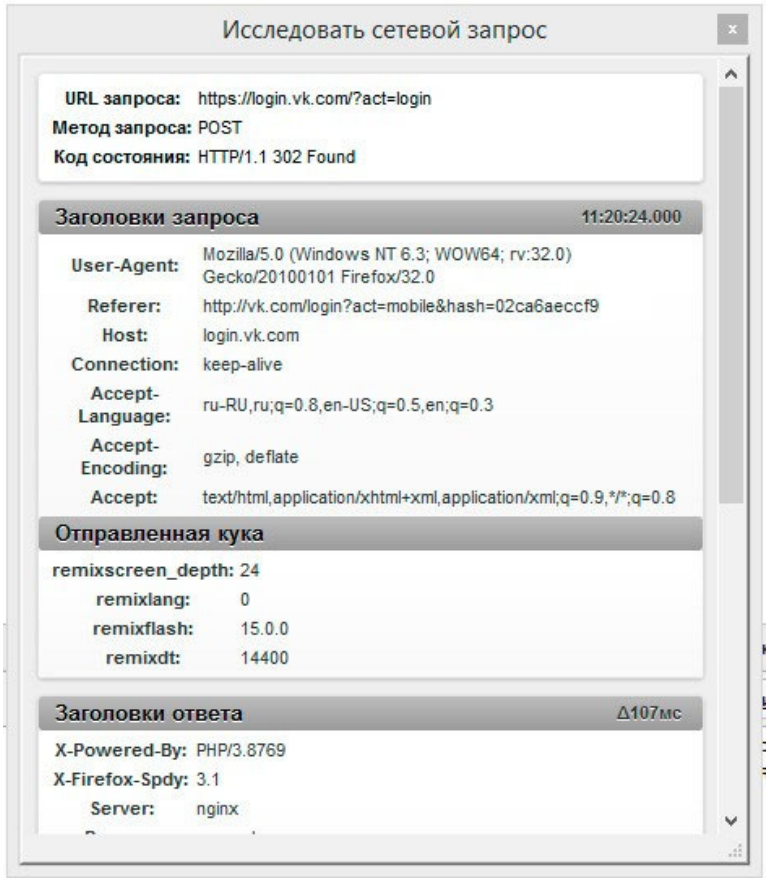

Рис. 8. Информация о запросе к *web*-ресурсу.

– листинг клиентской и серверной части *web*-приложения (шрифт `Consolas`, 10 кегель);

– заключение, содержащее выводы о безопасности, разработанного приложения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

– Что такое аутентификация и авторизация?

– Назовите и опишите базовые модели управления доступом?

– Назовите способы аутентификации в *web*-приложениях?

– Виды односторонней и двухсторонней аутентификации в *web*-приложениях?

– Что такое аутентификация с единым входом (*SSO*)?

– Опишите протоколы аутентификации *OpenID* и *OAuth* 2.0?

– Опишите протокол Диффи-Хеллмана для выработки общего секретного ключа?
– Что такое хеш-функция?
– Какие области применения хеш-функций в *web*-приложениях?

ОСНОВНАЯ ЛИТЕРАТУРА

1. Клод Шеннон. Теория связи в секретных системах / пер. с англ. В. Ф. Писаренко // Работы по теории информации и кибернетике : [пер. с англ.] / Под редакцией Р. Л. Добрушина и О. Б. Лупанова. — М. : Издательство иностранной литературы, 1963. — 829 с.

2. Коханович, Г.Ф. Компьютерная стеганография: теория и практика / Г.Ф. Коханович, А.И. Пузыренко. К.: МК - Пресс, 2006. – 288с.

3. Стеганография, цифровые водяные знаки и стеганоанализ, Монография, Аграновский А.В., Балакин А.В., Грибунин В.Г., Сапожников С.А., 2009