Итак, все чаще в кругах, работающих с документами все чаще звучат слова «электронный документ» и, связанное с ним почти неразрывно «электронная цифровая подпись», иначе — ЭЦП.

Данный цикл статей предназначен для того, чтобы раскрыть «тайное знание» о том, что это такое, когда и как это можно и нужно использовать, какие есть плюсы и минусы.

Естественно, статьи пишутся не для специалистов по криптографии, а для тех, кто эту самую криптографию будет использовать, или же только начинает ее изучение, желая стать специалистом, поэтому я старался максимально упростить понимание всего процесса, приводя аналогии и рассматривая примеры.

Зачем нам вообще что-то подписывать? Естественно, чтобы удостоверить, что мы ознакомились с содержимым, согласны (а иногда наоборот, не согласны) с ним. А электронная подпись еще и защищает наше содержимое от подмены.

Итак, начать, естественно, стоит с того, что такое электронная цифровая подпись.
В самом примитивном случае это — результат хэш-функции. Что это такое лучше меня разъяснит википедиа, в нашем же случае главное, что с высокой степенью вероятности ее результат не повторяется для разных исходных данных, а также что результат этой функции мало того, что короче исходных данных, так еще по нему исходную информацию восстановить нельзя. Результат функции называют хэшем, а применение этой функции к данным называют хешированием. Грубо, можно назвать хэш функцию архивированием, в результате чего мы получаем очень маленькую последовательность байт, но восстановить исходные данные из такого «архива» нельзя.

Итак, мы читаем файлик в память, хэшируем прочитанное. И что, уже получаем ЭЦП? Почти. Наш результат с большой натяжкой можно назвать подписью, но, все же, полноценной подписью он не является, потому что:

1. Мы не знаем, кто сделал данную подпись

2. Мы не знаем, когда была сделана подпись

3. Сама подпись не защищена от подмены никак.

4. Ну и да, хэш функций много, какая из них использовалась для создания этого конкретного хэша?

Поэтому применять к хэшу слово «подпись» еще нехорошо, будем называть его дальше просто хэш.

Вы посылаете ваш файл другому человеку, допустим, по почте, будучи уверенными, что он точно получит и прочитает именно то, что вы послали. Он же, в свою очередь, тоже должен хэшировать ваши данные и сравнить свой результат с вашим. Если они совпали — все хорошо. Это значит что данные защищены? **Нет.**
Ведь хэшировать может кто угодно и когда угодно, и вы никогда не докажете, что он хэшировал не то, что вы послали. То есть, если данные будут перехвачены по дороге злоумышленником, или же тот, кому вы посылаете данные — не очень хороший человек, то данные могут быть спокойно подменены и прохэшированы. А ваш получатель (ну или вы, если получатель — тот самый нехороший человек) никогда не узнает, что он получил не то, что вы отправляли, или сам подменил информацию от вас для дальнейшего использования в своих нехороших целях.
Посему, место для использование чистой хэш функции — транспорт данных в пределах программы или программ, если они умеют общаться между собой. Собственно, с помощью хэш функций вычисляются контрольные суммы. И эти механизмы защищают от *случайной* подмены данных, но не защищают от *специальной*.

Но, пойдем дальше. Нам хочется защитить наш результат хеширования от подмены, чтобы каждый встречный не мог утверждать, что это у него правильный результат. Для этого самое очевидное что (помимо мер административного характера)? Правильно, зашифровать. А ведь с помощью шифрования же можно и удостоверить личность того, кто хэшировал данные! И сделать это сравнительно просто, ведь есть ассиметричное шифрование. Да, оно медленное и тяжелое, но ведь нам всего-то и надо — зашифровать маленькую последовательность байт. Плюсы такого действия очевидны — для того, чтобы проверить нашу подпись, надо будет иметь наш открытый ключ, по которому личность зашифровавшего (а значит, и создавшего хэш) можно легко установить.
Суть этого шифрования в следующем: у вас есть закрытый ключ, который вы храните у себя. И есть открытый ключ. Открытый ключ вы можете всем показывать и раздавать, а закрытый — нет. Шифрование происходит с помощью закрытого ключа, а расшифровывание — с помощью открытого.
Приводя аналогию, у вас есть отличный замок и два ключа к нему. Один ключ замок открывает (открытый), второй — закрывает (закрытый). Вы берете коробочку, кладете в нее какую-то вещь и закрываете ее своим замком. Так, как вы хотите, чтобы закрытую вашим замком коробочку открыл ее получатель, то вы открытый, открывающий замок, ключик спокойно отдаете ему. Но вы не хотите, чтобы вашим замком кто-то закрывал коробочку заново, ведь это ваш личный замок, и все знают, что он именно ваш. Поэтому закрывающий ключик вы всегда держите при себе, чтобы кто-нибудь не положил в вашу коробочку мерзкую гадость и не говорил потом, что это ***вы*** ее положили и закрыли своим замком.

И все бы хорошо, но тут сразу же возникает проблема, а, на самом деле, даже не одна.

1. Надо как-то передать наш открытый ключ, при этом его должна понять принимающая сторона.

2. Надо как-то связать этот открытый ключ с нами, чтобы нельзя было его присвоить.

3. Мало того, что ключ надо связать с нами, надо еще и понять, какой зашифрованный хэш каким ключом расшифровывать. А если хэш не один, а их, скажем, сто? Хранить отдельный реестр — очень тяжелая задача.

Все это приводит нас к тому, что и закрытый ключ, и наш хэш надо хранить в каких-то форматах, которые нужно стандартизировать, распространить как можно шире и уже тогда использовать, чтобы у отправителя и получателя не возникало «трудностей перевода».

Как водится у людей, к чему-то единому прийти так и не смогли, и образовалось два больших лагеря — формат OpenPGP и формат S/MIME + X.509. Но об этом уже в следующей статье.

Продолжая раскрывать тайное знание о цифровой подписи простым языком, разберем, что же нам надо для удобной и эффективной работы с ними, а также главное различие между лагерями S/MIME + X.509 и PGP.

Перед тем, как рассматривать особенности этих двух больших лагерей, стоит рассмотреть, какая информация нужна получателю для проверки подписи (а наш зашифрованный хэш уже вполне можно называть подписью), и в каком виде ее можно ему передать.

Каждую из частей информации можно передать вместе с открытым ключом, или вместе с нашей подписью, а можно и с тем и с тем, для большего удобства. Конечно, можно не разделять информацию на передаваемую с открытым ключом и передаваемую с подписью. Но тогда каждый раз отправляя подписанную информацию мы отправляем одно и то же. Как если бы к каждому отправляемому нами бумажному письму (даже короткому, в две строки), мы бы прикладывали дополнение вида «Здравствуйте! Это я, В. Пупкин, которого вы встречали на Красной площади Москвы, где мы и познакомились, потом пошли в ресторан, потом <...>». Согласитесь, слегка неудобно.

Но вернемся к нашей информации, необходимой для проверки подписи.
Начнем с простого: информация, которая позволит нам узнать, кто же сделал эту подпись. Как мы уже договорились, ассиметричное шифрование позволяет однозначно связать наш открытый ключ и полученную подпись. Беда в том, что сам по себе открытый ключ – набор байт. При этом он, конечно, связан с закрытым, которым мы (то есть отправитель) владеем, но связь эта для получателя неочевидна. У него есть набор байт от В. Пупкина, от И. Петрова, от С. Сидорова… И от десятка других людей. И как ему их идентифицировать? Держать отдельный реестр для того, кому какой набор байт принадлежит? Это что же,

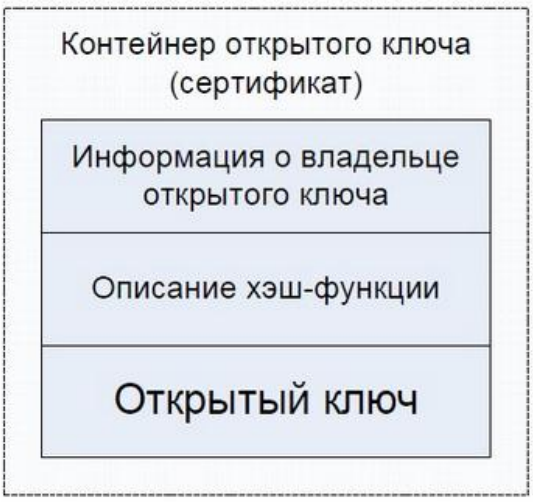
получается уже **второй** реестр (помимо того, где должно быть записано, с помощью какой хэш-функции какой хэш сделан)! И опять, неудобно!

Значит, надо связать каждый открытый ключ с информацией о том, кому этот ключ принадлежит, и присылать это все одним пакетом. Тогда проблема реестра решается сама собой – пакет (а если более правильно, контейнер) с открытым ключом можно будет просто посмотреть и сразу понять его принадлежность.

Но эту информацию все так же надо связать с подписью, пришедшей получателю. Как это сделать? Надо соорудить еще один контейнер, на сей раз для передачи подписи, и в нем продублировать информацию о том, кто эту подпись создавал.
Продолжая нашу аналогию с красивым замком, мы пишем на ключе «Этот ключ открывает замок В. Пупкина». А на замке тоже пишем «Замок В. Пупкина». Имея такую информацию, получатель нашей коробочки не будет каждый из имеющихся у него ключей вставлять наугад в наш замок, а возьмет наш ключ и сразу его откроет.

Теперь, по переданной информации при проверке можно найти контейнер открытого ключа, взять оттуда ключ, расшифровать хэш и…

А собственно, что «и»? Мы ведь пока так и не решили проблему, как донести до получателя информацию о том, какая хэш-функция применялась для хэша, а ведь для проверки подписи эта информация ***необходима***! Решить ее можно достаточно просто: положить эту информацию в контейнер вместе с нашим открытым ключом. Ведь именно связка «хэширование – шифрование результата хеширования» считается процедурой создания цифровой подписи, а ее результат – подписью. А значит, достаточно логичным представляется объединение в связку алгоритма шифрования хэша и хэш-функции, с помощью которой он сформирован. И доставлять эту информацию тоже надо в связке.



Теперь, ненадолго вернемся к информации о подписывающем. Какого рода эта информация должна быть? ФИО? Нет, В. Пупкиных много. ФИО + год рождения? Так и родившихся в один день В. Пупкиных тоже предостаточно! Более того, это может быть

Василий, Виктор, или даже Василиса или Виктория Пупкины. Значит, информации должно быть больше. Ее должно быть столько, чтобы совпадение всех параметров, по которым мы идентифицируем человека, было максимально невероятным.

Безусловно, такой пакет информации создать возможно. Вот только, работать с ним уже трудновато. Ведь надо наши контейнеры открытых ключей надо сортировать, хранить, использовать, в конце концов. А если для каждого использования придется указывать по полсотни параметров, то уже на втором контейнере станет понятно, что что-то надо менять. Решение этой проблемы, конечно же, было найдено.

Чтобы понять, в чем же оно заключалось, обратимся к бумажному документу, который есть у всех нас: к паспорту. В нем можно найти и ФИО, и дату рождения, и пол, и много другой информации. Но, главное, в нем можно найти серию и номер. И именно серия и номер являются той уникальной информацией, которую удобно учитывать и сортировать. Кроме того, они существенно короче всей оставшейся информации вместе взятой, и при этом все так же позволяют опознать человека.

Применяя этот же подход к контейнерам открытых ключей, мы получим, что у каждого контейнера должен быть некий номер, последовательность символов, уникальная для него. Эту последовательность символов принято называть ***идентификатором***, а сами контейнеры – ***сертификатами***, либо просто ключами.
Вот здесь и начинаются принципиальные различия в идеологиях OpenPGP и S/MIME + X.509. Для краткого понимания их, вернемся к нашей аналогии с паспортом.

Паспорт вы можете использовать при покупках билетов, при оформлении документов, для выдачи пропуска на какую-либо территорию и даже на территории других стран! То есть, вы используете его для идентификации вашей личности в самых различных, зачастую абсолютно не связанных друг с другом, местах, с самыми различными людьми. И везде ваш паспорт принимают. Гарантом того, что вы – это вы выступает третья сторона в ваших взаимоотношениях с другими: государство. Именно оно выдало вам ваш паспорт, специально оформленный, подписанный и заверенный, и именно поэтому ваш паспорт является таким универсальным документом.

С другой стороны, в кругу друзей, или внутри компании вам достаточно представиться так: «В. Пупкин из твоей группы в институте» или же «В. Пупкин из отдела продаж». И людям, с которыми вы контактируете в этом кругу уже не нужна третья сторона, они и так помнят Пупкина из группы с которым проучились пять лет, или Пупкина из отдела продаж, с которым недавно ходили обедать, и предоставленной вами информации им вполне достаточно.

Так же можно разделить и эти два лагеря.

Сертификат X.509 – это аналог нашего паспорта. Здесь сертификаты вам выдаются суровой

третьей стороной, гарантом вашей личности: Удостоверяющим Центром (УЦ). Получающий ваши подписи человек всегда может обратиться в УЦ и спросить интересующую его информацию по вот этому конкретному сертификату.

PGP же (и стандарт OpenPGP, появившийся в дальнейшем) создавался на основе так называемых сетей доверия. Такая идея подразумевает, что обмениваются подписями люди, которым третья сторона для их взаимоотношений не нужна, а нужна только защита от нехороших лиц.

Конечно, с течением времени такое разделение стало уже достаточно условным, так как на данный момент и в S/MIME+X.509 и в PGP можно использовать методы лагеря соперников. Но все же, стандарты достаточно продолжительное время развивались параллельно и развились до той степени, что взаимная совместимость между ними стала невозможной.

Более популярным стандартном, в силу своей ориентированности на участие более компетентной третьей стороны, стал стандарт S/MIME + X.509, однако и у PGP есть некоторое количество козырей за пазухой, с помощью которых он не только не погибает, но и продолжает успешно развиваться.
Более подробное рассмотрение каждого из форматов, а также рекомендации, когда, где и какой из них использовать вы сможете прочитать уже в следующих статьях.

В этой части сделаем небольшое отступление от цифровых подписей в сторону того, без чего непосредственно цифровых подписей, да и защиты информации в привычном понимании, не было бы: шифрования. Ведь первое, что приходит на ум, когда идет речь о защите наших данных — это не дать эти данные нехорошему человеку прочитать. Поэтому, перед тем, как продолжить рассмотрение стандартов PGP и S/MIME, стоит закрасить некоторые остающиеся в знаниях белые пятна, и рассмотреть процесс шифрования немного поподробнее.

Шифры и коды существуют, наверное, с того момента, как человечество научилось записывать свои впечатления об окружающем мире на носителях. Если немного вдуматься, даже обыкновенный алфавит — уже шифр. Ведь когда мы читаем какой-либо текст, в нашей голове каждому нарисованному символу сопоставляется некий звук, сочетание звуков, или даже целое понятие, а в голове соседа, который читать не умеет, этого уже не происходит.

Не зная, какому символу и что сопоставлено, мы никогда не сможем понять, что же именно писавший имел ввиду. К примеру, попробуйте взять и прочитать что-то, написанное на иврите, или на китайском языке. Сами алфавиты этих языков будут являться для вас непреодолимым препятствием, даже если с помощью этих символов написаны понятия вашего родного языка.

Но, тем не менее, простое использование чужого алфавита все же недостаточная мера для защиты ваших данных. Ведь любой алфавит, так или иначе, создавался для удобства пользования им и является неразрывно связанным с языком, которому данный алфавит характерен. А значит, выучив этот язык и некоторый набор базовых понятий на нем (а то и просто воспользовавшись услугами человека, знающего данный язык), нехороший человек может прочитать вашу информацию.

Значит, надо придумать алфавит, который знает только ограниченный круг лиц, и с его помощью записать информацию. Наверняка все читали (или, по крайней мере, слышали) цикл историй про Шерлока Холмса. В этом цикле фигурировал алфавит, составленный из пляшущих человечков (а многие, я думаю, в детстве на его основе составляли свой). Однако, как показывает данная история, наблюдательный человек может разгадать, какой символ и к чему относится. А значит наша информация опять попадет не в те руки.

Что же делать? Придумывать все более и более сложные алфавиты? Но чем более сложный и громоздкий алфавит, тем более неудобно с ним работать, хранить его в тайне. К тому же, насчет тайны есть замечательная поговорка: знают двое – знают все. Ведь самое слабое звено в любом шифре – это человек, который знает, как этот шифр расшифровать.

А почему бы не сделать так, чтобы способ шифрования был сразу известен всем, но расшифровать наши данные было бы нельзя без какого-то ключа? Ведь ключ (в отличие от всего алфавита) маленький, его достаточно легко сделать новый, если что (опять же, в отличие от переработки всего алфавита), легко спрятать. Наиболее наглядно плюсы ключевых систем показывает следующий пример: получателю надо прочитать сосланное вами сообщение. Обычное, на бумаге. Допустим, вы используете секретный алфавит. Тогда, чтобы прочитать сообщение, получатель должен знать алфавит, иметь большой пыльный талмуд, в котором описаны способы расшифровки (ведь алфавит должен быть сложным, чтобы быть надежным) и понимать, как же с этим талмудом работать. С ключами же все проще: вы кладете сообщение в коробку с замком, а получателю достаточно просто вставить подходящий ключик, а знать, как же устроен замок ему совершенно но нужно.
Итак, общеизвестные «алфавиты» и ключи — механизм, существенно более удобный, чем просто алфавиты. Но как же так зашифровать, чтобы все расшифровывалось простым ключом? И вот тут нам на помощь приходит математика, а конкретнее – математические функции, которые можно использовать для замены наших исходных символов на новые.

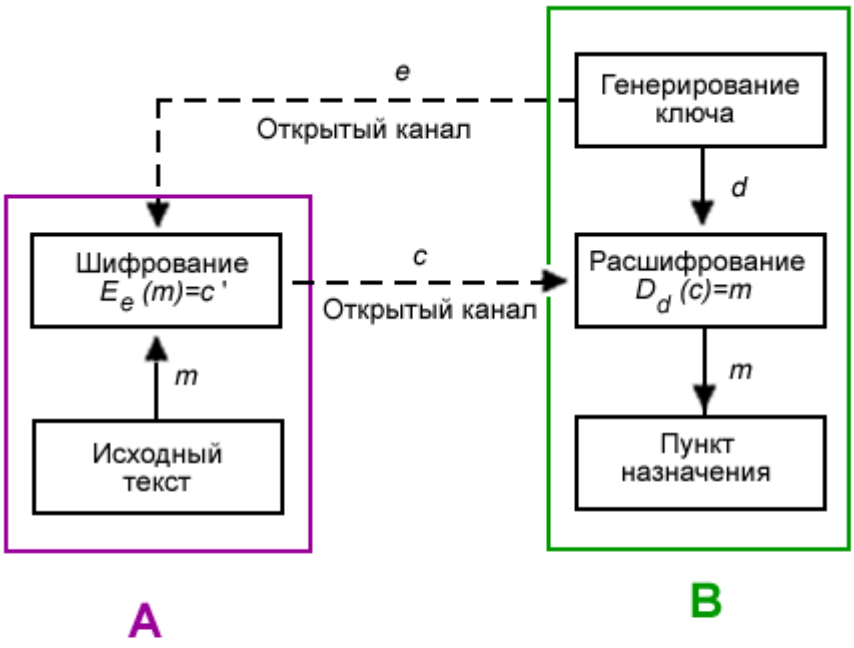
Вспомним же, что такое функция. Это некоторое соотношение, по которому из одного числа можно получить другое. Зная x и подставляя его в известное нам соотношение y=A*x, мы всегда получим значение y. Но ведь, как правило, верно и обратное: зная y, мы можем получить и x.
Как правило, но далеко не всегда. Для многих зависимостей получить y легко, тогда как x

– уже очень трудно, и его получение займет продолжительное время. Вот именно на таких зависимостях и базируется используемое сейчас шифрование.

Но, вернемся к самому шифрованию. Шифрование подразделяют на симметричное, асимметричное и комбинированное. Рассмотрим, в чем суть каждого из них.

Симметричное шифрование, по большому счету, достаточно слабо отличается от старого доброго секретного алфавита. Собственно говоря, отличается оно как раз наличием ключа – некоторой сравнительно маленькой последовательности чисел, которая используется для шифрования и расшифровывания. При этом, каждая из обменивающихся информацией сторон должна этот ключ знать и хранить в секрете. Огромным плюсом такого подхода является скорость шифрования: ключ, по сути, является достаточно простой и короткой инструкцией, какой символ, когда, и на какой надо заменять. И работает данный ключ в обе стороны (то есть с его помощью можно как заменить все символы на новые, так и вернуть все как было), за что такой способ шифрования и получил название симметричного. Столь же огромным минусом является именно то, что обе стороны, между которыми информация пересылается, должны ключ знать. При этом, стоит нехорошему человеку заполучить ключ, как он тут же прочитает наши столь бережно защищаемые данные, а значит проблема передачи ключа принимающей стороне становится в полный рост.

Асимметричное шифрование поступает несколько хитрее. Здесь и у нас, и у нашего получателя есть уже два ключа, которые называют открытый и закрытый. Закрытый ключ мы и получатель храним у себя (заметьте, каждый хранит только свой ключ, а значит, мы уже выходим за пределы той самой поговорки про двоих знающих), а открытый мы и получатель можем спокойно передавать кому угодно – наш закрытый, секретный, по нему восстановить нельзя. Итого, мы используем открытый ключ получателя для шифрования, а получатель, в свою очередь, использует свой закрытый ключ для расшифровывания. Плюс данного подхода очевиден: мы легко можем начать обмениваться секретной информацией с разными получателями, практически ничем (принимая условие, что наш получатель свой закрытый ключ не потерял/отдал и т.п., то есть не передал его в руки нехорошего человека) не рискуем при передаче информации. Но, без огромного минуса не обойтись. И здесь он в следующем: шифрование и расшифровывание в данном случае идут очень, очень, очень медленно, на два-три порядка медленнее, чем аналогичные операции при симметричном шифровании. Кроме того, ресурсов на это шифрование тратится также значительно больше. Да и сами ключи для данных операций существенно длиннее аналогичных для операций симметричного шифрования, так как требуется максимально обезопасить закрытый ключ от подбора по открытому. А значит, большие объемы информации данным способом шифровать просто невыгодно.



Пример использования асимметричного шифрования [Wikipedia]

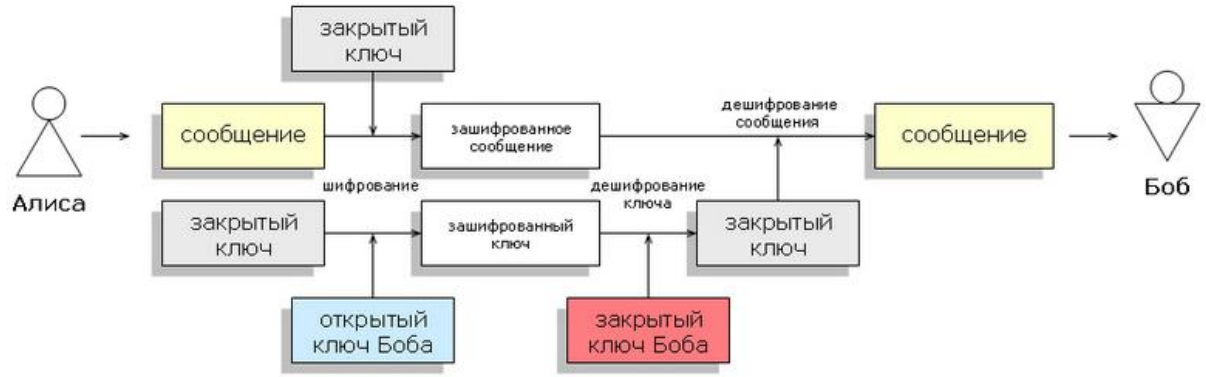
e — открытый ключ получателя B

d — закрытый ключ получателя B

m — исходная информация отправителя A

c — зашифрованная исходная информация

И снова возникает вопрос: что же делать? А делать нужно следующее: взять, и скомбинировать оба способа. Собственно, так мы и получаем комбинированное шифрование. Наш большой объем данных мы зашифруем по первому способу, а чтобы донести ключ, с помощью которого мы их зашифровали, до получателя, мы сам ключ зашифруем по второму способу. Тогда и получим, что хоть асимметричное шифрование и медленное, но объем зашифрованных данных (то есть ключа, на котором зашифрованы большие данные) будет маленьким, а значит расшифровывание пройдет достаточно быстро, и дальше уже в дело вступит более быстрое симметричное шифрование.



Пример применения комбинированной системы [Wikipedia]

Все эти механизмы нашли свое применение на практике, и оба наших больших лагеря PGP и S/MIME их используют. Как говорилось в первой статье, асимметричное шифрование используется для цифровой подписи (а именно, для шифрования нашего хэша). Отличие данного применения от обычного асимметричного шифрования в том, что для шифрования используется наш закрытый ключ, а для расшифровывания достаточно наличие связанного с ним (то есть, тоже нашего) открытого ключа. Поскольку открытый ключ мы не прячем, наш хэш можем прочитать кто угодно, а не только отдельный получатель, что и требуется для цифровой подписи.
Комбинированное же шифрование применяется в обоих стандартах непосредственно для шифрования отправляемых данных.

Таким образом, начиная пользоваться цифровыми подписями для защиты наших данных от подмены, мы автоматически (для этих двух стандартов) получаем и замечательную возможность защитить наши данные еще и от прочтения, что, согласитесь, весьма удобно.

Теперь, когда мы познакомились с общими принципами работы механизмов, используемых для защиты наших данных, можно наконец перейти к практике и рассмотрению, что же использовать. Но об этом в следующих статьях.