

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

Николай Жишкевич

1. Введение. Сферы применения ЭЦП и сертификатов

Следующим шагом нашего рассказа будет использование криптографических алгоритмов для защиты информации, передаваемой через Интернет, это то, с чего я и начинал свое повествование в прошлой статье.

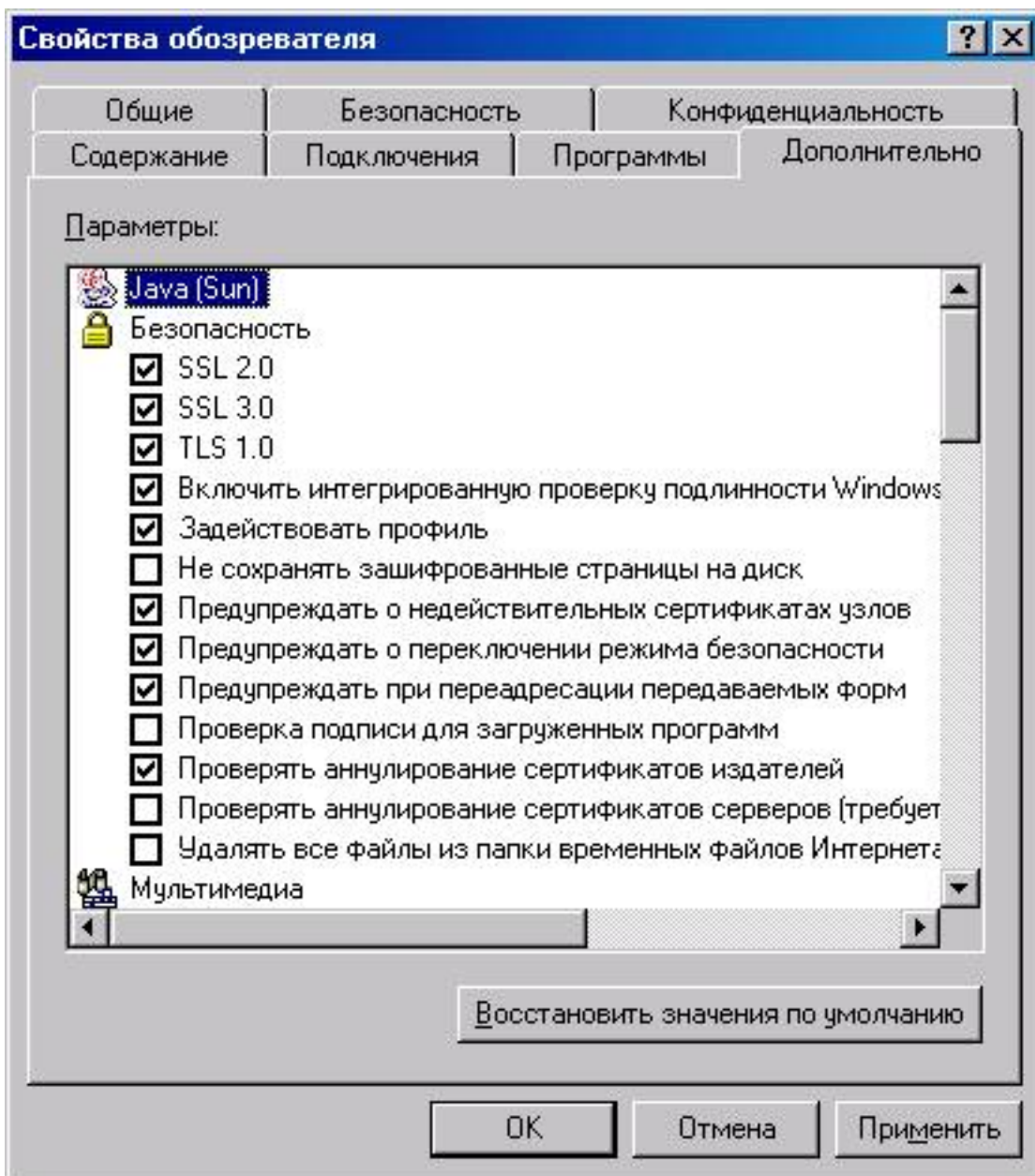
В общем случае систему защиты электронных коммуникаций и информации можно условно разделить на 4 направления:

- защита информации на стороне клиента;
- защита информации на стороне сервера;
- защита линий связи, соединений. Подразумевается что злоумышленник не может подключиться к линии передачи информации. Очевидно, что в случае Интернет это условие технически не осуществимо.
- Защита информации на пути от отправителя к адресату. В данном случае подразумевается использование алгоритмов шифрования. А для защиты от подмены и фальсификации передаваемой информации используют электронную подпись.

При обмене информацией сейчас приоритетно используется протокол http, в этом случае информация передается в открытом виде. Разумеется, для ответственных применений этот протокол не применим, вместо него следует использовать https. Чисто зрительно особых отличий при посещении страниц через протокол http и https вы не увидите. Разве что, в адресной строке браузера вы увидите не <http://xyz.com>, а

Практическая криптография в Java. Асимметричная криптография, алгоритм RSA

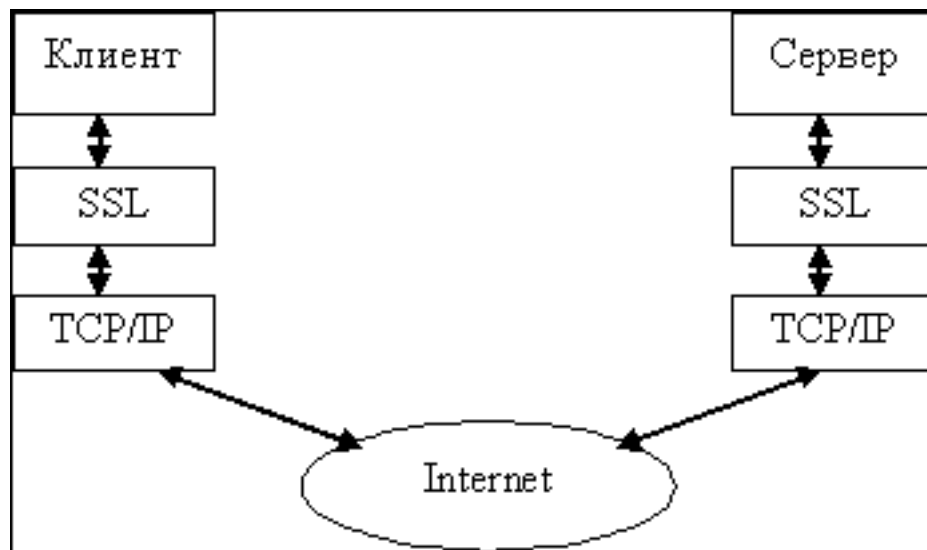
<https://xyz.com>, и еще внизу окна браузера в строке состояния вы также заметите значок сертификата. Настройками браузера при посещении защищенных сайтов можно управлять с помощью следующего диалогового окна.



1.1. Основные пункты и их расшифровка

- SSL2 (Secured Sockets Layer Level 2). Один из самых первых протоколов, поддерживается всеми безопасными веб-узлами. Когда клиент соединяется с защищенным сайтом то между им и сервером происходит процесс договора о том как будут передаваемые данные шифроваться в данном случае в ходе этого клиент должен будет принять (доверять) сертификату который есть у сервера.
- SSL3 (Secured Sockets Layer Level 3), данный протокол является следующим шагом развития SSL2 и обеспечивает большую степень защиты. И соответственно, как более новый протокол возможно что некоторые сайты его не поддерживают.
- TLS (Transport Layer Security), открытый стандарт обеспечения защиты, аналогичен протоколу SSL3 (Secure Sockets Layer). Данный протокол считается самым надежным.

Протокол SSL был разработан Netscape в 1994 году и его место в обмене информацией можно представить в виде следующей схемы.



Необходимо понимать, что слой SSL находится между прикладными протоколами, такими как http и другими; и слоем протоколов tcp/ip. И как принято в модели OSI вышестоящие уровни не догадываются о том, что на самом деле делают с передаваемой информацией на более нижнем уровне. Таким образом программе, которая общается по сети вовсе не обязательно иметь представление и поддержку

протоколов SSL. Чтобы установить соединение, браузер и сервер должны сначала идентифицировать себя. На этом этапе каждая сторона проверяет партнера до того как соединение будет установлено. После того как это произойдет, обмен информацией пойдет сплошным потоком, который будет шифроваться.

Безопасная передача информации в сети с помощью браузеров предполагает использование следующего набора средств:

- Электронная подпись
- Шифрование
- Проверка подлинности

Для решения всего этого набора задач используется шифрование с открытым ключом (ассиметричная криптография). Основное ее отличие от симметричной в том, что для расшифровки информации и ее шифрования используются два различных ключа. Вспомните, в решенной шагом ранее нами задаче с шифрованием информации с помощью алгоритма XOR для шифрования и расшифровки использовалось одно число. На данный момент существует большое количество отлаженных и надежных алгоритмов симметричного шифрования. Однако следует помнить, что криптосистема это не только алгоритм шифрования/дешифрования, но и управление ключами, которое включает в себя следующие действия:

- Генерацию ключей;
- Распределение ключей;
- Хранение ключей;
- Уничтожение ключей.

Распределение ключей в криптосистеме является самым сложным и тяжелым. Дело в том, что для безопасной передачи подобной секретной информации требуется наличие защищенного канала или иного способа, однако тогда возникает смежный вопрос: «А почему бы не воспользоваться этим способом и вместо передачи ключа не передавать саму информацию, и не забивать голову всякими криптографическими преобразованиями». Одной из сложнейших проблем криптографии также является проблема проверки принадлежности секретного ключа конкретному лицу, сайту, или организации. Данная проблема заключается в том, что необходимо иметь уверенность, что то лицо, с которым мы хотим обмениваться информацией, разумеется, зашифрованной, действительно то, за которое оно себя выдает.

2. Использование ЭЦП для подписи распространяемого программного обеспечения

Разумеется, что ЭЦП и методы двухключевой криптографии могут быть использованы не только для защиты сетевых коммуникаций, но и во всякой сфере, в которой возникает необходимость обеспечения гарантии не нарушения некоторой информации и кода; есть необходимость удостовериться в наличии определенных прав у пользователя. ЭЦП имеет особенно большую роль именно в связи с распространением Интернет. На сайте VeriSign есть документ в котором идет достаточно подробное объяснение о преимуществах получаемых в тех случаях, когда мы подписываем с помощью ЭЦП распространяемое ПО.

2.1. Зачем нужно подписывать программный код?

Когда вы покупаете программное обеспечение в магазине, вы получаете также подробную информацию о том, кто его произвел. И в случае необходимости вы можете обращаться к поставщику данного решения за помощью и разрешением возникших вопросов. Пользователь делает выбор будучи уверен в том, что компания разработчики потратила достаточно сил на то чтобы сделать качественный и безопасный продукт. Покупатели могут доверять, они знают, что в случае возникновения проблем они получают техническую поддержку и охотно делают покупки.

В отдельных ситуациях бывает более целесообразно распространять программное обеспечение через Интернет. Однако все вы слышали, что в сети много злых вирусов и плохих хакеров которые только и делают, что спят и видят, как украсть ваш номер кредитной карточки. Не забывайте, что в сети вы вынуждены воспринимать многое на веру, и возможно продукт XYZCad который вы загружаете с сайта известного поставщика САД решений XYZcorp. на самом деле не суперклассная программа для рисования графиков и чертежей, а злой троянский конь, а компании XYZcorp. вообще не существует, на самом деле это хакерская группа Crack&Hack ltd. подделала сайт или взломала, подменила страницы, ну и мало еще, что можно сделать. Одним словом у вас нет уверенности в вашей безопасности при использовании программного обеспечения загружаемого из сети.

Когда пользователь загружает страницы Интернет, он часто видит сообщения о том, что для корректного отображения данной страницы необходимо скачать и установить замечательный plugin ABCPlug, который также умеет варить кофе и протирать пыль на вашем рабочем столе. Не знаю как вы, а я таким сообщениям не верю. В отдельных ситуациях, однако, подобный плагин или компонент или Java applet следует и нужно установить. Например, мне по специфике своей деятельности иногда приходится отслеживать ситуацию на финансовых рынках Forex. Я прихожу на сайт www.akmos.com (без рекламы, я просто излагаю факты) и загружаю Java applet который потенциально (чтобы там не говорила Sun) может делать всякие гадости на моей машине (в основном конечно из-за дыр в Java VM для Windows). А когда а вижу сообщение о том, что данный апплет был подписан серьезной организацией вроде VeriSign, то начинаю немножко доверять.

Кроме того следует помнить, что исторически сложилось, что Java applets жили внутри песочной коробки и всякая попытка выйти за ее пределы завершалась убиением бедной программы. Когда я только начинал разрабатывать для Java, то меня остановили эти жесткие ограничения: нельзя обращаться к файловой системе, нельзя соединиться с другим сервером за исключением того, с которого applet был загружен и прочее. А если действительно надо, тогда приходилось придумывать обходные решения, которые, разумеется, были неэффективными и некрасивыми. К счастью, позже появилась идея подписи applets с помощью ЭЦП, и applet имеющий подобную подпись, и которому доверял пользователь получал дополнительные привилегии. Кто знает, если бы в самом начале этот момент был бы продуман, может быть Java на рынке applet&web занимал бы более серьезную роль, чем сейчас.

Не стоит также забывать, что для отдельных приложений не допустимы ошибки или погрешности в работе. Представим ситуацию, что мы используем программу XYZcalc внутри которой хранится некоторый ряд заранее вычисленных значений используемых для расчетов. А теперь представьте себе, что при загрузке по сети произошел сбой и один байт (даже один бит) был искажен. Вроде бы программа запускается и работает правильно, но в один замечательный момент ... Вы скажете, что в такой ситуации следует проверять CRC, я скажу что вы совершенно правы, однако следует понимать, что подобные средства могут защитить лишь в случае случайного искажения, а если мы имеем дело с целенаправленной атакой извне, то возможно провести такую модификацию программно кода, когда значение CRC «до» и «после» совпадут.

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

Единственным решением будет создание ЭЦП для CRC.

Благодаря использованию ЭЦП и сертификатов поставщики ПО могут добавить к нему информацию о самих себе и дать гарантию того, что именно они поставили данный продукт и гарантируют его качество.

Когда пользователь загружает ПО, которое имеет ЭЦП, то он может быть уверен в том, что:

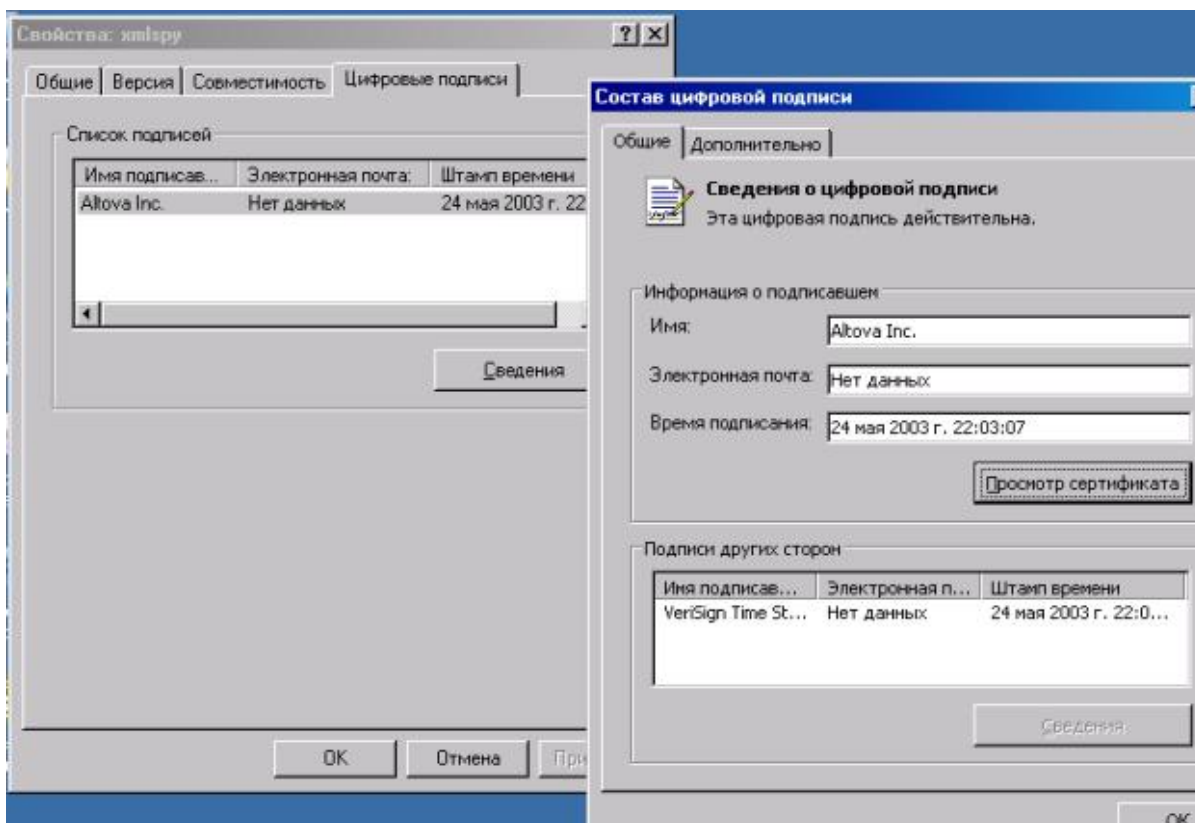
- Программный продукт был разработан именно той организацией, которая это говорит.
- Также можно быть уверенным в том, что программный продукт не был целенаправленно изменен или случайно поврежден с тех пор, как был подписан.

И кроме того не забывайте, что существуют определенные юридические аспекты использования ЭЦП. Если по вине программы были получены некоторые убытки а в заявлении EULA было сказано, что компания гарантирует и возмещает потери, то при попытке этой организации отказаться, заявив: «что я не я и собака не моя» (дословно, я не знаю откуда вы достали эту странную программу мы никогда ничего подобного не видели и не делали, все это происки злых хакеров), у покупателя будет возможность обратиться в суд и выиграть дело.

Разработчики также получают массу преимуществ от использования ЭЦП. Дело в том, что подписывая программный продукт, они строят доверительные отношения с пользователями и усложняют или делают невозможной подделку и фальсификацию своих программ или материалов на сайте. Только представьте себе возможные последствия, если на сайте серьезной аналитической организации по рынку hitech появится заявление, что компания XYZcorp. за прошедший квартал получила 100 миллионов убытка и ведутся переговоры о начале процедуры банкротства. Даже если через несколько часов обнаружится, что на самом деле была подмена и подлог со стороны конкурирующей компании ZYXcorp. нанявшей группу хакеров Trust&Hack ltd., для бедной компании XYZcorp. уже действительно можно начинать процедуру банкротства на самом деле. Конечно, таким способом мамонта вроде IBM||Microsoft не завалить, но для маленьких компаний это может быть смертельно.

Итак, серьезное программное обеспечение уже давно идет вместе с ЭЦП. Идут попытки создания аппаратных платформ, отказывающихся запускать софт которые не

был разрешен политикой компании, и в качестве критерия здесь выступает сертификат, прикладываемый к данному ПО. Вспомните Microsoft и ее XBox. Сертификаты применяются сейчас и при распространении ПО; если раньше для защиты программного продукта от модификации и подделки на сайте иногда выкладывали MD5-сигнатуры. То теперь уже в инсталляторе программы, как его свойства, находится информация о разработчике и сертификат, которым подтверждается данный продукт. Так, недавно скачав из Интернета XmlSpy5, я обнаружил в его свойствах закладку специально отвечающую за это.



3. Основные понятия асимметричной криптографии

В случае использования асимметричной криптографии все участники процесса обладают двумя ключами, связанными между собой хитрым математическим соотношением. Эти два ключа называются соответственно открытым и закрытым

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

ключами. Всякая информация, зашифрованная с помощью открытого ключа, может быть прочитана только владельцем закрытого ключа. Таким образом, главный недостаток симметричной криптографии, заключающийся в сложности обмена между участниками секретным ключом, уже исчезает. В действительности необходимо только опубликовать на общедоступном сайте свой открытый ключ, а закрытый спрятать и никогда не разглашать. Всякий кто хочет послать нам письмо, и при этом не хочет, чтобы его прочитал кто-нибудь кроме нас, должен будет воспользоваться открытым ключом, а получатель сообщения расшифрует данное письмо закрытым ключом. Разумеется, требуется, чтобы на основе известного открытого ключа было бы вычислительно невозможно найти секретный ключ. Для наилучшей иллюстрации хочется привести пример из одной очень хорошей книжки, не по криптографии правда, а скажем так, общего руководства по сети Интернет “Using Internet” Джерри Хонниката.

«При ассиметричном методе (иначе шифровании с открытым ключом) используются два ключа — секретный и открытый. Получается нечто вроде сейфа со щелью в крышке. Поместить туда документ может кто угодно, достать же — только тот, у кого есть секретный ключ»

Для некоторых алгоритмов открытый и закрытый ключи являются взаимно заменяющимися, например алгоритм RSA.

Зачастую имеет смысл совместно использовать методы симметричной и ассиметричной криптографии.

4. Направления применения методов ассиметричной криптографии

Существуют различные способы нападения на криптосистему и соответственно направления тех усилий, которые мы затрачиваем, чтобы не допустить ее компрометации:

- **Защита информации от несанкционированного чтения, обеспечение секретности информации.** По сути, говоря это то, что большинство понимает под словом защита информации: преобразование информации таким образом, чтобы прочитать ее можно было только законному обладателю некоторой секретной

информации о способе преобразования — ключа.

- **Защита от навязывания ложных сообщений.** Часто важным является необходимость убедиться в том, что пришедшее письмо не было искажено случайно, например, в результате сбоев, помех или иных случайных факторов, а также что ни какой злоумышленник не смог внедриться в нашу криптосистему и отправлять сообщения от имени другого участника. Если в первом случае, когда искажение было внесено случайно, еще можно анализируя его текст догадаться о подмене, то во втором варианте, когда действовал злоумышленник, или передавалась цифровая информация, анализ текста с точки зрения семантики нам уже не помогает и требуется более эффективное средство оценки. На основе информации вычисляется специальный небольшой код, который должен зависеть от всех байтов передаваемой информации и секретного ключа. Очевидно, что чем больше длина подобного кода (обычно его называют имитовставка), тем с большей долей вероятности самые незначительные изменения в исходном документе можно будет отследить. Разумеется, что 100% гарантии нельзя дать теоретически, но на практике подобный подход имеет право на жизнь. Для того, чтобы злоумышленник после модификации исходного сообщения не вычислил новое значение имитовставки и не подменил ее оригинал, имитовставка рассчитывается также на основе и секретного ключа отправителя. Когда адресат получает сообщение, он повторно вычисляет имитовставку и проверяет ее совпадение с оригинальной, используя открытый ключ отправителя.
- **Аутентификация пользователей.** Как узнать, что некоторый клиент нашей системы является кем-то, и имеет соответствующий набор прав? Общепринято, что законный пользователь системы должен знать некоторый секрет, или пароль, сообщить его системе проверке и она, сверив данный секрет с тем, что хранится в ее базе данных выдаст пользователю соответствующие права. А теперь давайте рассмотрим сугубо практический вопрос как именно этот пароль будет храниться и передаваться от клиента к серверу. К сожалению, протоколы Интернет открыты по своей природе и перехватить подобный пароль не составит большого труда. Более того, опасно хранить пароль в открытом виде и на сервере, ведь злоумышленник может легко их похитить и использовать для себя. На сервере хранится не открытый пароль, а, внимание, его образ, или выражаясь более формально, значение некоторой односторонней функции $y = f(x)$, где x — секретный пароль. К данной функции f предъявляется множество требований, но самым

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

главным является то, чтобы она была не обратима, т.е. по значению y невозможно было бы за приемлемое время вычислить значение x . Таким образом, схему аутентификации пользователя можно представить следующим образом:

- клиент подает запрос на соединение с сервером и называет свое имя;
- сервер передает приглашение на ввод пароля;
- клиент вводит пароль, который используется для вычисления значения односторонней функции $y = f(x)$, значение которой и отправляется серверу;
- сервер сверяет образ y с тем, что хранится в его реестре и наделяет пользователя соответствующими правами или сообщает об ошибке.

А, следовательно, при передаче информации по открытому каналу пароль не передается и соответственно не может быть перехвачен и использован злоумышленником. Однако есть недостаток и у подобной системы, и заключается он в том, что злоумышленник может перехватить посылаемый ключ и через некоторый промежуток времени выдать себя за законного пользователя, повторно отправив перехваченный образ пароля. С целью противодействия данной атаке часто вместе с паролем, точнее его образом, посылается текущее время или находят иной способ привязать запрос и ответ клиента к текущему моменту времени. Однако без использования ассиметричных алгоритмов сделать это достаточно тяжело.

На практике применяют следующую схему, которая является развитием ранее приведенного примера:

- клиент посылает запрос на соединение с сервером;
- сервер генерирует случайное число R и сообщает его клиенту;
- клиент шифрует данное случайное число R с помощью секретного ключа x и отправляет серверу $C = f(x, R)$;
- сервер повторно вычисляет значение $C = f(x, R)$ и сверяет с тем, что ему прислал клиент. Если присланное значение совпадает, то пользователь наделяется соответствующими правами.

Данный алгоритм будет работать даже в том случае, если злоумышленник сможет перехватывать посылаемые сервером клиенту случайные числа и ответы клиента серверу. Если в алгоритме генерации случайных чисел нет явных ошибок и период повторения серий чисел достаточно велик, то вполне возможно, что лишь спустя очень большой промежуток времени снова к клиенту будет послано случайное число, совпадающее с тем, что было отправлено ранее. Можно в развитие данного способа

рекомендовать отправлять не просто случайное число, но и текущее время. Тогда стойкость данной системы будет определяться стойкостью алгоритма генерации открытого и закрытого ключа.

- **Аутентификация информации** — необходимо иметь возможность удостовериться в том, что сообщение было послано именно оговоренным отправителем и не было отправлено злоумышленником. Решение этой задачи, как и предыдущей, основано на использовании электронной цифровой подписи. ЭЦП основывается на двухключевой криптографии. ЭЦП гарантирует не только целостность информации, но и играет роль подписи, означающей то, что отправитель прочитал и согласен с информацией содержащейся в письме. Наиболее известные алгоритмы ЭЦП — RSA и DSS, разработанный в рамках NIST. Важно понимать, то, что с помощью ЭЦП можно удостоверить все что угодно. Документы, программы, страницы html, даже бумажные документы. И это дает просто потрясающие перспективы.

5. Проблемы двухключевой криптографии и способы их решения

Несмотря на все преимущества, которые мы выделили, ассиметричная криптография не решает всех проблем. Остается не решенной проблема «первого контакта». Предположим, что два друга Вася и Петя хотят обмениваться секретной информацией, которую они решили шифровать с использованием средств ассиметричной криптографии. Как Васе быть уверенным в том, что открытый ключ, который он получил по почте или нашел на общедоступном сайте принадлежит действительно Пете, а не злоумышленнику Саше. Для решения данной проблемы служат сертификаты, которые устанавливают соответствие криптографических ключей, используемых для обмена информацией, конкретным людям или организациям. Мне очень нравится пример из одной книжки: *«Сертификат подобен гравировке на ключе от дома с именем его хозяина. Сертификат это удостоверение того, что данный ключ принадлежит конкретному человеку»*. Подлинность используемого сертификата подтверждается с помощью ЭЦП, ведь сертификат — просто файл с информацией, и к нему также применимы все те рассуждения, которые мы вели о документах подписываемых ЭЦП. Следовательно, мы можем проверить каждый сертификат на подлинность. Благодаря наличию ЭЦП невозможно подделать сертификат.

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

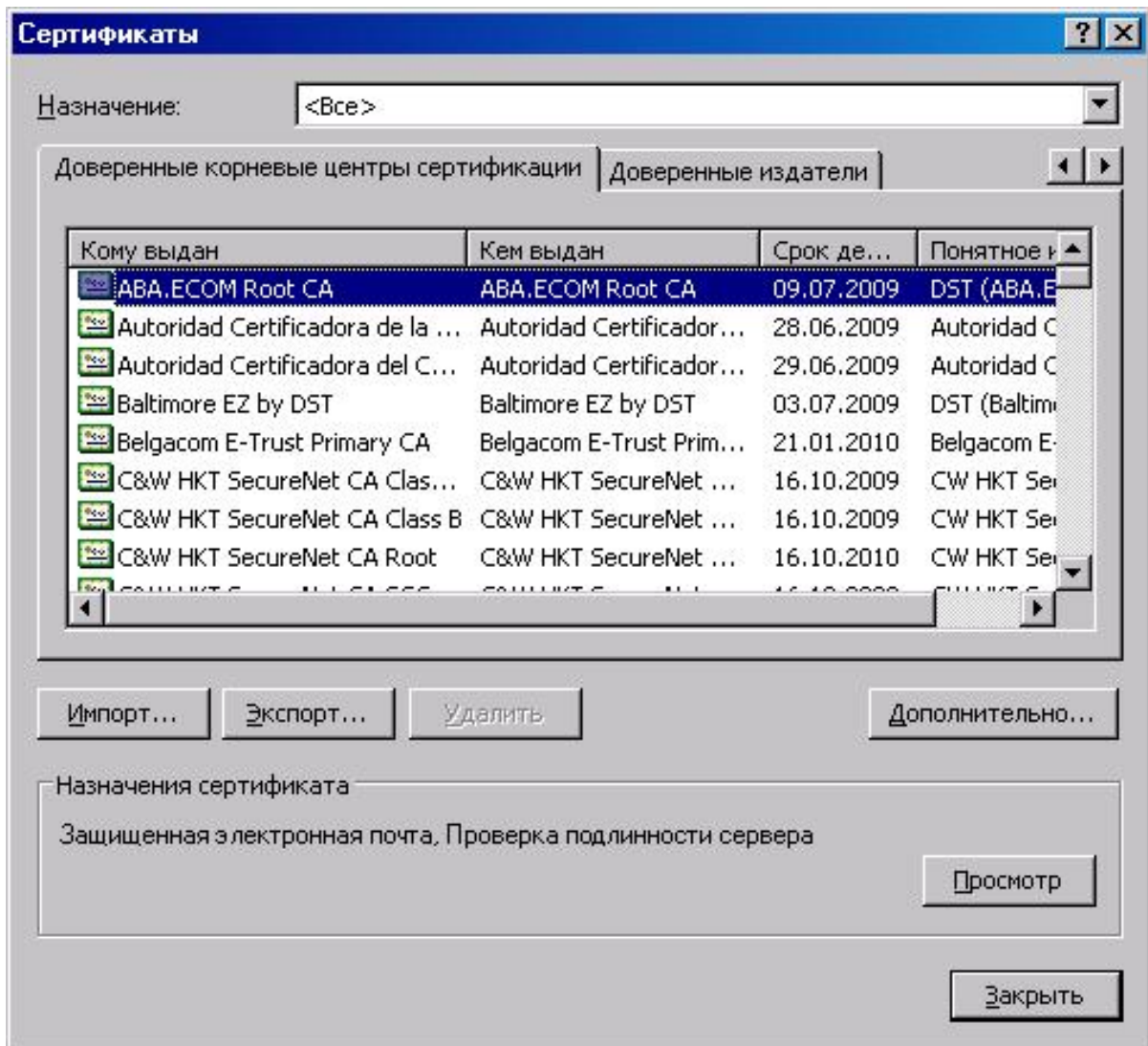
Сертификаты создаются и подписываются специальной организацией (CA — certificate authority). Если вы доверяете данной организации, то автоматически должны будете доверять и тем, кому доверяет CA. В сертификате содержится набор следующей информации.

- открытый ключ для проверки цифровых подписей или шифрования информации отправляемой издателю ключа;
- название организации, имя частного лица которому был выдан данный сертификат;
- информация о той организации, которая выдала сертификат;
- информация о сроке действия сертификата.

Обычно сертификаты делят на те, которые предоставляются:

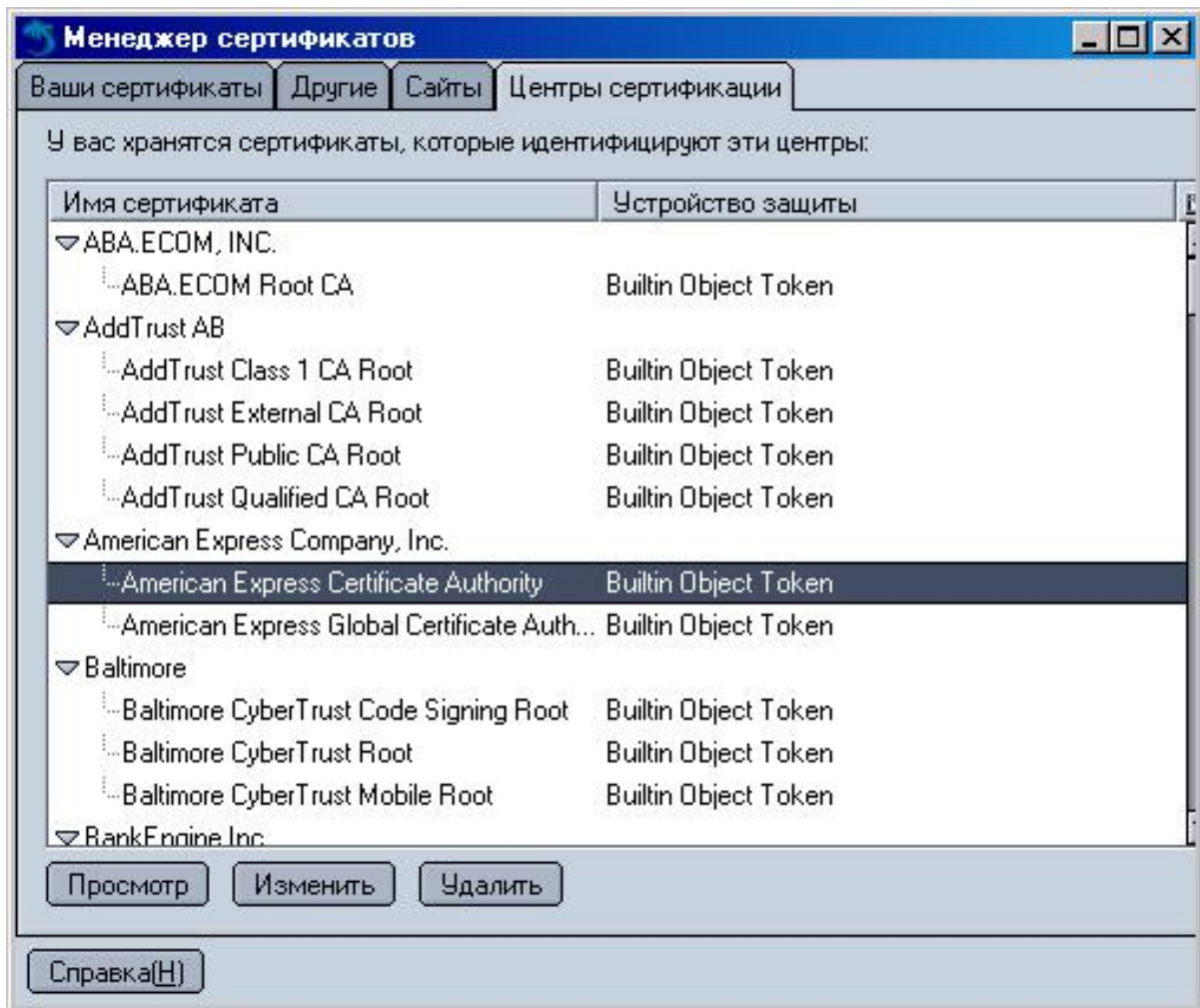
- поставщикам сертификатов;
- личные сертификаты лиц;
- сертификаты узлов Интернет;
- сертификаты, используемые поставщиками программного обеспечения для обеспечения надежного распространения своих продуктов.

Вы можете просмотреть список сертификатов каждого вида установленных на вашем компьютере и используемых вашим браузером с помощью следующего окна в настройках обозревателя IE.



Подобную возможность предоставляют также и другие браузеры. В качестве демонстрации приводятся изображения настроек Mozilla.

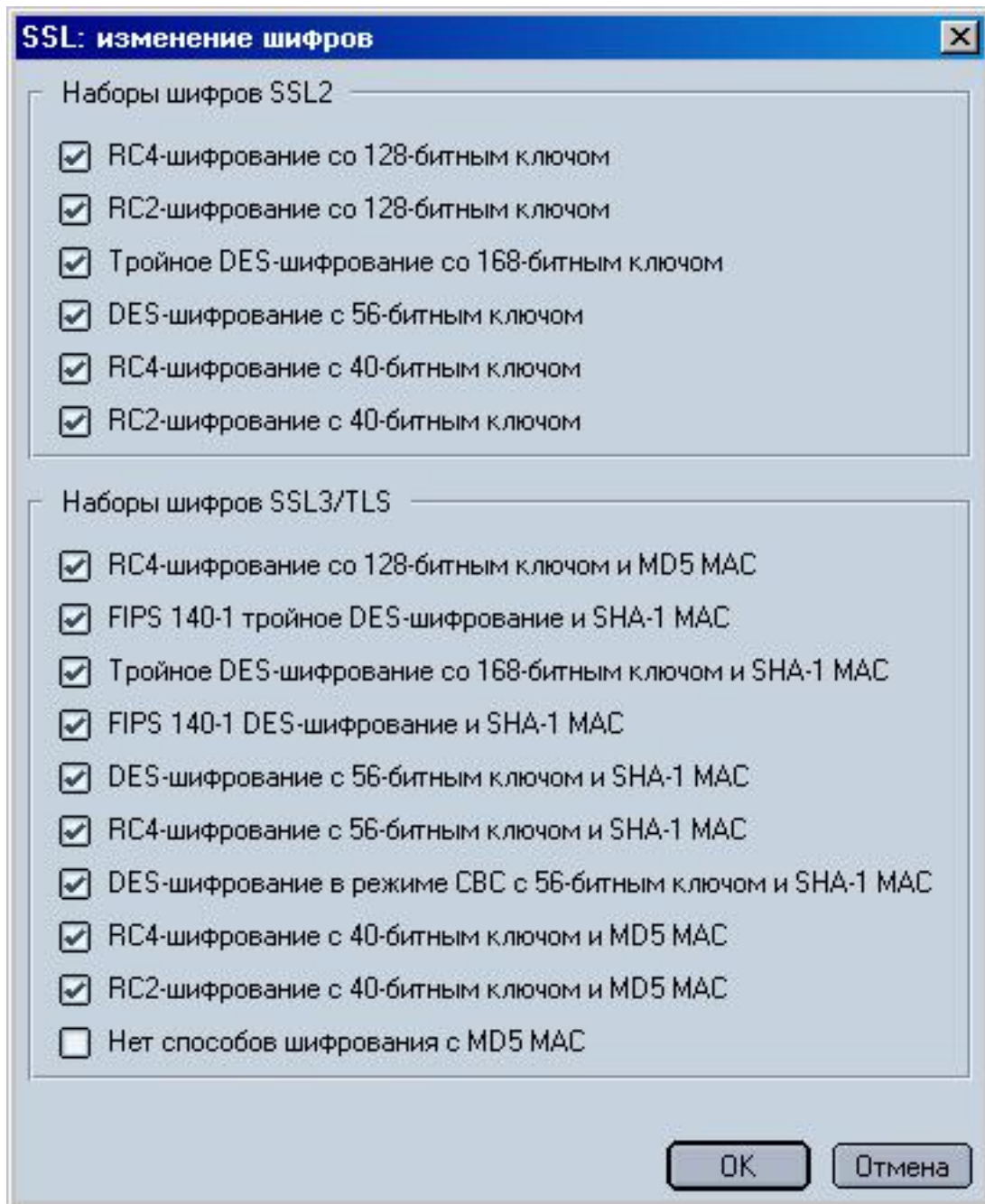
Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA



Каждый сертификат имеет определенное содержимое и может использоваться для целого круга задач.

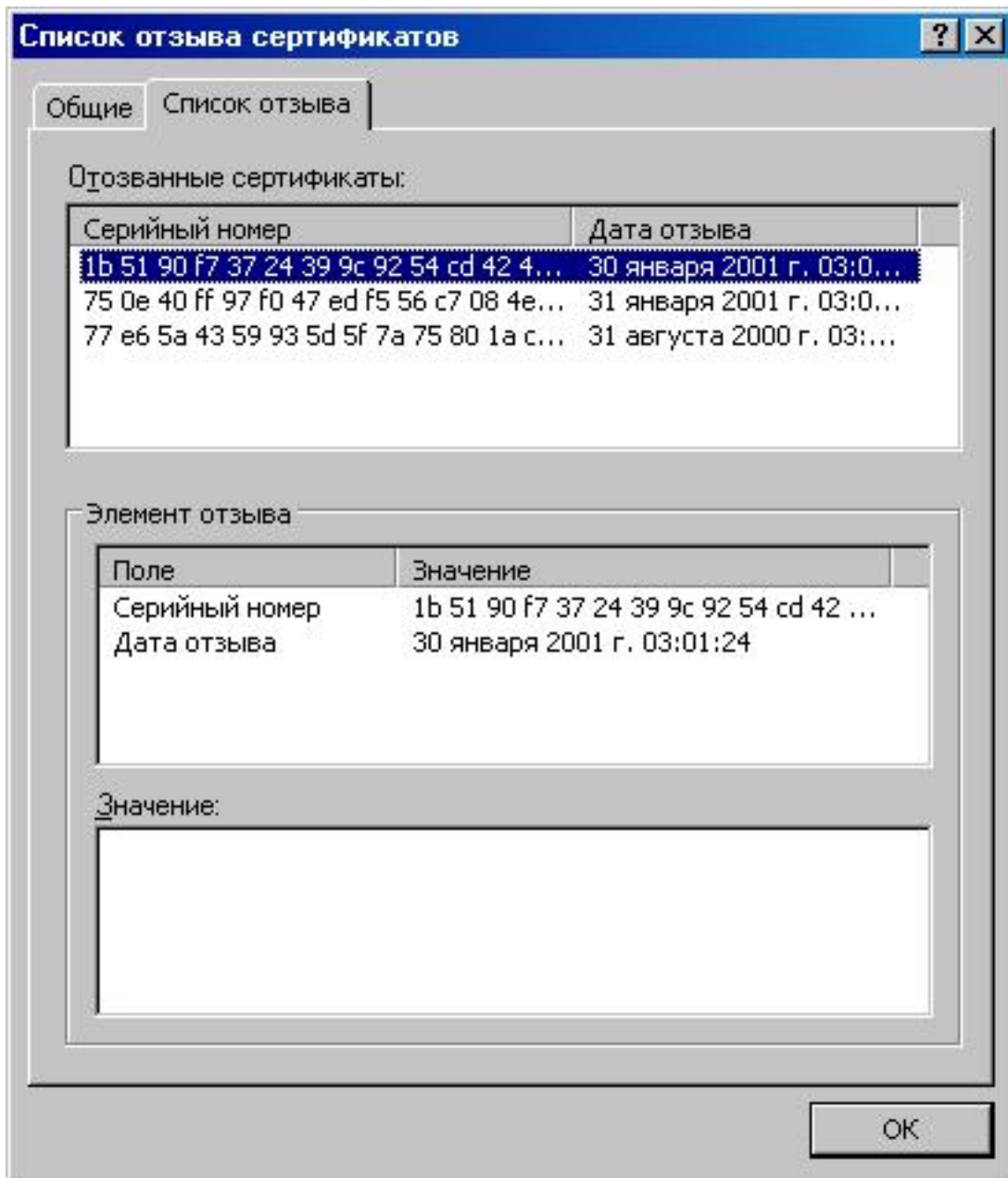


Протоколы SSL2 , SSL3, TLS представляют собой набор алгоритмов для вычисления сверток, шифрования. Просмотреть и настроить список используемых алгоритмов в сюите протокола можно с помощью окна настроек.



Для того чтобы получить сертификат следует обратиться к поставщикам CA. Наиболее известный поставщик это VeriSign. Очевидно, что сертификаты не только выдаются,

но и отзываюся. Поставщики СА периодически публикуют списки сертификатов, которые стали недействительны. Например, если вы предоставили службе сертификации (СА) неверные сведения, и об этом через некоторое время узнала ваша служба сертификации, то она имеет полное право на отзыв сертификата. Разумеется, что СА не может по мановению волшебной палочки физически уничтожить сертификаты на всех машинах во всем мире, а должна просто публиковать специальный список, в котором содержится перечисление тех сертификатов, которые более не должны считаться действительными, и, следовательно, доверять более его предъявителю не следует.



Итак, давайте подведем некоторый итог. Говоря на самом наглядном уровне, сертификаты представляют собой средство для идентификации клиента и сервера при установлении сообщения. Сертификаты это часть идеи SSL, именно благодаря SSL,

TLS возможно такое явление как электронная коммерция. Говорить о ее важности для современной экономики я не буду. Очевидно, что она очень велика. Итак, благодаря SSL&TLS&Certificates мы можем устанавливать безопасное соединение для передачи по нему конфиденциальных сведений (например, номера кредитных карт, или любой другой секретной и конфиденциальной информации). Рассмотрим более подробно процесс того, что происходит когда клиент запрашивает у сервера некоторую услугу. Перед тем как начнется обмен информацией, и клиент и сервер должны удостовериться в том, что их собеседник именно тот, за которого он себя выдает. Подобный процесс называется проверкой подлинности. Внутри сертификата, как вы помните, находится также ключ, который будет использоваться при пересылке информации для ее шифрования.

Следовательно, если вы запросили информацию по протоколу HTTPS, и вы доверяете серверу (точнее, тому сертификату, который у него есть), то вся информация будет при пересылке шифроваться. И, следовательно, хотя злоумышленник может перехватить эту информацию, но далее против него будет играть вся теория криптологии. Надеюсь, вы помните, что при пересылке информации с помощью протокола HTTP информация пересылается в сыром виде, т.е. она открыта для постороннего просмотра. Именно в SSL, я подчеркиваю SSL, обычно выделяют два вида сертификатов — это сертификат клиента и сертификат сервера. Каждый из этих двух сертификатов хранит некоторый объем персональной информации. Когда клиент просит услугу у сервера, он может предоставить подобный сертификат серверу в подтверждение того, что он имеет право на использование некоторого ресурса. Подобный подход является вполне приемлемым в большинстве прикладных систем, и заведомо лучше ранее принятой общепотребительной практики, когда для входа в систему клиент вводил на специальной вебформе пароль, который потом посылался (разумеется, в открытом виде) серверу. Как вы догадываетесь, перехватить и украсть пересылаемый таким образом пароль не составляет сложности. Всякие детские алгоритмы шифрования, которые иногда разрабатывались самим разработчиком, были смешны и практически бесполезны против серьезного взломщика.

6. Сертификаты и защита веб-серверов

Итак, как я уже отметил, существует два типа сертификатов, используемые в SSL

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

применительно к веб-коммерции. Каждый имеет свой формат и назначение. **Сертификаты клиента** содержат персональную информацию о клиентах, запрашивающих доступ к узлу, что позволяет однозначно их распознать до предоставления доступа к узлу. **Сертификаты сервера** содержат сведения о сервере, что позволяет клиенту однозначно идентифицировать сервер до передачи важной информации.

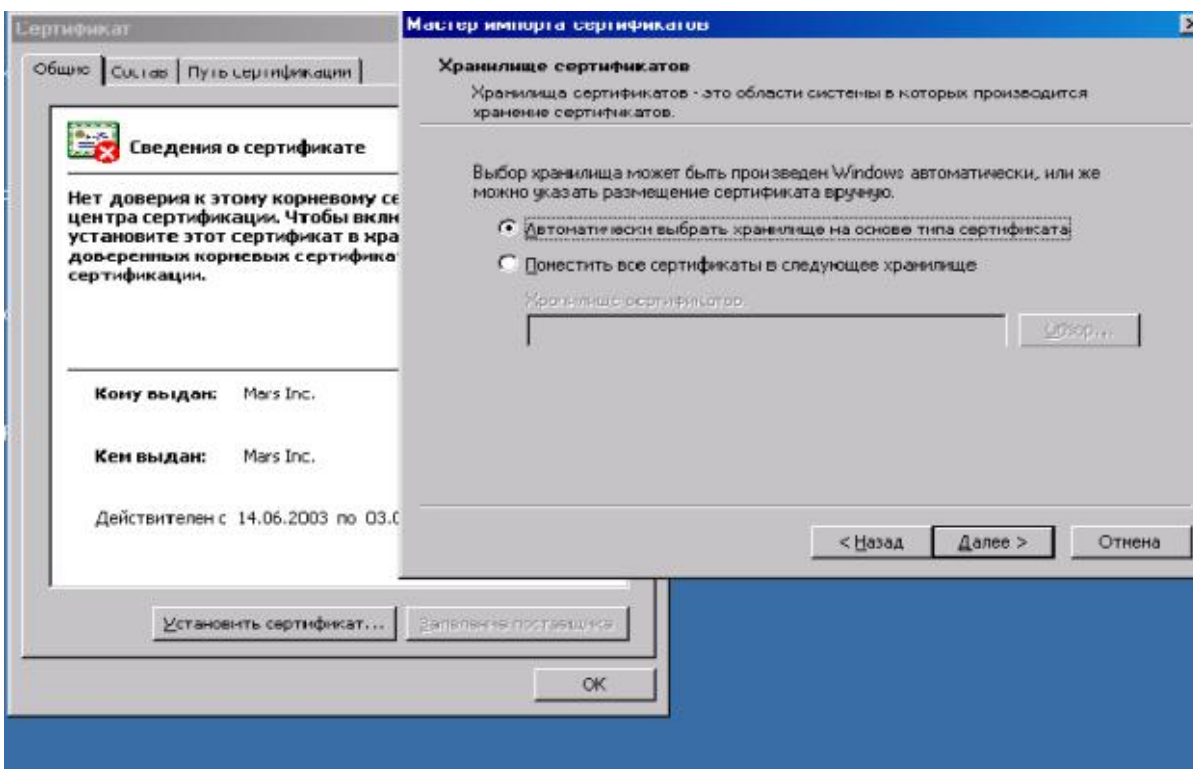
Любой веб-сервер, для того чтобы начать обслуживать входящие защищенные соединения, должен сначала получить данный электронный документ, с помощью которого клиент может убедиться в надежности программного обеспечения, информации которую клиент загружает с сервера. Когда клиент устанавливает соединение с защищенным сервером то IE проверяет, находится ли сертификат который был предоставлен сервером в списке тех сертификатов, или точнее тех организаций, которым клиент доверяет. Всякий сертификат перед своим использованием должен быть заверен. Выполняет данную задачу специальная организация СА (certificate authority). Разумеется, что сертификаты для СА выдают сами СА. Таким образом, клиент должен изначально доверять некоторому набору сертификатов, обычно подобный список идет вместе с установкой клиентского программного обеспечения или должен быть получен из надежного источника. Затем все сертификаты, которые предоставляют сайты, проверяются на предмет того, какая именно организация выдала данный сертификат, и если эта организация (СА) находится среди списка доверенных, то и сертификату сервера также будет оказано доверие. В отдельных ситуациях такое поведение можно изменить.

Клиент может просмотреть информацию о сертификате, которую представляет сервер и даже если он не доверяет никому из тех, кто выдал данный сертификат, он может начать доверять самому сертификату, просто установив его в хранилище доверенных сертификатов. Разумеется, можно пойти и в обратном направлении: клиент может просмотреть содержимое сертификата и отказаться доверять серверу. Если вы разрабатываете электронный магазин и хотите чтобы клиенты смело предоставляли сведения о кредитных картах и другую конфиденциальную информацию, то вам необходимо получить сертификат. У вас есть два направления:

вы обращаетесь к одному из зарекомендовавших себя служб сертификации, например, VeriSign. Получаете у данной организации сертификат и предоставляете его своим

клиентам. Разумеется, что сертификаты — это не бесплатные безделушки, которые СА рассылает щедрой рукой всем желающим. Вам будет необходимо оплатить денежную сумму, довольно чувствительную (по крайней мере, для 1/6 части суши) и предоставить полный объем информации о себе. В отдельных ситуациях, возможно, вам даже придется встретиться лично с представителями СА.

Если у вас налажены хорошие отношения с клиентами, или их круг невелик, то возможно создание и использование собственного сертификата. В таком случае все ваши клиенты должны будут загрузить ваш сертификат и установить его в системе как доверяемый.



Давайте рассмотрим сейчас задачу от обратного. На сервере хранится некоторый объем секретной информации, и она должна предоставляться не всем посетителям нашего ресурса, а некоторой категории доверенных лиц. В общем случае существуют следующие направления обеспечения проверки подлинности уже не сервера, как ранее, а клиента.

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

- Первый и самый простой прием основан на том, чтобы ничего не проверять. Нечто похожее любят показывать в глупых фильмах про умных хакеров, которые для того чтобы попасть на сверхсекретный сайт, должны на главной страничке нажать мышкой на пиксель с координатами (`rand()`, `rand()`). Насколько наивны подобные фильмы, настолько плох и этот подход. Без дополнительных комментариев.
- Обычная проверка подлинности пользователь должен ввести имя и пароль либо на специальной форме, либо в появляющемся диалоговом окне браузера, при получении им ответа от сервера с заголовком «Не авторизован». В случае Java servlets/JSP это что-то вроде следующего фрагмента кода:

```
response.setStatus(response.SC_UNAUTHORIZED);  
  
response.setHeader("WWW-Authenticate",  
    "BASIC realm=\"protected-area\"");
```

Недостаток подобного способа очевиден. Информация при отправке подвергается лишь простейшему преобразованию base64, и не составляет ни малейшей сложности при использовании снифферов и подобных им перехватчиков и анализаторов трафика сети получить пароль клиента. Помните, что base64 — это не шифрование данных. Преимуществом обычной проверки подлинности является то, что она является частью спецификации HTTP и поддерживается большинством обозревателей. Следовательно, обычный способ проверки подлинности не рекомендуется использовать, за исключением тех случаев, когда есть полная уверенность в безопасности соединения пользователя и веб-сервера (например, выделенная линия или безопасное подключение SSL (Secure Sockets Layer)). В том случае если все запросы проходят через прокси или подобный ему посредник, то можете быть уверены в том, что ваш пароль уже знает не только вы.

- Передача по сети не пароля, а результата некоторого криптопреобразования данного пароля. Например, при вычислении хеша. На сервере храниться не база паролей клиентов, а база хешей этих паролей. Таким образом, даже успешное нападение на сервер не приводит к дискредитации паролей клиентов.

7. Хеш-функции

Для вычисления хеша используется ряд специальных функций. Это так называемые

необратимые хеш-функции. К ним предъявляется целый ряд требований. Хеш-функция представляет собой криптографическую функцию, аргументом которой является сообщение произвольной длины, а значение сложным образом зависит от каждого бита данного сообщения. Обычно хеш-функции представляют собой некоторую итерацию по всем байтам/битам сообщения и выполнения некоторого вычисления. Важно отметить, что, несмотря на требование, чтобы аргумент функции мог быть произвольной длины, к значению накладывается ограничение по длине. Обычно длина результата вычисления хеш-функции не превосходит 128 бит. К хеш-функции предъявляются следующие требования обеспечивающие безопасность ее применения:

- вычислительно неосуществимо найти значение аргумента функции по известному ее значению;
- вычислительно неосуществимо найти два сообщения, не равных друг другу, но для которых значения хеш-функций были бы равны.

Очевидно, что если последнее требование не выполняется, то злоумышленник может легко подделать сообщение для подписанной хеш-функции. Все что для этого нужно это подготовить такой поддельный документ, для которого значение хеш-функции совпало бы с таковым для оригинального сообщения. Следует помнить, что значение хеш-функции вычисляется итеративно. И процесс ее вычисления можно представить по следующей формуле:

$$H_i = FH(H_{i-1}, M_i)$$

В качестве начального значения H_0 принимают специальное число. В качестве хеш-функции имеет смысл использовать не очень сложную функцию шифрования. Помните, что как к хеш-функции, так и к функции шифрования предъявляются достаточно схожие требования. Невозможность вычисления на базе известного значения результата вычисления данной функции значения ее аргумента предъявляется и к функции хеша и к функции шифрования.

Давайте более подробно рассмотрим характеристики, которыми обладают алгоритмы вычисления хешей.

Прежде всего, следует четко понимать, что результатом вычисления хеша будет последовательность байтов, она не велика по размеру и не зависит от длины исходного

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

сообщения. Обычно длина хеша составляет либо 128 или 160 бит.

Даже незначительные изменения в исходном тексте сообщения, всего несколько различающихся бит должны приводить к значительному изменению значения хеш-функции.

Говорить о определенности или детерминированности данного алгоритма пожалуй не стоит. Мы должны быть четко уверены в том, что одинаковые исходные последовательности должны порождать одинаковые выходные последовательности бит.

Замечание:

Невозможность вычисления на основе значения хеш-функции значения аргумента (исходного сообщения) обеспечивается за счет того, что множество исходных сообщений более чем множество результатов вычисления.

Разумеется, более подробную информацию вы всегда можете получить прочитав RFC 1321 который размещается по адресу <http://www.rfc-editor.org/rfc/rfc1321.txt>.

Любой современный веб-сервер обладает возможностью устанавливать защищенные соединения. Если даже вам захочется создать собственный веб-сервер на Java, то добавить к нему поддержку SSL не составит большого труда.

8. Ассиметричные алгоритмы. Теория и методы реализации

Далее я хочу рассмотреть внутреннее устройство метода ассиметричного алгоритма RSA. Я изложу все необходимые теоретические сведения, а затем мы напишем рабочий алгоритм и проверим правильность его действия.

Официальное рождение двухключевой криптографии относится к 1976 г., когда Диффи и Хелман опубликовали свою знаменитую статью «New Directions In Cryptography//IEEE Transactions in Information Theory». В данной статье была высказана и доказана мысль о том, что возможно построение практически стойких секретных систем, которые не требуют передачи секретного ключа. Диффи и Хелман ввели понятие односторонней функции с потайным ходом. Не вдаваясь в избыточные

технические детали можно сказать, что односторонняя функция это функция $y = f(x)$, значение которой легко вычисляется для любого x из области значения, но вычислить аргумент функции по известному y практически невозможно. Пока может показаться, что применение данной функции бессмысленно если даже законный получатель сообщения не в состоянии вычислить значение исходного аргумента (открытого сообщения). Дело в том, что на самом деле данная функция имеет следующий вид $y = f(x, z)$, в данном случае z — это и есть та самая лазейка с секретом. Ее знает получатель нашего сообщения и он может используя z на основании известного значения функции y вычислить и ее аргумент.

8.1. RSA. Математические основания

Наиболее известной ассиметричной криптосистемой является RSA. В курсе математики любого уважающего себя ВУЗа, и даже в некоторых классах школ с математическим уклоном, упоминаются различные теоремы, связанные с числами, например, малая теорема Ферма, теорема Эйлера. Но даже если вы ни разу с этим не сталкивались, не страшно, потому что на самом деле все очень просто. Теорема Эйлера формулируется следующим образом:

Если у нас есть два числа M и n , где $M < n$, а также что очень важно чтобы данные числа были взаимно простыми, то выполняется соотношение. $M^{j(n)} = 1 \pmod{n}$. В данной формуле мы вычисляем значение степени для числа M равной функции Эйлера для n . В качестве числа M мы возьмем исходное сообщение которое мы хотим зашифровать. Для того чтобы обеспечить взаимную простоту чисел M и n , следует выбирать число n , равным произведению двух больших простых множителей. Надеюсь, вы помните что число является простым если оно не имеет других делителей кроме себя и единицы. Предполагается, что если $n = p \cdot q$, где p и q являются большими простыми числами, то вероятность того что случайное сообщение M будет делителем n , т.е. не будет взаимно простым с данным числом, будет достаточно мала и ей можно пренебречь. В алгоритме RSA в качестве односторонней функции с потайной лазейкой, про которую я упоминал ранее, будет взято возведение в степень по модулю простого числа. В общем случае $SecretText = PlainText ^ (e) \pmod{(n)}$. Значение величины e мы будем считать общедоступным, это будет нашим открытым ключом. Значение данной величины будет известно всем, кто хочет посылать сообщения лицу владеющим другим значением d — и именно он будет

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

использовано для того чтобы вычислить открытый текст на базе закрытого (зашифрованного). И данное значение будет известно только получателю сообщения.

Идея расшифровки информации основана на повторном возведении зашифрованного текста в в степень, но уже числа d , для того чтобы такая идея работала необходимо чтобы между e и d существовало особое соотношение. И действительно есть требования чтобы $e*d = 1 \pmod{\phi(n)}$. В данной формуле $\phi(n)$ — это значения функции Эйлера. Следовательно эти две операции возведения в степень будут взаимно обратными и мы получим $(M^e)^d = M^{(e*d)} = M \pmod{n}$. Итак значение d — секретный ключ, а значение e — открытый ключ. Наиболее сложный вопрос — как вычислить значение данных переменных. Очевидно, что первым шагом в поиске значение e и d будет определение значения правой части уравнения. Необходимо узнать значение функции Эйлера для переменной n . В общем случае значение функции Эйлера от некоторой величины x равно количеству чисел взаимно простых с x и меньших его. Для произвольного аргумента нам придется заняться перебором всех значений меньших x и проверять каждое из них на предмет того кратно он x или нет. В простейшем варианте данный фрагмент кода будет выглядеть так.

```
package arti.security;
import java.io.*;
public class Euler {

    public static void main(String[] args) throws IOException{
        int x;

        if (args.length != 1) {
            System.out.println("Usage: java.arti.security.Euler x");
            System.out.println("Enter X Value");
            x = Integer.parseInt(new DataInputStream(System.in).readLine());
        }
        else
            x = Integer.parseInt(args[0]);

        int c = 0;

        for (int i = 2; i < x; i++)
```

```
        if (x % i != 0)
            c++;
        System.out.println("Euler (" + x + ") = " + (c+1));
    }
}
```

И хотя в общем случае к программному коду очень тяжело придаться и найти ошибки в реализации. Увы, но написанный код абсолютно бесполезен. Дело в том, что в ассиметричной криптографии оперируют над ключами очень большой величины. Для коммерческих приложений длина ключей составляет 1024 или более бита, и раз длина n соизмерима с длиной e , то мы должны понимать, что ни один стандартный тип данных Java нам не сможет помочь. Но даже если мы воспользуемся арифметикой больших чисел, это не будет иметь никакого практического значения, т.к. время расчета будет очень велико. Однако, я надеюсь, вы помните что мы решили наложить на значение n требование простоты, а для простого числа значение функции Эйлера равно $(n-1)$.

Схема RSA может быть использована не только для получения шифротекста для его безопасной пересылке по сети, но и прежде всего для создания ЭЦП некоторой информации. Ведь расшифровать сообщение может владелец секретного ключа d , а послать сообщение только владелец открытого ключа. Но раз мы наложили единственное ограничение на соотношение e и d чтобы $e*d = 1 \pmod{\phi(n)}$, то эти величины являются взаимозаменяемыми. И если владелец секретного ключа вычислит значение $X = Y^d \pmod{n}$, то получатель данной величины X сможет вычислить значение $Y_1 = X^e \pmod{n}$, и если оно совпадет с исходным значением $Y = Y_1$, это будет означать, что данное сообщение было отправлено именно и только владельцем секретного ключа d .

8.2. Алгоритм RSA. Блок схема работы

Сохранив принятые ранее обозначения, алгоритм RSA и сферу его применения можно оформить следующим образом.

Предположим необходимо установить секретное сообщение между участниками процесса взаимодействия Васей и Петей. Каждый из них выбирает два больших простых числа, разумеется что они должны быть не равны, обозначим данные числа как p и q . Затем мы найдем число $n = p*q$ и вычислим значение функции Эйлера от

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

данного произведения $\phi(n) = \phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1)$. Размер n в битах и задает длину ключа. Как я уже упоминал, рекомендуется использовать для ответственных применений ключ длиной не менее чем 512 бит.

Каждый участник коммуникационного процесса должен выбрать целое число e , данное число должно удовлетворять условию $e < \phi(n)$. А также наибольший общий делитель для чисел e и $\phi(n)$ должен быть равен 1. Дело в том, что известна следующая зависимость: если есть два целых числа X и Y которые удовлетворяют условию $0 < X < Y$ и НОД их равен 1, т.е. данные числа являются взаимно простыми, то существует некоторое число Z , находящееся в отрезке от 0 до Y , такое что произведение между Z и X дает единицу по модулю Y . Данный факт был известен еще более двух тысяч лет назад, и более того, древнегреческий математик Евклид нашел способ вычисления и величины Z . Данный алгоритм называется расширенным алгоритмом Евклида. Если мы заменим X на e , а значение Y на m и значение Z на d , то вернувшись к старой системе обозначений сможем вычислить значение секретного ключа для каждого из участников по расширенному алгоритму Евклида.

Таким образом, значения e и $n = p \cdot q$ являются открытым ключом и должны быть открыто опубликованы. Будьте внимательны: опубликовать можно только значение n , но ни в коем случае не значение переменных p и q , ведь если злоумышленник получит их значения, то он может также легко проделать все те шаги которые провели и мы вычисляя на основе известного открытого ключа значение закрытого. Вообще, наилучший вариант сразу после вычисления e и d сразу значения p и q уничтожить. Секретный ключ соответственно это значение величин d и $n = p \cdot q$.

Как я отмечал ранее, величины открытого и закрытого ключа являются взаимно заменяемыми.

Схема шифрования данных заключается в вычислении значения следующей функции для аргумента M – это исходное сообщение. $C = M^e \pmod{n}$.

Схема подписи данных заключается в похожем вычислении, только в нем мы заменяем значение e на d . $S = M^d \pmod{n}$.

Схема расшифровки информации заключается в действии аналогичном подписи сообщения, а проверка подписи соответственно эквивалента операции шифрования.

Практически целесообразно соединить операции шифрования данных и подписи. Дело в том, что математически не имеет значения порядок возведения некоторой величины в степени. Действительно $(X^Y)^Z = X^{(Y*Z)} = X^{(Z*Y)} = (X^Z)^Y$. Следовательно при обмене информацией имеет смысл сначала информацию подвергать операции подписи, а затем выполнить ее шифрование. Таким образом, получатель сообщения с помощью своего закрытого ключа сможет расширить информацию, а затем с помощью общедоступного ключа отправителя проверить действительно ли он послал ему данное письмо с информацией.

Операции вычисления значения больших степеней чисел очень трудоемко и применение ассиметричных алгоритмов для шифрования большого объема информации не рационально из-за низкой скорости, более того в приложениях где возникает необходимость выполнения криптопреобразований в режиме реального времени это вообще не допустимо. Применяется на практике следующая схема. В начале взаимодействия между адресатами выполняется проверка того, что наш собеседник именно тот, за кого он себя выдает, и обмениваются ключами для достаточно стойкого симметричного алгоритма, например для рассмотренного в предыдущей статье алгоритма blowfish. Данный ключ подвергается ассиметричному шифрованию и дальнейший обмен информацией идет именно с помощью симметричного шифрования. Ключ для симметричного алгоритма генерируется случайным образом, и даже в большинстве случаев клиенты не догадываются о том, что происходит на самом деле.

Далее я приведу пример фрагмента кода, в котором определяется набор классов служащих для генерации ключей по приведенной ранее схемы для алгоритма RSA, а также класс который позволяет шифровать/дешифровать сообщения и выполнять соответственно подпись/проверку подписи информации.

```
package arti.security;

import java.math.*;

public class RSAKeyGen {

    BigInteger n , e , d;
```

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

```
public RSAKeyGen (int len){
    BigInteger p = new BigInteger (len/2 , 5000 ,
        new java.util.Random ());
    BigInteger q = new BigInteger (len/2 , 5000 ,
        new java.util.Random ());
    BigInteger n = p.multiply(q);
    BigInteger fi = p.subtract(BigInteger.ONE).
        multiply(q.subtract(BigInteger.ONE));

    BigInteger e;

    do{
        e = new BigInteger (len / 3, 5000 , new java.util.Random ( ));
        //System.out.println("Try Number " + e);
    }while (e.gcd(fi).compareTo(BigInteger.ONE) != 0);

    BigInteger d = e.modInverse(fi);

    this.n = n;
    this.e = e;
    this.d = d;
}

public RSAPrivateKey getPrivateKey (){
    return new RSAPrivateKey (d , n);
}

public RSAPublicKey getPublicKey (){
    return new RSAPublicKey (e , n);
}
}

package arti.security;

import java.io.*;
import java.math.*;

public class RSAPublicKey {
    protected BigInteger e , n;
```

*Практическая криптография в Java. Ассиметричная криптография,
алгоритм RSA*

```
public RSAPublicKey (BigInteger ee , BigInteger nn){
    e = ee;  n =  nn;
}

public boolean equals(Object parm1) {

    if (! (parm1 instanceof RSAPublicKey )) return false;
    return ((RSAPublicKey)parm1).e.equals(e) &&
        ((RSAPrivateKey)parm1).n.equals(n);
}

public String toString() {
    return "RSAPublicKey (e="+ e+ ",
        n="+n + ")";
}
}

package arti.security;

import java.io.*;
import java.math.*;

public class RSAPrivateKey implements Serializable{

    protected BigInteger d , n;

    public RSAPrivateKey (BigInteger dd , BigInteger nn){
        d = dd;  n =  nn;
    }

    public boolean equals(Object parm1) {
        if (! (parm1 instanceof RSAPrivateKey )) return false;
        return ((RSAPrivateKey)parm1).d.equals(d) &&
            ((RSAPrivateKey)parm1).n.equals(n);
    }

    public String toString() {
        return "RSAPrivateKey (d="+ d+ ", n="+n + ")";
    }
}
}
```

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

```
package arti.security;

import java.math.*;
import java.util.*;

public class RSASign {

    /**
     * Данный метод служит для шифрования/проверки
     * подписи к некоторому сообщению
     * @param exp – показатель степени для
     * ключа RSA используемого для данного
     * преобразования
     * @param n – показатель модуля для вычислений
     * @param mes – сообщение которое должно быть подписано
     * @return – результат преобразования
     */
    private static BigInteger _sign (BigInteger exp ,
        BigInteger n, BigInteger mes){

        int len = n.bitLength();

        if (len < (mes.bitCount())) throw new
            IllegalArgumentException (
                "Длина сообщения для подписи не должна
                превышать длину ключа по модулю");

        BigInteger C = mes.modPow(exp, n);
        return C;
    }

    /**
     * В данном методе просто происходит вызов
     * метода _sign и выполняющего всю черновую работу
     */
    public static BigInteger sign (RSAPrivateKey pk , BigInteger mes){
        return _sign(pk.d , pk.n, mes);
    }
}
```

*Практическая криптография в Java. Ассиметричная криптография,
алгоритм RSA*

```
/**
 * Данный метод практически идентичен ранее приведенному
 * отличается только используемым показателем степени
 */
public static BigInteger sign (RSAPublicKey pk , BigInteger mes){
    return _sign(pk.e , pk.n, mes);
}
}

package arti.security;

import java.math.*;

public class genkeys {

    public static void main(String[] args) {
        long now = System.currentTimeMillis();
        RSAKeyGen kg = new RSAKeyGen (1024);
        System.out.println("Генерация ключа заняла "
            + (System.currentTimeMillis() - now) +
            " ms.");

        RSAPrivateKey privk = kg.getPrivateKey();
        RSAPublicKey publк = kg.getPublicKey();

        System.out.println(privk);
        System.out.println(publk);

        BigInteger res = RSASign.sign(privk ,RSASign.sign(publk ,
            new BigInteger("1234567890")));

        System.out.println("After transformation = " + res );
    }
}
}
```

В результате выполнения данного фрагмента кода был получен следующий вывод.

```
Try    Number    3747781628    2540335956    091769842    4637163237
6040466452    4900142880    7328096117    3039205052    2560055663
5126047398 3583
```

Практическая криптография в Java. Ассиметричная криптография, алгоритм RSA

Генерация ключа заняла 1421 ms.

```
RSAPrivateKey (d=1662422307 7011384449 266370914 4576806246
1115025619 1623296906 4656639487 0381181870 0192807561
6786470345 0139664915 2031216853 2137811075 7941043846
0466424351 5385698710 9392870849 1572533487 2912715442
6890078623 0642813811 9430098259 8239097271 2745494718
5432541376 4471216266 8730510693 6115664879 6920484083
1431727899 626166655, n=7587793895 7204147625 2769345987
2335265992 6775661045 2620118950 9175906372 6058304676
8234596804 8231523126 7078034189 9134657300 3230029632
5284506839 0641626215 8619819793 2829968668 4197069530
8439456807 9859281730 2817617351 9175560446 4715475064
4524755682 7083971362 6384865614 7729039083 9903967558
0975995369 0617245411 95057007)
```

```
RSAPublicKey (e=3747781628 2540335956 0917698424 6371632376
0404664524 9001428807 3280961173 0392050522 5600556635
1260473983 583, n=7587793895 7204147625 2769345987 2335265992
6775661045 2620118950 9175906372 6058304676 8234596804
8231523126 7078034189 9134657300 3230029632 5284506839
0641626215 8619819793 2829968668 4197069530 8439456807
9859281730 2817617351 9175560446 4715475064 4524755682
7083971362 6384865614 7729039083 9903967558 0975995369
0617245411 95057007)
```

After transformation = 1234567890

Разумеется, данный фрагмент кода еще далек от практического применения. Необходимо продумать механизм преобразования исходного сообщения в общем случае представляющего собой последовательность байт в набор объектов `BigInteger`. И если это не представляет большой сложности, ведь необходимо просто разделить исходный массив байт на группы не превышающие длиной число битов для модуля сообщения. Мы должны разбить данные на группы и выполнить преобразование и представить результат преобразования в виде массива байтов хранящего в себе результат представления `BigInteger` в виде массива байтов с

помощью метода `toByteArray()` для класса `BigInteger`. Обратная же операция по преобразованию массива байтов в объекты `BigInteger` достаточно сложна ведь размеры объектов `BigInteger` могут варьироваться и необходимо будет в результирующий массив преобразования помещать информацию о том, как именно было проведено разбиение исходного сообщения на массив блоков, каждый из которых подвергся отдельному криптопреобразованию.

9. Заключение

Было сделано краткое рассмотрение направлений использования методов ассиметричной криптографии. Рассмотрены способы безопасной передачи информации через общедоступные сети. Рассмотрены проблемы, которые характерны для двухключевой криптографии и способы их решения. Был сделан краткий обзор сертификатов. Введено понятие хеш-функций и способы ее использования. Также был рассмотрен один из наиболее популярных алгоритмов RSA и была создана его программная реализация. В следующей статье будет более подробно рассмотрено создание сертификатов с использованием средств идущих в JSDK. Также будет попытка создания безопасного веб-сервера работающего по протоколу SSL. Будет большее внимание уделено вопросам возможных направлений атак на криптосистему.

10. Ресурсы

- Хорошая книжка, настоятельно рекомендую. Молдовян А.А., Молдовян Н.А., Советов Б.Я. Криптография.
- <http://www.onjava.com/pub/a/onjava.2001/05/03/java.security.html?page=1>. O'Reilly — синоним качества, отличная серия статей.
- <http://www.apache-ssl.org/> — в следующей статье я уделю внимание вопросам ssl. Подготовьтесь теоретически.
- <http://www.Verisign.com> — отличная документация по сертификатам и вопросам их получения.
- <http://www.oreilly.com/catalog/javacrypt/chapter/ch06.html> — прекрасная выдержка из хорошей книжки по безопасности в Java. К прочтению обязательна.