Solving a 40-Letter Playfair Challenge with CrypTool 2

George Lasry
The CrypTool Team
george.lasry@cryptool.org

Abstract

Playfair is a manual substitution cipher invented in 1854 by Charles Wheatstone. Its name and popularity came from the endorsement of his friend Lord Playfair. The Playfair cipher encrypts bigrams (pairs of letters), and is considered more secure than monoalphabetic substitution ciphers which encrypt single letters. It was used by several countries in the 19th century and in the first half of the 20th century.

Playfair ciphers can often be solved with the help of a crib. Ciphertext-only attacks usually require hundreds of letters when carried out manually (Mauborgne, 1918). More recently, computerized attacks based on hill climbing and simulated annealing have been published, that require between 60 to 100 letters of ciphertext (Cowan, 2008; Al-Kazaz et al., 2018).

In this article, the author presents a novel ciphertext-only attack, implemented in the open-source e-learning CrypTool 2 (CT2) platform, that is effective against ciphertexts as short as 40 letters (CrypTool 2 Team, 2019). This attack is based on a specialized adaptation of simulated annealing and uses hexagrams in the scoring method. With CT2, a Playfair public challenge with only 40 letters was solved, establishing an unofficial world record for decrypting short Playfair messages, encrypted with random keys, from ciphertext only (Schmeh, 2018b). The author also offers a series of new Playfair challenges.[1]

[1]This work has been supported by the Swedish Research Council, grant 2018-06074, DECRYPT - Decryption of historical manuscripts.

1 Description of Playfair

Playfair enciphering and deciphering are based on a key square, with a $5 \cdot 5$ grid of letters. Each of the 25 letters must be unique and one letter of the alphabet (usually J) is omitted from the square, as there are only 25 positions in the square, but 26 letters in the alphabet.

While it is possible to use a random key square, it is often more convenient to derive a key square from a keyword (or sentence). The keyword is written horizontally with duplicate letters being removed. The rest of the square is filled with the remaining letters of the alphabet, in alphabetical order.

For example, the key square derived from the keyphrase HELLO WORLD is:

```
H E L O W
R D A B C
F G I K M
N P Q S T
U V X Y Z
```

To encrypt a message, the plaintext is split into bigrams. If there is an odd number of letters, a Z or X is added as the last letter. To encrypt the message HIDE THE GOLD, we first split it into bigrams, and add Z at the end:

```
HI DE TH EG OL DZ
```

Next, for each pair, we locate its two letters in the square. We replace them according to the following rules:

- If the two letters are corners of a rectangle, take the letters on the horizontal opposite corners of the rectangle. For example, HI is encrypted as LF.
- If both letters are in the same column, select the letters below each one in the square (go-

ing back to the top if at the bottom). For example, DE is encrypted as GD.

- If both letters are in the same row, select the letters to the right of each one (going back to the left if at the farthest right). For example, OL is encrypted as WO.

Using these rules, we obtain:

LF GD NW DP WO CV

2 Cryptanalysis of Playfair

In this section, the security of Playfair is discussed, and a survey of prior attacks on Playfair is presented.

2.1 Security of Playfair

The Playfair cipher has several weaknesses, which may be exploited when trying to recover a Playfair key square:

- With long enough ciphertexts, statistical analysis can be applied on bigrams, and matched against the frequencies of common bigrams in the language (e.g., English). The ciphertext bigrams corresponding to the most common bigrams in the language (such as TH or IN in English) can often be easily identified.
- The most frequent ciphertext letters are likely to be near the most frequent plaintext letters (e.g., E, T, I, O, N) in the key square.
- Each mapping of a plaintext bigram to a ciphertext bigram reveals the mapping of another bigram, where the letters of the bigrams have been reversed. For example, if HI is encoded as LF, then IH will necessarily be encoded as FL.
- If the Playfair key is derived from a keyword, then the last row often contains the last alphabet letters such as X, Y and Z. Also, the letters which did not appear in the keyword and are used to fill the bottom part of the square will always appear in alphabetical order.

2.2 Prior Cryptanalysis of Playfair

Historically, Playfair was often solved by hand with the help of cribs (partially-known plaintext attack). Based on the crib, some entries of the key square can be guessed or reproduced, and additional entries reconstructed by trial and error.

Manual ciphertext-only cryptanalysis involves frequency analysis of ciphertext bigrams, and usually requires hundreds of ciphertext letters, not an uncommon scenario if multiple messages were encoded using the same key. In (Mauborgne, 1918), a manual method is described, to solve a ciphertext composed of 800 letters. The frequencies of the most common ciphertext bigrams are matched against those most common in English, e.g. TH, ER, and ET. A tentative initial square is built, and completed in a trial-and-error process.

In (Monge, 1936), a challenge ciphertext with only 30 letters is solved by taking advantage of the characteristics of a key square built from a keyword (see Section 2.1). This is considered to be the shortest Playfair ciphertext ever solved, that was encrypted using a key derived from a keyword.[2]

(Cowan, 2008) presents an attack based on simulated annealing. It uses quadgrams frequencies (applied on a logarithmic scale) as the scoring function. A constant temperature is employed (Hoos and Stützle, 2004, p. 76). With this method, ciphertexts as short as 80 letters can be solved. Also, in the now-defunct website www.cryptoden.com[3], (Cowan, 2015) proposes a *churn* algorithm, described in Section 3.2. The *churn* algorithm was designed to mimic the process of simulated annealing with constant temperature, while reducing software code complexity and runtime. Cowan describes how his *churn* method was found superior to hill climbing for attacks on various ciphers (Cowan, 2015). Cowan's implementation of *churn* also produces an interesting but probably unintended side-effect, described in Section 3.3.

In (Al-Kazaz et al., 2018), a compression-based technique combined with simulated annealing is described, and demonstrated on several ciphertexts. The shortest one, with only 60 letters, was successfully decrypted with only two errors. The com-

[2]The unicity distance for Playfair and English is 22.69 letters (Deavours, 1977). For any Playfair cryptogram of that length or shorter, it is likely that there exist one or more keys, different from the original key, which decrypt the cryptogram so that the resulting decryption is a plausible English text (and different from the original plaintext). The unicity distance can be viewed as a theoretical lower-bound for the length of a cryptogram, so that its key may be recovered via cryptanalysis. The length of the cryptogram solved by Monge (30 letters) is very close to that limit. On the other hand, the unicity distance is only 16.56 letters if it can be assumed that the last row in the key square is *VWXYZ* (as for most keys derived from keywords) (Deavours, 1977).

[3]www.cryptoden.com is still accessible via www.wayback.com (Cowan, 2015).

pression technique proposed in (Al-Kazaz et al., 2018) is essentially analog to using hexagram statistics (on a logarithmic scale) as the scoring method.

3 A New Ciphertext-only Attack

In this section, a novel attack, successfully employed to solve several public challenges, is presented.

This new attack extends Cowan's method (Cowan, 2008). While it is also based on constant-temperature simulated annealing, it uses hexagrams statistics, instead of quadgrams, converted to a logarithmic scale. It implements an extended set of transformations applied to candidate keys, adding new types of transformations, as described in Section 3.4. Furthermore, the new attack exhaustively applies and tests the full set of transformations on candidate keys, at each step of simulated annealing (instead of applying only a random subset of transformations as in (Cowan, 2008)).

3.1 An Initial Implementation

Initially, this new attack was implemented using a standard constant-temperature simulated annealing algorithm. As described in Listing 1, higher scores are always accepted. If the new score is lower than the current score, the probability of acceptance p is computed using the Metropolis formula (Hoos and Stützle, 2004, p. 75), based on the score degradation d and the constant temperature t.

$$p = e^{-d/t} \tag{1}$$

The first implementation of this new attack, after tuning and optimizing the temperature t, was able to solve ciphertexts with only 70 letters, and rarely, with 60 letters (for comparison, (Cowan, 2008) requires between 80 to 100 letters). Also, hexagrams were found to be more effective than quadgrams or pentagrams as the scoring method (all using a logarithmic scale).

3.2 Improved Implementation Using Churn

The attack was modified to use Cowan's *churn* implementation of constant-temperature simulated annealing (Cowan, 2015). The *churn* acceptance function is described in Listing 2.[4] Cowan does not explain why he employs the term *churn*, however, the process could be described as candidate keys being accepted with a decreasing probability, or discarded ('churned') with a increasing probability, as the score of the current key increases over time during simulated annealing.[5]

A lookup table with degradation values, D, is precomputed. Cowan does not describe how he computed D, but his original values can be reproduced and closely approximated. From Equation 1, it follows that:

$$d = t \cdot \ln(1/p) \tag{2}$$

D has 100 entries (with an index i from 0 to 99). For each i, the acceptance probability is computed as follows:

$$p_i = (i+1)/100 \tag{3}$$

and therefore:

$$D_i = t \cdot \ln(100/(i+1)) \tag{4}$$

The *churn* acceptance function selects a (random) degradation threshold from the lookup table by generating a random index i from 0 to 99. If the actual degradation d is lower than this threshold, then the new key is accepted.[6]

After modifying the new attack on Playfair to use the *churn* acceptance function, the attack was again tested, and surprisingly, not only its runtime could be reduced, but the attack's performance was also improved. The algorithm was able to consistently solve ciphertexts with 50 letters. Cowan's *churn* algorithm was originally designed to mimic a constant-temperature simulated annealing process. There was therefore no apparent reason for such an improvement. After further investigation, the root cause of this phenomena was found, as described in Section 3.3.

3.3 An Unintended Side Effect

With a regular constant-temperature simulated annealing process (without *churn*), since the Metropolis acceptance function is a continuous function

[4]The code in Listing 2 is different from the original code given in (Cowan, 2015). It was adapted for clarity, but it preserves the original functionality.

[5]The acceptance probability decreases exponentially for candidate keys with a score lower than the score for the current key, as a function of the score degradation - see Equation 1. As the score of the current key increases over time (as better current keys are being selected), the degradation for a given candidate key increases, and its probability of acceptance decreases.

[6]The same functionality could in principle be achieved without a lookup table. However, the implementation using a lookup table plays an important role, described in Section 3.3.

(see Equation 1), a candidate key with a score significantly lower than the score of the current key can theoretically be accepted, albeit with a low but non-zero probability p. In other words, a candidate key resulting in a very high degradation d might still be accepted.

With *churn*, the lookup table stores 100 discrete degradation values, and from Equation 4, it can be seen that the highest degradation value is:

$$D_0 = t \cdot ln(100/(0+1)) = t \cdot ln(100) \quad (5)$$

As a result, with *churn*, no key resulting in degradation d greater than D_0 may ever be accepted. Similarly, it can be seen that there is a also lower bound for the acceptance probability, p_{min}, so that:

$$p_{min} = (0+1)/100 = 0.01 \quad (6)$$

Therefore, with *churn*, keys with a score degradation resulting in an acceptance probability $p < p_{min} = 0.01$ will always be rejected, unlike with regular simulated annealing, where there is a low but non-zero probability they might be accepted. It was suspected that particular side-effect of the implementation of *churn* could be the root cause for the higher performance of the attack with *churn* compared to the attack with regular constant-temperature simulated annealing.

To validate this hypothesis, a third version of the attack was implemented, using the standard acceptance function (for constant-temperature simulated annealing – see Listing 1), but this time only accepting keys with acceptance probabilities $p \geq 0.01$. With this modification (described in Listing 3), the attack on Playfair achieved the same performance as when using *churn*, confirming the hypothesis. The p_{min} parameter was further fine-tuned and set to an optimal value of 0.0085.[7]

3.4 Transformations on Candidate Keys

In all versions of the new attack on Playfair, the set of transformations applied at each stage of simulated annealing includes:

- Swaps of any two elements in the square
- Swaps of any two rows in the square
- Swaps of any two columns in the square
- Permutations of the five rows
- Permutations of the five columns
- Permutations of the five elements of any row
- Permutations of the five elements of any column

All possible transformations listed here are tested at each step of simulated annealing. In contrast, in (Cowan, 2008), only randomly selected transformations are applied and tested (from a smaller set of transformation types, which only includes the swaps, as well as a few special transformations).

4 A New Partly-Known Plaintext Attack

The algorithm described in Section 3 was also adapted to support a crib-based attack. The scoring function was modified, so that the score (computed using hexagrams statistics) is increased for each known-plaintext symbol correctly reproduced, when decrypting the ciphertext with a candidate key. With this modification, ciphertexts with 40 letters can easily be solved given a crib of 10 letters.

5 Solving Playfair Challenges with CrypTool 2 (CT2)

The new attacks described in Section 3 were first implemented as command-line programs. A first ciphertext-only challenge with 50 letters published by Klaus Schmeh was solved (Schmeh, 2018c). It took a few seconds on a 10-core Intel Core i7 6950X 3.0 GHz PC to complete the attack and to recover the key and the plaintext.

The attack was also integrated into CT2, taking advantage of the convenient user interface of CT2, which shows useful details about the progress of the attack, such as a list of top keys (CrypTool 2 Team, 2019). Klaus Schmeh published a second challenge, this time with only 40 letters, stating that its solution would constitute a world record for solving the shortest Playfair ciphertext encrypted with a random key (Schmeh, 2018b).

This new challenge was attacked with CT2. Initial runs only produced spurious solutions. At some stage, CT2 displayed a decryption (in the 4th place

[7] In preliminary experiments with attacks on other ciphers, this seemingly minor adaptation of simulated annealing significantly improved their performance. One possible explanation is that accepting candidate keys with scores significantly lower than the score of the current key, might completely disrupt the convergence of simulated annealing towards the correct key. Whereas accepting keys with score slightly or moderately lower than for the current keys helps in surveying more diverse areas of the keyspace.

in the list), starting with MEETYOU, but only for a few seconds, before the decryption quickly disappeared from the list as new higher-score decryptions were inserted. The partial known-plaintext attack (see Section 4) was then run with MEETYOU as a crib, and the solution was quickly found (see Figure 1). The plaintext, after adding spaces, is as follows:

MEET YOU TOMORROW AT FOUR TWENTY
AT MARKET PLACE

6 CrypTool 2

CrypTool 2 (CT2) is an open-source e-learning tool that helps pupils, students, and crypto enthusiasts to learn cryptology.[8] CT2 is part of the CrypTool project which includes widely-used e-learning tools for cryptography and cryptanalysis.[9] CT2 is the successor of CrypTool 1 (CT1), and it is based on a graphical programming language allowing the user to cascade different ciphers and methods and to follow the encryption or decryption steps in real-time (CrypTool 2 Team, 2019).

CT2 is maintained by the CrypTool team. Contributions and voluntary support to this open-source project come from all over the world. CT2 implements classical and modern cryptographic methods, including cryptanalytic methods. It is also used to implement real-world prototypes of distributed cryptanalysis using the so-called CrypCloud. CT2 is maintained in English and German.

State-of-the-art algorithms, such as the attack against the double transposition cipher described in (Lasry et al., 2014) and shown in Figure 2, are also integrated in CT2.

7 New Challenges

A series of new Playfair challenges is presented in Table 1 (Appendix 9), with short ciphertexts. The plaintexts were extracted from English books. The keys were generated either from an English keyphrase, or randomly. For some, the first eight letters of the plaintext are given as a crib. This crib is always PLAYFAIR, but the continuation of the plaintext is a sentence unrelated to Playfair.

In addition, (Schmeh, 2018a) has published a new challenge with 30 letters only and encrypted using a random keysquare.

[8] https://www.cryptool.org/en/cryptool2

[9] https://en.wikipedia.org/wiki/cryptool

References

Noor R Al-Kazaz, Sean A Irvine, and William J Teahan. 2018. An Automatic Cryptanalysis of Playfair Ciphers Using Compression. In *Proceedings of the 1st International Conference on Historical Cryptology HistoCrypt 2018*, number 149, pages 115–124. Linköping University Electronic Press.

Michael J Cowan. 2008. Breaking short Playfair ciphers with the simulated annealing algorithm. *Cryptologia*, 32(1):71–83.

Michael J Cowan. 2015. Churn Algorithm. https://web.archive.org/web/20150308125149/http://www.cryptoden.com:80/index.php/algorithms/churn-algorithm, [Accessed: January, 18th, 2019].

CrypTool 2 Team. 2019. CrypTool Portal – Cryptography for Everybody. http://www.cryptool.org/, [Accessed: January, 18th, 2019].

C.A. Deavours. 1977. Unicity Points in Cryptanalysis. *Cryptologia*, 1(1):46–68.

Holger H. Hoos and Thomas Stützle. 2004. *Stochastic local search: Foundations and applications*. Elsevier.

George Lasry, Nils Kopal, and Arno Wacker. 2014. Solving the Double Transposition Challenge with a Divide-and-Conquer Approach. *Cryptologia*, 38(3):197–214.

Joseph Oswald Mauborgne. 1918. *An advanced problem in cryptography and its solution*. Army Service Schools Press.

Alf Monge. 1936. *Solution of a Playfair cipher*. US Signal Corps.

Klaus Schmeh. 2018a. Playfair cipher: Is it unbreakable, if the message has only 30 letters? http://scienceblogs.de/klausis-krypto-kolumne/2019/04/15/, [Accessed: May, 18th, 2019].

Klaus Schmeh. 2018b. Playfair cipher: Is it unbreakable, if the message has only 40 letters? http://scienceblogs.de/klausis-krypto-kolumne/2018/12/08/, [Accessed: January, 18th, 2019].

Klaus Schmeh. 2018c. Playfair cipher: Is it unbreakable, if the message has only 50 letters? http://scienceblogs.de/klausis-krypto-kolumne/2018/04/07/, [Accessed: January, 18th, 2019].

8 Appendix – Listings and Figures

Listing 1: Simulated Annealing Acceptance Function – Constant Temperature

```
package common;

import java.util.Random;

public class FixedTemperatureSimulatedAnnealing {

    private Random random = new Random();

    // Fixed temperature optimized for hexagram scoring
    private static final double FIXED_TEMPERATURE = 20_000.0;

    /**
     * Simulated annealing acceptance function.
     *
     * @param newKeyScore - score for the ney key
     * @param currentKeyScore - score for the current key
     * @return true if new key should be accepted
     */
    boolean accept(double newKeyScore, double currentKeyScore) {

        // Always accept better keys
        if (newKeyScore > currentKeyScore) {
            return true;
        }

        // Degradation between current key and new key.
        double degradation = currentKeyScore - newKeyScore;

        double acceptanceProbability =
                Math.pow(Math.E, - degradation / FIXED_TEMPERATURE);

        return random.nextDouble() < acceptanceProbability;
    }
}
```

Listing 2: Simulated Annealing Acceptance Function – With Churn Lookup Table

```
package common;
import java.util.Random;

public class ChurnSimulatedAnnealing {

    private Random random = new Random();

    // Fixed temperature optimized for hexagram scoring
    private static final double FIXED_TEMPERATURE = 20_000.0;

    // Size of degradation threshold lookup table.
    private static final int LOOKUP_TABLE_SIZE = 100;

    // The churn algorithm lookup table of degradation thresholds.
    private final double[] degradationLookupTable = new double[LOOKUP_TABLE_SIZE];

    // Compute the churn algorithm lookup table of degradation thresholds.
    void computeDegradationLookupTable() {
        for (int index = 0; index < LOOKUP_TABLE_SIZE; index++)
            degradationLookupTable[index] =
                    FIXED_TEMPERATURE * Math.log(LOOKUP_TABLE_SIZE / (index + 1));
    }

    /**
     * Simulated Annealing acceptance function - Churn implementation.
     *
     * @param newKeyScore - score for the ney key
     * @param currentKeyScore - score for the current key
     * @return true if new key should be accepted.
     */
    boolean accept(double newKeyScore, double currentKeyScore) {

        // Always accept better keys
        if (newKeyScore > currentKeyScore) return true;

        // Fetch a random degradation threshold from the lookup table.
        int randomIndex = random.nextInt(LOOKUP_TABLE_SIZE);
        double degradationRandomThreshold = degradationLookupTable[randomIndex];

        // Degradation between current key and new key.
        double degradation = currentKeyScore - newKeyScore;

        return degradation < degradationRandomThreshold;
    }
}
```

Listing 3: Simulated Annealing Acceptance Function – Constant Temperature – Modified

```java
package common;

import java.util.Random;

public class ImprovedFixedTemperatureSimulatedAnnealing {

    private Random random = new Random();

    // Fixed temperature optimized for hexagram scoring
    private static final double FIXED_TEMPERATURE = 20_000.0;

    /**
     * Simulated Annealing acceptance function.
     *
     * @param newKeyScore - score for the ney key
     * @param currentKeyScore - score for the current key
     * @return true if new key should be accepted.
     */
    boolean accept(double newKeyScore, double currentKeyScore) {

        // Always accept better keys
        if (newKeyScore > currentKeyScore) {
            return true;
        }

        // Degradation between current key and new key.
        double degradation = currentKeyScore - newKeyScore;

        double acceptanceProbability =
                Math.pow(Math.E, - degradation / FIXED_TEMPERATURE);

        return acceptanceProbability > 0.0085
                && random.nextDouble() < acceptanceProbability;
    }
}
```

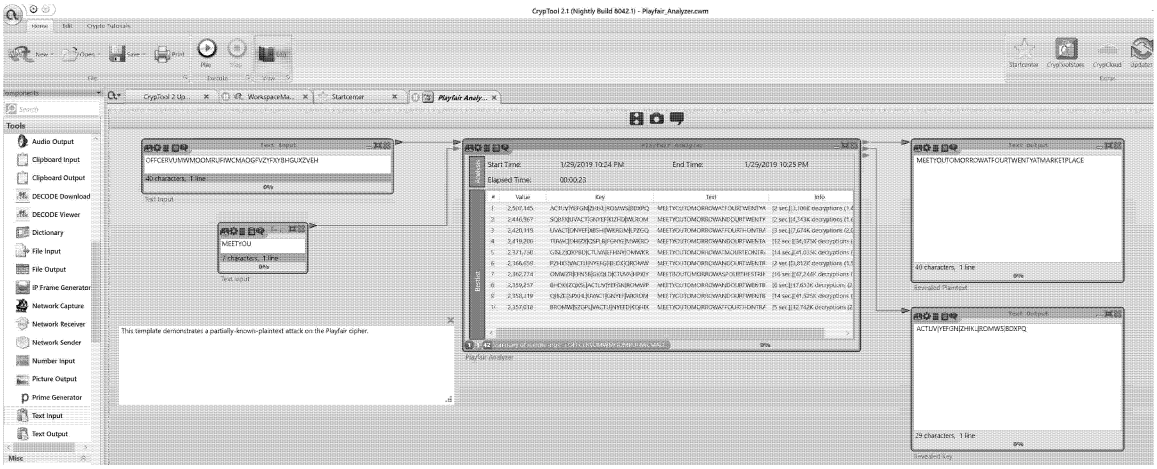

Figure 1: CT2 – Cryptanalysis of Playfair with Crib

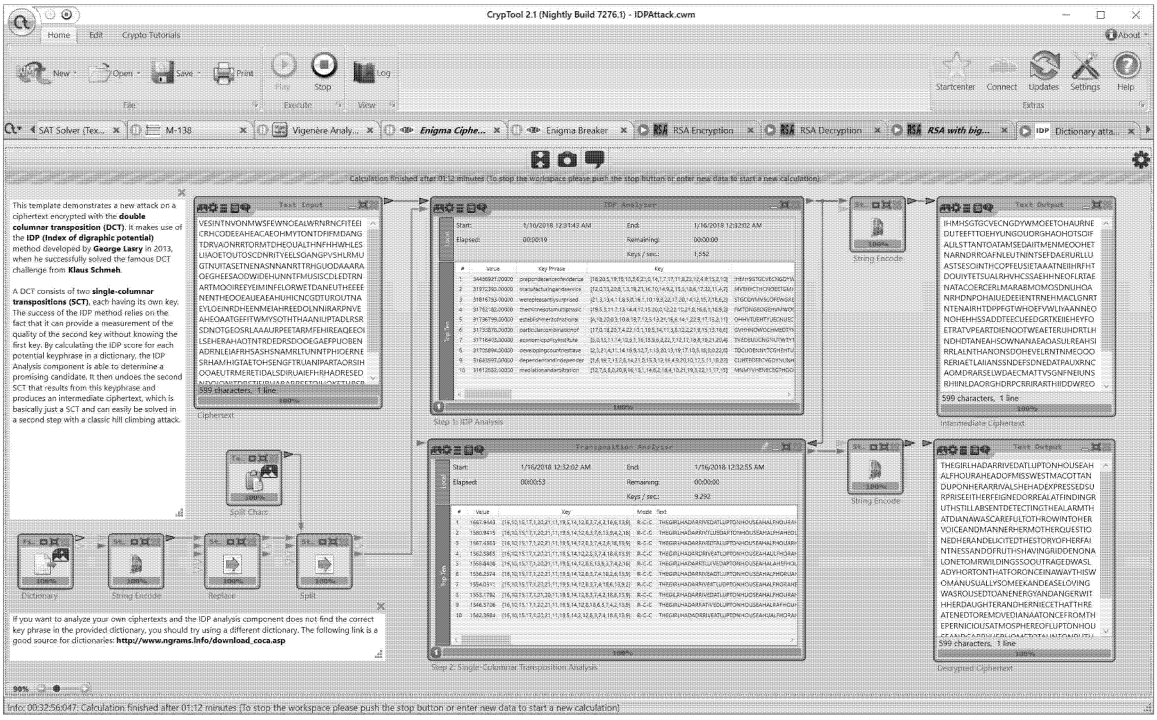

Figure 2: CT2 – Cryptanalysis of the Double Transposition Cipher

9 Appendix – Challenges

	Ciphertext	Length	Key	Crib
1	QONACDBLNKHIOTWDUEISOITFIDQBVOUTNRZOUCPC	40	From key phrase	PLAYFAIR
2	LVNXDNMHHLHIUIEGENXTHEGQHXUHFQ	30	From key phrase	PLAYFAIR
3	HNGFDIRFAMAVHFOVXLGLTVOAZMYLQGRXHAHRNHGF	40	Random key	PLAYFAIR
4	ZAYNWPSYEMYQTIRXICMCKVHQHTHUKY	30	Random key	PLAYFAIR
5	IROAWMDQLRNCTUOCFHMQQKMAALCQMGHIQOQKLCAP	40	From key phrase	
6	BQUWLODQTOODLXWKEGAQOGHQQTOQZI	30	From key phrase	
7	ILPMPEOIIZIRTPPRQRUYFUVXLIRCVANBVTPRWRCE CRVSLIQOVS	50	Random key	
8	TVCIYVGFVOGWPEFPDASNIXWKDISDRQVQLGSDZQXB	40	Random key	
9	PBILKMXFPGDMDHCYHIVECOOUTGBNUC	30	Random key	
10	PROMGDUGVBNYXKEADCHTHM	22	Random key	

Table 1: New Playfair Challenges