Cryptography midterm report

Tran Van Sang, Department of Computer Science, 48-186106
Eguchi Shingo, Department of Computer Science, 48-186016

June 10, 2018

Contents

Topic

We presented basic checksum algorithms such as the one used in credit card numbers (Chapter 3). Explain what is a quasigroup (Section 4.1). Present and analyze Damms method based on quasigroups to detect all single-digit and transposition errors (Section 4.2). Are there other quasigroups that work? (Section 4.3)

Chapter 1

Hash function

Hash function is a class of function that takes an input of arbitrary length, called message, and produces an output of fixed length, called hash. Hash function has important role in cryptography and many applications due to its one-wayness. Hash function ability to resist cryptanalytic attack is mainly determined by 3 properties

- Pre-image resistance. Sometime referred as one-wayness. Given a hash value h, it should be infeasible within acceptable time to find original domain value m such that the hash function maps m into h.
- Second pre-image resistance. It should be infeasible within acceptable time to find another input m that has same image mapped by the hash function, given input m.
- Collision resistance. It should be infeasible within acceptable time to find 2 input messages m, m such that the hash function maps them into same hash.

It is obvious that pre-image resistance hash function is also pre-image resistance. Similarly, functions that resists collision also have second pre-image resistance property.

Chapter 2

ISBN

2.1 Introduction

International Standard Book Number (ISBN) is an unique 13 digits long number assigned to a book. Affiliates of International ISBN Agency have ability to release a number whenever being requested by publisher. The last digit is the hash of 10 preceding digits. In other words,

$$ISBN = m_{13}, m_{12}, m_{11}, \ldots, m_1$$

$$m_1 = isbn(m_{13}, m_{12}, m_{11}, \ldots, m_2)$$

Whether

$$isbn(m_{13}, m_{12}, m_{11}, \ldots, m_2) = \sum_{i \equiv 1 \pmod 2, 2 \leq i \leq 13} m_i + 3 \times \sum_{i \equiv 0 \pmod{()2}, 2 \leq i \leq 13} m_i$$

2.2 Detection ability

Using ISBN system, one can detect all single-digit errors and adjacent transposition errors unless that those numbers' difference is 5 (i.e. swapping numbers are one of (0, 5), (1, 6), (2, 7), (3, 8), or (4, 9))

To check if there is any modification on the received ISBN number, one can check by calculating checksum $C = \sum_{i \equiv 1 \pmod 2, 2 \leq i \leq 13} m_i + 3 \times \sum_{i \equiv 0 \pmod 2, 2 \leq i \leq 13} m_i - m_1 \mod 10$ equal to 0.

2.3 Proof

2.3.1 Single digit error

We will prove with this method 2 kinds of modification mentioned above can be detected.
Assume that ith number is changed to m_i' while other numbers keep the same. Absolute of difference between new sum and old sum is

$$|C - C'| \mod 10 = \begin{cases} |m_i - m_i'| \mod 10 & \text{if} i \equiv 1 \pmod 2 \\ 3 \times |m_i - m_i'| \mod 10 & \text{otherwise} \end{cases}$$

Because $GCD(3, 10) = 0$, $|C - C'| \mod 10 \neq 0$

2.3.2 Adjacent transposition error

If 2 digits m_i and m_{i+1} are swapped, and $|m_i - m_{i+1}| \neq 5$
Similar to previous subsection

$$\begin{aligned} |C - C'| \mod 10 &= \begin{cases} |m_i + 3 \times m_{i+1} - 3 \times m_i - \times m_{i+1}| \mod 10 & \text{if} i \equiv 1 \pmod 2 \\ |3 \times m_i + m_{i+1} - m_i - 3 \times m_{i+1}| \mod 10 & \text{otherwise} \end{cases} \\ &= 2 \times |m_i - m_{i+1}| \end{aligned} \tag{2.1}$$

Thus, $|C - C'| \mod 10 \neq 0$

2.4 Generalization

The ISBN checksum is one special case of the hash function $H(m) = \sum_{i \geq 2} a_i m_i \mod 10$ where $a_i (i \geq 2)$ are fixed
In the following chapter we will discuss on other form of checksum with better error detection ability.

Chapter 3

Credit card checksum

Credit card number contains n number $m_n, m_{n-1}, \ldots, m_2, m_1$ where n varies and depends on credit card provider. For example, $n = 16$ for Visa, Master card, $n = 19$ for UnionPay card
The following hash function is used in credit card to detect error

$$m_1 = H(m)$$

is chosen such that

$$\sum_{iisodd} m_i + \sum_{iiseven} f(m_i) \equiv 0 \pmod{10}$$

whether

$$f(x) = 2x + \left[\frac{x}{5}\right] \mod 10$$

This $f(x)$ is called Luhn (1954) function.
By this checksum, it is able to detect all single digit errors and adjacent transposition errors unless 2 adjacent numbers are not (0, 9) or (9, 0) pairs.

Before giving proof to the error detection, we observe some properties of the Luhn function

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	2	4	6	8	1	3	5	7	9
$(x - f(x)) \mod 10$	0	9	8	7	6	4	3	2	1	0

Permutation property: f maps $\{0, 1, \ldots, 9\}$ to the same set $\{0, 1, \ldots, 9\}$ without collision. In other words, for all $x_1 \neq x_2$ $f(x_1) \neq f(x_2)$. Functions having this property are called permutation functions.

Second property: there are only 2 fixed points (0 and 9), i.e. $x = f(x)$ and for all x, and $(x - f(x)) \mod 10$ are almost different unless $x = 0$ or $x = 9$.

Now we move on proving the error detection ability.

If only one digit is changed, m_i is changed to m'_i,

$$|C - C'| \mod 10 = \begin{cases} |m_i - m'_i| \mod 10 & \text{if} i \equiv 1 \pmod 2 \\ f(m_i) - f(m'_i) \mod 10 & \text{otherwise} \end{cases}$$

Because of permutation property, this value never be zero. Thus, the error can be detected

If 2 adjacent numbers m_i, m_{i+1} are swapped.

$$\begin{aligned} |C - C'| \mod 10 &= \begin{cases} |m_i + f(m_{i+1}) - m_{i+1} - f(m_i)| \mod 10 & \text{if} i \equiv 1 \pmod 2 \\ |f(m_i) - m_{i+1} - f(m_{i+1}) - m_i| \mod 10 & \text{otherwise} \end{cases} \\ &= |(m_i - f(m_i)) - (m_{i+1} - f(m_{i+1}))| \mod 10 \end{aligned} \tag{3.1}$$

If $m_i = m_{i+1}$, there is no need of detection. If m_i, m_{i+1} are not $(0, 9)$ or $(9, 0)$ pair, from the second property of Luhn function, $m_i - f(m_i) \neq m_{i+1} - f(m_{i+1})$. Thus, $|C - C'| \mod 10 \neq 0$

Chapter 4

Damm's method

Damm's method is check digit algorithm based on *quasigroup*. It can detect all single-digit errors and all adjacent transposition errors.

4.1 Quasigroup

Quasigroup $(Q, *)$ is a set Q with a binary operation "$*$" such that *Latin square property* holds: for every $a, b \in Q$, there exists unique x and y satisfy the following.

$$a * x = b$$
$$y * a = b$$

Let us consider a *Cayley table* of the operation "$*$". Latin square property can be characterize by whether the table is *Latin square*: each element occurs exactly once in each row and exactly once in each column. For example, the operation expressed by the following table satisfies Latin square property.

$*$	0	1	2
0	0	1	2
1	1	2	0
1	2	0	1

The following lemma is derived immediately, which is significant for error detection of damm's method.

Lemma 1 *Suppose* $(Q, *)$ *is quasigroup. For arbitrary* x, x' *and* a *in* Q*, the following hold.*

$$x \neq x' \Rightarrow a * x \neq a * x' \tag{4.1}$$
$$x \neq x' \Rightarrow x * a \neq x' * a \tag{4.2}$$

4.2 Damm's method

This algorithm is based on quasigroup $(Q, *)$ of 10-order with a special property: for all $c, x, y \in Q$ the followings hold.

$$(c * x) * y = (c * y) * x \ \Rightarrow x = y \tag{4.3}$$

$$x * x = 0 \tag{4.4}$$

The example of such a quasigroup is shown below.

*	0	1	2	3	4	5	6	7	8	9
0	0	3	1	7	5	9	8	6	4	2
1	7	0	9	2	1	5	4	8	6	3
2	4	2	0	6	8	7	1	3	5	9
3	1	7	5	0	9	8	3	4	2	6
4	6	1	2	3	0	4	5	9	7	8
5	3	6	7	4	2	0	9	5	8	1
6	5	8	6	9	7	2	0	1	3	4
7	8	9	4	5	3	6	2	0	1	7
8	9	4	3	8	6	1	7	2	0	5
9	2	5	8	1	4	3	6	7	9	0

Figure 4.1: Example:Quasigroup for damm's method

The validity of a digit sequence $m_1 m_2 ... m_n$ is judged by whether $(...((0 * m_1) * m_2) * ... * m_n) = 0$ holds. Given a digit sequence $m_1 m_2 ... m_{n-1}$, a valid sequence can be generated by adding $m_n = (...((0 * m_1) * m_2) * ... * m_{n-1})$ to the tail: $(...((0 * m_1) * m_2) * ... * m_n) = 0$ holds from the equation 4.4.

This method can detect all single-digit errors and all adjacent transposition errors. The proof is as follow. We denote k_i^m as $(...(0 * m_1) * ... * m_i)$.

- single-digit errors

  Suppose digit sequences m and m' are same except for i'th element: $m_i \neq m'_i$.

  $$k_i^m = k_{i-1}^m * m_i$$
  $$k_i^{m'} = k_{i-1}^m * m'_i$$

  From the implication 4.1,

  $$k_i^m \neq k_i^{m'}$$

  is derived.

  $$k_{i+1}^m = k_i^m * m_i$$
  $$k_{i+1}^{m'} = k_i^m * m'_i$$

From the implication 4.2,

$$k_{i+1}^{m} \neq k_{i+1}^{m'}$$

Therefore, inductively,

$$k_{n}^{m'} \neq k_{n}^{m} = 0$$

- adjacent transposition errors

  Suppose a digit sequence m' is a result of swapping m_i and m_{i+1} in a sequence m. Here, we assume $m_i \neq m_{i+1}$.

$$k_{i+1}^{m} = (k_{i-1}^{m} * m_i) * m_{i+1}$$
$$k_{i+1}^{m'} = (k_{i-1}^{m} * m_{i+1}) * m_i$$

  From (4.3),

$$k_{i+1}^{m} \neq k_{i+1}^{m'}$$

  is derived. Hence, same as the case single-digit errors ,

$$k_{n}^{m'} \neq k_{n}^{m} = 0$$

4.3 Sufficient condition

We modified the original validation method such that the additional property is as weak as possible.

Given a digit sequence $m_1 m_2 \ldots m_{n-1}$, since *Latin square property* of Quasigroup, there exists unique m_n such that $(((0 * m_1) * m_2) * \ldots * m_{n-1}) * m_n = 0$. Valid sequence can be generated by adding this m_n value to the tail.

single-digit errors proof requires implication 4.1 and 4.2, that can be implied from *Latin square property* of Quasigroup. And, adjacent transposition errors proof requires only 4.3. We wrote a program to find all such Quasigroups that they satisify 4.3 but not 4.4. The result is shown in Figure 4.2, 4.3, 4.4.

C++ code is as following

```
#include <iostream>
using namespace std;

#define SIZE 10
//number of tables to find
#define NFIND 3
int cnt = NFIND;

void tprint(int a[][SIZE]){
  cout << "Result " << (NFIND - cnt + 1) << endl;
  for(int row = 0; row < SIZE; row++){
    for(int col = 0; col < SIZE; col ++)
    cout << a[row][col] << " ";
```

```
    cout << endl;
    //print for latex
    //cout << row;
    //for(int col = 0; col < SIZE; col ++)
      //cout << " & " << a[row][col];
    //cout << "\\\\\n\\hline\n";
  }
  cout << endl;
}
bool f(int a[][SIZE], int row, int col, int val){
  if (!cnt) return false;
  a[row][col] = val;
  int nrow, ncol;
  if (col < SIZE - 1){
    ncol = col + 1;
    nrow = row;
  } else if (row < SIZE - 1){
      ncol = 0;
      nrow = row + 1;
    } else {
      bool found = false;
      for(int i = 0; i < SIZE; i++)
        if (a[i][i] != i){
          found = true;
          break;
        }
      if (!found) return false;
      tprint(a);
      cnt--;
      return !cnt;
    }
  bool checked[SIZE];
  fill(checked, checked + SIZE * sizeof(bool), false);
  for(int i = 0; i < ncol; i++)
    checked[a[nrow][i]] = true;
  for(int i = 0; i < nrow; i++)
    checked[a[i][ncol]] = true;
  for(int v = 0; v < SIZE; v++)
    if (!checked[v] && f(a, nrow, ncol, v))
      return true;
  return false;
}

int main(){
  int a[SIZE][SIZE];
  for(int i = 0; i < SIZE; i++)
```

```
    f(a, 0, 0, i);
  if (cnt == NFIND)
    cout << "not found" << endl;
  return 0;
}
```

*	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	0	3	2	5	4	7	6	9	8
2	2	3	0	1	6	7	8	9	4	5
3	3	2	1	0	7	6	9	8	5	4
4	4	5	6	7	8	9	0	1	2	3
5	5	4	7	6	9	8	1	0	3	2
6	6	7	8	9	2	3	4	5	0	1
7	7	6	9	8	3	2	5	4	1	0
8	8	9	4	5	0	1	2	3	6	7
9	9	8	5	4	1	0	3	2	7	6

Figure 4.2: Example1: Non anti-symmetric Quasigroup for damm's method

*	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	0	3	2	5	4	7	6	9	8
2	2	3	0	1	6	7	8	9	4	5
3	3	2	1	0	7	6	9	8	5	4
4	4	5	6	7	8	9	0	1	2	3
5	5	4	7	6	9	8	1	0	3	2
6	6	7	8	9	2	3	4	5	0	1
7	7	6	9	8	3	2	5	4	1	0
8	8	9	4	5	0	1	2	3	7	6
9	9	8	5	4	1	0	3	2	6	7

Figure 4.3: Example2: Non anti-symmetric Quasigroup for damm's method

*	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	0	3	2	5	4	7	6	9	8
2	2	3	0	1	6	7	8	9	4	5
3	3	2	1	0	7	6	9	8	5	4
4	4	5	6	7	8	9	0	1	2	3
5	5	4	7	6	9	8	1	0	3	2
6	6	7	8	9	2	3	4	5	0	1
7	7	6	9	8	3	2	5	4	1	0
8	8	9	4	5	0	1	3	2	6	7
9	9	8	5	4	1	0	2	3	7	6

Figure 4.4: Example3: Non anti-symmetric Quasigroup for damm's method