Саратовский Государственный Университет им. Н. Г. Чернышевского

**Курсовая работа**

**«Хеш-функции в криптосистемах»**

Выполнил: студент 112гр. КниИТ

Иванченко Е. С.

*г. Саратов 2001*

**Содержание**

Введение 3

Метод хэширования 3

Коллизии и реверс 3

Односторонние хэши 4

Список литературы и сайтов последняя

**Введение**

В наше время большую роль в информатике играют сетевые технологии, базирующиеся на объединении огромного числа машин в единую сеть . Одним из ярких примеров такой сети является Internet. Она основана на многопользовательских операционных системах, позволяющих управлять данными, хранящимися на удалённых машинах (серверах) сразу нескольким людям. Иногда требуется сделать доступной для всех только часть документов. Например, зачастую требуется скрыть програмный код cgi-скрипта от посторонних глаз, но весьма нежелательно запрещать его исполнение. Для этого операционной системе необходимо “объяснить”, кто является владельцем. В большинстве операционных систем идентификация производится по логину и паролю. Но так как с файлом, в котором содержится этот пароль, работают не один, а несколько пользователей, то хранение его в открытом виде представляет угрозу сохранности документов. Для этого потребовалось шифрование данных.

**Метод хэширования**

Одним из наиболее распространённых методов криптования является хэширование. Метод хеширования позволяет хранить элементы из множества **A** в линейном массиве **X**. Математически это можно записать так:

**h: A  {0, x-1}**

т. е. Функция **h** отображает каждый элемент множества **A** в индекс множества **X**.

Например: пусть даны два множества **A {‘a’, ’b’, ’c’, …}** и **X {0, 1, 2, …}**, тогда функция **h:AX** ставит в соответствие каждому элементу из множества **A** элемент из множества **B**. Таким образом **h(‘a’)=0**, **h(‘c’)=2** и т. д.

**Коллизии и реверс**

Однако, возможно существование такого интервала на области определения функции, в границах которого она становится инъективной (т. е. если **h(‘a’)=0,** то существует такая функция, **g: XA**, для которой **g(0)=’a’**). Это означает, что только для одного элемента из множества **A** существует индекс **x1.** Функция будет инъективна и в том случае, если ни один элемент из **A** не отображается на интервал **(x1, x2)** при условии, что последний не равен нулю. В любом другом случае на каждый индекс множества **X** отображается более одного элемента из **A**. Это так называемая коллизия хэш-функции.

Реверс хэш-функции заключается в поиске всех отображаемых на данный индекс элементов. Для любого конечного множества это разрешимая задача, которая имеет наиболее простое решение на инъективных интервалах хэш-множества.

**Односторонние хэши**

В криптовании используются особые хэш-функции, называемые односторонними. Функция **: XY** называется односторонней, если **(x)** может быть легко вычислена для любого элемента из множества **X,** тогда для всех элементов из множества **Y** вычисление такого аргумента **x,** для которого **(x)=y**, не разрешимо полиномиально. Системы, построенные на односторонних функциях взлому, как правило, не поддаются.

**Основные аспекты написания**

При написанием алгоритма **kript** особое внимание уделялось следующим аспектам:

требования пользователя к алгоритму;

возможные варианты утечки зашифрованного пароля;

наиболее действенные методы расшифровки.

**1. Требования пользователя**

Основные требования к алгоритму с точки зрения пользователя являются:

надёжность;

скорость работы;

системные требования (необходимые ресурсы).

**2. Варианты утечки пароля**

Одной из главной причиной утечки пароля при использовании этого алгоритма служит его хранение в открытом виде самим владельцем, поэтому большая часть атак в наше время рассчитана на доверие пользователя (например, по телефону звонит мнимый администратор сети и просит пароль для проведения профилактических работ). В этом случае защита сводится к идентификации не только пользователя, но и машины, с которой производится запрос.

Второй причиной служит его расшифровка.

**3. Методы расшифровки**

Этот метод связан с использованием большинством пользователей слишком простых паролей (длиной менее 8 символов, или, пароль, несущий на сбе какую-то смысловую нагрузку (отчество прабабушки по маминой линии)). В этом случае атаки сводятся к перебору возможных паролей, а защита - к их усложнению.

Для расшифровки пароля вторым методом, требуется знать его длину и алгоритм шифования. В случае, когда длина пароля составит менее восьми символов, можно воспользоваться следующим алгоритмом:

1. Перевернуть зашифрованный пароль.

2. Так как размер блока не может быть более 5 байт и менее 1 байта, то разобьём его на 8 блоков и запишем в список (список первых блоков, список вторых, и т. д.). Получим восьмиподсписковый список списков, каждый подсписок которого представляет собой все возможные блоки шифрованных символов.

3. Пробегаем в цикле по подсписку, сверяя каждый элемент со всеми символами из ASCII следующим образом:

If j\*generate(x,n,j) = <элемент подсписка> then write(ord(j)), где j десятичный код символа, x - ключ, n - последовательный номер символа в пароле (в диапазоне [1, 8]). Если выполнилось это условие, то выведем на экран найденный символ.

После выполнения алгоритма на выходе получим либо пароль, либо такую последовательность, из которой его можно получить.

**Описание**

В основе алгортма лежит функция от трёх аргументов **generate=trunc(k\*(abs(sin(ln(a)\*x)+ sin(cos(b)\*x))))**:

1. ключа (**x**);

2. десятичного код символа (**a**);

3. номера символа во введённой строке (**b**).

Она используется для преобразования десятичного кода символа в число, лежащее в интервале от **0** до **2\*k**, где **k** - любое число целого типа. Чем больше число k - тем меньше вероятность коллизий в дальнейшем.

После обработки символа он добавляется в список списков процедурой **add\_in\_list(x: integer; s: string; var gr: llist)** следующим образом - **l^.inf:=ord(s[k])\*generate(x,ord(s[k]),k)**, где **l^.inf**-элемент списка списков, **x** - ключ (для функции **generate**), **s** - строка, разбиваемая на блоки по 8 символов. Каждый подсписок имеет длину не более 8 элементов размером до 5 байт.

Третим шагом является сложение соответствующих элементов процедурой **summ\_all(gr: llist; var a:array\_type)** из каждого подсписка **l** в 8 элментный массив **a**, т.е. первый элемент из первого элемента складывается с первым элементом второго, третьего и т.д. подсписка и записывается в **a[1]**.

Так - же поступаем и с другими элементами подсписков.

Следующим щагом записываем в файл ключ и по очереди все элементы массива **a,** обработанные функцией FromIntToString(), которая переводит численный тип в символьный и переворачивает.

Для сверки пароля его требуется зашифровать заново по известному ключу и сверить с зашифрованным экземпляром.

Вот исходный текст программы:

**kriptmod.pas**

unit kriptmod;

interface

type Plist=^list;

list=record

inf: word;

num: 1..8;

next: Plist;

end;

Llist=^List\_of\_list;

List\_of\_list=record

nb: Plist;

inf: 1..32;

next: Llist;

end;

array\_type=array[1..8] of longint;

function generate(x: integer; a, b: byte):integer;

procedure add\_in\_llist(x: integer; s: string; var gr: llist);

procedure print\_llist(gr: llist);

procedure summ\_all(gr: llist; var a:array\_type);

function FromIntToString(L: longint):string;

implementation

{--Эта функция переводит из целочисленного типа в символьный----------------------------------------------}

function FromIntToString;

var s: string;

l1: longint;

begin

l1:=l; s:='';

while (l1 div 10>0) do

begin

case l1 mod 10 of

0: s:=s+'0';

1: s:=s+'1';

2: s:=s+'2';

3: s:=s+'3';

4: s:=s+'4';

5: s:=s+'5';

6: s:=s+'6';

7: s:=s+'7';

8: s:=s+'8';

9: s:=s+'9';

end;

l1:=l1 div 10;

end;

case l1 mod 10 of

0: s:=s+'0';

1: s:=s+'1';

2: s:=s+'2';

3: s:=s+'3';

4: s:=s+'4';

5: s:=s+'5';

6: s:=s+'6';

7: s:=s+'7';

8: s:=s+'8';

9: s:=s+'9';

end;

FromIntToString:=s;

end;

{--Функция генерации (основная)----------------------------------------------}

function generate;

begin

generate:=trunc(abs(122.5\*(sin(ln(a)\*x)+sin(cos(b)\*x))));

end;

{--Процедура добавления в список списков----------------------------------------------}

procedure add\_in\_llist;

var g: llist;

l: plist;

k, i, j: byte;

begin

k:=1; i:=1;

while (k<=length(s)) do

begin

new(g);

g^.inf:=i;

g^.nb:=nil;

j:=1;

while (j<=8) and (k<=length(s)) do

begin

new(l);

l^.inf:=ord(s[k])\*generate(x,ord(s[k]),k);

l^.num:=j;

l^.next:=g^.nb;

g^.nb:=l;

k:=k+1;

j:=j+1

end;

g^.next:=gr;

gr:=g;

i:=i+1

end;

end;

{--Процедура заполнения массива a-----------------------------------}

procedure summ\_all;

var g: llist;

l: plist;

i: 1..8;

begin

g:=gr;

while g<>nil do

begin

l:=g^.nb;

i:=1;

while l<>nil do

begin

a[i]:=a[i]+l^.inf;

l:=l^.next;

i:=i+1

end;

g:=g^.next;

end;

end;

{------------------------------------------------}

end.

**kript.pas:**

program kuzik;

uses crt, kriptmod;

var x: integer;

i: 1..8;

pass: string;

l: Llist;

arr: array\_type;

f: text;

begin

clrscr;

randomize;

{--Генерируем число--}

x:=abs(random(9999-101))+101;

write('Password: '); textcolor(0);readln(pass);

add\_in\_llist(x,pass,l);

summ\_all(l,arr);

assign(f, 'shadow');

rewrite(f);

writeln(f,x);

for i:=1 to 8 do write(f,FromIntToString(arr[i]));

writeln(f);

close(f);

textcolor(2);

writeln('User added in base.');

repeat until keypressed;

end.

**unkript.pas:**

program kuzik;

uses crt, kriptmod;

var x: integer;

i: 1..8;

pass, pass1: string;

l: Llist;

arr: array\_type;

f: text;

s, s1: string;

begin

clrscr;

write('Password: '); textcolor(0);readln(pass);

{--Открываем файл с паролями--}

assign(f,'shadow');

reset(f);

{--Читаем ключ--}

readln(f,x);

{--Читаем зашифрованный пароль--}

readln(f,pass1);

close(f);

{--Шифруем только что введённый пароль--}

add\_in\_llist(x,pass,l);

summ\_all(l,arr);

for i:=1 to 8 do s1:=s1+FromIntToString(arr[i]);

{--Сверяем его с паролем из shadow--}

if (pass1=s1)

then begin

textcolor(2);

writeln('Password correct.')

end

else begin

textcolor(4);

writeln('Password incorrect!');

end;

repeat until keypressed;

end.