5.1 Загвоздка с передачей ключей

Пока зашифровку и расшифровку производит одно и то же лицо, с симметричным шифрованием нет никаких проблем. Они начинаются, когда зашифровывать хочет один, а расшифровывать – другой. Допустим, Алиса хочет передать Бобу секретные сведения. Она шиф-

рует сообщение и отправляет шифротекст Бобу. Теперь ему нужен Алисин ключ, чтобы расшифровать его. Сейчас Алисе нужно думать, как же передать ключ Бобу, минуя обозревательницу Еву. Алиса может зашифровать ключ другим ключом, но как передать Бобу второй ключ, чтобы Ева не подслушала? Можно зашифровать второй ключ третьим, но как тогда… В общем, смысл ясен. Необходимость передать ключ завела нас в рекурсивный тупик.

Пиши пропало, если таких Бобов у Алисы человек десять. Даже если она лично передаст ключ каждому, в случае утечки ключа ей придется снова посетить каждого. Притом что вероятность утечки ключа возросла в десять раз, как и сложность его смены. Алиса может, конечно, шифроваться с каждым собеседником разным ключом. Но загвоздку с их раздачей это по-прежнему не решает, в чем-то даже усугубляет. Именно эта незадача – одна из причин изобретения асимметричного шифрования.

5.2 Асимметричное шифрование

Если алгоритму шифрования – например, AES – требуется один и тот же ключ для зашифровки и расшифровки, то это *симметричное шифрование*. Если для зашифровки применяется один ключ, а для расшифровки – другой, то это асимметричное *шифрование*. Эти два ключа называют *парой ключей*.

Пара состоит из *закрытого ключа* и *открытого*. Закрытый ключ владелец держит в секрете. Открытый ключ держать в секрете не нужно, им можно делиться с кем угодно. Закрытым ключом можно расшифровать то, что зашифровано открытым, и наоборот. Асимметричное шифрование еще называют *криптосистемой с открытым ключом*.

Асимметричное шифрование – типичный ответ на задачу с передачей ключей. Его механизм показан на рис. 5.1. Допустим, Алиса хочет отправить Бобу секретные сведения с помощью криптосистемы с открытым ключом. Боб генерирует пару ключей. Закрытый ключ он оставляет себе, открытый отправляет Алисе по незащищенному каналу связи. Ева может обозревать этот канал связи – это не имеет значения, открытый ключ на то и открытый. Алиса зашифровывает сообщение открытым ключом Боба и отправляет ему шифрованный текст по незащищенному каналу. Боб получает шифротекст и расшифровывает его закрытым ключом. Никаким другим ключом расшифровать текст не получится.

Таким образом решаются две задачи. Во-первых, задача с передачей ключа больше не стоит. Даже если Ева подслушает открытый ключ Боба и шифротекст Алисы, злоумышленница не сможет расшифровать послание. Только закрытым ключом Боба можно расшифровать то, что зашифровано его открытым ключом. Во-вторых, теперь Алисе не составит труда общаться с любым количеством собе-

седников. Каждому из них просто нужно создать свою собственную пару ключей и отправить Алисе открытый ключ. Даже если чей-то закрытый ключ вдруг окажется достоянием Евы, на других собеседников это никак не повлияет.

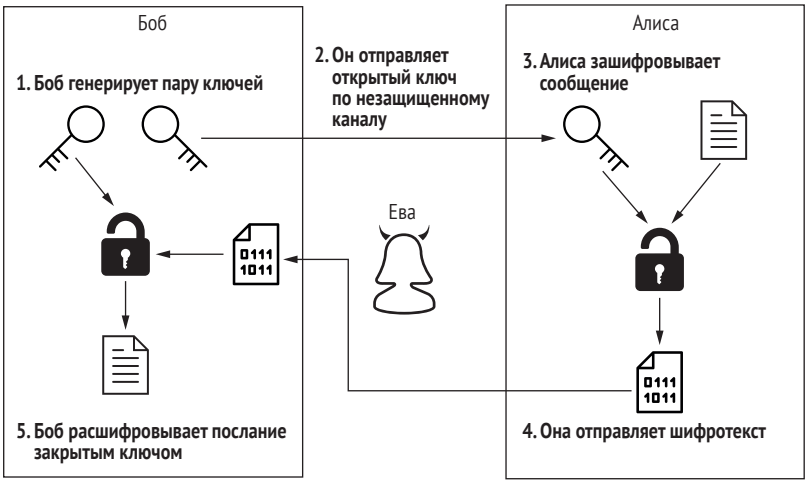

Рис. 5.1 Данные, отправленные Алисой Бобу с помощью асимметричного шифрования, остаются неразглашенными Еве

Этот раздел вкратце поведал об асимметричном шифровании. Следующий же наглядно покажет, как использовать в Python самую широко распространенную криптосистему с открытым ключом всех времен и народов.

5.2.1 RSA

RSA – классический пример криптосистемы с открытым ключом, которая выдержала проверку временем. Она была создана в конце 70-х, ее создатели – Рон Ривест (Ron Rivest), Ади Шамир (Adi Shamir) и Леонард Адлеман (Leonard Adleman). Название алгоритму дано по первым буквам их фамилий.

Ниже показан вызов `openssl`[1], который генерирует закрытый ключ RSA длиной 3072 бита. Для этого использована команда `genpkey`. На момент написания этой книги ключи RSA должны быть размером минимум 2048 бит.

```
$ openssl genpkey -algorithm RSA \        <-- Ключ для RSA
    -out private_key.pem \                <-- Файл с ключом положить по этому пути
    -pkeyopt rsa_keygen_bits:3072         <-- Длина ключа 3072 бита
```

[1] Чтобы воспользоваться `openssl` под Windows, установите Git for Windows (https://github.com/git-for-windows/git/releases) и запустите Git Bash. – *Прим. перев.*

Обратите внимание, насколько разнятся размеры ключа для RSA и AES. Первому для обеспечения сравнимой стойкости надлежит быть куда длиннее своего симметричного собрата. Ключ AES может быть 256 бит длиной максимум, ключ RSA же такой длины просто никуда не годится. Такая разница обусловлена математическими моделями, на которых строится шифрование. RSA применяет факторизацию целых чисел, AES использует подстановочно-перестановочную сеть. В общих чертах: ключ для асимметричного шифрования всегда будет длиннее ключа для симметричного.

С помощью команды `rsa` утилиты `openssl` можно извлечь открытый ключ из файла с закрытым:

```
$ openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Пара ключей иногда хранится в файловой системе. Важно отрегулировать права доступа к этим файлам. Права на чтение и запись файла с закрытым ключом должны быть только у владельца. Открытый ключ же можно читать кому угодно. Вот как ограничить доступ к файлам ключей на UNIX-подобных системах:

```
$ chmod 600 private_key.pem ◄—— Читать и писать может только владелец
$ chmod 644 public_key.pem  ◄—— Читать может кто угодно
```

> **ПРИМЕЧАНИЕ** Как и в случае с ключами от симметричного шифрования, ключам для асимметричного шифрования не место среди файлов боевой системы либо внутри ее исходного кода. Ключи должны находиться в службе управления ключами, где они будут в безопасности. Это может быть, например, Amazon's AWS Key Management Service (https://aws.amazon.com/kms/) либо Google's Cloud Key Management Service (https://cloud.google.com/security-key-management).

OpenSSL сохраняет ключи на диск в формате *Privacy-Enhanced Mail* (PEM, почта повышенной секретности). Это стандарт де-факто для представления пар ключей. Если вам уже приходилось сталкиваться с PEM-файлами раньше, вам может быть знаком заголовок, начинающийся с `-----BEGIN`:

```
-----BEGIN PRIVATE KEY-----
MIIG/QIBADANBgkqhkiG9w0BAQEFAASCBucwggbjAgEAAoIBgQDJ2Psz+Ub+VKg0
vnlZmm671s5qiZigu8SsqcERPlSk4KsnnjwbibMhcRlGJgSo5Vv13SMekaj+oCTl
...
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAydj7M/lG/lSoNL55WZpu
u9bOaomYoLvErKnBET5UpOCrJ548G4mzIXEZRiYEqOVb9d0jHpGo/qAk5VCwfNPG
...
```

Ключи можно создать и средствами пакета `cryptography`. Как сгенерировать закрытый ключ методом `generate_private_key` модуля

rsa, показано в листинге 5.1. Первый аргумент – это деталь реализации RSA, в которую я не буду вдаваться в рамках данной книги. Подробнее о ней можно узнать по ссылке https://www.imperialviolet.org/2012/03/16/rsae.html. Второй аргумент – длина ключа. Третьим аргументом мы выбираем использование под капотом библиотеки по умолчанию, это OpenSSL[1]. После создания закрытого ключа из него извлекается открытый.

Листинг 5.1 Создание пары ключей RSA через Python

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=3072,
    backend=default_backend(), )

public_key = private_key.public_key()
```

Нетривиальные низкоуровневые инструменты

Генерация закрытого ключа

Извлечение открытого ключа

ПРИМЕЧАНИЕ Ключи для боевых систем редко генерируются скриптами на Python. Как правило, используются консольные утилиты, как то `openssl` либо `ssh-keygen`.

Следующий пример показывает, как сохранить ключи из оперативной памяти на диск в формате PEM.

Листинг 5.2 Сохранение пары ключей RSA на Python

```
private_bytes = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption(), )

with open('private_key.pem', 'xb') as private_file:
    private_file.write(private_bytes)

public_bytes = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo, )

with open('public_key.pem', 'xb') as public_file:
    public_file.write(public_bytes)
```

Запаковка закрытого ключа в байтовую строку

Запись строки в файл

Запаковка открытого ключа в байтовую строку

Запись строки в файл

Ключи могут быть загружены и в обратном направлении – из файлов в оперативную память. Файлы ключей могут быть созданы любой программой, не обязательно предыдущим скриптом.

[1] Так как практически всегда в качестве backend указывается `default_backend()`, сейчас этот аргумент стал необязательным. – *Прим. перев.*

Листинг 5.3 Чтение пары ключей RSA на Python

```
with open('private_key.pem', 'rb') as private_file:
    loaded_private_key = serialization.load_pem_private_key(
        private_file.read(),
        password=None,
        backend=default_backend()
    )

with open('public_key.pem', 'rb') as public_file:
    loaded_public_key = serialization.load_pem_public_key(
        public_file.read(),
        backend=default_backend()
    )
```

Распаковка закрытого ключа

Распаковка открытого ключа

Следующий пример показывает, как зашифровать данные открытым ключом и как расшифровать закрытым. Как и симметричные блочные шифры, RSA перед зашифровкой дополняет открытый текст ничего не значащими данными.

> **ПРИМЕЧАНИЕ** Рекомендуемой схемой дополнения открытого текста для шифрования алгоритмом RSA является *оптимальное асимметричное шифрование с дополнением* (optimal asymmetric encryption padding – OAEP).

Листинг 5.4 Шифрование парой ключей RSA на Python

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

padding_config = padding.OAEP(
    mgf=padding.MGF1(algorithm=hashes.SHA256()),
    algorithm=hashes.SHA256(),
    label=None, )

plaintext = b'message from Alice to Bob'

ciphertext = loaded_public_key.encrypt(
    plaintext=plaintext,
    padding=padding_config, )

decrypted_by_private_key = loaded_private_key.decrypt(
    ciphertext=ciphertext,
    padding=padding_config)

assert decrypted_by_private_key == plaintext
```

Схема дополнения OAEP

Зашифровка открытым ключом

Расшифровка закрытым ключом

Асимметричное шифрование работает в обе стороны. Можно зашифровать послание открытым ключом, а расшифровать закрытым. Или наоборот, можно зашифровать закрытым, а расшифровать открытым. Таким образом, мы можем быть уверены только в чем-то одном: либо в неразглашении, либо в подлинности данных. То, что

зашифровано открытым ключом, может расшифровать только владелец закрытого ключа, и, следовательно, не может быть разглашено. При этом создателем содержимого может быть кто угодно. То, что зашифровано закрытым ключом, является подлинным и принадлежит руке владельца закрытого ключа, но расшифровать эти данные может кто угодно. Следовательно, данные разглашаются, но их авторство несомненно.

Мы обсудили, как добиться неразглашения с помощью зашифровки данных открытым ключом. В следующем разделе узнаем, как достичь неопровержимости деяния путем зашифровки закрытым ключом.

5.3 Неопровержимость деяния

В третьей главе рассказывалось о том, как Алиса и Боб благодаря хешированию с ключом могут убедиться в подлинности личности отправителя. Боб отправляет Алисе сообщение вместе с хешем. При получении сообщения она тоже высчитывает хеш. Если ее хеш совпадает с хешем отправителя, то Алиса может сделать два вывода: целостность данных сохранена, и их создателем является Боб.

Но давайте взглянем на ситуацию со стороны третьего участника переписки, Чарли. Знает ли он, кто автор сообщения? Нет, ведь одним и тем же ключом владеет как Алиса, так и Боб. Все, что знает Чарли, – что сообщение создал кто-то из них двоих, а вот кто конкретно – нет. Ничто не мешает Алисе создать сообщение и заявить, что она получила его от Боба. Ничто не мешает Бобу прислать Алисе сообщение, а затем заявить, что она сама его выдумала. Они-то оба знают, кто создал сообщение, но вот доказать это кому-то еще не имеют никакой возможности.

Если система не дает пользователю отрицать содеянное, то она обеспечивает *неопровержимость*, которая и не дала бы шанса Бобу отпираться. В жизни неопровержимость, как правило, требуется при отправке данных об операции на сервер. Например, это могут быть платежные операции на кассовом терминале. Неопровержимость здесь требуется, чтобы заставить процессинговый центр выполнить обязательства по договору. При этом третья сторона – например, надзорный орган – может проверять платежи.

Чтобы действия Алисы и Боба были неопровержимы, им придется отказаться от общего ключа и прибегнуть к цифровым подписям.

5.3.1 Цифровые подписи

Если требуется не просто проверка подлинности и целостности данных, а неопровержимость, то понадобится электронная цифровая подпись (ЭЦП). Ее применение позволяет кому угодно, а не только

получателю, ответить на два вопроса: кто отправил сообщение и дошло ли оно в первозданном виде. Цифровая подпись во многом похожа на рукописную:

- и та, и другая уникальна для каждой персоны;
- и та, и другая накладывает юридические обязательства на подписавшего;
- и ту, и другую трудно подделать.

ЭЦП обычно производится с помощью хеш-функции и зашифровки закрытым ключом. Чтобы подписать сообщение, сначала требуется высчитать его хеш. Затем полученное значение и закрытый ключ подаются на вход алгоритму асимметричного шифрования, и на выходе получается цифровая подпись. Таким образом владелец закрытого ключа подписывает созданное им сообщение. Если взглянуть со стороны алгоритма шифрования, то хеш – это открытый текст, а цифровая подпись – шифрованный. Сообщение и ЭЦП затем передаются вместе по сети связи. На рис. 5.2 показано, как подписал сообщение Боб.

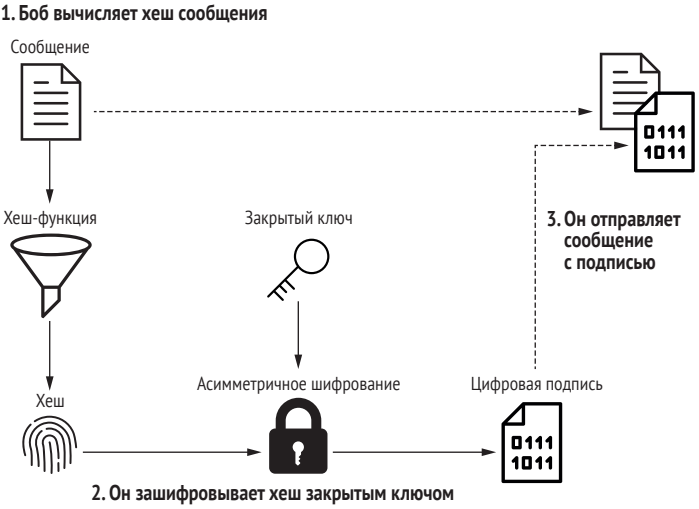

Рис. 5.2 Боб подписывает сообщение закрытым ключом, прикрепляет к нему полученную ЭЦП и отправляет Алисе

Цифровая подпись не хранится в секрете, а передается в открытом виде вместе с сообщением. На этом моменте некоторые разработчики ловят когнитивный диссонанс, ведь подпись – это шифротекст, который злоумышленник без труда может расшифровать открытым ключом. В этом нет ничего страшного. ЭЦП нужна для неопровержимости, ведь для ее генерации был задействован закрытый ключ владельца. Она не предназначена для неразглашения ее содержимого. Если взломщик расшифрует подпись, то все, что он получит, – хеш содержимого и никакой личной информации.

5.3.2 Подписание данных криптосистемой RSA

Боб облек процесс, изображенный на рис. 5.2, в показанный ниже код. Используются хеш-функция SHA-256, алгоритм шифрования RSA и дополнение данных по алгоритму PSS (probabilistic signature scheme). Метод `sign` класса `RSAPrivateKey` – сердце листинга, где и задействуется все перечисленное.

Листинг 5.5 Подписание данных криптосистемой RSA на Python

```
import json
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes

message = b'from Bob to Alice'

padding_config = padding.PSS(
    mgf=padding.MGF1(hashes.SHA256()),
    salt_length=padding.PSS.MAX_LENGTH)

private_key = load_rsa_private_key()
signature = private_key.sign(
    message,
    padding_config,
    hashes.SHA256())

signed_msg = {
    'message': list(message),
    'signature': list(signature),
}
outbound_msg_to_alice = json.dumps(signed_msg)
```

Схема дополнения PSS

Загрузка закрытого ключа, внутри этого метода код из листинга 5.3

Непосредственно подписание хеша SHA-256, высчитанного из сообщения

Сообщение для Алисы содержит изначальное послание и его ЭЦП

ВНИМАНИЕ! Схемы дополнения незначащими данными для подписания алгоритмом RSA и для зашифровки данных с помощью того же RSA не одинаковы. Стоит выбирать OAEP для шифрования и PSS для генерации ЭЦП. Они не взаимозаменяемы.

Итак, Алиса получила весточку от Боба. Но в самом ли деле от него? Для начала ей стоит проверить подпись.

5.3.3 Проверка подписи, созданной криптосистемой RSA

Как только Алиса получила сообщение и ЭЦП от Боба, она:

1. вычисляет хеш сообщения;
2. расшифровывает подпись открытым ключом Боба;
3. сравнивает хеш из открытого текста подписи с вычисленным ею ранее.

Если хеши равны, то она может доверять этому посланию. На рис. 5.3 изображено, как Алиса проверяет подпись при получении.

Рис. 5.3 Алиса получает сообщение от Боба, расшифровывает ЭЦП открытым ключом отправителя и сравнивает хеши. В этом и заключается проверка подписи

Алиса написала листинг 5.6 по мотивам схемы на рис. 5.3. Все три шага заключены внутри метода `verify` класса `RSAPublicKey`. Если высчитанный хеш не совпадает с хешем из ЭЦП Боба, то метод бросит исключение `InvalidSignature`. Если же они совпадают, то Алиса может быть уверена в двух вещах. Первое – что никто не исказил сообщение по пути. Второе – что отправил его некто, в чьем распоряжении есть закрытый ключ Боба, и высока вероятность, что он сам.

Листинг 5.6 Проверка подписи, созданной криптосистемой RSA, на Python

```
import json
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.exceptions import InvalidSignature

def receive(inbound_msg_from_bob):
    signed_msg = json.loads(inbound_msg_from_bob)
    message = bytes(signed_msg['message'])
    signature = bytes(signed_msg['signature'])

    padding_config = padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH)

    private_key = load_rsa_private_key()
    try:
        private_key.public_key().verify(
            signature,
            message,
            padding_config,
            hashes.SHA256())
        print('Сообщению можно доверять')
    except InvalidSignature:
        print('Сообщению нельзя доверять')
```

Получение сообщения и ЭЦП

Схема дополнения PSS

Загрузка закрытого ключа, внутри этого метода код из листинга 5.3

Метод verify проверяет все необходимое

Чарли, третий участник переписки, может убедиться в том, кто автор сообщения, точно так же, как это сделала Алиса. Цифровая подпись, таким образом, обеспечивает неопровержимость. Боб не сможет опровергнуть авторство, если только он не заявит, что его закрытый ключ был украден.

Злоумышленнице посередине, Мэллори, никак не получится успешно вмешаться в процесс переписки. Пусть она меняет как хочет хоть сообщение, хоть подпись. Хоть даже открытый ключ в момент передачи его от Боба Алисе по незащищенному каналу связи. Во всех этих случаях ЭЦП просто не пройдет проверку, и получательница не станет доверять посланию. В момент подсчета хеша Алисой искаженное сообщение даст неверный результат. Искаженные ЭЦП либо ключ обернутся проблемами с расшифровкой хеша либо его неверным значением[1].

В этом разделе для генерации цифровых подписей применялся алгоритм RSA. Время доказало, что он стойко справляется с задачей. Но увы, это затратный метод подписывать данные. В следующем разделе говорится о способе получше.

5.3.4 Подписание данных на базе эллиптических кривых

Как и в случае с RSA, в эллиптической криптографии тоже фигурируют пары ключей. Как и RSA, криптосистемами на основе эллиптических кривых можно подписывать данные и проверять созданную ЭЦП. Однако эллиптическая криптография не находит популярности в прикладных целях. Ее основная задача – генерация подписей, в то время как RSA используется для шифрования широкого спектра открытых текстов.

Почему же так вышло? Дело в том, что эллиптические криптосистемы затрачивают меньше вычислительной мощности на создание ЭЦП, подлинность которой потом можно проверить особого вида открытым ключом даже без получения открытого текста из шифротекста. Поэтому они теперь и стали использоваться для подписания сообщений вместо RSA.

Алгоритм RSA безопасен для применения, однако достаточно сравнить длины ключей, чтобы сделать выбор в пользу эллиптической криптографии для генерации ЭЦП. Ключ длиной 256 бит на основе эллиптических кривых по стойкости равен ключу RSA размером 3072 бита. Эта разница обусловлена математическими моделями, которые используются в алгоритмах. Эллиптическая крипто-

---

[1] Стоит заметить, что если злоумышленница сможет подменить сразу и открытый ключ, и сообщение, и ЭЦП, то сможет успешно посылать Алисе сообщения от имени Боба. То есть Алисе нужно быть стопроцентно уверенной в том, что открытый ключ принадлежит Бобу, и тогда она сможет отклонять искаженные сообщения. Для обеспечения такой уверенности применяются сертификаты открытого ключа. – *Прим. перев.*

графия прибегает к эллиптическим кривым над конечными полями, RSA применяет факторизацию целых чисел.

В листинге 5.7 Боб создает пару эллиптических ключей – открытый ключ можно будет вывести из закрытого – и подписывает закрытым ключом хеш SHA-256. По сравнению с RSA затрачивается меньше тактов центрального процессора, и даже строчек кода понадобилось меньше. Ключ сгенерирован по алгоритму SECP384R1, он же P-384, который одобрен Национальным институтом стандартов и технологий.

Листинг 5.7 Подписание данных эллиптической криптосистемой на Python

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec

message = b'from Bob to Alice'

private_key = ec.generate_private_key(ec.SECP384R1(), default_backend())

signature = private_key.sign(message, ec.ECDSA(hashes.SHA256()))
```

Подписание хеша SHA-256, высчитанного из сообщения

Листинг 5.8 продолжает предыдущий. Он изображает, как Алиса проверила бы подпись Боба. Открытый ключ получен из закрытого; если подпись неверна, метод `verify` кидает исключение `InvalidSignature` – все как в RSA.

Листинг 5.8 Проверка подписи, созданной эллиптической криптосистемой, на Python

```
from cryptography.exceptions import InvalidSignature

public_key = private_key.public_key()

try:
    public_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))
except InvalidSignature:
    pass
```

Вычисление открытого ключа из закрытого

Если подпись неверна, сделать то-то

Метод `sign`, который можно увидеть в листингах 5.5 и 5.7, высчитывает хеш от переданного сообщения. Однако в случае длинного послания либо их большого числа это может быть ресурсозатратная операция. Можно посчитать хеш предварительно эффективнее либо воспользоваться уже вычисленным хешем. Затем можно передать методу `sign` заготовленный хеш вместо сообщения, при этом передаваемую в метод хеш-функцию необходимо обернуть в класс `utils.Prehashed`. Это работает как для эллиптических криптосистем, так и для RSA.

Листинг 5.9 Подписание объемных сообщений на Python

```
import hashlib
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec, utils

large_msg = b'from Bob to Alice ...'
sha256 = hashlib.sha256()
sha256.update(large_msg[:8])
sha256.update(large_msg[8:])
hash_value = sha256.digest()
```

Вычисление хеша менее затратным путем

```
private_key = ec.generate_private_key(ec.SECP384R1(), default_backend())

signature = private_key.sign(
    hash_value,
    ec.ECDSA(utils.Prehashed(hashes.SHA256())))
```

Хеш-функция обернута в utils.Prehashed

На данный момент вы обрели практические навыки работы с хешированием, шифрованием и цифровыми подписями. Вы узнали, что:

- хеширование позволяет убедиться в целостности данных и их подлинности;
- шифрование обеспечивает неразглашение;
- цифровые подписи гарантируют неопровержимость.

В этой главе в образовательных целях неоднократно были использованы низкоуровневые инструменты из пакета `cryptography`. Это все ради фундамента для понимания высокоуровневой технологии, о которой мы поговорим в следующей главе, а именно сетевого протокола Transport Layer Security. В нем сходится воедино все, что вы узнали о вычислении хешей, криптосистемах и ЭЦП.

Итоги

- Алгоритмам асимметричного шифрования требуются разные ключи для зашифровки и расшифровки.
- Разделение ключей на закрытый и открытый решает трудности с их передачей.
- RSA – классический и безопасный выбор для асимметричного шифрования.
- Цифровые подписи обеспечивают неопровержимость.
- ЭЦП на основе эллиптических кривых вычисляются быстрее подписей на базе RSA.