





the left-most place in the register is filled with the XOR of bits 1, 5, and 7. Consider what happens if we try to encrypt the message "01000001" and fill the register with a secret key of [0,0,1,1,0,1,1] from left to right. Then the first bit of keystream is  $f(0;1;0;1) = 0 \oplus 1 \oplus 0 \oplus 1 = 0$ , which gets XORed with the first bit of plaintext, 0, to produce the ciphertext bit of 0. The register then shifts one bit to the right and the empty spot is filled with  $0 \oplus 0 \oplus 1 = 1$ . Then the second keystream bit is generated in a similar fashion using  $f(0;0;1;1)$  and the register is updated again. This continues until the entire plaintext message is encrypted. If we continued to the end, our keystream would be [0,1,1,0,0,1,1], meaning our plaintext message of "01000001" would be encrypted as "00100110." Since XOR is its own inverse, decryption works in a similar fashion. The receiver simply generates the same stream and XORs it with the ciphertext to recover the plaintext. Real stream ciphers are larger and more complex than this example. The ciphers we dealt with, for example, had 48-bit LFSRs, multiple NLFs, and initialization protocols that set up the register before generating keystream.



Fig. 1. A simple stream cipher



$$\begin{aligned} &x[21], \\ &x[22] + 1, \\ &x[23] + 1, \\ &x[24] + 1 \end{aligned}$$

Attempts are then made to solve the polynomial system. A variety of methods exist for solving systems of polynomial equations. These include Gröbner basis algorithms (such as Faugere's  $F_4$  [9] and  $F_5$  [10] algorithms) which focus on expanding the ideal generated by the polynomials, eXtended Linearization algorithms (such as XL [3], MutantXL [7], and MXL2 [12]) which use linear algebra, and logic based SAT-solvers (such as MiniSAT [8] and CryptoMiniSAT [13]) since the work is typically done over  $\mathbb{F}_2$ . In general, finding the solutions of a polynomial system over a finite field is NP-complete. However, the polynomial systems that result from modeling ciphers are often very sparse (i.e., there are a large number of coefficients equal to zero) and highly structured, and therefore often permit successful cryptanalysis. While algebraic attacks have had limited success on other types of ciphers, stream ciphers have permitted a number of successful algebraic cryptanalysis attempts. Many of these successful attacks have used SAT-solvers as their method for solving. For a thorough introduction to algebraic cryptanalysis, see [1].

## 2.2 SAT-solvers in Cryptanalysis

SAT-solver programs try to determine whether or not a given set of boolean constraints has a solution. Conflict-driven SAT-solvers, such as MiniSat [8] (the program used for our experiments), use tree-based search algorithms with learning. MiniSat guesses the value of a variable and propagates the value throughout. If this guess causes a conflict with an earlier guess, MiniSat creates a new learned constraint, backtracks up the tree to the highest guess allowed by the new constraint, and changes the value of that guess. These learned constraints restrict the search space by trimming branches of the search tree, making it faster than brute force provided a solution exists. For an introduction to SAT-solvers, see Chapter 14 of [1].

SAT-solvers operate on constraints in their conjunctive normal form (CNF). In CNF, each variable and monomial from a polynomial system becomes a boolean variable. The disjunction (ORing) of boolean variables is called a clause, and CNF is the conjunction (ANDing) of these clauses. To use a SAT-solver on a polynomial system, the polynomials must first be converted to CNF. A variety of programs exist for this. Soos [14] even has a program, Grain of Salt, that translates stream ciphers directly to CNF. We used a Perl script written by Jeremy Erickson<sup>2</sup> to convert our polynomials into CNF. The CNF of the polynomial system from our earlier example can be found in Appendix A.

<sup>2</sup> A former NKU/UC REU participant. Now a Computer Science graduate student at the University of North Carolina.

### 3 Crypto1 and Hitag2

Crypto1 and Hitag2 are two stream ciphers that have received a lot of negative attention in recent years, as both ciphers have been successfully attacked on multiple occasions, often with algebraic cryptanalysis. Both ciphers have been used in industrial applications making their lack of security more glaring.

#### 3.1 Crypto1

Crypto1 is a proprietary stream cipher used in the MiFare Classic smart card and is manufactured by NXP (formerly Phillips) Semiconductors. The MiFare Classic smart card is used in a variety of public transportation payment systems, most notably the London Oyster card. Since MiFare Classic has been reverse engineered[11], the Crypto1 cipher has been attacked and broken numerous times. Some of these attacks used SAT-solver based algebraic cryptanalysis[4][13]. These attacks broke Crypto1 in 200 seconds [4] and 40 seconds [13] respectively. The Crypto1 cipher consists of a 48-bit LFSR and six NLFs that are combined to generate keystream. The LFSR has an update function  $f: \mathbb{F}_2^{18} \rightarrow \mathbb{F}_2$  that simply XORs 18 bits of register to produce the new bit. There are 20 bits of register that are fed into five degree-3 NLFs  $g_i: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2, i = 0; \dots; 4$ . The outputs of these functions are subsequently fed into one degree-4 NLF  $g_5: \mathbb{F}_2^5 \rightarrow \mathbb{F}_2$  that produces the keystream output (see Figure 1 of Appendix B).

The cipher is initialized using a 48-bit secret key, a 32-bit serial number, and two 32-bit initialization vectors (IVs) in the following manner. The secret key is used to fill the register. Then, for 32 clocks, the LFSR shifts one position to the left and one bit of serial is XORed with one bit from the Tag IV and the output from  $f$  to fill the new position in the register. The cipher continues with a second initialization phase for 32 more clocks. The LFSR shifts one position to the left, and one bit of Reader IV gets XORed with one bit of keystream (the output of  $g_5(g_0; \dots; g_4)$ ) and the output from  $f$ , to fill the empty space in the register. After this, initialization is complete. Then keystream generation begins. Keystream generation operates for  $n$

---

**Algorithm 1** Crypto1

---

$f$   $x_1 \dots x_{48}$  are register bits  
 $n$  is the number of plaintext bits to encrypt  
 $f$   $g_0 = g_3 = acd + bcd + ab + ac + ad + bc + cd + a + bg$   
 $f$   $g_1 = g_2 = g_4 = abc + acd + bcd + ab + ac + ad + bc + bd + cg$   
 $f$   $g_5 = abce + bcde + abd + abe + acd + ade + bde + ac + ad + ae + de + a + eg$

$f$  First initialization phase  $g$

**for**  $i$  in 1..32 **do**  
   $t := x_1 \ x_6 \ x_{10} \ x_{11} \ x_{13} \ x_{15} \ x_{16} \ x_{18} \ x_{20} \ x_{25} \ x_{26} \ x_{28} \ x_{30}$   
   $x_{36} \ x_{40} \ x_{42} \ x_{43} \ x_{44} \ \text{Serial}[i] \ \text{TagIV}[i]$   
   $(x_1; \dots; x_{48}) := (x_2; \dots; x_{48}; t)$   
**end for**

$f$  Second initialization phase  $g$

**for**  $i$  in 1..32 **do**  
   $t := x_1 \ x_6 \ x_{10} \ x_{11} \ x_{13} \ x_{15} \ x_{16} \ x_{18} \ x_{20} \ x_{25} \ x_{26} \ x_{28} \ x_{30} \ x_{36} \ x_{40}$   
   $x_{42} \ x_{43} \ x_{44} \ \text{ReaderIV}[i] \ g_5(g_0(x_{10}; x_{12}; x_{14}; x_{16}); \dots; g_4(x_{42}; x_{44}; x_{46}; x_{48}))$   
   $(x_1; \dots; x_{48}) := (x_2; \dots; x_{48}; t)$   
**end for**

$f$  Keystream generation  $g$

**for**  $i$  in 1.. $n$  **do**  
   $t := x_1 \ x_6 \ x_{10} \ x_{11} \ x_{13} \ x_{15} \ x_{16} \ x_{18} \ x_{20} \ x_{25} \ x_{26} \ x_{28} \ x_{30}$   
   $x_{36} \ x_{40} \ x_{42} \ x_{43} \ x_{44}$   
   $k_i := g_5(g_0(x_{10}; x_{12}; x_{14}; x_{16}); \dots; g_4(x_{42}; x_{44}; x_{46}; x_{48}))$   
   $(x_1; \dots; x_{48}) := (x_2; \dots; x_{48}; t)$   
**end for**

---

Hitag2 also consists of a 48-bit LFSR and six NLFs that are combined to generate keystream. The LFSR's feedback function  $f : \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$  XORs 16 register bits to produce the new register bit. There are 20 bits of register that are fed into five degree-3 NLFs  $g_i : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2; i = 0; \dots; 4$ . The outputs of these functions are subsequently fed into one degree-4 NLF  $g_5 : \mathbb{F}_2^5 \rightarrow \mathbb{F}_2$  that produces the keystream output.

Hitag2's initialization process differs slightly from Crypto1. Hitag2 is initialized using a 48-bit secret key, a 32-bit serial number, and a 32-bit IV. Its LFSR is initially filled with the 32-bit serial and the 16 lowest ranking bits of key. For 32 clocks, the LFSR shifts one position to the left, and the empty register bit is filled with the XOR of three bits: one bit of IV, one bit of key not already in the register, and one bit of output from the NLFs. Then keystream generation begins. The LFSR is updated by  $f$ , and keystream is generated using  $g_0; \dots; g_5$ . The encryption algorithm is listed in Algorithm 2, and a diagram of the cipher can be found in Figure 2 of Appendix B.

---

**Algorithm 2 Hitag2**

---

$fX_1:::X_{48}$  are register bits  
 $fn$  is the number of plaintext bits to encrypt  
 $f g_0 = g_4 = abc + ac + bc + ad + a + b + d + 1g$   
 $f g_1 = g_2 = g_3 = abd + acd + bcd + ab + ac + bc + a + b + d + 1g$   
 $f g_5 = abce + abde + acd + ade + bcd + bce + cde + ab + bc + bd + be + ce + de + b + d + 1g$

$f$ Initialization phase  
 $g$

**for**  $i$  in 1..32 **do**  
   $t := IV[i] \oplus KEY[i+16] \oplus g_5(g_0(X_2; X_3; X_5; X_6)); g_1(X_8; X_{12}; X_{14}; X_{15}); g_2(X_{17}; X_{21}; X_{23}; X_{26}); g_3(X_{28}; X_{29}; X_{31}; X_{33}); g_4(X_{34}; X_{43}; X_{44}; X_{46}))$   
   $(X_1; :::; X_{48}) := (X_2; :::; X_{48}; t)$   
**end for**

$f$ Keystream generation  
 $g$

**for**  $i$  in 1.. $n$  **do**  
   $t := X_1 \oplus X_3 \oplus X_4 \oplus X_7 \oplus X_8 \oplus X_9 \oplus X_{17} \oplus X_{23} \oplus X_{24} \oplus X_{27} \oplus X_{31} \oplus X_{42} \oplus X_{43} \oplus X_{44} \oplus X_{47} \oplus X_{48}$   
   $k_i := g_5(g_0(X_2; X_3; X_5; X_6)); g_1(X_8; X_{12}; X_{14}; X_{15}); g_2(X_{17}; X_{21}; X_{23}; X_{26}); g_3(X_{28}; X_{29}; X_{31}; X_{33}); g_4(X_{34}; X_{43}; X_{44}; X_{46}))$   
   $(X_1; :::; X_{48}) := (X_2; :::; X_{48}; t)$   
**end for**

---

## 4 Experimental Comparison of Crypto1 and Hitag2

The main goal for this portion of our research was to determine if the difference in register taps was the main factor in the security disparity between the two ciphers. Since the non-linear functions (NLFs) and register taps (the inputs of the NLFs) were the most apparent differences between the ciphers, these were the variables we concentrated on. To analyze the effect of NLFs and register taps on the stream ciphers Crypto1 and Hitag2, SAT-solver timing tests were conducted on the following systems:

- { Crypto1
- { Hitag2
- { Crypto1 with Hitag2 NLFs
- { Hitag2 with Crypto1 NLFs
- { Crypto1 with Hitag2 Taps
- { Hitag2 with Crypto1 Taps

For each system, ten instances were created and tested. Each of these instances used a different randomly generated key, IV, and serial to encrypt 56 bits of plaintext. For Hitag2 this is not a practical attack (as noted in [5] encrypting a message of this length would require additional initialization vectors), but 56 bits were chosen to ensure a unique solution for each system, and to stay consistent with the attacks in [13]. The encryptions and polynomial systems were



generated using MAGMA<sup>3</sup> [2], converted to CNF, and then solved using MiniSat 2.0<sup>4</sup>. All tests were run on Wittenberg University's WARP2 next generation computing cluster.

The SAT-solver running times (in seconds) of Crypto1, Hitag2, and hybrids are listed in Table 1. The  $\infty$  symbol indicates that the given test was killed after 14 days. Each system used the same ten seeds to generate the random keys, IVs, and serials.

**Table 1.** SAT-solving times for Crypto1, Hitag2, and Hybrid Systems with swapped NLFs(in seconds)

| Cipher  |         |            | C1 w/   | H2 w/      | C1 w/      | H2 w/   |
|---------|---------|------------|---------|------------|------------|---------|
|         | Crypto1 | Hitag2     | H2 NLFs | C1 NLFs    | H2 Taps    | C1 Taps |
| Test 1  | 26.88   |            | 516.12  | 157,367.20 | 2958.50    | 5994.41 |
| Test 2  | 127.12  | 726,967.74 | 206.65  | 210,117.45 | 63,960.98  | 1052.12 |
| Test 3  | 276.56  | 799,127.10 | 724.55  | 108,557.26 | 5860.80    | 3677.58 |
| Test 4  | 153.89  |            | 145.93  | 53,260.86  |            | 4131.03 |
| Test 5  | 1752.91 |            | 448.61  | 46,375.79  |            | 285.55  |
| Test 6  | 152.72  |            | 50.64   |            | 850,311.27 | 6092.15 |
| Test 7  | 712.28  |            | 1059.98 | 32,949.12  | 30,202.11  | 862.00  |
| Test 8  | 380.74  | 102,639.52 | 217.46  |            | 57,846.76  | 448.47  |
| Test 9  | 1340.23 |            | 362.25  | 98,052.59  | 50,286.20  | 4372.88 |
| Test 10 | 86.63   |            | 1361.33 | 101,613.74 | 314,860.44 | 1262.57 |
| Mean    | 500.90  | N/A        | 509.35  | N/A        | N/A        | 2817.91 |

Based on this data, it appears that the greatest difference between Crypto1 and Hitag2 in terms of security is the difference in register taps, as expected. Swapping NLFs had relatively small effects (increasing the average runtime by nine seconds in Crypto1 and causing five more tests to finish in less than 14 days in Hitag2), but swapping taps had a large impact. Swapping the taps caused Crypto1 to go from having an average runtime of about 8 minutes, to having 2 tests fail to finish in under fourteen days. Similarly, Hitag2 only had three tests finish in under fourteen days, but using Crypto1 taps, all 10 finished with an average runtime of about 47 minutes.

## 5 Manual and Random Tap Configurations

Based on our results with Crypto1, Hitag2, and the four hybrid systems, we discovered the choice of register taps to be an important factor in the security of

<sup>3</sup> Special thanks to the Computational Algebra Group at the University of Sydney for providing MAGMA for this project.

<sup>4</sup> Available at <http://minisat.se/MiniSat.html>

stream ciphers. Since the choice of taps is so important, we wanted to quantify how different aspects of tap configurations affected security. In order to do this, we ran tests on cryptosystems following Crypto1's protocol but with manually chosen taps.

### 5.1 Manually Configured Taps

The characteristics we chose to focus on were regularity (whether the taps followed a regular pattern), distance (how far apart the taps were), and register position (whether the taps were concentrated in the left, middle, or right portion of the register). Additional details on these characteristics can be found in Section 6.1. Tests were conducted in the same manner as those on Crypto1 and Hitag2, and were run on the following systems:

- { Left-adjusted Crypto1 | every other tap is used with the collection beginning on the left side of the register
- {

negatively correlated with security. To further explore this, we tested randomly generated tap configurations.

## 5.2 Randomly Generated Taps

We randomly generated taps for six different tap configurations (A-F) and ran tests on them in the same fashion as our tests above. These tests used Crypto1's protocol and functions and the results are shown in the Table 3.

**Table 3.** SAT-solving times for Crypto1 with randomly generated taps (in seconds)

| Taps    | Rand A     | Rand B     | Rand C     | Rand D       | Rand E     | Rand F     |
|---------|------------|------------|------------|--------------|------------|------------|
| Test 1  | 622,821.10 | 39,462.55  | 49,157.27  | 663,549.73   | 423,270.91 | 29,106.78  |
| Test 2  | 14,242.05  | 23,763.62  | 72,341.33  |              | 589,014.20 | 210,869.40 |
| Test 3  | 391,570.56 | 1964.12    | 3714.08    | 180,678.58   | 412,970.57 | 446,532.26 |
| Test 4  | 50,378.07  | 13,024.08  | 763,204.95 | 658,268.54   | 192,860.62 | 62,713.09  |
| Test 5  | 103,576.18 | 9887.10    | 12,915.87  |              | 620,232.68 | 611,247.11 |
| Test 6  | 736,915.77 | 662,668.36 | 738,421.84 | 1,028,884.23 | 258,563.13 | 197,543.79 |
| Test 7  | 162,965.86 | 5474.42    | 335,768.48 | 28,760.83    | 32,282.07  | 69,156.34  |
| Test 8  | 84,086.51  | 57,873.49  | 172,888.32 | 40,447.96    |            | 11,764.21  |
| Test 9  | 138,650.05 | 97,395.13  | 155,567.60 |              | 283,487.27 | 264,012.74 |
| Test 10 | 575,653.63 | 15,869.65  | 2569.52    |              |            | 32,328.55  |
| Mean    | 288,085.98 | 92,738.25  | 230,654.93 |              |            | 193,527.43 |

These results further strengthened our notion that distance was positively correlated with security and regularity was negatively correlated with security, as the random tap configurations had more distance and less regularity, and took significantly longer to break. To better quantify this effect, we decided to develop a security score that assigns a number to tap configurations in order to predict security.

## 6 A Security Score for Tap Configurations

In order to quantify the ideas we saw in our experimental results, we developed a score that predicts security based on tap configuration characteristics. From our experimental results it was clear that distance and regularity subscores should be part of this score. However, capturing these qualities (especially regularity) in one measure proved difficult<sup>5</sup>, so we ran a linear regression on six subscores that each help describe the distance and regularity characteristics of a tap configuration. These subscores were computed using a Java program we wrote which can be

## 6.1 Distance

For the notion of distance, we wanted a measurement of how much spread there was between register taps. To do this, we decided to measure the distance of individual taps from a baseline configuration, and sum these distances. As baselines, we chose the Right Consecutive and Left Consecutive configurations. The reason for these baselines is the close spacing of the taps (they are consecutive), and the lack of security these tap configurations offered. We chose two baseline configurations to account for differences in register position, using the minimum of the two scores calculated.

As an example of how we calculated distance, consider the Crypto1 tap configuration: 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48. Since the Left Consecutive taps are 1 through 20, and the Right Consecutive taps are 29 through 48, our distance scores are:

$$LeftDistance = j1 + 10j + j2 + 12j + j3 + 14j + \dots + j19 + 46j + j20 + 48j = 370$$

$$RightDistance = j29 + 10j + j30 + 12j + j31 + 14j + \dots + j47 + 46j + j48 + 48j = 190$$

So Crypto1's final distance score would be the minimum of these two scores, which is 190.

## 6.2 Repetition

Regularity is a concept that is easy to understand, but difficult to quantify. We wanted a measure that shows whether a configuration has a regular, evenly-spaced tap pattern or not. Our first method of doing this, repetition, was to look at the four taps used in each of the five NLFs and see how many times those taps were used together in an NLF. For each set of four taps, we counted how many times all four those taps appeared in the same function, how many times three appeared in the same function, and how many times two appeared in the same function after 48 clocks (one complete cycle through the register). Instances where all four appeared in the same function were weighted with a four, three with a three, and two with a two. We did this for all five sets of taps.

The result was a score that reflected our intentions as patterned those-29iap(thos346(re ect3(thos38(Crypto1 conp66ritha an theetitioSscor(re)-526(metho)-28(d)2525(of)-526(our)-526Jaw

having one and two spaces between taps, so the number of distinct spacings was two. The complete algorithm can be found in the `spaceCount()` method of our Java source code in Appendix D.

#### **6.4 Spacing Deviation**

Spacing deviation was another measure we used to capture the regularity of a tap con guration. In order to calculate our spacing deviation, we took the set of spacings and computed the standard deviation of that set. So evenly spaced con gurations like Crypto1 had a spacing deviation of 0, while sporadically spaced con gurations like Hitag2 (1.808) had higher spacing deviations. The complete algorithm can be found in the `spaceDev()` method of our Java source code in Appendix D.

#### **6.5 Span Deviation**

Span deviation was our last score that focused solely on regularity. We de ned a span as the length of a set of continuous taps. Our span deviation was then the standard deviation of the set of spans for each tap con guration. For example our consecutive taps had one span of 20 taps and Crypto1 had 20 spans of 1 tap, but both of their span deviations were 0. The complete algorithm can be found in the `spanDev()` method of our Java source code in Appendix D.

#### **6.6 Inclusion**

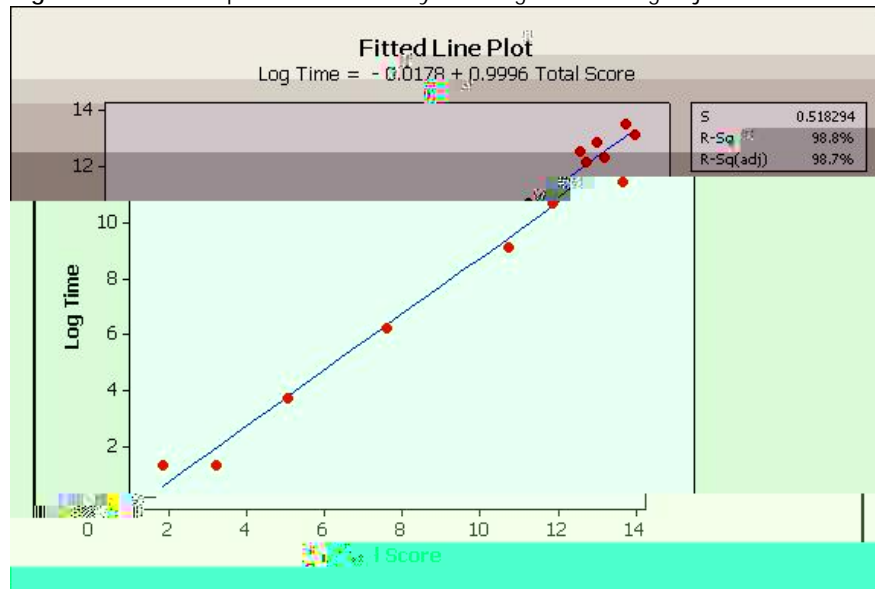
**Table 4.** Configuration subscores, final score, and log-adjusted time

|            | Distance | Regularity | Inclusion | Space | Count  | Space  | Dev | Span | Dev | Final Score | Log Time |
|------------|----------|------------|-----------|-------|--------|--------|-----|------|-----|-------------|----------|
| Right Con  | 0        | 183        | 70        | 1     | 0      | 0      |     |      |     | 0.5947      | 1.2953   |
| Crypto1    | 190      | 183        | 37        | 1     | 0      | 0      |     |      |     | 6.3534      | 6.2164   |
| 2-2 Taps   | 280      | 162        | 37        | 2     | 0.9986 | 0      |     |      |     | 9.4820      | 9.1246   |
| 1-2-1 Taps | 280      | 162        | 19        | 2     | 0.4993 | 0      |     |      |     | 10.6229     | 10.7159  |
| Hitag2     | 230      | 109        | 29        | 5     | 1.8080 | 0.4949 |     |      |     | 12.3692     | 12.8600  |
| Rand A     | 277      | 125        | 36        | 6     | 1.8018 | 2.0494 |     |      |     | 11.9428     | 12.5710  |
| Rand B     | 250      | 102        | 31        | 6     | 1.6560 | 1.4025 |     |      |     | 12.4309     | 11.4375  |
| Rand C     | 255      | 108        | 29        | 6     | 1.4643 | 1.1513 |     |      |     | 12.5731     | 12.3487  |
| Rand D     | 263      | 117        | 28        | 6     | 1.8727 | 1.1923 |     |      |     | 13.1319     | 13.5197  |
| Rand E     | 266      | 104        | 33        | 6     | 2.1353 | 0.7454 |     |      |     | 13.3546     | 13.1677  |
| Rand F     | 271      | 64         | 27        | 4     | 1.2230 | 0.7458 |     |      |     | 12.1083     | 12.1732  |
| 19-1 Taps  | 28       | 168        | 66        | 2     | 6.2523 | 9.000  |     |      |     | 3.8043      | 3.7298   |
| 10-10 Taps | 10       | 182        | 66        | 2     | 0.2233 | 0      |     |      |     | 1.9640      | 1.2953   |

The data for this regression can be found in the Table 4.

As seen in the Figure 2, our regression analysis resulted in a very good fit. Using the equation of  $LogTime = -0.0178 + 0.9996 \cdot FinalScore$ , we obtained a fit with an R-squared value of 98.8%. These six qualities have an apparent relationship with cipher security.

**Fig. 2.** A fitted line plot of our security score against the log-adjusted mean times



<sup>6</sup> For configurations where a mean was not possible (as not all instances finished), we used our cutoff time (2 weeks) as the time for those tests.



## References

1. G. Bard. *Algebraic Cryptanalysis*. Springer, New York, NY, 2009.
2. W. Bosman, J. Cannon, and C. Playhoust. *The Magma algebra system. I. The User Language*. In: *J. of Symbolic Comput.*, 24(3-4), pp. 235-265, 1997.
3. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. *Efficient Algorithms for Solving Overdetermined Systems of Multivariate Polynomial Equations*. In: EUROCRYPT 2000, LNCS 1807, pp. 392-497.
4. N.T. Courtois, K. Nohl, and S. O'Neil. *Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards*. Cryptology ePrint Archive, Report 2008/166, 2008. <http://eprint.iacr.org/>.
5. N.T. Courtois, S. O'Neil, and J.J. Quisquater. *Practical Algebraic Attacks on the Hitag2 Stream Cipher*. In: P. Samarati et al. (Eds.): ISC 2009, LNCS 5735, pp. 167-176, 2009.
6. D.K. Dalai, K.C. Gupta, and S. Maitra. *Notion of Algebraic Immunity and Its evaluation Related to Fast Algebraic Attacks*. Cryptology ePrint Archive, Report 2006/018, 2006. <http://eprint.iacr.org/>.
7. J. Ding, J. Buchmann, M.S.E. Mohamed, W.S.A Mohamed, and R.P. Weinmann. *MutantXL*. In: Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC 2008), LMIB, pp. 23-32.
8. N. Een N. Sorenson. *An extensible SAT-solver*. In: E. Giunchiglia and E. Tacchella (Eds.): SAT. LNCS 2919, pp. 502-518, 2003.
9. J.C. Faugere. *A New Efficient Algorithm for Computing Grobner Bases ( $F_4$ )*. In: *Journal of Pure and Applied Algebra*, 139, pp. 61-88, 1999.
10. J.C. Faugere. *A New Efficient Algorithm for Computing Grobner Bases Without Reduction to Zero ( $F_5$ )*. In: Proceedings of ISSAC, pp. 75-83, 2002.
11. F.D. Garcia, G. de Koning Gans, R. Muijrrers, P. van Rossum, R. Verdult, R.W. Schreur, and B. Jacobs. *Dismantling MIFARE Classic*. In: S. Jajodia, and J. Lopez (Eds.): ESORICS 2008, LNCS 5283, pp. 97-114, 2008.
12. M.S.E. Mohamed, W.S.A.E. Mohamed, J. Ding, and J. Buchmann. *MXL2: Solving Polynomial Equations over  $GF(2)$  Using an Improved Mutant Strategy*. In: J. Buchmann and J. Ding (Eds.): PQCrypto 2008, LNCS 5299, pp. 203-215, 2008.
13. M. Soos, K. Nohl, and C. Castelluccia. *Extending SAT Solvers to Cryptographic Problems*. In: Oliver Kullmann (Ed.): LNCS 5584, pp. 244-257, 2009.
14. M. Soos. *Grain of Salt | An Automated Way to Test Stream Ciphers through SAT Solvers*. (2010). <http://planete.inrialpes.fr/~soos/GrainOfSalt/index.php>



## A CNF of our Simple Cipher Example

There are 73 boolean variables and 502 clauses. Each number is a boolean variable (with '-' indicating the negation of that variable) and each line is a clause. Each variable in a clause is ORed together and each clause is ANDed together. The variables 1 through 24 correspond to  $x[1]$  through  $x[24]$ .

```
p cnf 73 502
25 0
1 -26 0 // 26 corresponds the monomial  $x[1]*x[3]*x[5]$ 
3 -26 0 // 3 is True ( $x[3] = 1$ ) or 26 is False (the monomial = 0)
5 -26 0
26 -1 -3 -5 0
3 -27 0
5 -27 0
6 -27 0
27 -3 -5 -6 0
1 -28 0
3 -28 0
28 -1 -3 0
3 -29 0
5 -29 0
29 -3 -5 0
5 -30 0
6 -30 0
30 -5 -6 0
-26 27 28 29 30 31 0
-26 -27 -28 29 30 31 0
-26 -27 -28 -29 -30 31 0
-26 -27 -28 -29 30 -31 0
-26 -27 -28 29 -30 -31 0
-26 -27 28 -29 30 31 0
-26 -27 28 -29 -30 -31 0
-26 -27 28 29 -30 31 0
-26 -27 28 29 30 -31 0
-26 27 -28 -29 30 31 0
-26 27 -28 -29 -30 -31 0
-26 27 -28 29 -30 31 0
-26 27 -28 29 30 -31 0
-26 27 28 -29 -30 31 0
-26 27 28 -29 30 -31 0
-26 27 28 29 -30 -31 0
26 -27 28 29 30 31 0
26 -27 -28 -29 30 31 0
26 -27 -28 -29 -30 -31 0
26 -27 -28 29 -30 31 0
```

26 -27 -28 29 30 -31 0  
26 -27 28 -29 -30 31 0  
26 -27 28 -29 30 -31 0  
26 -27 28 29 -30 -31 0  
26 27 -28 29 30 31 0  
26 27 -28 -29 -30 31 0  
26 27 -28 -29 30 -31 0  
26 27 -28 29 -30 -31 0  
26 27 28 -29 30 31 0  
26 27 28 -29 -30 -31 0  
26 27 28 29 -30 31 0  
26 27 28 29 30 -31 0  
-31 6 17 0  
-31 -6 -17 0  
31 -6 17 0  
31 6 -17 0  
-1 3 7 8 0  
-1 -3 -7 8 0  
-1 -3 7 -8 0  
-1 3 -7 -8 0  
1 -3 7 8 0  
1 -3 -7 -8 0  
1 3 -7 8 0  
1 3 7 -8 0  
2 -32 0  
4 -32 0  
6 -32 0  
32 -2 -4 -6 0  
4 -33 0  
6 -33 0  
7 -33 0  
33 -4 -6 -7 0  
2 -34 0  
4 -34 0  
34 -2 -4 0  
4 -35 0  
6 -35 0  
35 -4 -6 0  
6 -36 0  
7 -36 0  
36 -6 -7 0  
-32 33 34 35 36 37 0  
-32 -33 -34 35 36 37 0  
-32 -33 -34 -35 -36 37 0  
-32 -33 -34 -35 36 -37 0

-32 -33 -34 35 -36 -37 0  
-32 -33 34 -35 36 37 0  
-32 -33 34 -35 -36 -37 0  
-32 -33 34 35 -36 37 0  
-32 -33 34 35 36 -37 0  
-32 33 -34 -35 36 37 0  
-32 33 -34 -35 -36 -37 0  
-32 33 -34 35 -36 37 0  
-32 33 -34 35 36 -37 0  
-32 33 34 -35 -36 37 0  
-32 33 34 -35 36 -37 0  
-32 33 34 35 -36 -37 0  
32 -33 34 35 36 37 0  
32 -33 -34 -35 36 37 0  
32 -33 -34 -35 -36 -37 0  
32 -33 -34 35 -36 37 0  
32 -33 -34 35 36 -37 0  
32 -33 34 -35 -36 37 0  
32 -33 34 -35 36 -37 0  
32 -33 34 35 -36 -37 0  
32 33 -34 35 36 37 0  
32 33 -34 -35 -36 37 0  
32 33 -34 -35 36 -37 0  
32 33 -34 35 -36 -37 0  
32 33 34 -35 36 37 0  
32 33 34 -35 -36 -37 0  
32 33 34 35 -36 37 0  
32 33 34 35 36 -37 0  
-37 7 18 0  
-37 -7 -18 0  
37 -7 18 0  
37 7 -18 0  
-2 4 8 9 0  
-2 -4 -8 9 0  
-2 -4 8 -9 0  
-2 4 -8 -9 0  
2 -4 8 9 0  
2 -4 -8 -9 0  
2 4 -8 9 0  
2 4 8 -9 0  
3 -38 0  
5 -38 0  
7 -38 0  
38 -3 -5 -7 0  
5 -39 0

7 -39 0  
8 -39 0  
39 -5 -7 -8 0  
3 -40 0  
5 -40 0  
40 -3 -5 0  
5 -41 0  
7 -41 0  
41 -5 -7 0  
7 -42 0  
8 -42 0  
42 -7 -8 0  
-38 39 40 41 42 43 0  
-38 -39 -40 41 42 43 0  
-38 -39 -40 -41 -42 43 0  
-38 -39 -40 -41 42 -43 0  
-38 -39 -40 41 -42 -43 0  
-38 -39 40 -41 42 43 0  
-38 -39 40 -41 -42 -43 0  
-38 -39 40 41 -42 43 0  
-38 -39 40 41 42 -43 0  
-38 39 -40 -41 42 43 0  
-38 39 -40 -41 -42 -43 0  
-38 39 -40 41 -42 43 0  
-38 39 -40 41 42 -43 0  
-38 39 40 -41 -42 43 0  
-38 39 40 -41 42 -43 0  
-38 39 40 41 -42 -43 0  
38 -39 40 41 42 43 0  
38 -39 -40 -41 42 43 0  
38 -39 -40 -41 -42 -43 0  
38 -39 -40 41 -42 43 0  
38 -39 -40 41 42 -43 0  
38 -39 40 -41 -42 43 0  
38 -39 40 -41 42 -43 0  
38 -39 40 41 -42 -43 0  
38 39 -40 41 42 43 0  
38 39 -40 -41 -42 43 0  
38 39 -40 -41 42 -43 0  
38 39 -40 41 -42 -43 0  
38 39 40 -41 42 43 0  
38 39 40 -41 -42 -43 0  
38 39 40 41 -42 43 0  
38 39 40 41 42 -43 0  
-43 8 19 0

-43 -8 -19 0  
43 -8 19 0  
43 8 -19 0  
-3 5 9 10 0  
-3 -5 -9 10 0  
-3 -5 9 -10 0  
-3 5 -9 -10 0  
3 -5 9 10 0  
3 -5 -9 -10 0  
3 5 -9 10 0  
3 5 9 -10 0  
4 -44 0  
6 -44 0  
8 -44 0  
44 -4 -6 -8 0  
6 -45 0  
8 -45 0  
9 -45 0  
45 -6 -8 -9 0  
4 -46 0  
6 -46 0  
46 -4 -6 0  
6 -47 0  
8 -47 0  
47 -6 -8 0  
8 -48 0  
9 -48 0  
48 -8 -9 0  
-44 45 46 47 48 49 0  
-44 -45 -46 47 48 49 0  
-44 -45 -46 -47 -48 49 0  
-44 -45 -46 -47 48 -49 0  
-44 -45 -46 47 -48 -49 0  
-44 -45 46 -47 48 49 0  
-44 -45 46 -47 -48 -49 0  
-44 -45 46 47 -48 49 0  
-44 -45 46 47 48 -49 0  
-44 45 -46 -47 48 49 0  
-44 45 -46 -47 -48 -49 0  
-44 45 -46 47 -48 49 0  
-44 45 -46 47 48 -49 0  
-44 45 46 -47 -48 49 0  
-44 45 46 -47 48 -49 0  
-44 45 46 47 -48 -49 0  
44 -45 46 47 48 49 0

44 -45 -46 -47 48 49 0  
44 -45 -46 -47 -48 -49 0  
44 -45 -46 47 -48 49 0  
44 -45 -46 47 48 -49 0  
44 -45 46 -47 -48 49 0  
44 -45 46 -47 48 -49 0  
44 -45 46 47 -48 -49 0  
44 45 -46 47 48 49 0  
44 45 -46 -47 -48 49 0  
44 45 -46 -47 48 -49 0  
44 45 -46 47 -48 -49 0  
44 45 46 -47 48 49 0  
44 45 46 -47 -48 -49 0  
44 45 46 47 -48 49 0  
44 45 46 47 48 -49 0  
-49 9 20 0  
-49 -9 -20 0  
49 -9 20 0  
49 9 -20 0  
-4 6 10 11 0  
-4 -6 -10 11 0  
-4 -6 10 -11 0  
-4 6 -10 -11 0  
4 -6 10 11 0  
4 -6 -10 -11 0  
4 6 -10 11 0  
4 6 10 -11 0  
5 -50 0  
7 -50 0  
9 -50 0  
50 -5 -7 -9 0  
7 -51 0  
9 -51 0  
10 -51 0  
51 -7 -9 -10 0  
5 -52 0  
7 -52 0  
52 -5 -7 0  
7 -53 0  
9 -53 0  
53 -7 -9 0  
9 -54 0  
10 -54 0  
54 -9 -10 0  
-50 51 52 53 54 55 0

-50 -51 -52 53 54 55 0  
-50 -51 -52 -53 -54 55 0  
-50 -51 -52 -53 54 -55 0  
-50 -51 -52 53 -54 -55 0  
-50 -51 52 -53 54 55 0  
-50 -51 52 -53 -54 -55 0  
-50 -51 52 53 -54 55 0  
-50 -51 52 53 54 -55 0  
-50 51 -52 -53 54 55 0  
-50 51 -52 -53 -54 -55 0  
-50 51 -52 53 -54 55 0  
-50 51 -52 53 54 -55 0  
-50 51 52 -53 -54 55 0  
-50 51 52 -53 54 -55 0  
-50 51 52 53 -54 -55 0  
50 -51 52 53 54 55 0  
50 -51 -52 -53 54 55 0  
50 -51 -52 -53 -54 -55 0  
50 -51 -52 53 -54 55 0  
50 -51 -52 53 54 -55 0  
50 -51 52 -53 -54 55 0  
50 -51 52 -53 54 -55 0  
50 -51 52 53 -54 -55 0  
50 51 -52 53 54 55 0  
50 51 -52 -53 -54 55 0  
50 51 -52 -53 54 -55 0  
50 51 -52 53 -54 -55 0  
50 51 52 -53 54 55 0  
50 51 52 -53 -54 -55 0  
50 51 52 53 -54 55 0  
50 51 52 53 54 -55 0  
-55 10 21 0  
-55 -10 -21 0  
55 -10 21 0  
55 10 -21 0  
-5 7 11 12 0  
-5 -7 -11 12 0  
-5 -7 11 -12 0  
-5 7 -11 -12 0  
5 -7 11 12 0  
5 -7 -11 -12 0  
5 7 -11 12 0  
5 7 11 -12 0  
6 -56 0  
8 -56 0

10 -56 0  
56 -6 -8 -10 0  
8 -57 0  
10 -57 0  
11 -57 0  
57 -8 -10 -11 0  
6 -58 0  
8 -58 0  
58 -6 -8 0  
8 -59 0  
10 -59 0  
59 -8 -10 0  
10 -60 0  
11 -60 0  
60 -10 -11 0  
-56 57 58 59 60 61 0  
-56 -57 -58 59 60 61 0  
-56 -57 -58 -59 -60 61 0  
-56 -57 -58 -59 60 -61 0  
-56 -57 -58 59 -60 -61 0  
-56 -57 58 -59 60 61 0  
-56 -57 58 -59 -60 -61 0  
-56 -57 58 59 -60 61 0  
-56 -57 58 59 60 -61 0  
-56 57 -58 -59 60 61 0  
-56 57 -58 -59 -60 -61 0  
-56 57 -58 59 -60 61 0  
-56 57 58 -59 -60 61 0  
-56 57 58 -59 60 -61 0  
-56 57 58 59 -60 -61 0  
56 -57 58 59 60 61 0  
56 -57 -58 -59 60 61 0  
56 -57 -58 -59 -60 -61 0  
56 -57 -58 59 -60 61 0  
56 -57 -58 59 60 -61 0  
56 -57 58 -59 -60 61 0  
56 -57 58 -59 60 -61 0  
56 -57 58 59 -60 -61 0  
56 57 -58 59 60 61 0  
56 57 -58 -59 -60 61 0  
56 57 -58 -59 60 -61 0  
56 57 -58 59 -60 -61 0  
56 57 58 -59 60 61 0  
56 57 58 -59 -60 -61 0



56 57 58 59 -60 61 0  
56 57 58 59 60 -61 0  
-61 11 22 0  
-61 -11 -22 0  
61 -11 22 0  
61 11 -22 0  
-6 8 12 13 0  
-6 -8 -12 13 0  
-6 -8 12 -13 0  
-6 8 -12 -13 0  
6 -8 12 13 0  
6 -8 -12 -13 0  
6 8 -12 13 0  
6 8 12 -13 0  
7 -62 0  
9 -62 0  
11 -62 0  
62 -7 -9 -11 0  
9 -63 0  
11 -63 0  
12 -63 0  
63 -9 -11 -12 0  
7 -64 0  
9 -64 0  
64 -7 -9 0  
9 -65 0  
11 -65 0  
65 -9 -11 0  
11 -66 0  
12 -66 0  
66 -11 -12 0  
-62 63 64 65 66 67 0  
-62 -63 -64 65 66 67 0  
-62 -63 -64 -65 -66 67 0  
-62 -63 -64 -65 66 -67 0  
-62 -63 -64 65 -66 -67 0  
-62 -63 64 -65 66 67 0  
-62 -63 64 -65 -66 -67 0  
-62 -63 64 65 -66 67 0  
-62 -63 64 65 66 -67 0  
-62 63 -64 -65 66 67 0  
-62 63 -64 -65 -66 -67 0  
-62 63 -64 65 -66 67 0  
-62 63 -64 65 66 -67 0  
-62 63 64 -65 -66 67 0

-62 63 64 -65 66 -67 0  
-62 63 64 65 -66 -67 0  
62 -63 64 65 66 67 0  
62 -63 -64 -65 66 67 0  
62 -63 -64 -65 -66 -67 0  
62 -63 -64 65 -66 67 0  
62 -63 -64 65 66 -67 0  
62 -63 64 -65 -66 67 0  
62 -63 64 -65 66 -67 0  
62 -63 64 65 -66 -67 0  
62 63 -64 65 66 67 0  
62 63 -64 -65 -66 67 0  
62 63 -64 -65 66 -67 0  
62 63 -64 65 -66 -67 0  
62 63 64 -65 66 67 0  
62 63 64 -65 -66 -67 0  
62 63 64 65 -66 67 0  
62 63 64 65 66 -67 0  
-67 12 23 0  
-67 -12 -23 0  
67 -12 23 0  
67 12 -23 0  
-7 9 13 14 0  
-7 -9 -13 14 0  
-7 -9 13 -14 0  
-7 9 -13 -14 0  
7 -9 13 14 0  
7 -9 -13 -14 0  
7 9 -13 14 0  
7 9 13 -14 0  
8 -68 0  
10 -68 0  
12 -68 0  
68 -8 -10 -12 0  
10 -69 0  
12 -69 0  
13 -69 0  
69 -10 -12 -13 0  
8 -70 0  
10 -70 0  
70 -8 -10 0  
10 -71 0  
12 -71 0  
71 -10 -12 0  
12 -72 0

13 -72 0  
72 -12 -13 0  
-68 69 70 71 72 73 0  
-68 -69 -70 71 72 73 0  
-68 -69 -70 -71 -72 73 0  
-68 -69 -70 -71 72 -73 0  
-68 -69 -70 71 -72 -73 0  
-68 -69 70 -71 72 73 0  
-68 -69 70 -71 -72 -73 0  
-68 -69 70 71 -72 73 0  
-68 -69 70 71 72 -73 0  
-68 69 -70 -71 72 73 0  
-68 69 -70 -71 -72 -73 0  
-68 69 -70 71 -72 73 0  
-68 69 -70 71 72 -73 0  
-68 69 70 -71 -72 73 0  
-68 69 70 -71 72 -73 0  
-68 69 70 71 -72 -73 0  
68 -69 70 71 72 73 0  
68 -69 -70 -71 72 73 0  
68 -69 -70 -71 -72 -73 0  
68 -69 -70 71 -72 73 0  
68 -69 -70 71 72 -73 0  
68 -69 70 -71 -72 73 0  
68 -69 70 -71 72 -73 0  
68 -69 70 71 -72 -73 0  
68 69 -70 71 72 73 0  
68 69 -70 -71 -72 73 0  
68 69 -70 -71 72 -73 0  
68 69 -70 71 -72 -73 0  
68 69 70 -71 72 73 0  
68 69 70 -71 -72 -73 0  
68 69 70 71 -72 73 0  
68 69 70 71 72 -73 0  
-73 13 24 0  
-73 -13 -24 0  
73 -13 24 0  
73 13 -24 0  
-8 10 14 15 0  
-8 -10 -14 15 0  
-8 -10 14 -15 0  
-8 10 -14 -15 0  
8 -10 14 15 0  
8 -10 -14 -15 0  
8 10 -14 15 0

8 10 14 -15 0  
-17 0  
-18 25 0  
18 -25 0  
-19 25 0  
19 -25 0  
-20 0  
-21 0  
-22 25 0  
22 -25 0  
-23 25 0  
23 -25 0  
-24 25 0  
24 -25 0

## B Crypto1 and Hitag2 Diagrams



Fig. 3. The stream cipher Crypto1



Fig. 4. The stream cipher Hitag2

## C Tap Configurations

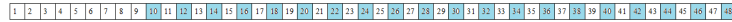


Fig. 5. Crypto1 Taps



Fig. 6. Hitag2 Taps



Fig. 7. \Left" Taps



Fig. 8. \Mid" Taps



Fig. 9. Left Consecutive Taps



Fig. 10. Mid Consecutive Taps



Fig. 11. Right Consecutive Taps



Fig. 12. \2-2" Taps



Fig. 13. \1-2-1" Taps

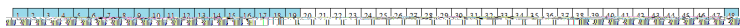


Fig. 14. \19-1" Taps

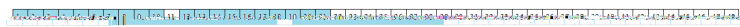


Fig. 15. \10-10" Taps



Fig. 16. Random A Taps



Fig. 17. Random B Taps

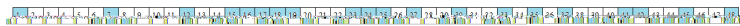


Fig. 18. Random C Taps

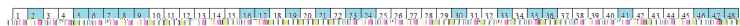


Fig. 19. Random D Taps



Fig. 20. Random E Taps

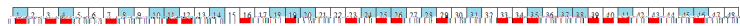


Fig. 21. Random F Taps



## D Java Program Source Code

```
import java.util.Random;
import java.util.Scanner;
import java.util.ArrayList;

public class TapConfig{

    // Instance variables
    private ArrayList<Integer> taps;
    private int distScore, diffScore, repScore, distScore1, distScore2;
    private String configName;

    // Static variables
    private static final Integer[] DIST_BASIS1 = {29, 30, 31, 32, 33, 34,
        35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48};
    private static final Integer[] DIST_BASIS2 = {1, 2, 3, 4, 5, 6, 7, 8,
        9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
    private static final Integer[] DIST_BASIS3 = {14, 15, 16, 17, 18, 19,
        20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33};

    private static int numTapConfigs = 0;
    private static double sumOfTotalScores = 0;

    // Tap Configurations
    public static final Integer[] RIGHT_CON = {29, 30, 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48};
    public static final Integer[] CRYPTO1 = {10, 12, 14, 16, 18, 20, 22,
        24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48};
    public static final Integer[] HITAG2 = {2, 3, 5, 6, 8, 12, 14, 15, 17,
        21, 23, 26, 28, 29, 31, 33, 34, 43, 44, 46};
    public static final Integer[] TWO_TWO = {6, 7, 10, 11, 14, 15, 18, 19,
        22, 23, 26, 27, 30, 31, 34, 35, 38, 39, 42, 43};
    public static final Integer[] ONE_TWO_ONE = {1, 3, 6, 8, 11, 13, 16,
        18, 21, 23, 26, 28, 31, 33, 36, 38, 41, 43, 46, 48};
    public static final Integer[] RAND_A = {1, 2, 5, 7, 8, 11, 14, 17, 22,
        24, 25, 33, 37, 38, 39, 40, 41, 42, 43, 44};
    public static final Integer[] RAND_B = {2, 4, 7, 14, 18, 19, 20, 23, 25,
        26, 27, 28, 29, 34, 36, 39, 40, 41, 42, 46};
    public static final Integer[] RAND_C = {1, 7, 12, 15, 16, 17, 18, 19,
        23, 24, 25, 27, 30, 33, 35, 37, 41, 42, 45, 48};
    public static final Integer[] RAND_D = {2, 5, 6, 7, 8, 9, 16, 17, 19,
        21, 23, 24, 30, 33, 35, 36, 41, 46, 47, 48};
    public static final Integer[] RAND_E = {1, 3, 5, 6, 7, 11, 15, 17, 18,
        23, 26, 36, 37, 38, 40, 41, 43, 44, 46, 47};
    public static final Integer[] RAND_F = {1, 4, 8, 10, 11, 14, 18, 19, 20,
        24, 25, 26, 28, 31, 35, 37, 38, 41, 45, 46};
    public static final Integer[] NINETEEN_ONE = {1, 2, 3, 4, 5, 6, 7, 8, 9,
```

```

        10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 48};
public static final Integer[] TEN_TEN = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
        12, 13, 14, 15, 16, 17, 18, 19, 20, 21};

// Constructor
public TapConfig(String name, Integer[] tapas){
    this.tapas = new ArrayList<Integer>(20);
    for(int i = 0; i < 20; i++){
        this.tapas.add(i, tapas[i]);
    }
    this.configName = name;
    numTapConfigs++;
    sumOfTotalScores = sumOfTotalScores + totalScore();
}

// Static method
public static Double getAverageScore(){
    return TapConfig.sumOfTotalScores / TapConfig.numTapConfigs;
}

// Random Tap Generator
public static Integer[] randTaps(){
    ArrayList<Integer> ret = new ArrayList(20);
    Integer[] ret2 = new Integer[20];
    Random rand = new Random();
    Integer entry;

    int i = 0;
    while(i < 20){
        entry = rand.nextInt(47);
        if(! ret.contains(entry+1)){
            ret.add(i, entry+1);
            i++;
        }
    }

    for(int j = 0; j < 20; j++){
        ret2[j] = ret.get(j);
    }
    Sort2.insertionSort(ret2);
    return ret2;
}

/** Tap Configuration Subscores */

```

```

public int getDistanceScore(){
    distScore1 = 0;
    distScore2 = 0;

    for(int i = 0; i < 20; i++){
        distScore1 = distScore1 + (DIST_BASIS1[i] - taps.get(i));
    }
    for(int i = 0; i < 20; i++){
        distScore2 = distScore2 + (taps.get(i) - DIST_BASIS2[i]);
    }

    int minimum = Math.min(distScore1, distScore2);

    return minimum;
}

public Double getDistStanDev(){
    Double distAvg = getDistanceScore()/20.0;
    Double distVar = 0.0;
    int x;
    if(distScore1 < distScore2){
        for(int i = 0; i < 20; i++){
            x = (DIST_BASIS1[i]-taps.get(i));
            distVar = distVar + .05*Math.pow((distAvg-x), 2.0);
        }
    }
    else{
        for(int j = 0; j < 20; j++){
            x = (taps.get(j) - DIST_BASIS2[j]);
            distVar = distVar + .05*Math.pow((distAvg-x), 2.0);
        }
    }
    return Math.sqrt(distVar);
}

public int getDiffusionScore(){
    diffScore = 0;
    for(int i = 1; i < 5; i++){
        for(int j=19; j >=0; j--){
            if(taps.contains(taps.get(j)-i)){
                diffScore++;
            }
        }
    }
    return diffScore;
}

public int getRepetitionScore(){

```

```

ArrayList<Integer> f1, f2, f3, f4, f5;

f1 = new ArrayList<Integer>(4);
f2 = new ArrayList<Integer>(4);
f3 = new ArrayList<Integer>(4);
f4 = new ArrayList<Integer>(4);
f5 = new ArrayList<Integer>(4);

for(int i = 0; i < 4; i++){
    f1.add(taps.get(i));
    f2.add(taps.get(i+4));
    f3.add(taps.get(i+8));
    f4.add(taps.get(i+12));
    f5.add(taps.get(i+16));
}

int f5Score = 0;

for(int j = 0; j < 48; j++){
    int f1Num = 0;
    int f2Num = 0;
    int f3Num = 0;
    int f4Num = 0;
    int f5Num = 0;

    for(int k = 0; k < 4; k++){
        if(f5.contains(f5.get(k)-j)){
            f5Num++;
        }
        else if(f4.contains(f5.get(k)-j)){
            f4Num++;
        }
        else if(f3.contains(f5.get(k)-j)){
            f3Num++;
        }
        else if(f2.contains(f5.get(k)-j)){
            f2Num++;
        }
        else if(f1.contains(f5.get(k)-j)){
            f1Num++;
        }
    }

    if(f1Num == 1){
        f1Num = 0;
    }
    if(f2Num == 1){
        f2Num = 0;
    }
    if(f3Num == 1){

```

```

        f3Num = 0;
    }
    if(f4Num == 1){
        f4Num = 0;
    }
    if(f5Num ==1){
        f4Num = 0;
    }

    f5Score = f5Score + f1Num + f2Num + f3Num + f4Num + f5Num;
}

int f4Score = 0;

for(int j = 0; j < 44; j++){
    int f1Num = 0;
    int f2Num = 0;
    int f3Num = 0;
    int f4Num = 0;

    for(int k = 0; k < 4; k++){
        if(f4.contains(f4.get(k)-j)){
            f4Num++;
        }
        else if(f3.contains(f4.get(k)-j)){
            f3Num++;
        }
        else if(f2.contains(f4.get(k)-j)){
            f2Num++;
        }
        else if(f1.contains(f4.get(k)-j)){
            f1Num++;
        }
    }

    if(f1Num == 1){
        f1Num = 0;
    }
    if(f2Num == 1){
        f2Num = 0;
    }
    if(f3Num == 1){
        f3Num = 0;
    }
    if(f4Num == 1){
        f4Num = 0;
    }

    f4Score = f4Score + f1Num + f2Num + f3Num + f4Num;
}

```

```

int f3Score = 0;

for(int j = 0; j < 40; j++){
    int f1Num = 0;
    int f2Num = 0;
    int f3Num = 0;

    for(int k = 0; k < 4; k++){
        if(f3.contains(f3.get(k)-j)){
            f3Num++;
        }
        else if(f2.contains(f3.get(k)-j)){
            f2Num++;
        }
        else if(f1.contains(f3.get(k)-j)){
            f1Num++;
        }
    }

    if(f1Num == 1){
        f1Num = 0;
    }
    if(f2Num == 1){
        f2Num = 0;
    }
    if(f3Num == 1){
        f3Num = 0;
    }

    f3Score = f3Score + f1Num + f2Num + f3Num;
}

int f2Score = 0;

for(int j = 0; j < 36; j++){
    int f1Num = 0;
    int f2Num = 0;

    for(int k = 0; k < 4; k++){
        if(f2.contains(f2.get(k)-j)){
            f2Num++;
        }
        else if(f1.contains(f2.get(k)-j)){
            f1Num++;
        }
    }

    if(f1Num == 1){

```

```

        f1Num = 0;
    }
    if(f2Num == 1){
        f2Num = 0;
    }

    f2Score = f2Score + f1Num + f2Num;
}

int f1Score = 0;

for(int j = 0; j < 36; j++){
    int f1Num = 0;

    for(int k = 0; k < 4; k++){
        if(f1.contains(f1.get(k)-j)){
            f1Num++;
        }
    }

    if(f1Num == 1){
        f1Num = 0;
    }

    f1Score = f1Score + f1Num;
}
return f5Score+f4Score+f3Score+f2Score+f1Score;
}

public Double spaceCount(){
    Integer[] xArray =
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    Double xNum = 0.0;
    for(int i = 0; i < 19; i++){
        xArray[taps.get(i+1)-taps.get(i)-1] =
            xArray[taps.get(i+1)-taps.get(i)-1] + 1;
    }
    for(Integer j: xArray){
        if(j != 0){
            xNum = xNum + 1.0;
        }
    }
    return xNum;
}

public Double spaceDev(){
    int sum = 0;
    Double num = 0.0;
    for(int i = 0; i < 19; i++){

```

```

        int x = (taps.get(i+1)-taps.get(i));
        if(x != 0){
            sum = sum + x;
            num = num + 1.0;
        }
    }
    Double avg = sum/num;
    Double variance = 0.0;
    for(int i = 0; i < 19; i++){
        int x = (taps.get(i+1)-taps.get(i));
        if(x != 0){
            variance = variance+(1.0/num)*Math.pow((x-avg), 2.0);
        }
    }
    return Math.sqrt(variance);
}

public Double spanDev(){
    boolean connected = true;
    Integer[] spans= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    int i = 0;
    int j = 0;
    int run = 1;
    while((i < 19)){
        if((taps.get(i+1) - taps.get(i)) == 1){
            run++;
            i++;
        }
        else{
            spans[j] = run;
            run = 1;
            i++;
            j++;
        }
    }
    spans[j] = run;
    int sum = 0;
    Double num = 0.0;
    for(Integer span : spans){
        if(span != 0){
            sum = sum + span;
            num = num + 1.0;
        }
    }
    Double avg = sum/num;
    Double variance = 0.0;
    for(Integer k: spans){
        if(k != 0){

```



```

        variance = variance+(1.0/num)*Math.pow((k - avg),2.0);
    }
}

return Math.sqrt(variance);
}

public Double totalScore(){
    return 8.05 + (.014 * this.getDistanceScore())
        - (.0939 * this.getDiffusionScore())
        - (.0081 * this.getRepetitionScore())
        + (.6 * spaceCount() + 1.1 * spaceDev()
        -(.573 * spanDev()));
}

/** Display Methods**/

public String toString(){
    String str;

    str = "===== \nName: " + configName
        + "\nDistance Score: " + this.getDistanceScore()
        + "\nDiffusion Score: " + this.getDiffusionScore()
        + "\nRepetition Score: " + this.getRepetitionScore()
        + "\nSpacing Count Standard Dev: " + spaceCount()
        + "\nSpace Dev: " + spaceDev() + "\nSpan Standard Dev: "
        + spanDev() + "\nDist Stand Dev: " + getDistStanDev()
        + "\nTotal Score: " + this.totalScore()
        + "\n===== \n";

    return str;
}

public String toString2(){
    String str;

    str = "===== \n[ ";

    for(int i = 0; i < 19; i++){
        str = str + this.taps.get(i) + ", ";
    }

    str = str + this.taps.get(19) + " ]";

    str = str + "\nTotal Score: " + this.totalScore()
        + "\n===== \n";

    return str;
}

```

```

}

/** The main test methods**/

// Random Tests

public static void runRandSample(){
    Scanner in;
    TapConfig rand;
    int n;
    int x;
    double maxVal = 0;
    TapConfig maxTaps = new TapConfig("Maximum", TapConfig.randTaps());
    ArrayList<TapConfig> maxes = new ArrayList<TapConfig>();

    in = new Scanner(System.in);
    System.out.print("Enter the number of random
        tap configurations you want to test: ");
    n = in.nextInt();
    System.out.println();
    System.out.print("Enter 1 for keeping track, 2 for no: ");
    x = in.nextInt();

    for(int i = 0; i < n; i++){
        String name = "Random " + i;
        rand = new TapConfig(name, TapConfig.randTaps());

        if(rand.totalScore() > maxVal){
            maxTaps = rand;
            maxVal = rand.totalScore();
        }
        if(((i % 100) == 0) && (x == 1)){
            maxes.add(maxTaps);
        }
        if((x == 2) && ((i % 1000) == 0)){
            System.out.println((i/((double)n))*100 + " %");
        }
    }

    for(TapConfig m : maxes){
        System.out.println(m.totalScore());
    }

    System.out.println( maxTaps.toString2());
}

// Our basic configs

```

```
public static void standardScores(){
    TapConfig rightCon = new TapConfig("Right Con", TapConfig.RIGHT_CON);
    TapConfig crypto1 = new TapConfig("Crypto1", TapConfig.CRYPTO1);
    TapConfig hi tag2 = new TapConfig("Hi tag2", TapConfig.HI_TAG2);
    TapConfig oneTwoOne = new TapConfig("1-2-1 Taps", TapConfig.ONE_TWO_ONE);
    TapConfig twoTwo = new TapConfig("2-2 Taps", TapConfig.TWO_TWO);
    TapConfig randA = new TapConfig("Random A", TapConfig.RAND_A);
    TapConfig randB = new TapConfig("Random B", TapConfig.RAND_B);
    TapConfig randC = new TapConfig("Random C", TapConfig.RAND_C);
    TapConfig randD = new TapConfig("Random D", TapConfig.RAND_D);
    TapConfig randE = new TapConfig("Random E", TapConfig.RAND_E);
    TapConfig randF = new TapConfig("Random F", TapConfig.RAND_F);
    TapConfig ni neteenOne = new TapConfig("19-1 Taps", TapConfig.NI_NETEEN_ONE);
    TapConfig tenTen = new TapConfig("10-10 Taps", TapConfig.TEN_TEN);

    System.out.println(rightCon);
    System.out.println(crypto1);
    System.out.println(hi tag2);
    System.out.println(oneTwoOne);
    System.out.println(twoTwo);
    System.out.println(randA);
    System.out.println(randB);
    System.out.println(randC);
    System.out.println(randD);
    System.out.println(randE);
    System.out.println(randF);
    System.out.println(ni neteenOne);
    System.out.println(tenTen);
}
}
```