Rambler&Co 85,36
Компания

20 апреля 2016 в 14:08

AES шифрование и Android клиент

Разработка под Android*, Java*, Блог компании Rambler&Co

Как говорится, ничего не предвещало беды. Мобильный клиент потихоньку пилился, кофе стыл, задачки закрывались одна за другой, пока в внезапно не пришло письмо на корпоративную почту:

Срочно внедряем новый функционал. Все необходимые параметры для построения бизнес модели, в целях безопасности, будут передаваться зашифрованном виде ***AES/CBC/PKCS5Padding*** *с вектором инициализации* ***AAACCCDDDYYUURRS*** *и ключом шифрования* ***ZZHHYYTTUUHH***
Пример зашифрованных данных:

```
p+oJjsGEULNSptP5Sj1BM5w65hMjkqzahORd8ybIkqyJD0V/608c1tYuKIvDLUIa
RQ9jQ6+EwbyMFjlMa6xuEnxOx4sez001hd3NsLO7p00XoTqAvi9zwUBII+
nPphP6Zr0P4icvODpmhlmRILgSBsUf1H/3VN1lNXjo4LTa
```

```
GxLqW3VSg9iV9yFq4VMWqsRF
```

Попытки быстрого поиска решения ~~выдали кучу неработающих примеров~~ показали, что задача выходит за рамки привычной верстки layout'о написания Presenter'ов и требует изучения доков и чтения мануалов. Отличная возможность изучить что-то новое и обогатить свой опыт.

Но для начала, давайте разберемся, что же это такое — шифрование и зачем оно вообще нужно.

Немного теории об AES шифровании

Advanced Encryption Standard (AES) — симметричный алгоритм блочного шифрования, принятый правительством США на основе результа проведенного конкурса в качестве стандарта шифрования и заменивший собой менее надежный алгоритм **Data Encryption Standard (DES)** Утвержденный алгоритм в качестве единого стандарта шифрования стал повсеместно применяться для защиты электронных данных.

Основу алгоритма составляют замены, подстановки и линейные преобразования, каждое из которых выполняется блоками по 128 бит (цифр значениями от 0 или 1), являющихся основой структуры входных и выходных данных, поэтому он и носит называние блочного шифра. Повто операций происходит неоднократно и в процессе каждой итерации (раунда) вычисляется уникальный ключ на основе ключа шифрования и встраивается в дальнейшие вычисления.

Криптографический ключ для алгоритма AES представляет собой последовательность из 128, 192 или 256 бит. Другие параметры входных и выходных данных и криптографического ключа не допускается стандартом AES.

Надежность шифрования обеспечивается тем, что изменение даже одного блока влечет за собой изменение последующих блоков и полное изменение конечных данных на выходе.

Данный подход обеспечивает высокую надежность алгоритма, в которой можно убедиться, рассмотрев следующий несложный пример:

▾ Пример расчета времени на взлом шифротекста

Таблица 1: Зависимость количества комбинаций от длины ключа

Размер ключа	Возможное количество комбинаций
1 бит	2
2 бита	4
4 бита	16
8 бит	256
16 бит	65536
32 бита	4.2 * 10^9
56 бит (DES Алгоритм)	7.2 * 10^16
64 бита	1.8 * 10^19
128 бит (AES алогритм)	3.4 * 10^38
192 бита (AES алогритм)	6.2 * 10^57
256 бит (AES алогритм)	1.1 * 10^77

Самый быстрый суперкомпьютер: 10,51 ПетаФлопс = 10,51 x 10^15 Флопс (операций с плавающей точкой в секунду)

Пусть примерное количество операций в секунду, необходимых для проверки комбинации оптимистично будет: 1000

Количество проверок комбинации в секунду = (10,51 x 10^15) / 1000 = 10,51 x 10^12

Количество секунд в течение одного года = 365 x 24 x 60 x 60 = 31536000

Количество лет, чтобы взломать AES с 128-битным ключом = (3,4 x 10^38) / [(10,51 x 1012) x 31536000] = (0,323 x 10^26) / 31536000 = 1 10^18 = 1 миллиард миллиардов лет.

По материалам: How secure is AES against brute force attacks?

Подробное описание алгоритма на английском языке: ADVANCED ENCRYPTION STANDARD
Также можно почитать эту замечательную статью: Как устроен AES

Вектор инициализации

Initialization vector (IV) — вектор инициализации, представляет собой произвольное число, которое может быть использовано вместе с секретным ключом для шифрования данных.

Использование IV предотвращает повторение шифрования данных, что делает процесс взлома более трудным для хакера с помощью атаки п словарю, в попытках найти шаблоны и сломать шифр. Например, последовательность может появиться два раза и более в теле сообщения. Е повторяются последовательности в зашифрованных данных, злоумышленник может предположить, что соответствующие последовательности

сообщении также были идентичны. IV предотвращает появление соответствующих повторяющихся последовательностей символов в зашифрованном тексте.

Математическая основа

Для ~~вспоминания~~ изучения математической основы, воспользуемся материалом из документации к алгоритму ADVANCED ENCRYPTION STAND также этим хорошим материалом на русском языке: Общее описание криптоалгоритма AES

Соответственно, для описания алгоритма используется конечное поле Галуа GF(2^8), построенное как расширение поля GF(2) = {0,1} по мо неприводимого многочлена m(x) = x^8 + x^4 + x^3 + x + 1. Элементами поля GF(2^8) являются многочлены вида

$$b_7 \cdot x^7 + b_6 \cdot x^6 + b_5 \cdot x^5 + b_4 \cdot x^4 + b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0$$

Операции в поле выполняются по модулю m(x). Всего в поле GF(2^8) насчитывается 2^8 = 256 многочленов.

Основные математические операции в поле GF(2^8)

1. Сложение байт можно выполнить любым из трех способов:
   - представить байты битовыми многочленами и сложить их по обычному правилу суммирования многочленов с последующим приве коэффициентов суммы по модулю 2 (операция XOR над коэффициентами);
   - суммировать по модулю 2 соответствующие биты в байтах;
   - сложить байты в шестнадцатеричной системе исчисления.
2. Умножение байт выполняется с помощью представления их многочленами и перемножения по обычным алгебраическим правилам. Полученное произведение необходимо привести по модулю многочлена m(x) = x^8 + x^4 + x^3 + x + 1 (результат приведения равен остатку от деления произведения на m(x))
3. Для любого ненулевого битового многочлена b(x) в поле GF(2^8) существует многочлен b^-1(x), обратный к нему по умножению, т.е. b(x) · b^-1(x) = 1 mod m(x)

Многочлены с коэффициентами, принадлежащими полю GF(2^8)

Многочлены третьей степени с коэффициентами из конечного поля $a_i \in GF(2^8)$ имеют вид: $a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$ (1)

Таким образом, в этих многочленах в роли коэффициентов при неизвестных задействованы байты вместо бит. Далее эти многочлены будем представлять в форме слова [a_0, a_1, a_2, a_3]. В стандарте AES при умножении многочленов вида (1) используется приведение по модулю другого многочлена x^4 + 1.

Для изучения арифметики рассматриваемых многочленов введем дополнительно многочлен $b(x) = b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0$, где GF(2^8). Тогда

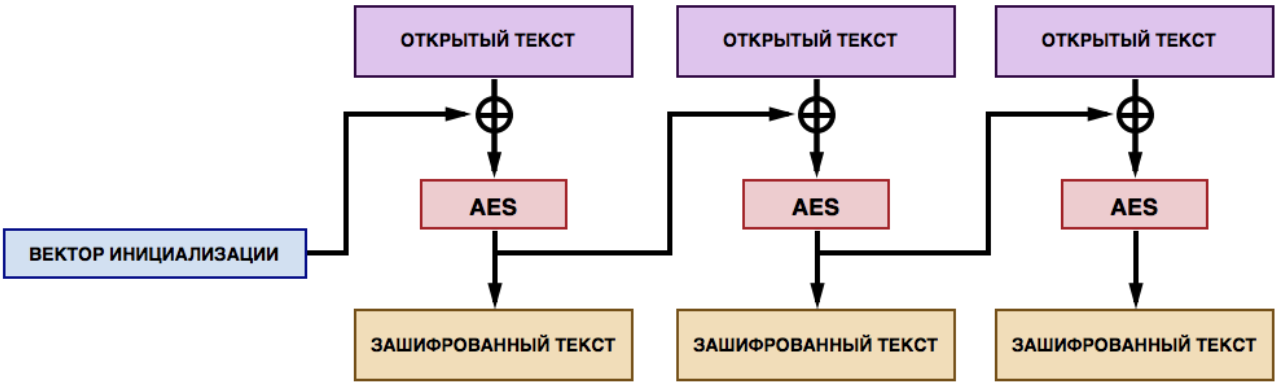
1. Сложение
   $a(x) + b(x) = (a_3 \oplus b_3)\, x^3 + (a_2 \oplus b_2)\, x^2 + (a_1 \oplus b_1)\, x + (a_0 \oplus b_0)$
2. Умножение
   Для представления результата четырехбайтовым словом, берется результат по модулю многочлена степени не более 4. Авторы шифра выбрали для этой цели многочлен x^4+1, для которого справедливо x_i mod(x^4 + 1) ≡ x_i mod 4. Дальнейшее приведение по модулю *x^4+1* позволяет получить результат в виде:

   $$d(x) = a(x) \cdot b(x) = d_3 \cdot x^3 + d_2 \cdot x^2 + d_1 \cdot x + d_0$$

Параметры шифрования

Ну что есть AES и вектор инициализации стало понятно. Теперь попытаемся понять и остальные слова в строке **AES/CBC/PKCS5Padding**

Cipher block chaining (CBC) — режим сцепления блоков шифротекста — один из режимов шифрования для симметричного блочного шифра использованием механизма обратной связи. Каждый блок открытого текста (кроме первого) побитово складывается по модулю 2 с предыдущ результатом. Одна ошибка в бите блока шифротекста влияет на расшифровку всех последующих блоков. Перестройка порядка блоков зашифрованного текста вызывает повреждения результата дешифрования.



Другой параметр **PKCS5Padding**, указывает на то, каким образом должны обрабатываться неполные блоки. При использовании одного из об алгоритмов заполнения, нужно включить размер блока в зашифрованные данные, гарантируя то, что когда вы попытаетесь расшифровать зашифрованное сообщение, вы получите нужное количество байт.

Для работоспособности всех параметров шифрования AES, каждая реализация платформы Java должна поддерживать следующие стандартны алгоритмы шифрования с ключевыми размерами (в скобках):

Standard Cipher transformations

- AES/CBC/NoPadding (128)
- AES/CBC/PKCS5Padding (128)
- AES/ECB/NoPadding (128)
- AES/ECB/PKCS5Padding (128)
- DES/CBC/NoPadding (56)
- DES/CBC/PKCS5Padding (56)
- DES/ECB/NoPadding (56)
- DES/ECB/PKCS5Padding (56)
- DESede/CBC/NoPadding (168)
- DESede/CBC/PKCS5Padding (168)
- DESede/ECB/NoPadding (168)
- DESede/ECB/PKCS5Padding (168)
- RSA/ECB/PKCS1Padding (1024, 2048)
- RSA/ECB/OAEPWithSHA-1AndMGF1Padding (1024, 2048)
- RSA/ECB/OAEPWithSHA-256AndMGF1Padding (1024, 2048)

Источник: Cipher

Ларчик просто открывался

Разобравшись с теорией, можно приступать к реализации самого алгоритма расшифровки серверного сообщения.

В отличии от стандартного набора JDK, для работы нам потребуется *android.util.Base64* для преобразования строки:

```
import android.util.Base64;

import java.security.GeneralSecurityException;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public static String decrypt(String key, String iv, String encrypted)
 throws GeneralSecurityException {

 //Преобразование входных данных в массивы байт

   final byte[] keyBytes = key.getBytes();
   final byte[] ivBytes = iv.getBytes();

   final byte[] encryptedBytes = Base64.decode(encrypted, Base64.DEFAULT);

  //Инициализация и задание параметров расшифровки

   SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
   cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, new IvParameterSpec(ivBytes));

  //Расшифровка

   final byte[] resultBytes = cipher.doFinal(encryptedBytes);
   return new String(resultBytes);
}
```

Стоит также принимать во внимание, что размер вектора инициализации должен быть 16 байт (128 — бит). Это связано с тем, что стандарт шифрования AES включает в себя три типа блочных шифров: AES — 128, AES — 192 и AES — 256. Каждый из этих шифров имеет 128 — бит размер блока, с размерами ключа 128, 192 и 256 бит соответственно и принимая во внимание то, что для всех типов блочного шифра, вектор инициализации такого же размера, как и размер блока шифра мы получаем, что вектор инициализации всегда имеет 128 — битный размер.

В противном случае, даже если мы попытаемся использовать вектор другого размера, то шифротекст не будет расшифрован и мы получим следующее исключение:

```
java.security.InvalidAlgorithmParameterException: Wrong IV length: must be 16 bytes long
```

Результат

Как видно из реализации, решение оказалось достаточно простым и тривиальным в контексте задач подобного рода. Но тем не менее, иногда бывает очень полезно покопаться в доках и реализовать то, что встречается не так уж и часто в трудовых буднях Android разработчика.

Для самых любознательных — под спойлером то, что было зашифровано в сообщении:

▾ ответ к задачке

```
{
   "items": [
      {
         "name": "star",
         "color": "green",
         "id": 21
      },
      {
         "name": "dog",
         "color": "brown",
         "id": 43
      }
   ],
   "lucky_item_id": 43
}
```

aes, android

+6 10,6k 79

Автор: @v555

Rambler&Co рейтинг 85,36

ПОХОЖИЕ ПУБЛИКАЦИИ

26 октября 2015 в 19:21
Конвейерное производство Android приложений
+12 16,4k 137 18

16 сентября 2015 в 12:30
Тестирование на Android: Robolectric + Jenkins + JaCoCo
+20 20k 187 5

1 сентября 2015 в 13:02
Открытый митап Rambler.Android
+11 3,8k 16 9

Комментарии (10)

vikS 20 апреля 2016 в 18:41 #

спасибо за статью, только AES/CBC/PKCS5Padding встречается довольно часто в буднях разработчиков… И не только Android

deinlandel 20 апреля 2016 в 18:58

TL;DR: google android aes, первая ссылка…

Ну серьёзно, тема освещалась неоднократно, в том числе на Хабре, какой-то хитрой специфики в Android относительно обычной Java нет.

v555 20 апреля 2016 в 23:48

Согласен с Вами по поводу обилия материала на данную тему.

Но эта статья имеет цель на легком примере показать, что шифрование это не что-то страшное и непонятное, а очень нужный и полезный инструмент, имек под собой красивую математическую основу.

Ну а Android — живой пример использования и внедрения.

Kolyuchkin 22 апреля 2016 в 08:30

Шифрование действительно не является чем-то страшным и непонятным))) Но эта эйфория проходит, когда Вам нужно применить его для «серьезных» з требующих сертификации у «регуляторов» — вот тут начинаются «танцы с бубном».
А «для поиграться» — это пожалуйста)))

UncleAndy 20 апреля 2016 в 22:57

Вопрос к знатокам… Что лучше с точки зрения безопасности: генерировать вектор инициализации IV случайным образом при каждом шифровании и передават сообщением или использовать его в фиксированном виде?

Я так понимаю, что в описанной задаче и ключ и IV являются константами, зашитыми в код. Это так?

v555 20 апреля 2016 в 23:49

Вопрос безопасности является достаточно непростой темой для обсуждения и не имеет однозначного решения, т.к. устройство у пользователя может быть рутованным, трафик может перехватываться, плюс существую и другие источники опасности для каких либо засекреченных данных. Зашивать ключ просто явно менее надежный способ, чем предложенный Вами.

SannX 21 апреля 2016 в 08:31 +

IV нужен для первого раунда шифрования, т.к. в последующих раундах очередной блок исх текста «суммируется» с результатом шифровки предыдущего бл текста. Поскольку для первого раунда у нас нет результата шифровки предыдущего блока исх. текста, то вместо него используется IV. Поэтому для одного и исх. текста в плане безопасности лучше использовать случайный IV. А если у вас исх. тексты всегда разные, то IV менять каждый раз не обязательно. Но см ли вы гарантировать, что исх текст у вас больше не повториться при шифровке? Поэтмоу лучше всегда использовать случайный IV. И его необязательно скр

agee 21 апреля 2016 в 15:43 +

> и ключ и IV являются константами

Нет, не верно. В CBC-режиме **случайный** IV генерируется перед шифрованием первого блока исходного сообщения, а затем в явном виде сохраняется/пер вместе с шифротекстом, в частности путем конкатенации IV || C, где C — шифротекст. При дешифровании всего шифротекста IV считывается (так как его дл фиксирована) и используется для дешифрования первого блока, т.е.:
если

```
c[0] = E(k, IV⊕m[0])
```

, то

```
m[0] = D(k, c[0])⊕IV
```

, где E и D — ф-ции шифрования и дешифрования соответственно, `m[0]` — первый блок открытого текста, `c[0]` — первый блок шифротекста.

Доказано, что использование неслучайного IV делает блочный шифр неустойчивым к атаке с подобранным открытым текстом (chosen plaintext attack). Более неустойчивым к этой атаке считается и шифр, IV которого можно предугадать.

shuron 21 апреля 2016 в 22:51

Да IV должен быть случайным особенно в связке с CBC и не повторятся…
IV не секретная информация он передается не шифрованым

SannX 21 апреля 2016 в 08:31 *(комментарий был изменён)*

del

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.