Модуль 5. Основы информационной безопасности

Тема 5.3. Дайджест сообщения и алгоритм MD5

2 часа

Оглавление

5.3. Дайджест сообщения и алгоритм MD5

5.3.1 Дайджест сообщения

В модуле 3 в разделе 3.6 были рассмотрены функции хеширования и их применение для хранения данных в виде хеш-таблиц с целью более эффективного поиска, операций добавления и удаления элементов. Однако хеш-функции могут применяться и в сфере информационной безопасности для защиты *целостности* сообщений, *аутентификации* отправителя и гарантии *неаннулируемости*.

Итак, *функция хеширования h(x)* представляет собой преобразование исходных данных *x*, причем вне зависимости от объема входной информации результат – дайджест – *y=h(x)* будет представлять собой строку фиксированной длины l=128 или 160 бит. Таким образом, хеш-функции позволяют построить короткий дайджест обрабатываемой ими информации, своего рода, отпечаток или краткое содержание. Если данные были изменены в процессе передачи или хранения, устаревший дайджест уже будет недействительным. Следовательно, целостность данных может проверяться периодическим вычислением дайджеста и сравнением с ранее зафиксированными значениями, даже если сами данные хранятся в небезопасном месте. Очевидно, что при легальном (санкционированном) изменении информации соответствующий дайджест должен обновляться. Поэтому алгоритмы вычисления дайджеста применяются для вычисления *электронной цифровой подписи (ЭЦП),* о которой речь пойдет в разделе 5.4.

Следует отметить, что *множество исходных текстов (сообщений) X* может быть конечным или неограниченным, но *множество значений хеш-функции Y* всегда конечно, поскольку дайджесты имеют ограниченную длину. В том случае, если X – конечно, то *h(x)* называют функцией сжатия (*сжимающим отображением* $h(x)$: $X \to Y$). Тогда выполняется неравенство $|X| \geq |Y|$.

Для того, чтобы использовать хеш-функции с целью защиты информации, следует выбирать такие алгоритмы, которые обладают следующими свойствами:

1. Функция *h* может быть вычислена для входных данных любого размера.
2. Результат вычисления *y=h(x)* должен иметь фиксированный размер l.
3. Вне зависимости от исходных данных результат хеширования *y* должен быть вычислительно неотличим от строки битов, распределенных случайным образом. Иными словами, вероятность появления 0 в строке результата должна быть примерно равна вероятности появления 1.
4. Вычисление функции для любого аргумента *x* должно быть достаточно простым и быстрым для применения в приложениях реального времени.
5. Для хеш-функции $h(x)$: $X \to Y$ и любого элемента $x \in X$, задача поиска *прообраза* x (исходного сообщения, такого что $y = h(x)$) должна быть вычислительно невыполнимой.

Если для известного y поиск прообраза не может быть выполнен эффективно, то такую функцию называют *однонаправленной (односторонней).*

6. Для хеш-функции $h(x)\colon X \to Y$ и любого элемента $x \in X$, задача поиска *второго прообраза* x'(такого сообщения, что $h(x) = h(x')$ причем $x \neq x'$) должна быть вычислительно невыполнимой.

7. Для хеш-функции $h(x)\colon X \to Y$, задача поиска *коллизии* должна быть вычислительно невыполнимой. То есть поиск двух значений исходных аргументов $x, x' \in X$, причем $x \neq x'$ и $h(x) = h(x')$, должен быть вычислительно невыполним.

Свойства 6 и 7 гарантируют, что злоумышленник не сможет найти такое сообщение, которое даст тот же результат хеширования, что и исходное.

Особую роль играют ключевые хеш-функции $h_K(x)$, с помощью которых вычисляются *коды аутентификации сообщений (Message authentication code, MAC)*. Допустим, Элис и Боб владеют общим секретным ключом *K,* который определяет хеш-функцию h_K. Элис хочет отправить сообщение x Бобу, но не уверена в безопасности канала передачи данных. Тогда она вычисляет $y = h_K(x)$ и отправляет Бобу пару (x, y). Когда Боб получает эту пару (x, y), он может удостовериться, что $y = h_K(x)$. Если равенство выполняется, то он уверен, что ни само сообщение, ни y не были изменены в процессе передачи (если функция хеширования безопасна). Кроме того, он может быть уверен, что сообщение послано именно Элис, следовательно, выполнена *аутентификация отправителя*.

Кроме того, в криптосистемах с открытым ключом функции хеширования используются для верификации правильности зашифрованной информации. Также они применяются в качестве компонентов генераторов псевдослучайных чисел, векторов инициализации. При распространении файлов программного обеспечения владелец может снабжать их контрольной суммой, вычисленной в соответствии с открытым алгоритмом хеширования. Затем пользователь, получив информацию, повторяет вычисления. Если контрольные суммы не совпадают, то можно говорить о несанкционированных изменениях в исходных файлах или ошибках в процессе передачи по каналу связи.

5.3.2 Алгоритм MD5

Алгоритм MD5 является улучшенной версией хеш-функции MD4. Он более сложен, но использует тот же набор операций и вычисляет 128-битовый дайджест, который обычно представлен числом, содержащим 32 шестнаднадцатеричные цифры. Алгоритм был опубликован Рональдом Ривестом в 1992 году. В настоящее время его альтернативой являются бесключевые функции хеширования семейства SHA-2.

MD5 используется, в том числе для хранения паролей в операционной системе. При регистрации нового пользователя вычисляется хеш-функция от значения его пароля для входа в систему. Именно значение дайджеста записывается в системный файл, а не сам пароль. При повторном входе в систему пользователь вводит свой пароль, вычисляется результат хеширования этого значения, и, если он совпадает с ранее зарегистрированным значением, то вход в систему

предоставляется. В противном случае, пользователь не получает к ней доступ. Таким бразом, даже если злоумышленник сможет прочитать файл паролей, он узнает лишь их дайджесты, но не сможет восстановить исходные последовательности и их использовать.

Следует также отметить, что MD5 не используется как хеш-функция в классе String вследствие более значительных затрат на вычисления, требующих дополнительного времени для получения результата.

Итак, на вход алгоритма подается сообщение x длиной $l > 0$ бит. MD5 не использует ключей. Для вычисления дайджеста необходимо выполнить следующие преобразования.

1. Добавление битов заполнения

Сообщение будет обрабатываться блоками, длина которых равна 512 бит. Причем последний блок дополняется до длины, сравнимой с 448 по модулю 512, поскольку последние 64 бита используются для записи размера сообщения. Расширение выполняется даже в том случае, если длина исходного сообщения уже сравнима с 448 по модулю 512. В соответствии с алгоритмом в конец последнего блока сначала дописывается бит, равный 1, затем - нулевые биты (их количество может варьироваться от 0 до 512). Затем записывается битовое представление длины сообщения – в блоке из 64 бит как два 32-разрядных слова, при этом младшее слово дописывается первым. Если длина исходного сообщения больше, либо равна $2^{64} - 1$ бит, используются только младшие 64 бита представления (эквивалентно взятию остатка от деления на 2^{64}). Таким образом, сообщение можно представить в виде:

$$x' = \underbrace{x \| 1 \| 0 \dots 0 \| l}_{\textbf{448 mod 512 бит}}$$

Тогда длина сообщения x' будет кратна 512 битам или кратна 16 32-разрядным словам. Обозначим $M_0, M_1, \dots, M_N$ слова дополненного сообщения

2. Инициализация буфера

Буфер из четырех слов *(A, B, C, D)* используется для последовательного вычисления дайджеста сообщения. В этих переменных будут храниться результаты промежуточных вычислений. В начале каждое слово инициализируется значением:

A=0x01234567

B=0x89abcdef

C=0xfedcba98

D=0x76543210

Начальное состояние *ABCD* называется *инициализирующим вектором,* который стандартен. Кроме того, задается набор констант *K* – 64 слова.

3. Обработка сообщения блоками по 16 (32-разрядных) слов

Каждый блок (512 бит) обрабатывается в течение 4 раундов, которые, в свою очередь, заключаются в выполнении стандартных операций 16 раз. Для каждого раунда определяется свой тип операции, включающей вычисление нелинейной функции от трех аргументов (*A, B, C* или *D*). Эти функции основаны на выполнении поразрядных операций И, ИЛИ, НЕ, XOR. Выходное значение функции (32-битовое слово) складывается с четвертым значением буфера (не участвовавшим в вычислении функции), со словом сообщения M_i и с заданной константой K_i. Затем производится циклический сдвиг результата влево на число бит *s*, определяемое номером раунда и номером выполняемой операции, и, наконец, сложение со значением буфера *A, B, C* или *D*. В итоге, происходит замещение результатом соответствующего (предыдущего) значения буфера.

Схема выполнения одной из операций представлена на рисунке ниже[1] (всего выполняется 64 операции, по 16 в каждом из четырех раундов). *F* - нелинейная функция, определяемая для каждого раунда, сложение выполняется по модулю 2^{32}.

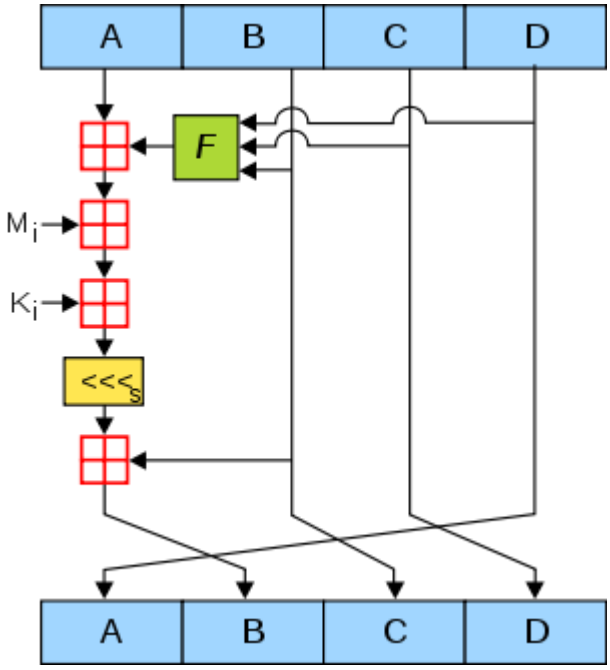

Рисунок. Схема одной из операций MD5

Таким образом, дайджест блока содержится в буфере *ABCD*. Затем преобразования совершаются над следующим блоком и т.д. Следует отметить, что вывод результата начинается с младшего байта слова *A* и заканчивается выводом старшего байта слова *D.*

[1] MD5 [Электронный ресурс] : Материал из Википедии — свободной энциклопедии / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2015. — Режим доступа: http://ru.wikipedia.org/?oldid=69890820

5.3.3 Безопасность MD5

В настоящее время известно несколько видов взлома хешей, вычисленных по MD5: перебор по словарю, полный перебор, поиск коллизий. Информация об уязвимостях алгоритма публиковалась с 1993 года, поэтому в конце 2008 US-CERT (Компьютерная команда экстренной готовности США) рекомендовал разработчикам программного обеспечения, владельцам сайтов и пользователям не использовать MD5 в любых целях, так как исследования продемонстрировали его ненадежность.

Упражнение 5.3.1

Алгоритм хеширования MD5 в Java можно реализовать при помощи одноименного класса MessageDigest из пакета java.security. Рассмотрим простой пример Android-приложения, позволяющего реализовать алгоритм MD-5. Приложение будет состоять из одной Activity, на которой будет располагаться редактируемое текстовое поле типа EditText для ввода пароля, ниже кнопка типа Button для реализации механизма преобразования пароля в хеш-сумму по алгоритму MD-5, а за ней текстовое поле типа TextView для вывода результата работы.

Следуя вышеописанному составим разметку для нашего layout с названием acitivity_main. Она будет выглядеть следующим образом:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:padding="16dp" >

      <EditText
            android:id="@+id/editTextPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/et_password_hint"
            android:maxLines="1"
             android:singleLine="true" />

      <Button
            android:id="@+id/buttonPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/btn_add_password"
            android:onClick="addPassword"/>

      <TextView
            android:id="@+id/textViewResult"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="16sp" />

</LinearLayout>
```

Единственная примечательная часть в этой разметке, это указание для EditText, что бы он не позволял писать в несколько строк, последние две строчки в его описании за это и отвечают. Конечно же, без представления файла со строковыми ресурсами, не совсем понятно, какие названия отражаются в элементах нашего интерфейса. Продемонстрируем и это:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

      <!-- Application -->
      <string name="app_name">MD5 Hashing</string>

      <!-- ActivityMain -->
      <string name="et_password_hint">Введите пароль</string>
      <string name="btn_add_password">Добавить пароль</string>

</resources>
```

Далее самое интересное – рассмотрение кода логики приложения. Она достаточно простая, состоит из двух самописных методов и реализации щелчка кнопки. Ниже приведён конечный вариант класса MainActivity:

```
package ru.itschool.md5hashing;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashSet;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

      HashSet<String> md5_of_passwords = new HashSet<String>();

      @Override
      protected void onCreate(Bundle savedInstanceState) {
                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.activity_main);
      }

      public void addPassword(View v) {
                  try {
                        md5Hashing();
                  } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                  } catch (NoSuchAlgorithmException e) {
                        e.printStackTrace();
                  }
      }
```

```
	private String convertToHex(byte[] thedigest) {
			StringBuilder result = new StringBuilder();
			for (int i = 0; i < thedigest.length; ++i) {
				result.append(Integer.toHexString(thedigest[i]));
			}
			return result.toString();
	}

	private void md5Hashing() throws UnsupportedEncodingException,
			 NoSuchAlgorithmException{
			String input = ((EditText) findViewById(R.id.editTextPassword))
				.getText().toString();
			byte[] bytesOfMessage = input.getBytes("UTF-8");
			MessageDigest md = MessageDigest.getInstance("MD5");
			byte[] thedigest = md.digest(bytesOfMessage);
			md5_of_passwords.add(convertToHex(thedigest));
		((TextView)findViewById(R.id.textViewResult))
				.setText(md5_of_passwords.toString());
	}
}
```

Разберём построчно представленный листинг. В самой первой строчке указывается пакет, в котором находится класс MainAcitivity. Затем описаны импортируемые библиотеки, сразу после которых начинается реализация класса MainAcitivty унаследованного от базового класса Acitivity. В его определении объявляется и инициализируется множество HashSet с названием «md5_of_passwords» и дженериком String. Далее перегружен метод onCreate(), сгенерированный средой разработки Eclipse.

Самая важная часть приложения, где собственно и представлен метод хеширования по алгоритму MD-5, расположена далее и представлена тремя методами:

- addPassword() – реализация механизма нажатия на кнопку, вызывающая метод md5Hashing() и обрабатывающая возникающие исключения;
- md5Hashing() – метод получения хеш-суммы по алгоритму MD-5;
- convertToHex(byte[] thedigest) – метод преобразования входящего через параметры дайджеста в шестнадцатеричное строковое представление.

Рассмотрим каждый из методов по отдельности.

С того момента, как пользователь нажмёт на кнопку с названием «Добавить пароль», запустится на выполнение метод addPassword(), который в свою очередь вызывает метод md5Hashing(). Так как md5Hashing() может породить два исключения, соответственно в addPassword() они обрабатываются при помощи конструкции try-catch, с указанием вывода в журнал сообщений (Log) трассировки стэка.

Когда вызывается метод md5Hashing(), первое что в нём выполняется – получение значения редактируемого текстового поля с идентификатором «editTextPassword» и его запись в строковую переменную «input». Затем, она преобразуется в массив байтов именуемый «bytesOfMessage» с указанием кодировки «UTF-8». Далее создаётся объект (дайджест сообщение), основанный на алгоритме MD-5. После этого формируется еще один массив байтов с названием «thedigest», отличающийся от первого тем, что он базируется на хеш-значениях. Следующим шагом идёт

заполнение множества HashSet при помощи вызова метода convertToHex(). В методе преобразования создаётся объект StringBuilder и при помощи цикла заполняется элементами массива байтов «thedigest» с представлением каждого в шестнадцатеричном виде. Результатом работы метода является возвращаемая хеш-сумма, полученная при помощи алгоритма MD-5. Всё что остаётся – вывести данные значения на экран, что и делает последняя строчка метода md5Hashing().

Задание 5.3.1

Доработать приложение, чтобы оно выводило MD5 последнего введенного пароля. Изучить насколько сильно изменяется MD5 при изменении одного символа пароля.

Задание 5.3.2 (сквозное задание 9)

В приложении, разработанном в теме 3.7 (задание 8), создать дополнительно пользователя, у которого будут ограниченные права, например доступен только калькулятор. Зашифровать пароли пользователей по MD5.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателям IT ШКОЛЫ SAMSUNG Пономарчук Юлии Викторовне и Канаеву Константину Александровичу.