DES

Кевролетин В.В.

25 декабря 2011 г.

Задание5.1

Условие

Продемонстрировать лавинный эффект в DES: написать программу (Mathematica, Scheme, Sage), которая вычисляет расстояние Хемминга для изменений в тексте и в ключе. Сгенерировать сообщение и ключ, а затем, последовательно изменяя в сообщении по одному биту, рассчитать расстояние Хемминга с исходным при неизменном ключе. Вычислить также среднее расстояние по всем вариантам. Аналогичные действия проделать для фиксированного сообщения, изменяя ключ. Программу предварительно протестировать

Решение

Реализация шифратора/дешифратора находится в самом конце документа.
Ниже проверка алгоритма на тесте со страницы курса (http://people.csail.mit.edu/rivest/Destest.txt)
:

```
my $x_0 = BitsArray::from_hex(lc('9474B8E8C73BCA7D'));
for my $i (0 .. 15) {
    my $encoder = Des->new(key => $x_0);
    unless ($i % 2) {
        $x_0 = $encoder->encode_block($x_0);
    } else {
        $x_0 = $encoder->decode_block($x_0);
    }
}
print BitsArray::to_hex($x_0); # 1B1A2DDB4C642438 == 1b1a2ddb4c642438
```

Ниже демонстрация лавинного эффекта DES. Ключ и данные генерируются случайным образом, количество 0 и 1 примерно равно. На каждом шаге инфертируется i-й бит данных(ключа), данные кодируются с использованием ключа и подсчитывается расстояние Хемминга для открытого текста и шифротекста. Ниже подводится среднее.

```
===changing data test===
0: 29
1: 25
2: 37
3: 37
4: 34
```

5: 31
6: 31
7: 34
8: 28
9: 33
10: 29
11: 28
12: 33
13: 33
14: 30
15: 39
16: 39
17: 26
18: 25
19: 26
20: 29
21: 35
22: 37
23: 34
24: 38
25: 32
26: 34
27: 30
28: 30
29: 34
30: 36
31: 26
32: 33
33: 33
34: 39
35: 32
36: 29
37: 39
38: 30
39: 33
40: 31
41: 26
42: 34
43: 34
44: 30
45: 29
46: 36
47: 29
48: 31
49: 37
50: 29
51: 30
52: 34
53: 30
54: 29

```
55: 34
56: 41
57: 31
58: 30
59: 38
60: 31
61: 28
62: 28
63: 38
64: 30
average: 32.123077
==changing key test==
0: 34
1: 37
2: 34
3: 38
4: 29
5: 36
6: 34
7: 26
8: 26
9: 25
10: 30
11: 26
12: 30
13: 31
14: 42
15: 33
16: 33
17: 32
18: 37
19: 35
20: 31
21: 29
22: 33
23: 34
24: 34
25: 24
26: 32
27: 26
28: 32
29: 27
30: 31
31: 29
32: 29
33: 38
34: 32
35: 35
36: 26
37: 34
```

```
38: 34
39: 29
40: 29
41: 37
42: 34
43: 34
44: 29
45: 30
46: 36
47: 20
48: 20
49: 31
50: 28
51: 35
52: 35
53: 32
54: 37
55: 30
56: 30
57: 28
58: 37
59: 43
60: 30
61: 29
62: 37
63: 37
64: 37
average: 31.876923
```

Ниже приведён код для демонстрации лавинного эффенкта(результаты выше)

```
my $encoder = Des->new(key => $key);
my @res;
for my $i (0 .. 64) {
    my $res = $encoder->encode_block($data);
    my $dist = BitsArray::hamming_dist($data, $res);
    printf "%d: %d\n", $i, $dist;
    push @res, $dist;

    $data->[$i] = $data->[$i] ? 0 : 1;
}

printf "average: %f\n", ( sum(@res) / @res );


print "===changing key test===\n";
$key = $rand_64->();
$data = $rand_64->();;

@res = ();
```

```
for my $i (0 .. 64) {
    $encoder = Des->new(key => $key);
    my $res = $encoder->encode_block($data);
    my $dist = BitsArray::hamming_dist($data, $res);
    printf "%d: %d\n", $i, $dist;
    push @res, $dist;

    $key->[$i] = $key->[$i] ? 0 : 1;
}

printf "average: %f", ( sum(@res) / @res );
```

Реализация

Ниже приведеня реализация алгоритма DES. Язык реализации: Перл

```
package BitsArray;
use warnings;
use strict;

use List::Util 'sum';


sub split_32 {
    my ($bit_array) = @_;
    die "length != 64" unless @$bit_array == 64;

    ([@$bit_array[0 .. 31]], [@$bit_array[32 .. 63]])
}

sub split_28 {
    my ($b) = @_;
    die "length != 56" unless @$b == 56;

    ([@$b[0 .. 27]], [@$b[28 .. 55]])
}

sub split_6 {
    my ($b) = @_;
    die "length != 48" unless @$b == 48;

    my $res;
    for (0 .. 47) {
        $res->[$_ / 6][$_ % 6] = $b->[$_]
    }
    $res
}

sub join_arrays {
    my @res;
```

```
    for (@_) {
        push @res, $_ for @$_
    }
    \@res
}

sub map_xor {
    my ($f, $s) = @_;
    die "different lenghts" unless @$f == @$s;
    my @res;
    for my $i (0 .. $#{$f}) {
        push @res, ($f->[$i] xor $s->[$i]) ? 1 : 0
    }
    \@res
}

sub circle_shift {
    my ($b) = @_;
    my $t = shift $b;
    push $b, $t;
    $b;
}

sub to_dec {
    my ($bin_arr) = @_;
    my $i = $#{$bin_arr};
    my $res = 0;
    my $bin_pow = 1;
    while ($i >= 0) {
        $res += $bin_arr->[$i] * $bin_pow;
        $bin_pow *= 2;
        --$i;
    }
    $res
}

sub from_dec {
    my ($dec) = @_;
    my @res;
    while ($dec) {
        unshift @res, $dec % 2;
        $dec = int ($dec / 2);
    }
    unshift @res, 0 while @res < 4;
    \@res
}

sub from_hex {
    my ($hex) = @_;
    my @res;
```

```
    my $h = { 0 => [qw(0 0 0 0)],
              1 => [qw(0 0 0 1)],
              2 => [qw(0 0 1 0)],
              3 => [qw(0 0 1 1)],
              4 => [qw(0 1 0 0)],
              5 => [qw(0 1 0 1)],
              6 => [qw(0 1 1 0)],
              7 => [qw(0 1 1 1)],
              8 => [qw(1 0 0 0)],
              9 => [qw(1 0 0 1)],
              a => [qw(1 0 1 0)],
              b => [qw(1 0 1 1)],
              c => [qw(1 1 0 0)],
              d => [qw(1 1 0 1)],
              e => [qw(1 1 1 0)],
              f => [qw(1 1 1 1)] };
    for (split //, $hex) {
        push @res, $_ for @{$h->{$_}}
    }
    \@res
}

sub to_hex {
    my ($self) = @_;
    my $b = [@{$self}];
    my $a = [0 .. 9, 'a' .. 'f'];
    my $res;
    while (@$b) {
        my @p = map { shift $b } 1 .. 4;
        $res .= $a->[to_dec(\@p)];
    }
    $res
}

sub pretty_print {
    my ($self, $len) = @_;
    $len ||= 8;
    my $j = 0;
    for (@{$self}) {
        print $_;
        print ' ' unless ++$j % $len;
    }
    print "\n";
}

sub hamming_dist {
    my ($f, $s) = @_;
    sum @{map_xor($f, $s)}
}
```

```
1;

package Permutation;
use Moose;
use Moose::Util::TypeConstraints;
use Data::Dumper::Concise;


subtype 'BitsArray', as 'ArrayRef[Int]',
    where {
        my $ok = 1;
        for my $val (@$_) {
            $ok &&= $val ~~ [0, 1]
        }
        $ok
    },
    message { "Contains value not in [0, 1]: " . Dumper $_ };

subtype 'Parmutatin', as 'ArrayRef[Int]',
    where {
        my %h;
        my $max = $_->[0];
        for (@$_) {
            $h{$_} = 1;
            $max = $_ if $_ > $max;
        }
        unless ($max) {
            warn "empty permutation"
        } else {
            for (1 .. $max) {
                warn "$_ unused in permutation" unless $h{$_}
            }
        }
        1
    },
    message { "" };


subtype 'IntArray', as 'ArrayRef[Int]';

has table => (isa => 'Parmutatin',
              is => 'ro',
              required => 1 );

has length => (isa => 'Maybe[Int]',
               is => 'ro');

sub execute {
    my ($self, $bits) = @_;
    find_type_constraint('ArrayRef[Int]')->check($bits)
```

```
        or die "Bad bits sequence: " . Dumper $bits;
    my $len = defined $self->length() ? $self->length() :
                                        @{$self->table()};
    @$bits == $len
        or die "Bad sequence length: " . @$bits . " != " .
            @{$self->table()};

    my $res = [];
    my $i = 0;
    for (@{$self->table()}) {
        $res->[$i++] = $bits->[$_ - 1]
    }
    $res
}

1;

package Des;
use Moose;
use Moose::Util::TypeConstraints;

subtype 'Key', as 'BitsArray',
    where { @$_ == 64 },
    message { "Bad key length" . @$_ };

my $init_perm = Permutation->new(table => [qw(
58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9  1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7
)]);

my $final_perm = Permutation->new(table => [qw(
40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9  49 17 57 25
)]);

my $ex_perm = Permutation->new(length => 32, table => [qw(
32 1 2 3 4 5
4 5 6 7 8 9
```

```
8 9 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 1
)]);

my $p_perm = Permutation->new(length => 32, table => [
16, 7, 20, 21, 29, 12, 28, 17,
1, 15, 23, 26, 5, 18, 31, 10,
2, 8, 24, 14, 32, 27, 3, 9,
19, 13, 30, 6, 22, 11, 4, 25
]);


my $pc_1_perm;
my $pc_2_perm;
{
    local $SIG{__WARN__} = sub {  };
    $pc_1_perm= Permutation->new(length => 64, table => [qw(
57 49 41 33 25 17 9
1  58 50 42 34 26 18
10 2  59 51 43 35 27
19 11 3  60 52 44 36
63 55 47 39 31 23 15
7  62 54 46 38 30 22
14 6  61 53 45 37 29
21 13 5  28 20 12 4
)]);

    $pc_2_perm = Permutation->new(length => 56, table => [qw(
14 17 11 24 1 5
3 28 15 6 21 10
23 19 12 4 26 8
16 7 27 20 13 2
41 52 31 37 47 55
30 40 51 45 33 48
44 49 39 56 34 53
46 42 50 36 29 32
)]);

}

my @s =
([
14 ,4 ,13 ,1 ,2 ,15 ,11 ,8 ,3 ,10 ,6 ,12 ,5 ,9 ,0 ,7 ,
0 ,15 ,7 ,4 ,14 ,2 ,13 ,1 ,10 ,6 ,12 ,11 ,9 ,5 ,3 ,8 ,
4 ,1 ,14 ,8 ,13 ,6 ,2 ,11 ,15 ,12 ,9 ,7 ,3 ,10 ,5 ,0 ,
15 ,12 ,8 ,2 ,4 ,9 ,1 ,7 ,5 ,11 ,3 ,14 ,10 ,0 ,6 ,13 ] ,
```

```
[
15 ,1 ,8 ,14 ,6 ,11 ,3 ,4 ,9 ,7 ,2 ,13 ,12 ,0 ,5 ,10 ,
3 ,13 ,4 ,7 ,15 ,2 ,8 ,14 ,12 ,0 ,1 ,10 ,6 ,9 ,11 ,5 ,
0 ,14 ,7 ,11 ,10 ,4 ,13 ,1 ,5 ,8 ,12 ,6 ,9 ,3 ,2 ,15 ,
13 ,8 ,10 ,1 ,3 ,15 ,4 ,2 ,11 ,6 ,7 ,12 ,0 ,5 ,14 ,9 ] ,
[
10 ,0 ,9 ,14 ,6 ,3 ,15 ,5 ,1 ,13 ,12 ,7 ,11 ,4 ,2 ,8 ,
13 ,7 ,0 ,9 ,3 ,4 ,6 ,10 ,2 ,8 ,5 ,14 ,12 ,11 ,15 ,1 ,
13 ,6 ,4 ,9 ,8 ,15 ,3 ,0 ,11 ,1 ,2 ,12 ,5 ,10 ,14 ,7 ,
1 ,10 ,13 ,0 ,6 ,9 ,8 ,7 ,4 ,15 ,14 ,3 ,11 ,5 ,2 ,12 ] ,
[
7 ,13 ,14 ,3 ,0 ,6 ,9 ,10 ,1 ,2 ,8 ,5 ,11 ,12 ,4 ,15 ,
13 ,8 ,11 ,5 ,6 ,15 ,0 ,3 ,4 ,7 ,2 ,12 ,1 ,10 ,14 ,9 ,
10 ,6 ,9 ,0 ,12 ,11 ,7 ,13 ,15 ,1 ,3 ,14 ,5 ,2 ,8 ,4 ,
3 ,15 ,0 ,6 ,10 ,1 ,13 ,8 ,9 ,4 ,5 ,11 ,12 ,7 ,2 ,14 ] ,
[
2 ,12 ,4 ,1 ,7 ,10 ,11 ,6 ,8 ,5 ,3 ,15 ,13 ,0 ,14 ,9 ,
14 ,11 ,2 ,12 ,4 ,7 ,13 ,1 ,5 ,0 ,15 ,10 ,3 ,9 ,8 ,6 ,
4 ,2 ,1 ,11 ,10 ,13 ,7 ,8 ,15 ,9 ,12 ,5 ,6 ,3 ,0 ,14 ,
11 ,8 ,12 ,7 ,1 ,14 ,2 ,13 ,6 ,15 ,0 ,9 ,10 ,4 ,5 ,3 ] ,
[
12 ,1 ,10 ,15 ,9 ,2 ,6 ,8 ,0 ,13 ,3 ,4 ,14 ,7 ,5 ,11 ,
10 ,15 ,4 ,2 ,7 ,12 ,9 ,5 ,6 ,1 ,13 ,14 ,0 ,11 ,3 ,8 ,
9 ,14 ,15 ,5 ,2 ,8 ,12 ,3 ,7 ,0 ,4 ,10 ,1 ,13 ,11 ,6 ,
4 ,3 ,2 ,12 ,9 ,5 ,15 ,10 ,11 ,14 ,1 ,7 ,6 ,0 ,8 ,13 ] ,
[
4 ,11 ,2 ,14 ,15 ,0 ,8 ,13 ,3 ,12 ,9 ,7 ,5 ,10 ,6 ,1 ,
13 ,0 ,11 ,7 ,4 ,9 ,1 ,10 ,14 ,3 ,5 ,12 ,2 ,15 ,8 ,6 ,
1 ,4 ,11 ,13 ,12 ,3 ,7 ,14 ,10 ,15 ,6 ,8 ,0 ,5 ,9 ,2 ,
6 ,11 ,13 ,8 ,1 ,4 ,10 ,7 ,9 ,5 ,0 ,15 ,14 ,2 ,3 ,12 ] ,
[
13 ,2 ,8 ,4 ,6 ,15 ,11 ,1 ,10 ,9 ,3 ,14 ,5 ,0 ,12 ,7 ,
1 ,15 ,13 ,8 ,10 ,3 ,7 ,4 ,12 ,5 ,6 ,11 ,0 ,14 ,9 ,2 ,
7 ,11 ,4 ,1 ,9 ,12 ,14 ,2 ,0 ,6 ,10 ,13 ,15 ,3 ,5 ,8 ,
2 ,1 ,14 ,7 ,4 ,10 ,8 ,13 ,15 ,12 ,9 ,0 ,3 ,5 ,6 ,11 ]);

has 'key' => ( isa => 'Key',
               is => 'ro',
               required => 1 );

has 'round_keys' => ( isa => 'ArrayRef',
                      is => 'ro',
                      required => 0 );

sub BUILD {
    my ($self) = @_;
    my @keys;
    my $ext_key = $pc_1_perm->execute($self->key());
    my ($c, $d) = BitsArray::split_28($ext_key);
    for my $round (1 .. 16) {
```

```
        for (($round ~~ [1, 2, 9, 16]) ? 1 : (1, 2)) {
            BitsArray::circle_shift($c);
            BitsArray::circle_shift($d);
        }
        my $t_key = BitsArray::join_arrays($c, $d);
        push @keys, $pc_2_perm->execute($t_key);
    }
    $self->{round_keys} = \@keys;
};

sub _select_s {
    my ($self, $s_num, $bit_arr) = @_;
    die "should be 6-bit" unless @$bit_arr == 6;
    my $b = [@{$bit_arr}];
    my $t = shift $b;
    my $p = pop $b;
    unshift $b, $p;
    unshift $b, $t;

    my $n = BitsArray::to_dec($b);
    my $res =  BitsArray::from_dec( $s[$s_num][$n] );

    $res;
}

sub _f {
    my ($self, $round_key, $data) = @_;
    my $ex_d = $ex_perm->execute($data);
    my $t_res = BitsArray::map_xor($ex_d, $round_key);

    my $s_block_data = BitsArray::split_6($t_res);
    my @s_block_res;
    for my $s_block_num (0 .. 7) {
        my $t = $self->_select_s($s_block_num,
                                 $s_block_data->[$s_block_num]);
        push @s_block_res, $t;
    }

    $p_perm->execute(BitsArray::join_arrays(@s_block_res));
}

sub _process_block {
    my ($self, $data, $encode) = @_;
    die "bad block length" unless @$data == 64;

    my ($a_0, $b_0) = BitsArray::split_32(
                          $init_perm->execute($data)
                      );
    my $get_key = $encode ? sub { $_[0] - 1 } :
                            sub { 17 - $_[0] - 1 } ;
```

```
    for my $round (1 .. 16) {
        my $subkey = $self->round_keys()->[$get_key->($round)];

        my $f_res = $self->_f($subkey, $b_0);
        ($a_0 , $b_0) = ($b_0, BitsArray::map_xor($f_res, $a_0));
    }
    ($a_0 , $b_0) = ($b_0, $a_0);
    my $res = BitsArray::join_arrays($a_0, $b_0);

    $final_perm->execute($res)
}

sub encode_block {
    my ($self, $data) = @_;
    $self->_process_block($data, 1);
}

sub decode_block {
    my ($self, $data) = @_;
    $self->_process_block($data, 0);
}

1;
```