

SM3生日攻击

一.生日攻击定义

引入：最少有多少人，使得两个人是同一天生日的概率大于50%？

答：经过概率论公式的计算后，得出最少为 $1.17 \times 365^{0.5}$ 人

扩展到密码学中，即如果取出 $1.17 \times (2^n)^{0.5}$ 个样本进行哈希，那么就有50%的概率得到这样两个字符串：他们的哈希值的前n bit相同。

二.C++

可以碰撞32位

1.代码展示

首先定义一个Attack类，其中的方法调用已经实现并且优化的SM3接口。其中str是哈希的原像，result是哈希的结果。int size是碰撞的位数，想要碰撞几位，最后的哈希结果就会截取几位，作为result。

```
class Attack {
private:
    SM3_basic test;
    string str;
    string result;
    int size;
public:
    //对str进行hash，取前size个字节
    void init(string str_,int size) {
        str = str_;
        test.update(str);
        size = size;
        result = test.final().substr(0, size);
    }
    string get_result() {
        return result;
    }

    string get_str() {
        return str;
    }

    bool is_empty() {
        return str.empty();
    }

    bool empty() {
        bool result = str.empty();
        return result;
    }
};
```

对==符号进行重载，方便以后的代码

```
bool operator==(Attack& st1,Attack& st2)
{
    string s1 = st1.get_result();
    string s2 = st2.get_result();
    return (s1==s2);
}
```

然后定义生日碰撞函数，两个参数：n为碰撞的位数，size为样本数量。由于生日碰撞的成功率为50%，所以循环两次。在一次循环中，首先先生成随机字符串，并调用Attack类中的方法进行哈希。然后寻找哈希值一样的元素，如果找到了，那么输出碰撞结果；如果没找到，算法失败。

```
//对前n个半字节进行生日攻击
void birthday_attack(int n,long long size) {
    Attack* test = new Attack[size];
    //两轮
    for (int k = 0; k < 2; k++) {
        //生成随机字符串
        int i;
        int limit = size - 1;
        for (i = 0; i < limit; i += 2) {
            test[i].init(makeRandStr(56), n);
            test[i + 1].init(makeRandStr(56), n);
        }
        for (i; i < size; i++) {
            test[i].init(makeRandStr(56), n);
        }

        Attack* result = find_equal_string(test, 0, size);
        if (result!=nullptr) {
            cout << "*****碰撞结果*****" << endl;
            cout << "字符串1: " << result[0].get_str() << endl;
            cout << "字符串2: " << result[1].get_str() << endl;
            cout << "哈希结果1: " << result[0].get_result() << endl;
            cout << "哈希结果2: " << result[1].get_result() << endl;
            cout << "找到了" << n * 4 << "位的碰撞" << endl;
            return;
        }

    }
    cout << "失败" << endl;
    delete[] test;
}
```

对于其中的关键步骤：寻找哈希值一样的元素。这里采用的是依次对比的方法，时间复杂度为 $O(n^2)$ 。代码如下

```
//找到两个哈希值相同的元素
Attack* find_equal_string(Attack* test, long long start,long long end) {
```

```

Attack* result = new Attack[2];
for (int i = start; i < end; i++) {
    for (int j = i + 1; j < end; j++) {
        if (test[i] == test[j]) {
            result[0] = test[i];
            result[1] = test[j];
            find_equal = true;
            return result;
        }
    }
}
return nullptr;
}

```

最后在主函数中调用。按照生日攻击的定义，样本数量至少为 $1.17 \cdot 2^{(0.5n)}$ 。但在代码中，n每加1，碰撞数量加4（即一个n代表4bit），因此代码中的样本数量为 $1.17 \cdot 2^{(2n)}$ 。

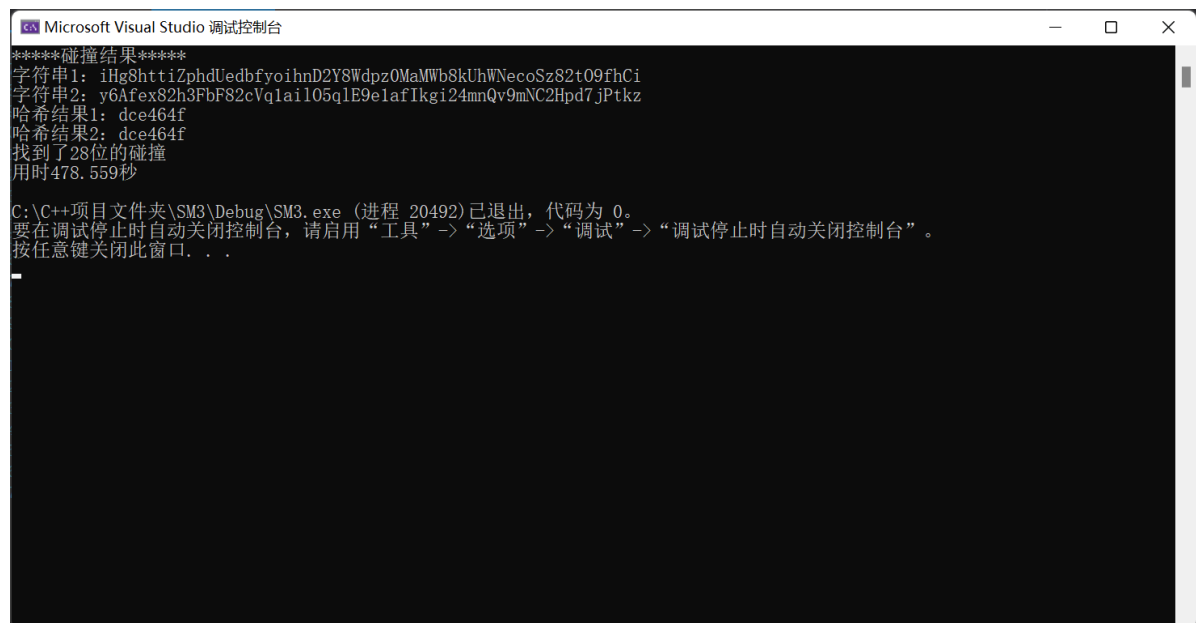
```

int main() {
    clock_t start, end;
    int n = 7;
    long long size = long long(ceil(1.17 * pow(2, (n - 1) * 2)));
    birthday_attack(n, size);
    end = clock();
    double endtime = (double)(end - start) / CLOCKS_PER_SEC;
    cout << "用时" << endtime << "秒" << endl;
}

```

2.结果分析

使用上述代码，最多可以碰撞28bit，结果如图。



The screenshot shows the Microsoft Visual Studio Debug Console window. The output text is as follows:

```

**** 碰撞结果 ****
字符串1: iHg8httiZphdUedbfoyihnd2Y8Wdpz0MaMWb8kUhWNecoSz82t09fhCi
字符串2: y6Afex82h3FbF82cVqlai105qlE9e1afIkgi24mnQv9mNC2Hpd7jPtkz
哈希结果1: dce464f
哈希结果2: dce464f
找到了28位的碰撞
用时478.559秒

C:\C++项目文件夹\SM3\Debug\SM3.exe (进程 20492)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。

```

经过测试，在28位碰撞下，Attack类生成对应的哈希值只需要5秒左右，而总用时在480秒左右，可见程序的主要制约因素在于寻找相同的哈希值这一步。我对这一步进行了优化，使用哈希表的方法用空间换时间，时间复杂度为 $O(n)$ ，空间复杂度为 $O(n)$ 。但是这种优化效果并不明显，因为哈希表本身还存在开销，使用了哈希表后，最多可以碰撞32位，如图。

```
C:\C++项目文件夹\SM3\Debug\SM3.exe
*****碰撞结果*****
字符串1: r5LiI1672jt8SnYlfzQ3dHaeXW2EMstTK23TLQ7SCuhMHPqZZvLFN8Z5
字符串2: HvdRvi4hYX9Vp0Q6VC7ASfEB1Ui68YzggjQjvYFXUhWy8xWQzsEGIoK7
哈希结果1: 603e9b19
哈希结果2: 603e9b19
找到了32位的碰撞
```

三.C++加python

可以碰撞48位

在发现纯C++实现效率不高后，我想到了使用python大数据分析的技术来帮助我找到哈希值相同的元素，因此我使用了pandas库。

1.pandas库介绍

(1) 数据结构

dataframe：二维表格

series：一维的行或列

(2) 基本方法

读取txt文件

```
sample = pd.read_csv(<文件路径>, sep=<列之间的分隔符>, names = <列名>)
```

表格去重

```
sample.drop_duplicates(subset=<去重的列名>, keep=False)
```

提取出重复的元素

```
#单加一列duplicate_bool，将重复元素的duplicate_bool值设为true
duplicate_bool = sample.duplicated(subset=<列名>, keep='first')
#挑选出duplicate_bool为true的元素
repeat = sample.loc[duplicate_bool == True]
```

2.代码展示

在用python进行数据分析后，C++的代码也发生了一些变化。首先定义一个attack_init，调用attack类中的方法计算哈希值。为了方便并行化，在参数中加上开始和结束的下标。并且为了更高的效率，函数中使用了循环展开技术。

```

void attack_init(Attack* test,int n,long long start,long long end) {
    int i;
    int limit = end - 1;
    for (i = start; i < limit; i += 2) {
        test[i].init(makeRandStr(56), n);
        test[i + 1].init(makeRandStr(56), n);
    }
    for (i; i < end; i++) {
        test[i].init(makeRandStr(56), n);
    }
}

```

在主函数中调用attack_init函数。经过测试，调用线程数小于4时，加速效果基本呈指数型增长；调用线程数大于4小于6时，加速效果逐渐减缓，但是依然可以加速；在调用线程数大于6时，没有任何加速效果；综上，使用6线程。同时，使用txt文件作为C++程序和python程序交互的接口，将哈希原像和哈希结果都输入到这个txt文件中。至此，C++的任务完成。

```

int main() {
    clock_t start, end;
    int n = 11;
    long long size = long long(ceil(1.17 * pow(2, (n) * 2)));
    fstream myfile;
    myfile.open("C:\\Users\\86139\\Documents\\test123.txt", ios::out);
    Attack* test = new Attack[size];
    start = clock();
    thread t1(attack_init, test, n, long long(0), long long(size / 6));
    thread t2(attack_init, test, n, long long(size / 6), long long(size / 3));
    thread t3(attack_init, test, n, long long(size / 3), long long(size / 2));
    thread t4(attack_init, test, n, long long(size / 2), long long(size / 3 *
2));
    thread t5(attack_init, test, n, long long(size / 3 * 2), long long(size / 6
* 5));
    thread t6(attack_init, test, n, long long(size / 6 * 5), long long(size));
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();
    t6.join();
    if (myfile.is_open())
    {
        for (int j = 0; j < size; j++) {
            string s = test[j].get_result();
            string s1 = test[j].get_str();
            myfile << s1 << "\\t" << ":" << s << "\\n";
        }
    }

    myfile.close();
    end = clock();
    double endtime = (double)(end - start) / CLOCKS_PER_SEC;
    cout << "用时" << endtime << "秒" << endl;
    delete[] test;
}

```

```
}
```

在python程序中，首先先读取C++程序生成的txt文件，而后调用pandas库中的数据分析的方法：首先对原像相等的进行去重，去重后，找到哈希值相等的并输出，即为碰撞结果。

```
import pandas as pd
if __name__ == "__main__":
    #打开txt文件
    sample =
pd.read_csv("C:\\Users\\86139\\Documents\\test123.txt", sep=":", names = ["原像", "哈希值"])
    #对于原像相等的进行去重
    data1 = sample.drop_duplicates(subset=['原像'], keep=False)
    print(data1.shape)
    #找到哈希值相等的
    duplicate_bool = data1.duplicated(subset=['哈希值'], keep='first')
    repeat = data1.loc[duplicate_bool == True]
    #哈希值相等的构造一个列表
    hash_ls = repeat.哈希值.values
    #在大表格中对这些相等的哈希值进行查找并打印
    for i in hash_ls:
        print(data1[data1["哈希值"] == i])
```

3.结果分析

在使用了python大数据库后，碰撞效果大大增强，最高可以碰撞44位。结果如下。

```
Process finished with exit code 0
```

	原像	哈希值
211084	PhU5sH4sJ0UAtpQIn9Gv0k87IfrJvUroGctIPA7ztJtXrn...	9f3ded51423
2312866	xaf7tKeLEGeM0h0gzqhAIyzHPR1vZlpgBmh1JIwbsHAqbh...	9f3ded51423
	原像	哈希值
2641694	EgM5SkS249nRljgArDiB7vnjwQ6sewkfgUR6DJZjIQLWbG...	a0fce4a5f9d
3520168	zZkVGLeyE56pjKkwJFotaxUrWX3MsT4NSGwS83cNfjcHWL...	a0fce4a5f9d
	原像	哈希值
134763	Bww0WeFQWp6MhGnTZ7dhiE2FuwgSAAUt9QTjWbeqFk61j4...	6f596ac9237
3654830	UYPaUmWzBvUp6WsUNZbQ4RCtBfUFzgDoEk5mWyR8Ng8iaD...	6f596ac9237

在使用python库后，程序的主要性能制约变为计算哈希值并输出到文件中这一步。以44位碰撞为例，C++程序的执行时间为1440秒，而python程序的执行时间为7秒，二者时间相差200倍左右。

4.更高位

当我把44位碰撞提升到48位碰撞时，C++程序出现异常。因为在运行Attack* test = new Attack[size]这一行代码时，size太大，所分配的空间超过了操作系统分配的虚拟内存的大小，因此程序报出异常。因此可以把test分割为4份，循环四次，每次循环运行完成后就及时删除那一份，这样可以保证同一时间test的大小仅仅为size/4，这样就不会出现异常。代码如下。在正常情况下，循环四次就可以，但是因为生日攻击成功率只有50%，所以在本次实验中循环8次，以增强成功率。

```
int main() {
```

```

clock_t start, end;
int n = 12;
long long size = long long(ceil(1.17 * pow(2, (n - 1) * 2)));
fstream myfile;
myfile.open("C:\\Users\\86139\\Documents\\test123.txt", ios::out);
for (int k = 0; k < 8; k++) {
    Attack* test = new Attack[size];
    start = clock();
    thread t1(attack_init, test, n, long long(0), long long(size / 6));
    thread t2(attack_init, test, n, long long(size / 6), long long(size /
3));
    thread t3(attack_init, test, n, long long(size / 3), long long(size /
2));
    thread t4(attack_init, test, n, long long(size / 2), long long(size / 3
* 2));
    thread t5(attack_init, test, n, long long(size / 3 * 2), long long(size
/ 6 * 5));
    thread t6(attack_init, test, n, long long(size / 6 * 5), long
long(size));
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();
    t6.join();
    if (myfile.is_open())
    {
        for (int j = 0; j < size; j++) {
            string s = test[j].get_result();

            string s1 = test[j].get_str();
            myfile << s1 << "\\t" << ":" << s << "\\n";
        }

    }
    delete[] test;
}
myfile.close();
end = clock();
double endtime = (double)(end - start) / CLOCKS_PER_SEC;
cout << "用时" << endtime << "秒" << endl;
}

```

在生成txt文件后运行python程序，得到如下结果。48bit碰撞完成。

```

                                原像          哈希值
4708946  4ZQMELPhbooKXWSRiCNec2ISKakGv0qimnqAvn6m9nd3ZL...  3e2cfa86b96b
11728146 cs6WJwIN2iUy75vhIBFMEDQnEM32qU0WwbJu2wrVefINZa...  3e2cfa86b96b
                                原像          哈希值
19562493 EP5mX4dUB6QXD3SftyBBZLQyJ6SgEo9mhF6YjNjQfa3R8W...  bc1f03e0f824
19615744 pfSQGYLE5iPU1rkP2QPqrrmkIg1v4HET1v0xNfZ4lCt3zp...  bc1f03e0f824
                                原像          哈希值
13885341 qj7sTtxpPE9QiDukZNeWDTA0v1uw27GLNYUWpngNupyN55...  24e3e9bf8911
28391666 7WNrgaE3xmFaJH6jA2s2l0KyG16FJ9bhov1QDpQpKwdHKJ...  24e3e9bf8911
                                原像          哈希值
12562343 PkrXoAiywxyAw6xzBhyGzJS3fjtRjMpwt0FMu0p4QkGL3A...  ae3061e7f283
30800162 Ke0lAixqcANTqcIcxRS6hTpxB7o5gLw6SPopMBJmzbwVyh...  ae3061e7f283
                                原像          哈希值
28639731 Gks4vzyD8MxQ057IgTMbpFXApQpRd4YUJE3nmhZu96UqwY...  dafbf488958d
37766141 QeNc05ohAYGALYZT7vDITbJQEk8uAWsTJrmzTchgT4fj7t...  dafbf488958d
                                原像          哈希值
36865965 wGrMnbYl907wxhjf976dCnL2gT3GGeLUzgm5gwfxU570A3...  ffb10df4a712
38959955 SccsD7jKB0p0TJNeLqe1Ib0v36YwIEgHXUqBuTD5up16FU...  ffb10df4a712

Process finished with exit code 0
|

```

时间分析：在高位碰撞中，性能制约还是在于C++程序，对于48bit而言，生成txt文件花费了约5个小时，而使用python大数据分析仅仅用了80秒。