Электронная цифровая подпись «для чайников». Часть 1

http://habrahabr.ru/blogs/infosecurity/97066/

Все чаще вокруг нас звучат слова «электронный документ» и почти неразрывно с ними «электронная цифровая подпись» или ЭЦП.
В данном материале мы постараемся раскрыть «тайное знание» о том, что же такое ЭЦП, когда и как ее можно и нужно использовать и какие в этом 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 и согласны (а иногда наоборот, не согласны) с ним. А электронная подпись еще и защищает содержимое от подмены.

Хэш-функция или ЭЦП?

Начнем, естественно с того, что такое электронная цифровая подпись.
В самом примитивном случае это — результат хэш-функции. Что это такое лучше меня разъяснит википедия. 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 — хешированием. Попросту говоря, можно назвать хэш-функцию архивированием, в результате чего мы получаем очень маленькую последовательность байт, но восстановить исходные данные из такого «архива» нельзя. Итак, считываем файл в память и хэшируем прочитанное. Мы уже получаем ЭЦП? Почти. Наш результат с большой натяжкой можно было бы назвать подписью, но, все же, строго говоря, 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» не слишком правильно, будем называть его дальше просто хэш.

Подмены случайные и умышленные.

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.

Поэтому, 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.

Ключи и шифры.

Пойдем дальше. Нам хочется защитить результат хеширования от подмены, чтобы каждый встречный не мог утверждать, что это у него правильный результат. Что мы можем для этого сделать (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.

Проводя 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 именно вы ее положили и закрыли своим замком.
И все бы хорошо, но тут сразу же возникает проблема, и, 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 — приверженцев формата OpenPGP и формата S/MIME + X.509. Но об этом уже в следующей статье.