Криптография
для Java-программиста

Сергей Владимиров

- 25 лет опыта разработки (с 1995 года)
  - 15 лет коммерческой разработки (с 2004)
- 18 лет пишу на Java (с 2001 года)

- Преподаватель курсов «Защита информации» и «Криптографические протоколы» в МФТИ с 2008 года

Криптография для Java-программиста

• Генератор случайных чисел	• java.security.SecureRandom
• Шифрование	• javax.crypto.Cipher
• Хеш-функции	• java.security.MessageDigest
• Шифрование с открытым ключом	• javax.crypto.Cipher
• ЭЦП	• java.security.Signature

Криптография для Java-программиста

- Генератор случайных чисел
- java.security.SecureRandom
- Шифрование
- javax.crypto.Cipher
- Хеш-функции
- java.security.MessageDigest
- Шифрование с открытым ключом
- javax.crypto.Cipher
- ЭЦП
- java.security.Signature

```
@since 1.4
```

Secure Random

Криптографически-стойкий генератор псевдослучайных чисел

Secure Random

- Генератор случайных чисел
- Почти как java.util.Random
- …только криптографически стойкий

Secure Random

- Генератор случайных чисел
- Почти как java.util.Random
- …только криптографически стойкий

- никто не предскажет следующий выход, даже зная предыдущие

Secure Random

```
var random = new SecureRandom();

var bytes  = new byte[20];

var result = random.nextBytes( bytes );
```

Secure Random

```
var random = new SecureRandom();
var bytes  = new byte[20];
var result = random.nextBytes( bytes );

var i = random.nextInt()
var l = random.nextLong()
var d = random.nextDouble()
```

Secure Random

```
var random = new SecureRandom();


String newPassword = random.ints( 30, 'A', 'Z' )
                           .mapToObj( Integer::toString )
                           .collect( joining() )
```

Secure Random: что делать точно не надо!

- Не нужно "усиливать безопасность":

```
//  так делать не нужно
new SecureRandom( new SecureRandom().generateSeed( 128 ) )

//  так тоже не надо
var nextByte = (byte) ( random.nextLong() && 0xFF00 >> 8 )
var nextInt = (int) ( random.nextLong() && 0xFFFFFFFF )
```

Secure Random: "максимальная защита"

```
var secureRandom = SecureRandom.getInstanceStrong()
```

- Обеспечивает защиту получше
- Но может привести к бесконечному ожиданию
  - В зависимости от версий и настроек ОС и JVM

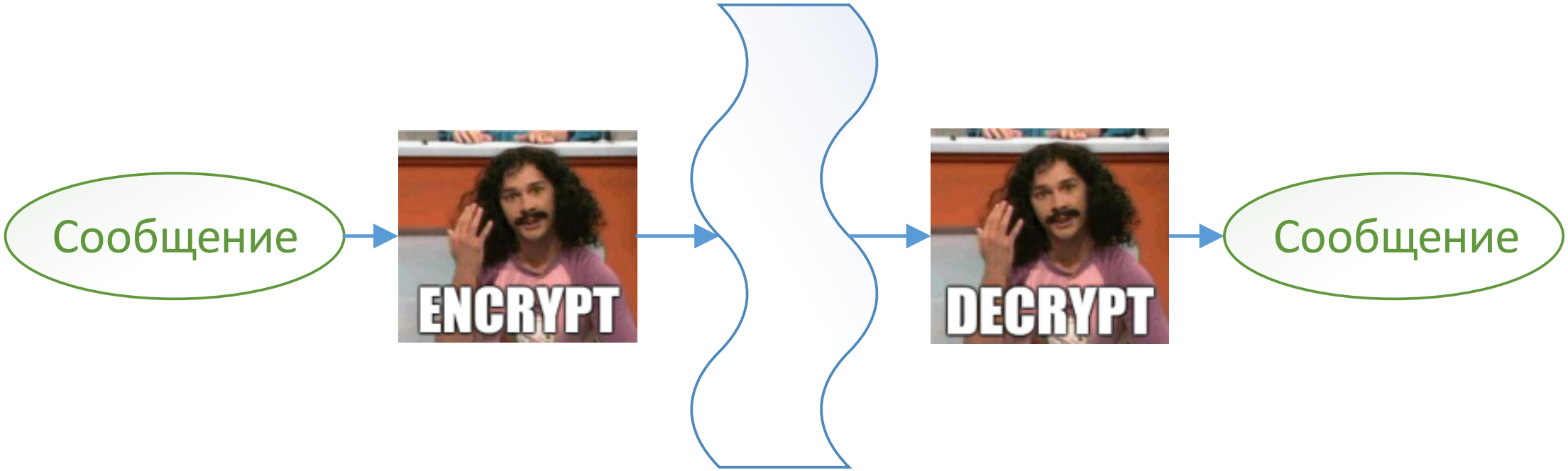
javax.crypto.Cipher

Симметричное шифрование

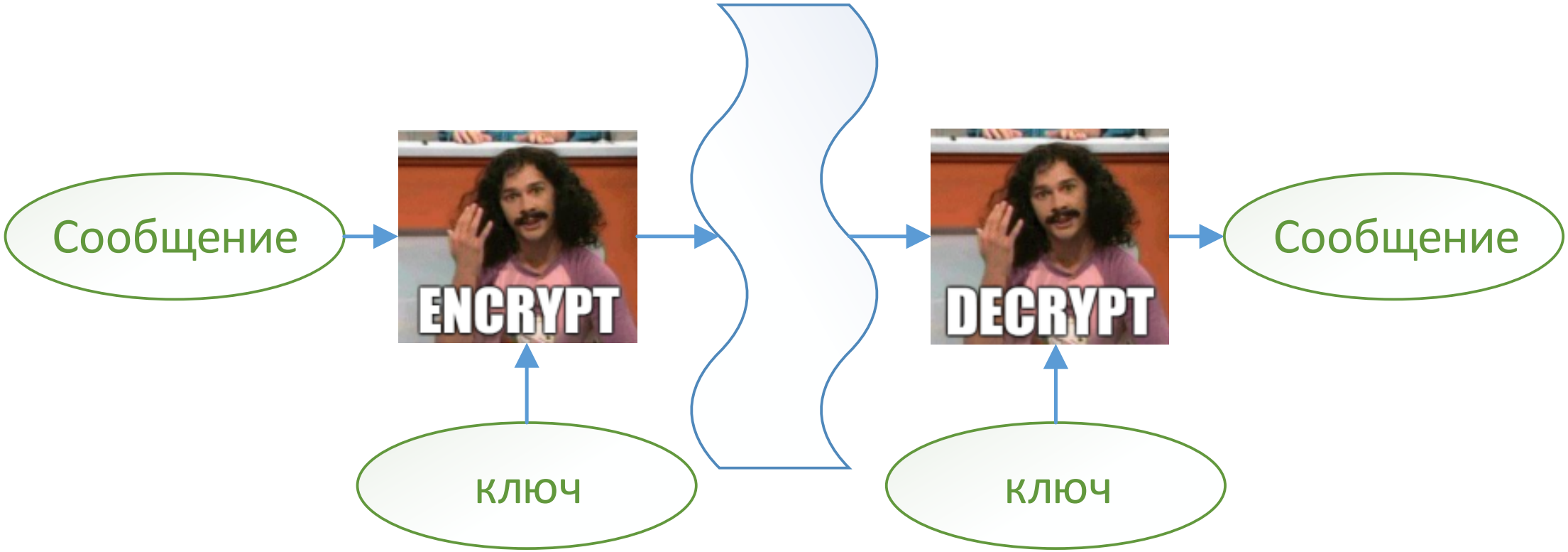
a.k.a.

шифрование на секретном ключе

javax.crypto.Cipher



javax.crypto.Cipher



javax.crypto.Cipher

javax.crypto.Cipher

javax.crypto.Cipher: выбор transformation

Алгоритм шифрования

a.k.a. transformation:

`AES/GCM/NoPadding`

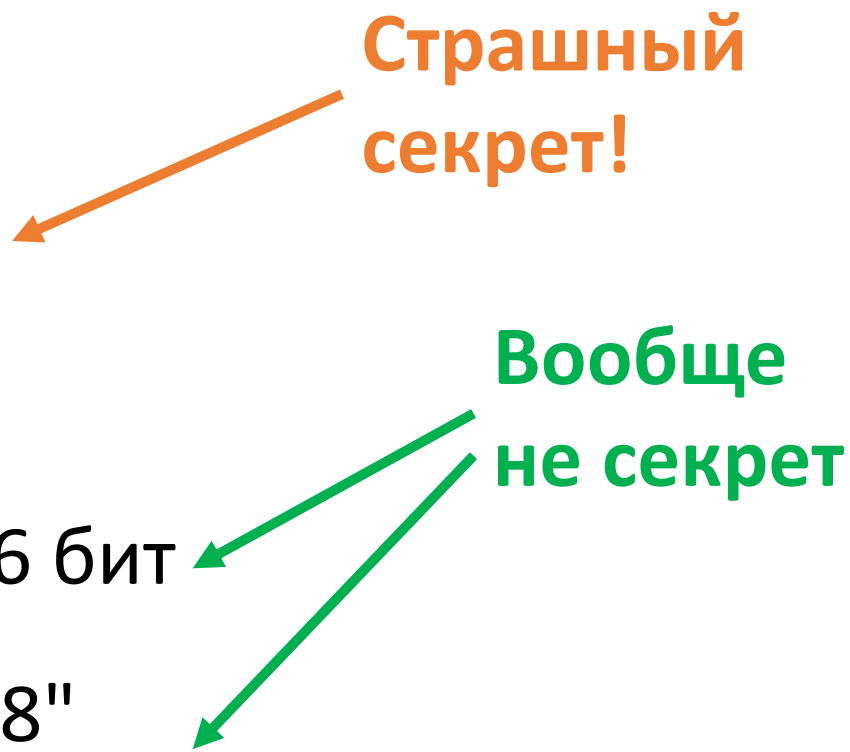
javax.crypto.Cipher

javax.crypto.Cipher: ключ и параметры

- AES/GCM/NoPadding:
  - Ключ: 128, 192 или 256 бит
  - Параметры:
    - Вектор инициализации IV: 96 бит
    - Длина тега целостности: "128" (бит)

javax.crypto.Cipher: ключ и параметры

- AES/GCM/NoPadding:
  - Ключ: 128, 192 или 256 бит
  - Параметры:
    - Вектор инициализации IV: 96 бит
    - Длина тега целостности: "128"



javax.crypto.Cipher: ключ и параметры

- AES/GCM/NoPadding:
  - Ключ: 128, 192 или 256 бит
  - Параметры:
    - Вектор инициализации IV: 96 бит
    - Длина тега целостности: "128"

Страшный секрет!

Вообще не секрет

При шифровании Java сгенерирует самостоятельно ✓

javax.crypto.Cipher: получение ключа

A. Просто записать случайные 16 байт в конфигурацию в base64 или в hex
   - `9sJ9Rg8ph04VGjC3jU5L`
   - `3f2f6ffce4b0f5eb25c5df6d94e724`

javax.crypto.Cipher

AES Cipher Example

@ GitHib gist

javax.crypto.Cipher: генерация ключа

B. Использовать генерацию ключа из пароля

```
byte[] SALT = "Salt1".getBytes(StandardCharsets.UTF_8);
char[] SECRET_PHRASE = "mySecretPassword".toCharArray();


SecretKey generateKey( char[] secretPhrase ) throws Exception {
  var factory = SecretKeyFactory.getInstance( "PBKDF2WithHmacSHA256" );
  var spec = new PBEKeySpec( secretPhrase, SALT, 1, 128 );
  var temp = factory.generateSecret( spec );
  return new SecretKeySpec( temp.getEncoded(), "AES" );
}
```

javax.crypto.Cipher: генерация ключа

B. Использовать генерацию ключа из пароля

```
byte[] SALT = "Salt1".getBytes(StandardCharsets.UTF_8);
char[] SECRET_PHRASE = "mySecretPassword".toCharArray();


SecretKey generateKey( char[] secretPhrase ) throws Exception {
  var factory = SecretKeyFactory.getInstance( "PBKDF2WithHmacSHA256" );
  var spec = new PBEKeySpec( secretPhrase, SALT, 1, 128 );
  var temp = factory.generateSecret( spec );
  return new SecretKeySpec( temp.getEncoded(), "AES" );
}
```

```
byte[] encrypt( SecretKey secretKey, byte[] openText ) throws Exception {
  var cipher = Cipher.getInstance( "AES/GCM/NoPadding" );
  cipher.init( Cipher.ENCRYPT_MODE, secretKey );

  byte[] cipherText = cipher.doFinal( openText );

  var gcmParams = cipher.getParameters().
                         getParameterSpec( GCMParameterSpec.class );
  byte[] iv = gcmParams.getIV();

  var result = new byte[ iv.length + cipherText.length ];
  System.arraycopy( iv, 0, result, 0, iv.length );
  System.arraycopy( cipherText, 0, result, iv.length, cipherText.length );
  return result;
}
```

Результат выполнения метода encrypt()

- `byte[] result`

  - 12 байт вектор инициализации
  - N байт – данные

javax.crypto.Cipher: расшифрование

1. получить из конфигурации / от пользователя ключ
2. извлечь из данных вектор инициализации IV
3. расшифровать

```java
int IV_LENGTH = 12; // bytes
int TAG_SIZE = 128; // bits

byte[] decrypt( SecretKey secretKey, byte[] message ) throws Exception {
  var iv = Arrays.copyOfRange( message, 0, IV_LENGTH );
  var cipherText = Arrays.copyOfRange( message, IV_LENGTH,
                                       message.length );

  var gcmParams = new GCMParameterSpec( TAG_SIZE, iv );
  var cipher = Cipher.getInstance( "AES/GCM/NoPadding" );
  cipher.init( Cipher.DECRYPT_MODE, secretKey, gcmParams );

  byte[] openText = cipher.doFinal( cipherText );
  return openText;
}
```

javax.crypto.Cipher

- Выход decrypt:
  a) `byte[]`
  b) `Exception`

Использование шифрования

- Шифрование – это создание ***защищённого канала связи***

Использование шифрования

- Другие каналы связи:
  - tcp
  - http
  - файловая система
  - реляционная база данных
  - redis
  - cookies

Использование шифрования

- …сделать ненадёжный канал связи более надёжным

Использование шифрования

- …сделать ненадёжный канал связи более надёжным

- …использовать ненадёжный канал связи вместо надёжного

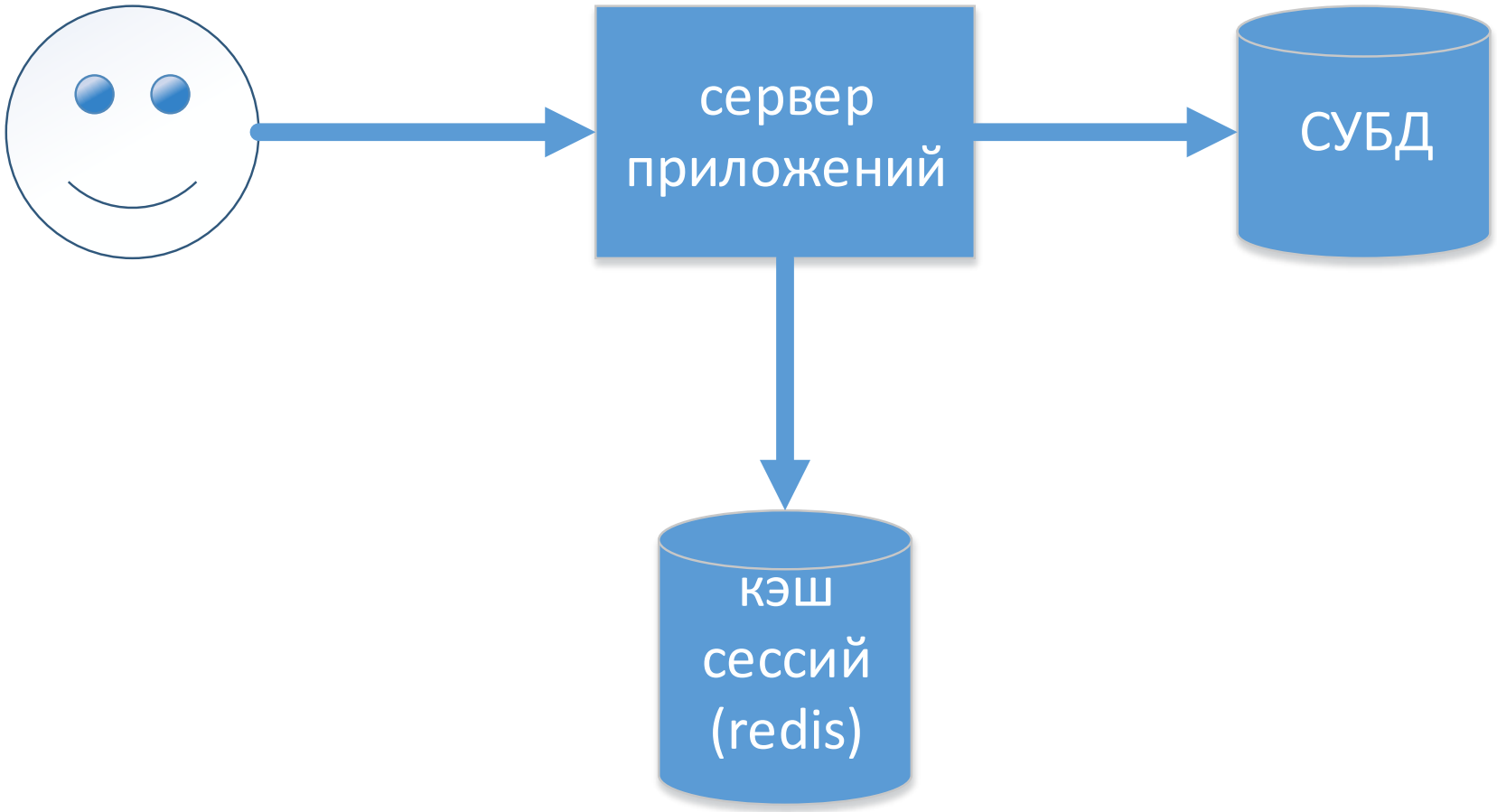
Использование шифрования

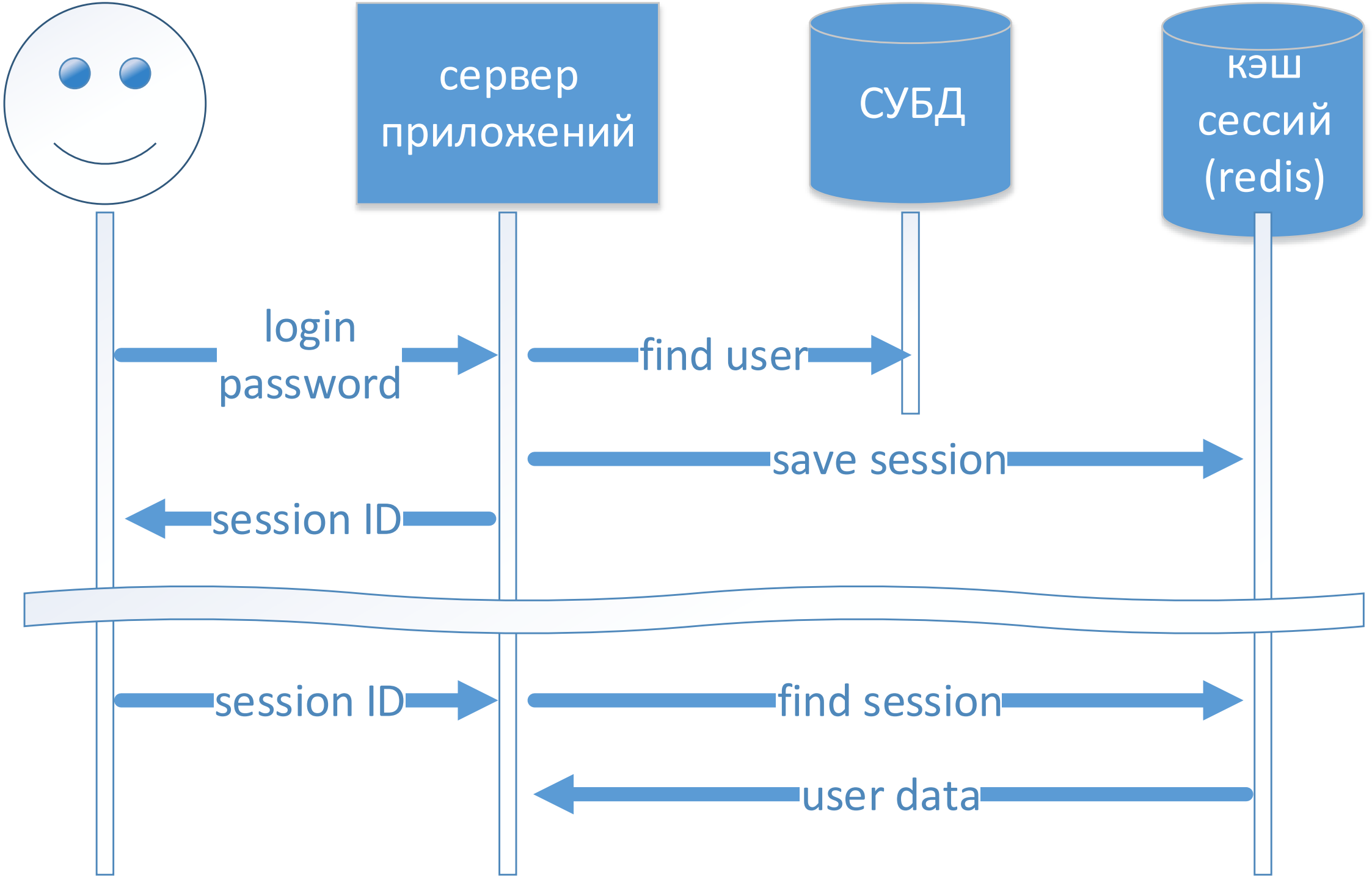
- …сделать ненадёжный канал связи более надёжным
- …использовать ненадёжный канал связи вместо надёжного
- …отказаться от лишнего канала связи…

Использование шифрования

- …сделать ненадёжный канал связи более надёжным
- …использовать ненадёжный канал связи вместо надёжного
- …отказаться от лишнего канала связи… например от базы данных

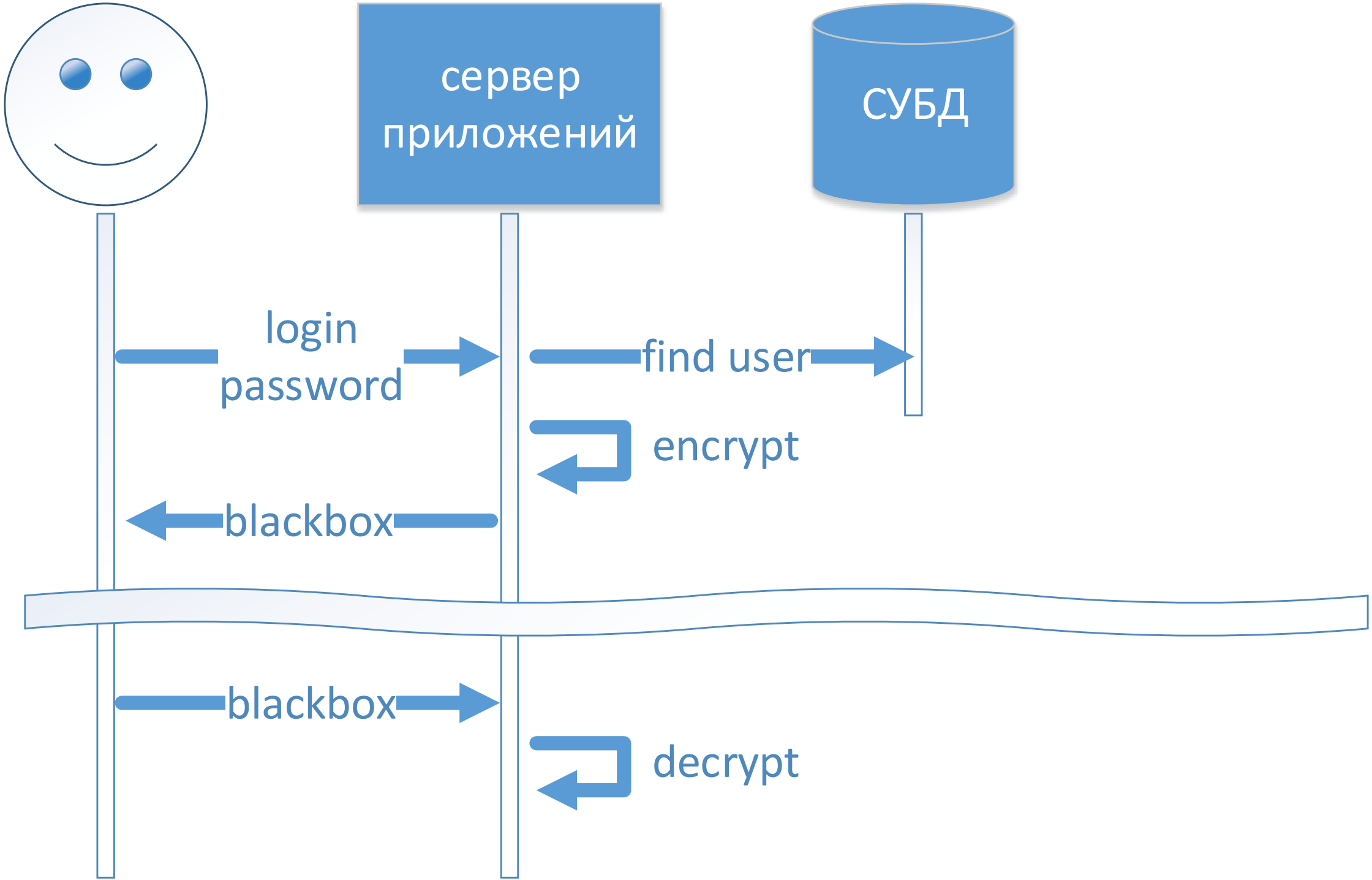
Использование шифрования




сервер приложений
СУБД
кэш сессий (redis)
login
password
find user
save session
session ID
session ID
find session
user data

Использование шифрования

- мы храним данные в redis, потому что не доверяем передачу данных пользователю:
  - пользователь может поменять userid / свои права в отправляемых данных (cookie)
  - может мы просто не хотим отдавать что-то пользователю (напр. карму)


сервер
приложений
СУБД
login
password
find user
encrypt
blackbox
blackbox
decrypt

Плюсы использования шифрования

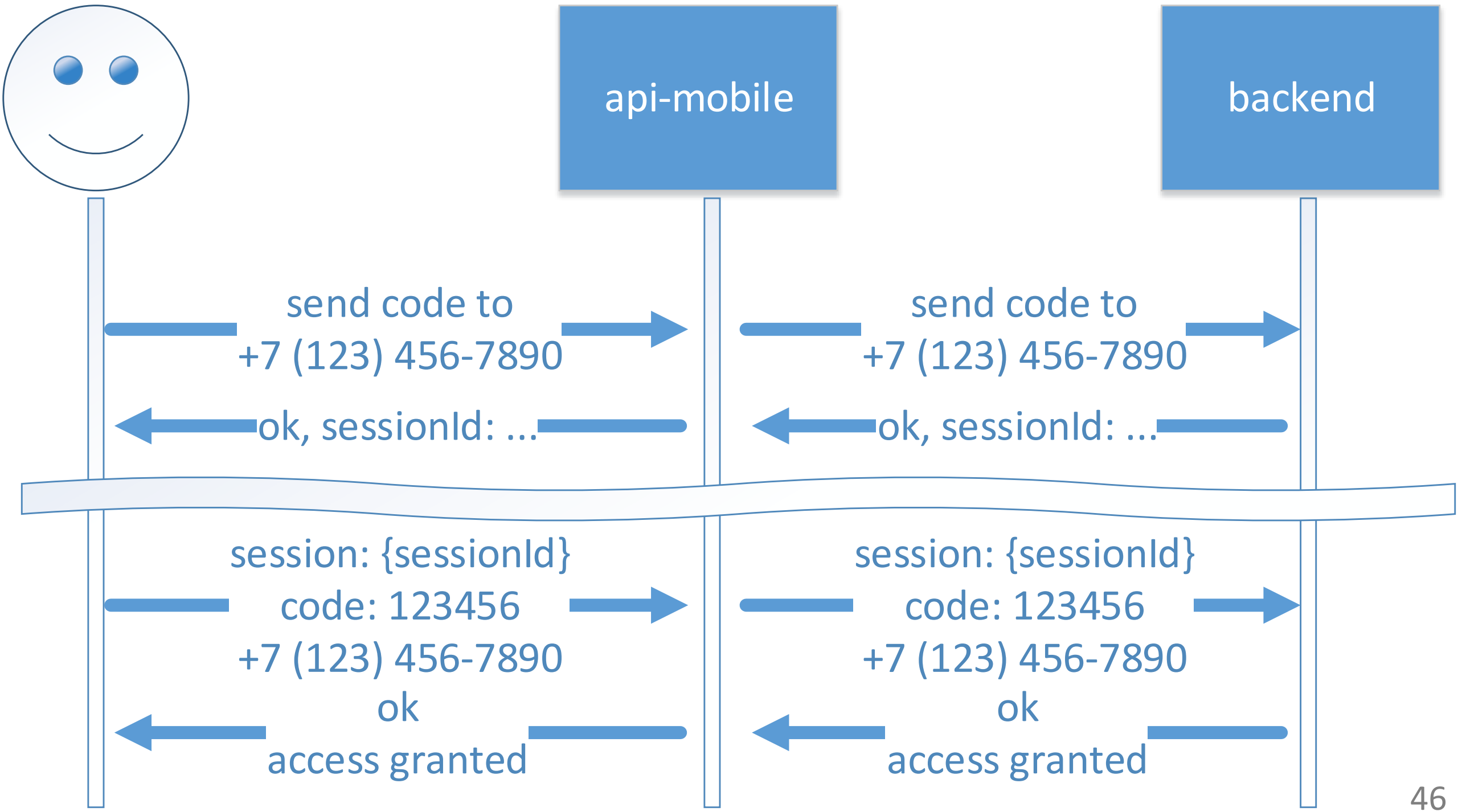
- не нужен дополнительный сервис (кэш)
- легкое масштабирование решения
  - (даже между дата-центрами)

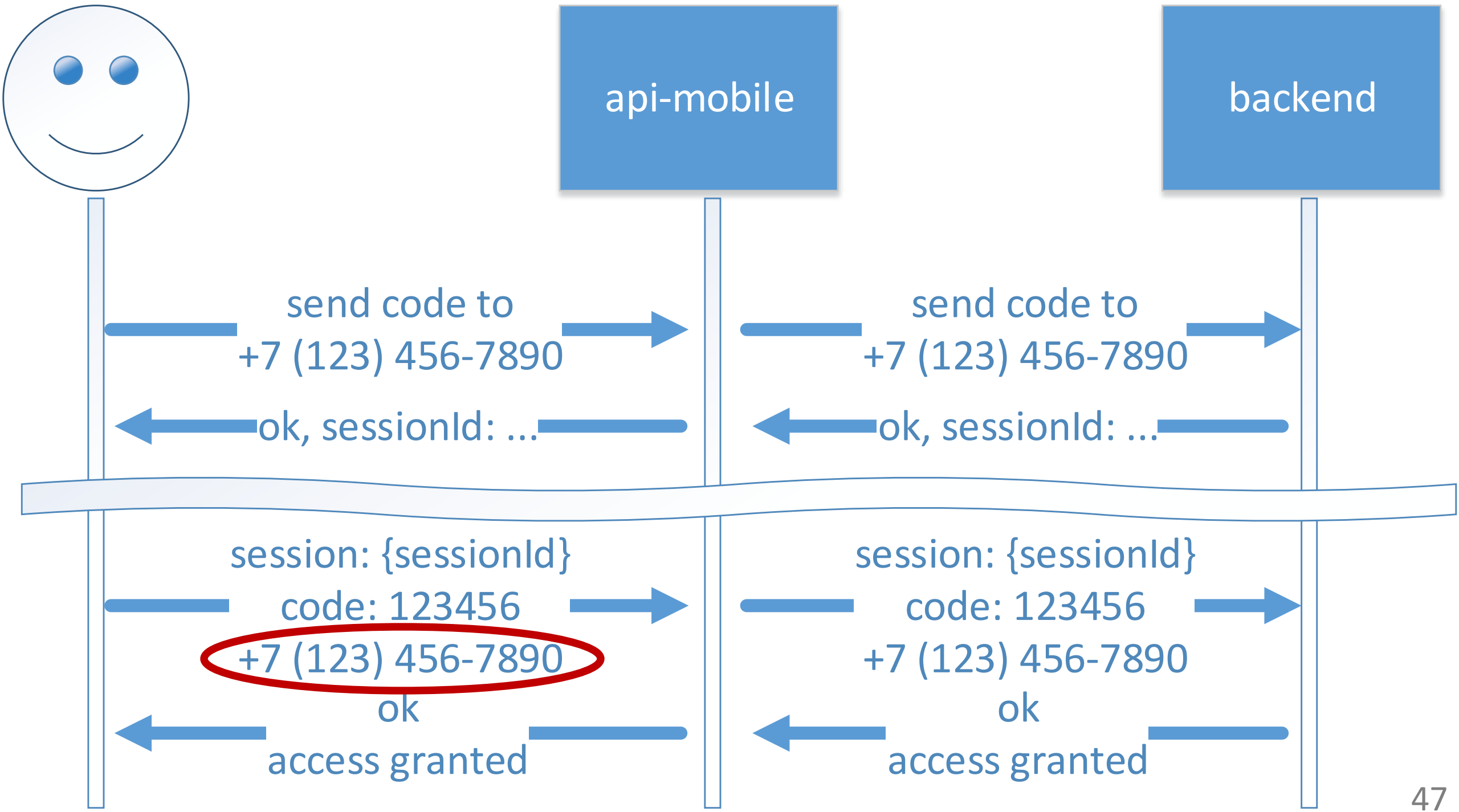
Минусы использования шифрования

Минусы использования шифрования

- нет инвалидации кэша

Использование шифрования: blackbox'инг


api-mobile
backend
send code to
+7 (123) 456-7890
send code to
+7 (123) 456-7890
ok, sessionId: ...
ok, sessionId: ...
session: {sessionId}
code: 123456
+7 (123) 456-7890
ok
access granted
session: {sessionId}
code: 123456
+7 (123) 456-7890
ok
access granted


api-mobile
backend
send code to
+7 (123) 456-7890
send code to
+7 (123) 456-7890
ok, sessionId: ...
ok, sessionId: ...
session: {sessionId}
code: 123456
+7 (123) 456-7890
ok
access granted
session: {sessionId}
code: 123456
+7 (123) 456-7890
ok
access granted

Как предотвратить изменение телефона?

- Валидировать связку sessionId <-> phone
  - где-то хранить эту связку?

Как предотвратить изменение телефона?

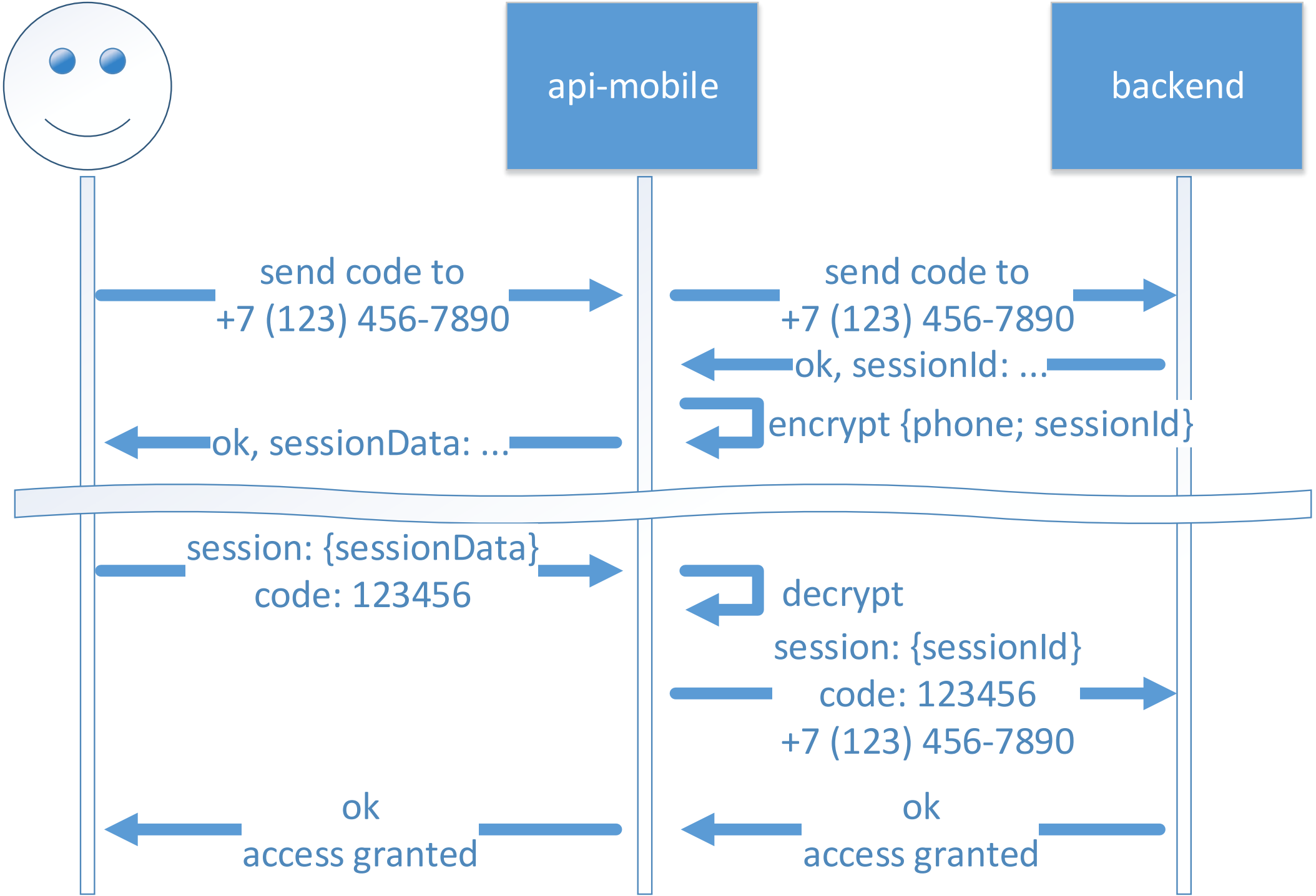
- Валидировать связку sessionId <-> phone
  - где-то хранить эту связку?

- Использовать шифрование?
  - а как?

Как предотвратить изменение телефона?

- Валидировать связку sessionId <-> phone
  - где-то хранить эту связку?

- Использовать шифрование?
  - посылать все данные в sessionId


api-mobile
backend
send code to
+7 (123) 456-7890
send code to
+7 (123) 456-7890
ok, sessionId: ...
encrypt {phone; sessionId}
ok, sessionData: ...
session: {sessionData}
code: 123456
decrypt
session: {sessionId}
code: 123456
+7 (123) 456-7890
ok
access granted
ok
access granted

java.security.MessageDigest

Криптографические хеш-функции

Хеш-функции

- `hashCode()` как составная часть [...]HashMap
  - преобразование ключа произвольной длины в значение заданной [фиксированной] длины

- `hashCode()`
  - быстрая
  - минимум коллизий

Хеш-функции

обычные

- быстрая
- минимум коллизий

криптографические

- быстрая
- "устойчивость"

Криптографические хеш-функции

- Преобразование аргумента произвольной длины в аргумент заданной длины
- Устойчивость
  - …к восстановлению прообраза
  - …к поиску второго прообраза
  - …к поиску коллизий
  - …к удлинению прообраза

Криптографические хеш-функции

- Преобразование аргумента произвольной длины в аргумент заданной длины
- Устойчивость
  - …к восстановлению прообраза
  - …к поиску второго прообраза
  - …к поиску коллизий
  - …к удлинению прообраза

java.security.MessageDigest

```
byte[] src = …

var md = MessageDigest.getInstance( "SHA-256" );

byte[] digest = md.digest( src );
```

java.security.MessageDigest

- (std) MD5
- (std) SHA-1
- (std) SHA-256
- MD2
- SHA-224
- SHA-384
- SHA-512

java.security.MessageDigest

- (std) MD5
- (std) SHA-1
- (std) SHA-256
- MD2
- SHA-224
- SHA-384
- SHA-512

Что можно делать с MessageDigest?

- Использовать как уникальный идентификатор
  - Некоторые виды GUID "внутри" используют хеш-функции
- Использовать для скрытия информации (т.е. хранения паролей)
- Использовать для поддержания целостности данных

Что не надо делать с MessageDigest?

- Не надо комбинировать разные функции

  для "повышения уникальности" или "защищённости"

  - `MD5( message ) || SHA256( message ) || GOST( message )`
  - `SHA256( MD5( message ) )`

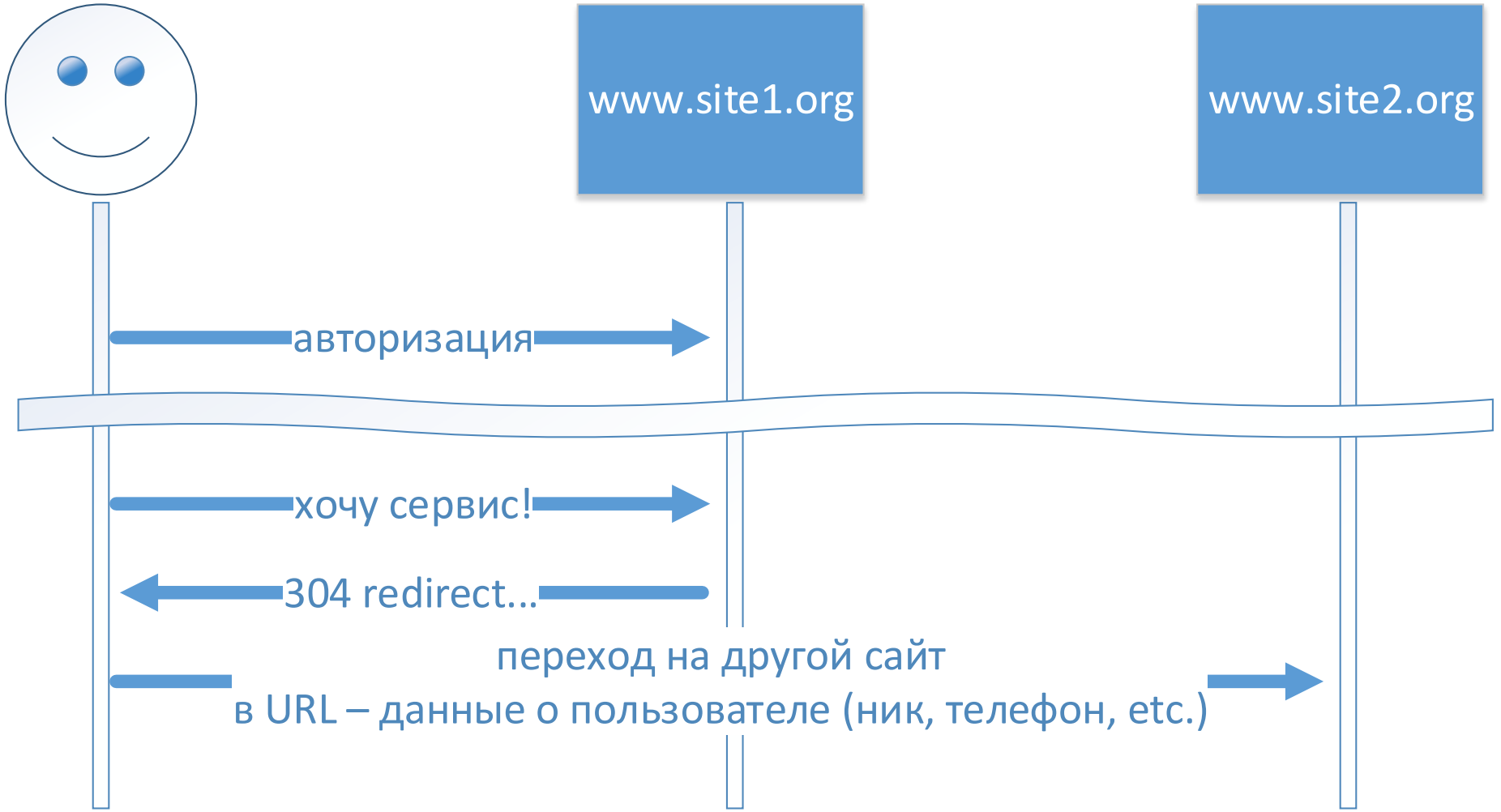
- Не надо "обрезать" функцию для повышения защиты

  - `SHA256( message ).substring( 0, length - 50 );`

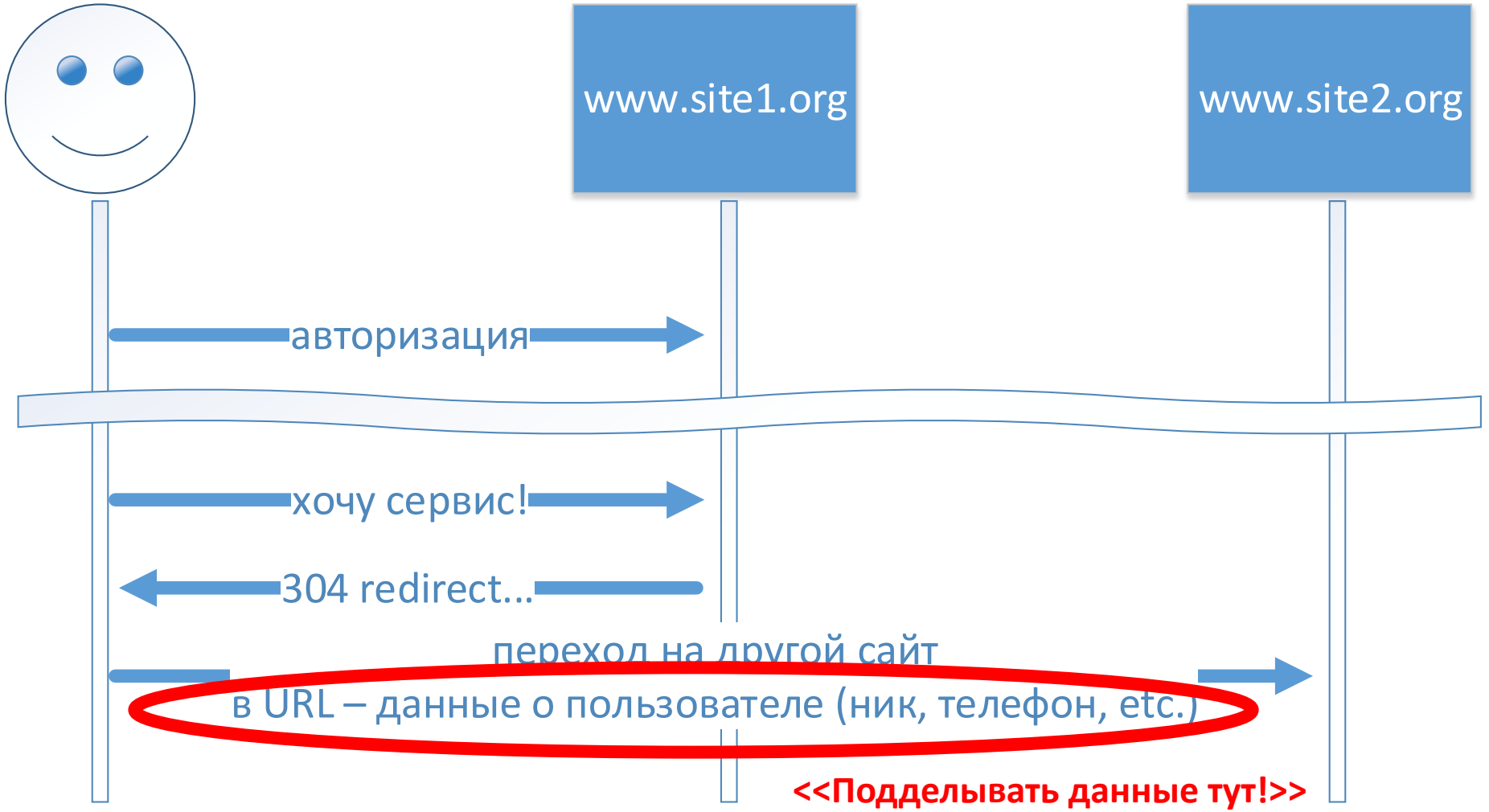
Что не надо делать с MessageDigest?

- Не хешировать то, что имеет ограниченное количество значений:
  - ENUM
  - номера телефонов
  - email'ы
  - номера кредитных карт
  - PIN-коды (*)

Интеграция с внешними системами



Интеграция с внешними системами



Интеграция с внешними системами

a) можно использовать дополнительный канал общения ("защищённый от пользователя" канал связи)

b) можно зашифровать данные (и передавать просто длинную строку)

c) можно передавать данные как обычно, но дополнить её "подписью" на основе хеш-функции

Интеграция с внешними системами

шифрование

- (+) ничего не видно
- (-) новый "endpoint"
- (-) ничего не видно

хеширование

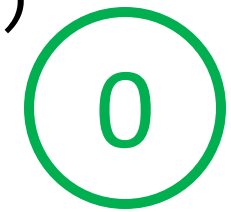
- (+) новый аргумент в старом API
- (+) данные видны в URL
- (-) данные видны в URL

MessageDigest

```
String digest( String... data ) {
  var srcString = Arrays.stream( data )
    .map( s -> s == null ? "" : s )
    .map( String::trim )
    .map( s -> normalize( s, Form.NFC ) )
    .collect( joining() );

  var srcBytes = srcString.getBytes(StandardCharsets.UTF_8);

  var md = MessageDigest.getInstance( "SHA-256" );
  var result = md.digest( srcBytes );
  return Base64.getEncoder().encodeToString( result );
}
```



```
String encode( String str ) {
  return URLEncoder.encode( str,
                            StandardCharsets.UTF_8.name() );
}

void redirectWithSignature(
          HttpServletResponse response,
          String name, String phone
  ) {
  response.sendRedirect( "https://site2.org/path"
    + "?name=" + encode( name )
    + "&phone=" + encode( phone )
    + "&digest=" + encode( digest( name, phone ) ) );
}
```

Интеграция с внешними системами

- Происходит перенаправление на адрес
  - https://site2.org/path?name=...&phone=...&digest=...

Интеграция с внешними системами

- Происходит перенаправление на адрес
  - https://site2.org/path?name=...&phone=...&digest=...

- Всё норм?

Интеграция с внешними системами

- Происходит перенаправление на адрес
  - https://site2.org/path?name=...&phone=...&digest=...

- ~~Всё норм?~~

- Ничего не мешает злоумышленнику вычислить digest самостоятельно!

Интеграция с внешними системами

- Происходит перенаправление на адрес
  - https://site2.org/path?name=...&phone=...&digest=...

- Не хватает секретной соли!
  - не передаётся, но участвует в вычислении digest
  - `digest( name || phone || secretSalt )`

```
String SALT = "Some secret salt for site2 only";


void redirectWithSignature(
          HttpServletResponse response,
          String name, String phone
  ) {
  response.sendRedirect( "https://site2.org/path"
    + "?name=" + encode(name)
    + "&phone=" + encode(phone)
    + "&digest=" + encode(digest(name, phone, SALT)));
}
```

```
String SALT = "Some secret salt for site2 only";


void redirectWithSignature(
          HttpServletResponse response,
          String name, String phone
  ) {
  response.sendRedirect( "https://site2.org/path"
    + "?name=" + encode(name)
    + "&phone=" + encode(phone)
    + "&digest=" + encode(digest(name, phone, SALT)));
}
```

Соль своя
для каждого сайта

Соль в конце!

Хранение паролей

Хранение паролей

- Не изобретайте велосипед, используйте Spring Security Crypto

Хранение паролей

- Не изобретайте велосипед, используйте Spring Security Crypto

```
@Autowired
private PasswordEncoder pEncoder;

// Запись значения хеш-функции от пароля в базе данных
String toStoreInDb = pEncoder.encode( newPlainPassword )

// Сравнение введённого пароля и хранимого хеша
boolean matches = pEncoder.matches( enteredPass, storedInDb );
```

javax.crypto.Cipher

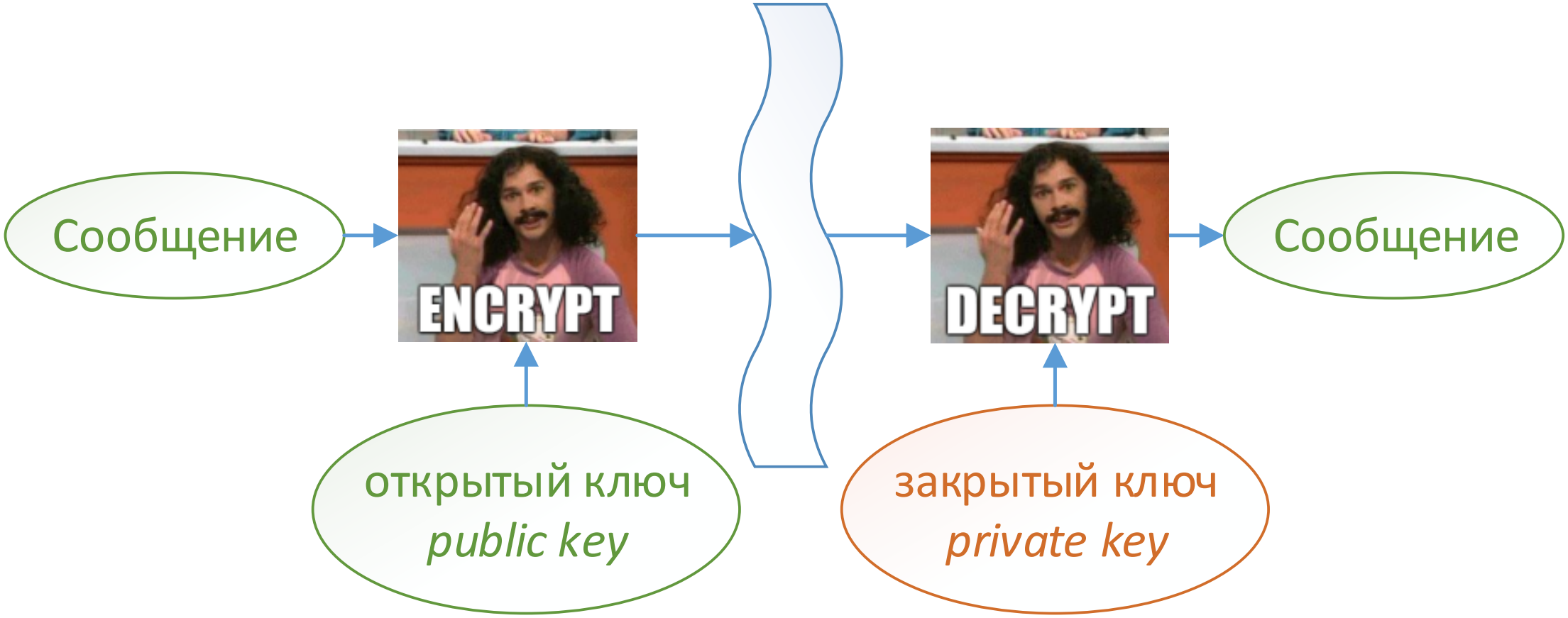
Асимметричное шифрование

a.k.a.

шифрование с открытым ключом

javax.crypto.Cipher

javax.crypto.Cipher



javax.crypto.Cipher

Форматы входных данных

- Сертификаты
  - PKCS#12 (binary)
  - DER
  - PEM "-----BEGIN CERTIFICATE-----"
- Ключи
  - PEM "----BEGIN PUBLIC KEY----"
  - PKCS#1 RSAPublicKey "----BEGIN RSA PUBLIC KEY----"
  - RFC 4253 "ssh-dsa …"
  - RFC 4253 "ssh-rsa …"
- Java keystore

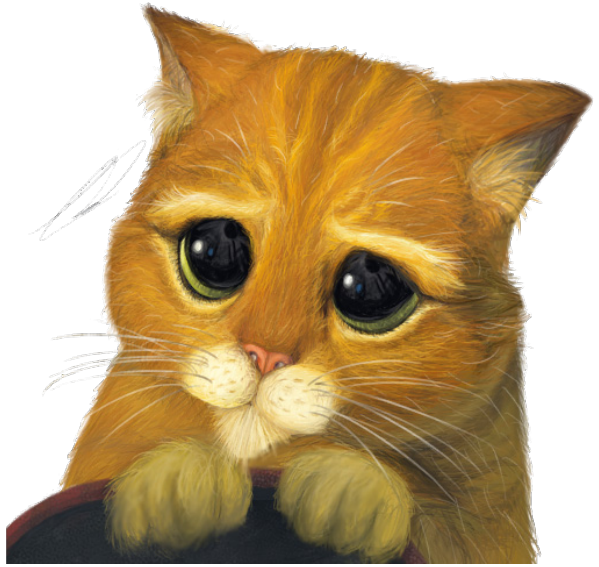
Форматы входных данных

- Сертификаты
  - PKCS#12 (binary)
  - DER
  - PEM "-----BEGIN CERTIFICATE-----"
- Ключи
  - PEM "----BEGIN PUBLIC KEY----"
  - PKCS#1 RSAPublicKey "----BEGIN RSA PUBLIC KEY----"
  - RFC 4253 "ssh-dsa …"
  - RFC 4253 "ssh-rsa …"
- Java keystore

Форматы входных данных

- Сертификаты
  - PKCS#12 (binary)
  - DER
  - PEM "-----BEGIN CERTIFICATE-----"
- Ключи
  - PEM "----BEGIN PUBLIC KEY----"
  - PKCS#1 RSAPublicKey "----BEGIN RSA PUBLIC KEY----"
  - RFC 4253 "ssh-dsa …"
  - RFC 4253 "ssh-rsa …"
- Java keystore

Пожалуйста, не шифруйте сертификаты и не защищайте их паролями!



Форматы входных данных

- `java.security.PublicKey`
  - `java.security.interfaces.DSAPublicKey`
  - `java.security.interfaces.ECPublicKey`
  - `java.security.interfaces.RSAPublicKey`

Форматы входных данных

```
// из сертификата X.509
X509Certificate certificate = … ;
PublicKey publicKey = certificate.getPublicKey();

// из Java-keystore
var keystore = KeyStore.getInstance( KeyStore.getDefaultType() );
var cert = keystore.getCertificate( alias );
PublicKey publicKey = cert.getPublicKey();
```

Шифрование с RSA

```
PublicKey publicKey = …;

byte[] data = …;

var cipher = Cipher.getInstance( "RSA/…" );

cipher.init( Cipher.ENCRYPT_MODE, publicKey );

byte[] cipherText = cipher.doFinal( data );
```

Ой, опять Transformation…

- RSA/ECB/ +
  - NoPadding,
    PKCS1Padding,
    OAEPWithMD5AndMGF1Padding,
    OAEPWithSHA1AndMGF1Padding,
    OAEPWithSHA-1AndMGF1Padding,
    OAEPWithSHA-224AndMGF1Padding,
    OAEPWithSHA-256AndMGF1Padding,
    OAEPWithSHA-384AndMGF1Padding,
    OAEPWithSHA-512AndMGF1Padding

Ой, опять Transformation…

- RSA/ECB/ +
  - ~~NoPadding,~~
    ~~PKCS1Padding,~~
    ~~OAEPWithMD5AndMGF1Padding,~~
    ~~OAEPWithSHA1AndMGF1Padding,~~
    ~~OAEPWithSHA-1AndMGF1Padding,~~
    ~~OAEPWithSHA-224AndMGF1Padding,~~
    OAEPWithSHA-256AndMGF1Padding,
    OAEPWithSHA-384AndMGF1Padding,
    OAEPWithSHA-512AndMGF1Padding

Ой, опять Transformation…

- RSA/ECB/ +
  - NoPadding,
    PKCS1Padding,
    OAEPWithMD5AndMGF1Padding,
    OAEPWithSHA1AndMGF1Padding,
    OAEPWithSHA-1AndMGF1Padding,
    OAEPWithSHA-224AndMGF1Padding,
    OAEPWithSHA-256AndMGF1Padding,
    OAEPWithSHA-384AndMGF1Padding,
    OAEPWithSHA-512AndMGF1Padding

Шифрование с RSA

```
String TRANSFORMATION = "RSA/ECB/OAEPWithSHA-256AndMGF1Padding";

byte[] encrypt( PublicKey publicKey, byte[] openText ) {
  var cipher = Cipher.getInstance( TRANSFORMATION );
  cipher.init( Cipher.ENCRYPT_MODE, publicKey );
  byte[] cipherText = cipher.doFinal( openText );
  return cipherText;
}
```

Расшифрование с RSA

```
String TRANSFORMATION = "RSA/ECB/OAEPWithSHA-256AndMGF1Padding";

byte[] encrypt( PrivateKey privateKey, byte[] cipherText ) {
  var cipher = Cipher.getInstance( TRANSFORMATION );
  cipher.init( Cipher.DECRYPT_MODE, privateKey );
  byte[] openText = cipher.doFinal( cipherText );
  return openText;
}
```

RSA в Java

- Нет "мучений" с параметрами сцепления блоков (IV)
- Не требуется комбинировать IV и данные

RSA в Java

- Нет "мучений" с параметрами сцепления блоков (IV)
- Не требуется комбинировать IV и данные

- Цена?

RSA в Java

```
Exception in thread "main" javax.crypto.IllegalBlockSizeException:
```

Data must not be longer than 62 bytes

```
        at com.sun.crypto.provider.RSACipher.doFinal(RSACipher.java:344)
        at com.sun.crypto.provider.RSACipher.engineDoFinal(RSACipher.java:389)
        at javax.crypto.Cipher.doFinal(Cipher.java:2165)
        at ru.nspk.vlsergey.test.RsaTest.encrypt(RsaTest.java:37)
        at ru.nspk.vlsergey.test.RsaTest.main(RsaTest.java:28)
```

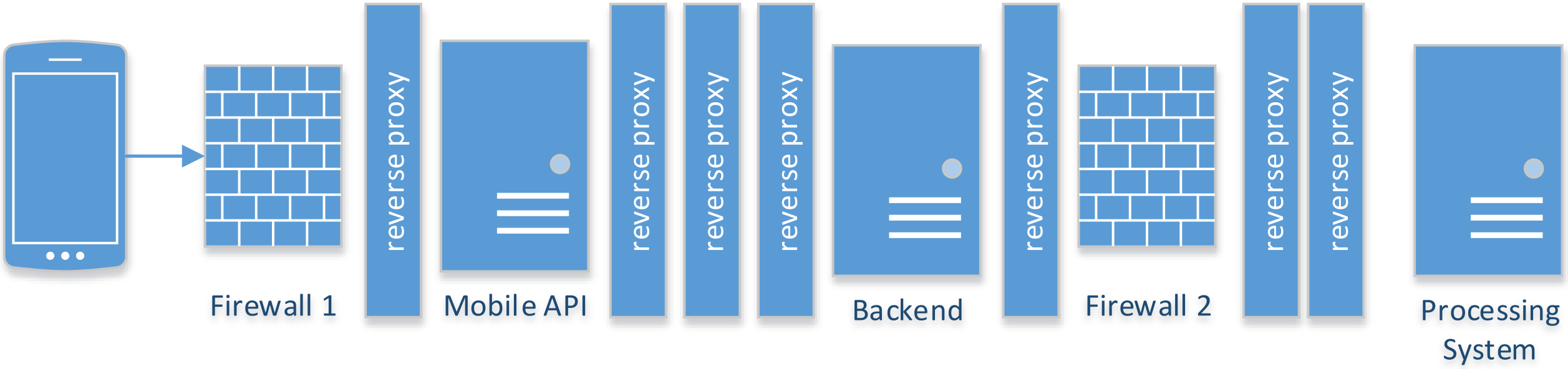
RSA в Java

Решение: *гибридное* шифрование

1. Придумываем новый ключ AES
2. Шифруем данные AES
3. Сам ключ шифруем RSA
4. Отправляем куда надо

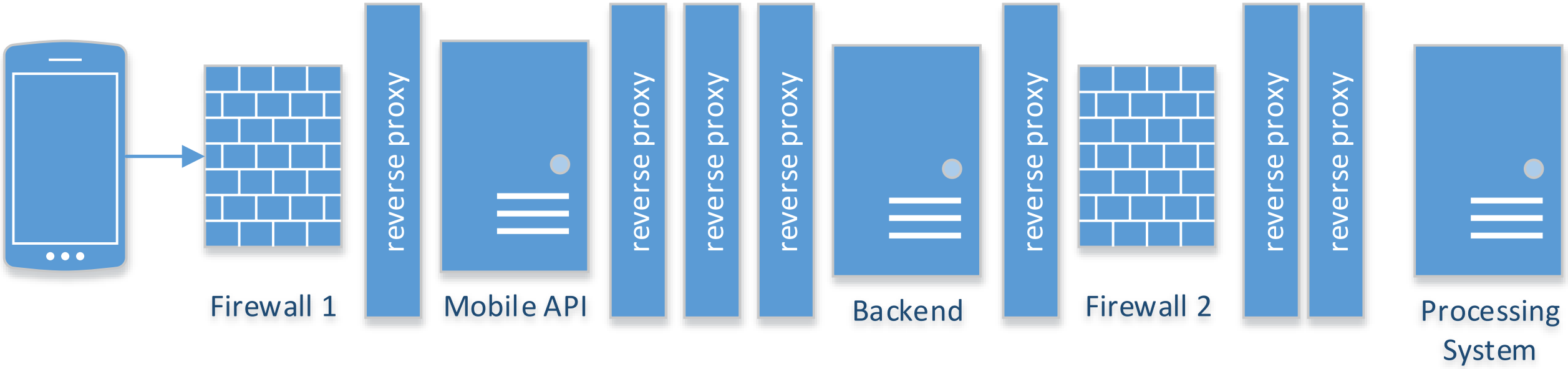
Использование шифрования на открытых ключах

- Передача данных через "полунадёжные" системы



Использование шифрования на открытых ключах

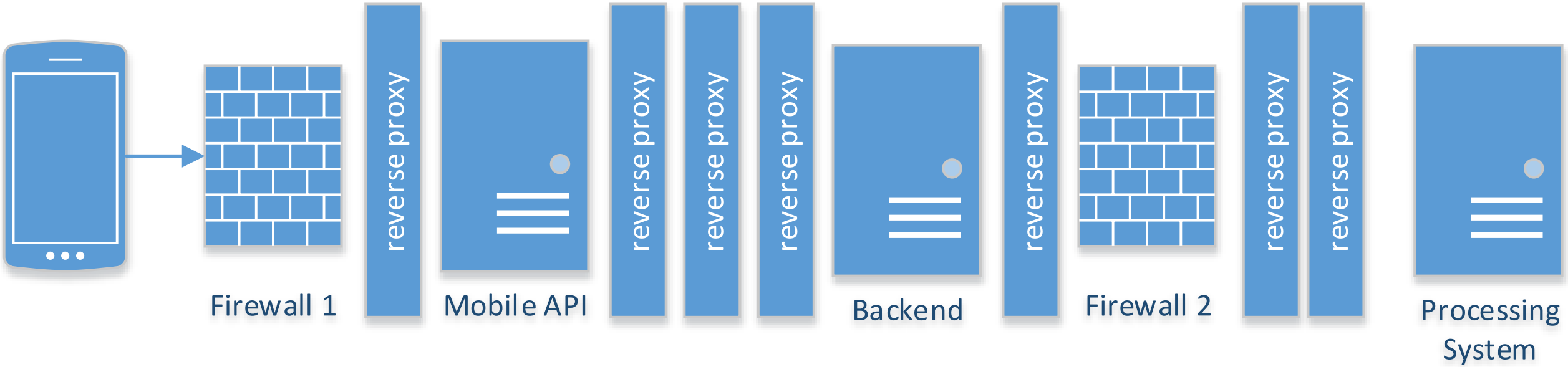
- Передача данных через "полунадёжные" системы



- Схема, конечно, шуточная…

Использование шифрования на открытых ключах

- Передача данных через "полунадёжные" системы



- Схема, конечно, шуточная…
- На самом деле на входе 3 Firewall'а разных типов + ещё один межу Mobile API и Backend'ом

Использование шифрования на открытых ключах

- Передача данных через "полунадёжные" системы

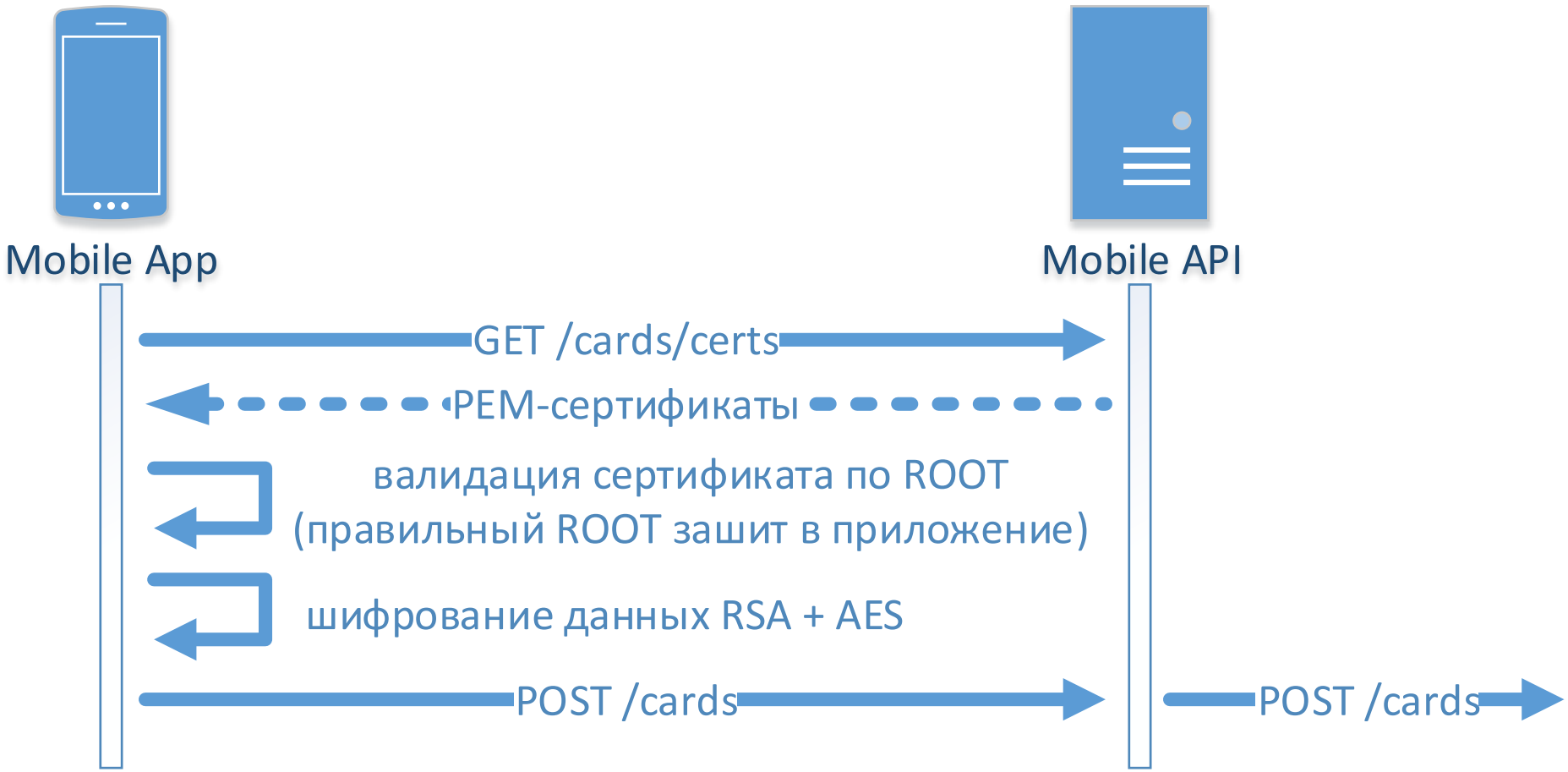
Использование шифрования на открытых ключах

- Передача данных через "полунадёжные" системы

- GET /api/cards/certs
  - получение списка сертификатов (X.509)
  - не те же самые, что и для HTTPS

- POST /api/cards
  - передача данных с шифрованием на открытом ключе сервера

Использование шифрования на открытых ключах



Обратите внимание

- Корневой сертификат – это сертификат ***вашего*** CA
  - имеет срок действия ≈100 лет

- Используйте алгоритмы, которые есть и на Java, и на Swift
  - `AES/CBC/PKCS5Padding`

  - `Java (bc):` `RSA/NONE/OAEPWithSHA256AndMGF1Padding`
  - iOS Swift: `rsaEncryptionOAEPSHA256`

java.security.Signature

Цифровая подпись

Цифровая подпись

- Не путать с "электронная цифровая подпись"
- Не путать с "электронная подпись"

- Строка, зависящая от сообщения и закрытого ключа.
- Легко проверяется без доступа к закрытому ключу.

Цифровая подпись

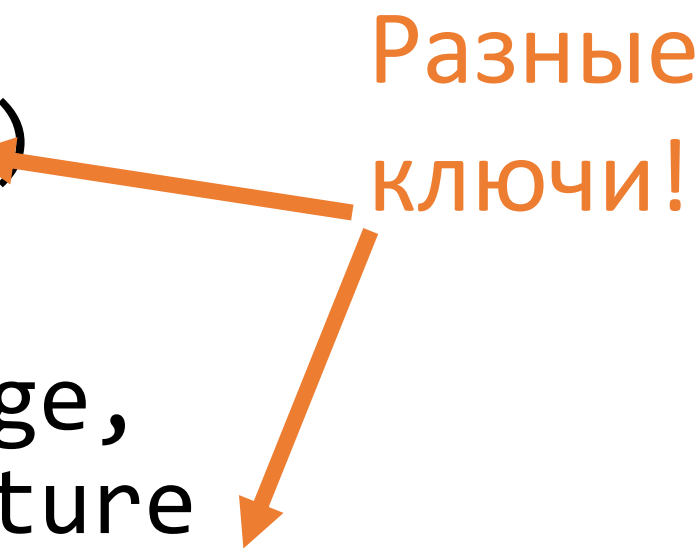
```
byte[] sign( byte[] message,
             PrivateKey key )

boolean verify( byte[] message,
                byte[] signature
                PublicKey publicKey )
```

Цифровая подпись

```
byte[] sign( byte[] message,
             PrivateKey key );

boolean verify( byte[] message,
                byte[] signature
                PublicKey publicKey )
```



Цифровая подпись: генерация ЦП

```
Signature signature = Signature.getInstance( … );
```

Цифровая подпись: генерация ЦП

- NONEwithDSA
- SHA1withDSA
- SHA224withDSA
- SHA256withDSA

- NONEwithRSA
- MD2withRSA
- MD5withRSA
- SHA1withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

Цифровая подпись: генерация ЦП

- ~~NONEwithDSA~~
- ~~SHA1withDSA~~
- SHA224withDSA
- SHA256withDSA

- ~~NONEwithRSA~~
- ~~MD2withRSA~~
- ~~MD5withRSA~~
- ~~SHA1withRSA~~
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

Цифровая подпись: генерация ЦП

- ~~NONEwithDSA~~
- ~~SHA1withDSA~~
- SHA224withDSA
- SHA256withDSA

- ~~NONEwithRSA~~
- ~~MD2withRSA~~
- ~~MD5withRSA~~
- ~~SHA1withRSA~~
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

Цифровая подпись: генерация ЦП

- ~~NONEwithDSA~~
- ~~SHA1withDSA~~
- SHA224withDSA
- SHA256withDSA

- **SHA3-512withECDSA из BC**

- ~~NONEwithRSA~~
- ~~MD2withRSA~~
- ~~MD5withRSA~~
- ~~SHA1withRSA~~
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

Цифровая подпись: генерация ЦП

```
byte[] sign( PrivateKey privateKey, byte[] openText ) {
  var dsa = Signature.getInstance( "SHA256withRSA" );
  dsa.initSign( privateKey );
  dsa.update( openText );
  byte[] signature = dsa.sign();
  return signature;
}
```

Цифровая подпись: валидация

```
boolean verify( PublicKey publicKey,
                byte[] openText, byte[] signature ) {
  var dsa = Signature.getInstance( "SHA256withRSA" );
  dsa.initVerify( publicKey );
  dsa.update( openText );
  return dsa.verify( signature );
}
```

Не надо использовать ЦП для…

- Контроля целостности данных
  одним и тем же приложением
  - в базе данных
  - в файловой системе

- Для контроля целостности ***потока*** данных (гигабиты)

Использование ЦП

- Передача данных в другие *не очень доверенные* системы

- Уменьшение потенциального фронта атаки в схемах аутентификации и авторизации

Использование ЦП: передача данных

- При использовании MessageDigest:

```
https://site2.org/path ? name=Vasya
                       & phone=75551234567
                       & digest=…

digest = SHA256( {name} || {phone} || {salt} )
```

- Система получатель может подделать запрос самой-себе

Использование ЦП: передача данных

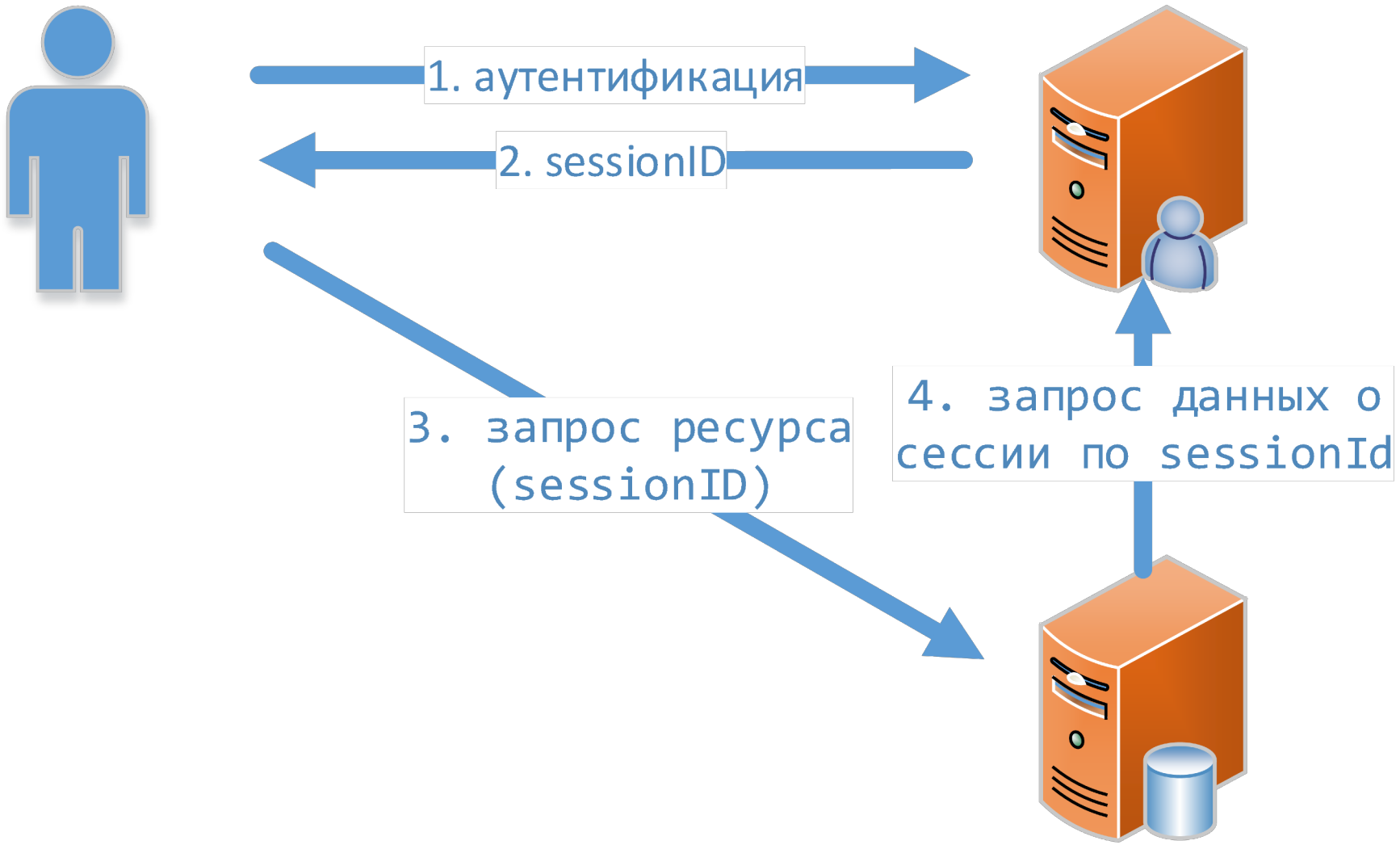
- При использовании Signature:

```
https://site2.org/path ? name=Vasya
                       & phone=75551234567
                       & sign=…

sign = sign( privateKey, {name} || {phone} )

pass = verify ( publicKey,  {name} || {phone} )
```

Использование ЦП: OAuth + JWT



Использование ЦП: OAuth + JWT

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)
  - JSON заголовок (header)
  - JSON полезная нагрузка (payload)
  - byte[] данные для валидации токена

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)

```
base64( headerJson )
    + "."
    + base64( payloadJson )
    + "."
    + base64( validateVata )
```

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWI
iOiIxMjM0NSIsIm5hbWUiOiJKb2huIEdvbGQiLCJhZG1
pbiI6dHJ1ZX0K.LIHjWCBORSWMEibq-
tnT8ue_deUqZx1K0XxCOXZRrBI
```

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWI
iOiIxMjM0NSIsIm5hbWUiOiJKb2huIEdvbGQiLCJhZG1
pbiI6dHJ1ZX0K.LIHjWCBORSWMEibq-
tnT8ue_deUqZx1K0XxCOXZRrBI
```

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)

```
{"alg": "HS512", "typ": "JWT" }

{ "sub": "12345", "name": "John Gold", "admin": true }

LIHjWCBORSWMEibq-tnT8ue_deUqZx1K0XxCOXZRrBI
```

Использование ЦП: OAuth + JWT

- JSON Web Token (RFC 7519)

```
{"alg": "HS512", "typ": "JWT" }
```

Способ валидации токена

```
{ "sub": "12345", "name": "John Gold", "admin": true }
```

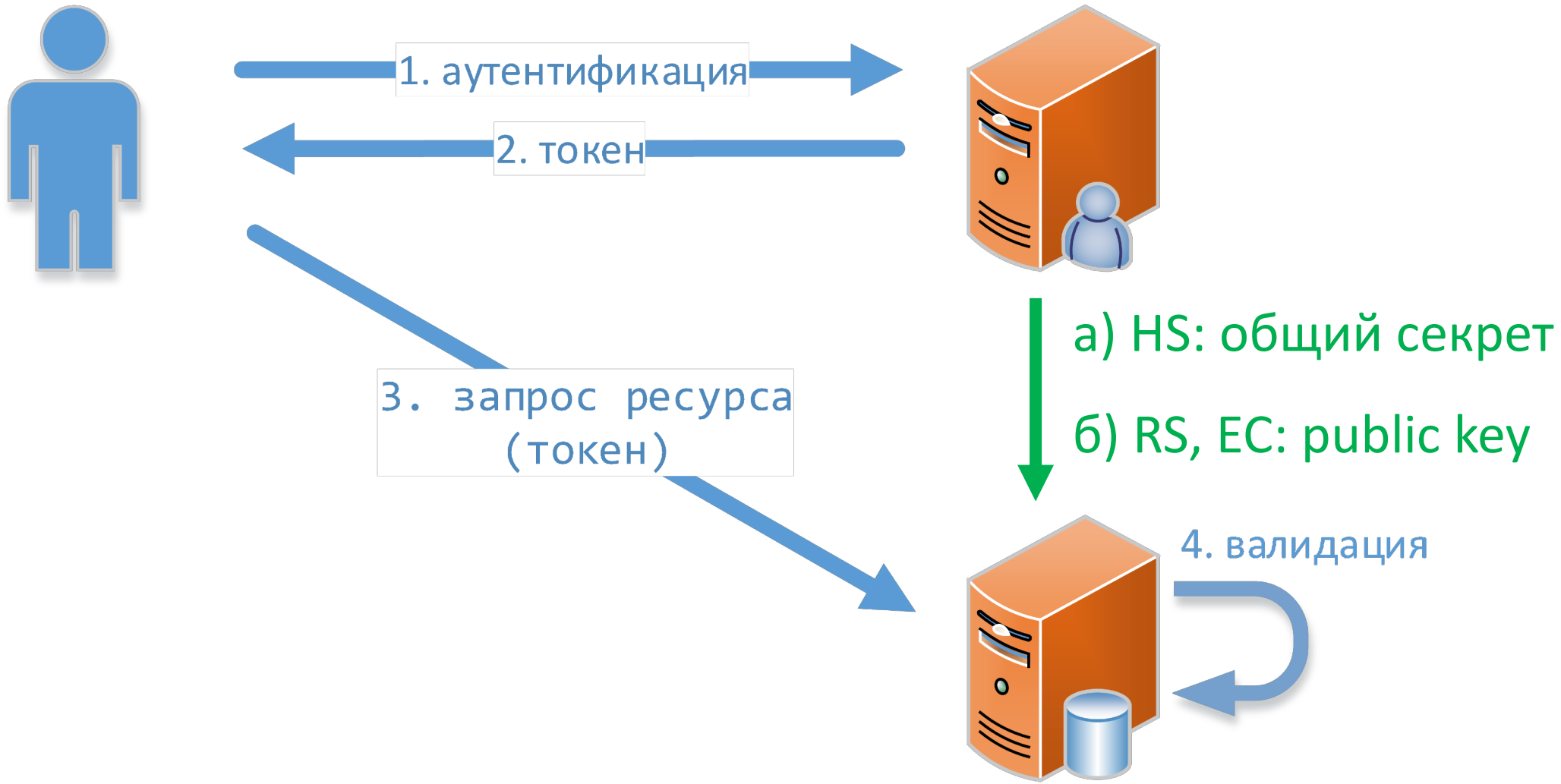
```
LIHjWCBORSWMEibq-tnT8ue_deUqZx1K0XxCOXZRrBI
```

Данные для валидации токена

Использование ЦП: OAuth + JWT

- Валидация JWT – в зависимости от alg
  - HS… : с помощью MessageDigest ("hash")
  - RS… : с помощью ЦП RSA
  - EC… : с помощью ЦП на эллептических кривых

Использование ЦП: OAuth + JWT



Пара-тройка советов…

Пулы объектов Cipher / MessageDigest

pro

- apache commons-pool
- Взять из пула – быстро
- Возможно мониторить

contra

- Хороший пул надо ещё настроить
- Надо не забывать об очистке
  - `Cipher.init()`
  - `MessageDigest.reset()`

Документируйте с примерами

- Примеры должны включать все тестовые ключи, данные и промежуточные результаты

1. «Пусть выбран тестовый ключ… (дамп в PEM-формате)»
2. «Пусть сгенерирован вектор IV … (дамп IV в base64)»
3. «Данные объединены в массив N байт (дамп в base64)»
4. «Получили выход N байт (дамп в base64)»
5. «Итоговый результат: N символов (строка)»

Q & A

вопросы