SHA-RNN的分析与破解

韩奉延 罗裕佳 金雨琳 孙宇涛 张熙至

1 摘要

SHA-RNN是一种利用了混沌系统、循环神经网络特点的sponge结构的Hash Function（https://github.com/Ashitemaru/Sharnn）。在本次实验中，我们组对SHA-RNN进行了分析和破解，得到如下主要结果：

- 随机性分析：在随机数据的测试下，随机性测试保持良好，但是在**高密和低密数据**的测试下，随机性测试表现较差。
- 发现了**重要性质**（漏洞）：SHA-RNN生成的80bit 哈希值中，若**前40bit相同，后40bit大概率相同。**
- 区分攻击：在不考虑上述重要性质的情况下，随机性的区分攻击能**较小程度区分**SHA-RNN生成序列与随机序列；在考虑上述重要性质的情况下，其**显然区分**于随机序列。
- 碰撞攻击：利用上述重要性质，我们**找到了多对SHA-RNN上的碰撞**，成功攻破了SHA-RNN。

我们的代码仓库位于https://github.com/Robin-Diddle/Sharnn。

2 随机性检验

- 序列生成
    - 随机序列：在 $[0, 2^{64} - 1)$ 中利用 `mt19937_64` 选取大量随机数，将数字对应的 64 位整数作为输入进行哈希，将输出保存至二进制文件中；
    - 高密序列：在 $[0, 2^{64} - 1)$ 中构造二进制表示中1多（多于40位）的数字，将数字对应的 64 位整数作为输入进行哈希，将输出保存至二进制文件中；
    - 低密序列：在 $[0, 2^{64} - 1)$ 中构造二进制表示中1少（少于20位）的数字，将数字对应的 64 位整数作为输入进行哈希，将输出保存至二进制文件中；
- 使用 NIST `Statistical Test Suite` 基于多种度量，对随机、高密、低密序列进行随机性测试。并根据要求参数选择 $n = 1024$ 和 $n = 131072$

2.1 随机序列

- $n = 1024$ 时，测试100组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	99/100	0.419021	-
02	Block Frequency	100/100	0.946308	$M = 20$
03	Cumulative Sums	2/2	通过	-
04	Runs	100/100	0.474986	-
05	Longest Run of Ones	100/100	0.574903	-
06	Approximate Entropy	100/100	0.455937	$m = \lfloor \log_2 n \rfloor - 6 = 4$
07	Serial	2/2	通过	$m = \lfloor \log_2 n \rfloor - 3 = 7$
注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。				

- $n = 131072$ 时，测试50组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	49/50	0.779188	-
02	Block Frequency	50/50	0.213309	$M = 1350$
03	Cumulative Sums	2/2	通过	-
04	Runs	50/50	0.699313	-
05	Longest Run of Ones	49/50	0.657933	-
06	Approximate Entropy	49/50	0.350485	$m = \lfloor \log_2 n \rfloor - 7 = 10$
07	Serial	2/2	通过	$m = \lfloor \log_2 n \rfloor - 3 = 14$
08	FFT	50/50	0.350485	-
09	Nonperiodic Template Matchings	148/148	通过	$m = 9$
10	Overlapping Template Matchings	48/50	0.739918	$m = 9$
11	Linear Complexity	49/50	0.213309	-
12	Rank	50/50	0.455937	-
13	Random Excursions	8/8	通过	-
14	Random Excursions Variant	18/18	通过	-

注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。

2.2 高密序列

- $n = 1024$ 时，测试100组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	99/100	0.816537	-
02	Block Frequency	99/100	0.719747	$M = 20$
03	Cumulative Sums	2/2	通过	-
04	Runs	99/100	0.897763	-
05	Longest Run of Ones	98/100	0.191687	-
06	Approximate Entropy	100/100	0.595549	$m = \lfloor \log_2 n \rfloor - 6 = 4$
07	Serial	2/2	通过	$m = \lfloor \log_2 n \rfloor - 3 = 7$

注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。

- $n = 131072$ 时，测试50组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	46/50	0.000818	-
02	Block Frequency	49/50	0.096578	$M = 1350$
03	Cumulative Sums	0/2	未通过	-
04	Runs	43/50	0.000000 *	-
05	Longest Run of Ones	28/50	0.000000 *	-
06	Approximate Entropy	0/50	0.000000 *	$m = \lfloor \log_2 n \rfloor - 7 = 10$
07	Serial	0/2	未通过	$m = \lfloor \log_2 n \rfloor - 3 = 14$
08	FFT	50/50	0.739918	-
09	Nonperiodic Template Matchings	127/148	未通过	$m = 9$
10	Overlapping Template Matchings	49/50	0.262249	$m = 9$
11	Linear Complexity	49/50	0.494392	-
12	Random Excursions	8/8	通过	-
13	Random Excursions Variant	18/18	通过	-
14	Rank	45/50	0.000000 *	-

注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。

2.3 低密序列

- $n = 1024$ 时，测试100组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	100/100	0.080519	-
02	Block Frequency	100/100	0.334538	$M = 20$
03	Cumulative Sums	2/2	通过	-
04	Runs	99/100	0.419021	-
05	Longest Run of Ones	99/100	0.616305	-
06	Approximate Entropy	100/100	0.202268	$m = \lfloor \log_2 n \rfloor - 6 = 4$
07	Serial	2/2	通过	$m = \lfloor \log_2 n \rfloor - 3 = 7$

注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。

- $n = 131072$ 时，测试50组，结果如下：

编号	测试类型	通过率	p 值均匀性	非默认参数
01	Frequency	41/50	0.000000 *	-
02	Block Frequency	50/50	0.699313	$M = 1350$
03	Cumulative Sums	0/2	未通过	-
04	Runs	47/50	0.023545	-
05	Longest Run of Ones	49/50	0.574903	-
06	Approximate Entropy	0/50	0.000000 *	$m = \lfloor \log_2 n \rfloor - 7 = 10$
07	Serial	0/2	未通过	$m = \lfloor \log_2 n \rfloor - 3 = 14$
08	FFT	50/50	0.137282	-
09	Nonperiodic Template Matchings	130/148	未通过	$m = 9$
10	Overlapping Template Matchings	48/50	0.171867	$m = 9$
11	Linear Complexity	50/50	0.008879	-
12	Rank	34/50	0.000000 *	-

注：其他的测试均因为n太小不满足文档中的要求，得不到有意义的结论。

2.4 测试结果总结

我们可以发现，在**随机数据**的测试下，随机性测试保持**良好**；

但是在**高密和低密数据**的测试下，随机性测试表现**较差。**

3 攻击

3.1 区分攻击

在进行区分攻击时，我还没有发现摘要中提到的重要性质（前40bit相同后40bit大概率相同），所以这里姑且不考虑该性质。

我们采用参考资料[1]中提到的方法进行区分攻击。思路如下：

1. 生成10000对字符串，每个字符串的长度都在10-1000之间，且都由可见字符组成；每对字符串之间只相差1个bit。
2. 使用我们提出的哈希函数加密这10000对字符串，得到10000 * 2个长为80bits的字符串（哈希结果）。
3. 比较每对结果之间相同ascii字符的个数（也就是以1byte为单位进行比较）。用x表示一对结果有x个字符相同，用$W(x)$表示在这10000对结果中，一对结果有x个字符相同的对的个数。记录并和理想情况比较。

在理想情况（完全等效于随机的情况下），$W(x)$的理论值为

$$W(x) = 10000 \times C_{10}^{x} \times (\frac{1}{2^8})^{x} \times (\frac{1}{2^8})^{(10-x)}$$

其中10000是字符串对的数量，10为hash得到的结果的长度（byte为单位）。

我们运行后得到如下结果：

```
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3/Sharnn/attack/distinguish$ ./run_attack.sh
g++ distinguish_attack.cpp -o distinguish_attack
Finish hash
The theorical distirbution is:
{0: 9616.170416164263, 1: 377.1047222025202, 2: 6.654789215338591, 3: 0.06959256695778919, 4: 0.0004775960477495336, 5: 2.2475108129389817e-06,
6: 7.3448065782319664e-09, 7: 1.645895031536575e-11, 8: 2.420433869906728e-14, 9: 2.1093105620102206e-17, 10: 8.271806125530277e-21}
The distribution of our function is:
{0: 9584, 1: 410, 2: 6, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0}
```

我们SHA-RNN运行结果和理想情况、我们组提出的hash（RCA）进行对比

	0	1	2	3	4	5
随机序列	9616	377	7	0	0	0
SHA-RNN	9584	410	6	0	0	0
RCA	9626	368	6	0	0	0

可以发现SHA-RNN在0个相同和1个相同的分布上和理想情况还有一定差距，使用这种方法的区分攻击可以**较小程度**区分随机序列和SHA-RNN生成的序列。

如果考虑重要性质，SHA-RNN**显然区分**于随机序列，在下一部分会进行碰撞攻击，这里不再赘述。

3.2 碰撞攻击（利用重要性质的生日攻击）

在经过了大量的奇奇怪怪的实验和奇奇怪怪的“注意到”之后，我发现了SHA-RNN有一个重要性质：**SHA-RNN生成的80bit 哈希值中，若前40bit相同，后40bit大概率相同。**

对于这个性质的补充说明：我试验了很多组，但我目前并没有找到前40bit相同，而后40bit不同的反例（也就是这里的“大概率”有可能是“必定”），但我并没法实验所有组，因此这里姑且称为“大概率”。我没法证明它，因为混沌系统和RNN里可能发生很多奇奇怪怪而我搞不明白的事情，我也是在实验中发现偶然这点的。我怀疑SHA-RNN并不是一个到$\{0,1\}^{80}$上的满射。在应用中，我们暂且认为这个性质是对的。

在有重要性质的情况下，我们只需要寻找前40bit的碰撞。这是容易的(甚至不需要用TMTO或者彩虹表)，只需要运用暴力方法。我们的方法是：一个哈希表存储`(SHARNN(x)[0:40], x)`的映射。不断随机生成长度为80bit的序列x，得到其哈希值SHARNN(x)，将SHARNN(x)[0:40]与哈希表中已有内容进行比较。如果发生了碰撞，就代表（大概率）找到了一组碰撞，之后进行验证即可。算法伪代码如下：

```
hash2text = {}
while true:
    x = random_text(length = 80)    # 生成长为80bit的随机序列
    key = SHARNN(x)[0:40]           # 取其hash值前40bit
    if key in hash2text:            # 如果已经有前40bit相等的
        if SHARNN(x) == SHARNN(hash2text[key]): # 在测试中，这句话一直是true
            return x, hash2text[key]
    else:
        hash2text[SHARNN(x)] = x    # 将摘要和明文对加入到哈希表中
```

分析这种算法的复杂度：因为只需要找前40bit的碰撞，生日攻击次数的期望是1.7×2^{20}；对于具有这种性质的、生成长为n（这里n=80）的摘要的hash算法，进行这种攻击的时间和空间复杂度是$O(2^{\frac{n}{4}})$，这是比较小的开销，因此这种算法是有效的。

下面是几个破解成功的例子（还有很多组，这里不放了）：

```
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ./run_attack.sh
g++ collision_attack.cpp -o collision_attack
404e4149503e31445c33
4c4348687b3e5840454b
@NAIP>1D\3
LCHh{>X@EK
Find collision in first 40 bits
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "@NAIP>1D\3"
Hash = 29303069255a2bc9fd3a
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "LCHh{>X@EK"
Hash = 29303069255a2bc9fd3a
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
```

```
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ./run_attack.sh
g++ collision_attack.cpp -o collision_attack
526b3c3e334662624775
405b4c68416757596b3c
Rk<>3FbbGu
@[LhAgWYk<
Find collision in first 40 bits
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "Rk<>3FbbGu"
Hash = 93e71480f65ed0f99396
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "@[LhAgWYk<"
Hash = 93e71480f65ed0f99396
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
```

```
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ./run_attack.sh
g++ collision_attack.cpp -o collision_attack
5942675242344b3f7d50
6338367853677b5e3242
YBgRB4K?}P
c86xSg{^2B
Find collision in first 40 bits
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "YBgRB4K?}P"
Hash = 3712233d594ef5a94996
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
zhangxz18@LAPTOP-C31OG9K1:/mnt/d/college_2022_spring/cryptogrophy/homework3_group1/SHARNN/attack/collision$ ../../bin/main -s "c86xSg{^2B"
Hash = 3712233d594ef5a94996
Size = 136 bytes (after padding)
Time = 0 ms
Speed = inf Mbps
```

可见碰撞攻击是成功的。

4 代码说明与运行方法

```
$ git clone https://github.com/Robin-Diddle/Sharnn
# randomness_analysis目录下为随机性分析结果
# attack文件夹下为区分攻击和碰撞攻击
# 区分攻击
$ cd ./attack/distinguish
$ ./run_attack.sh
# 碰撞攻击
$ cd ./attack/collision
$ ./run_attack.sh
```

5 参考文献

[1] Alawida, Moatsum, et al. "A novel hash function based on a chaotic sponge and DNA sequence." *IEEE Access* 9 (2021): 17882-17897.