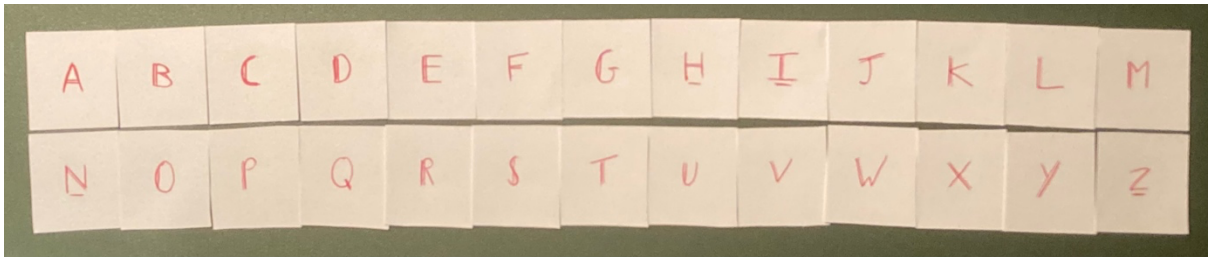
Spec. on the Encryption and Decryption of the Cascade cipher

It is not required for the cipher encryption or decryption, but for simplicity, I have created a series of 26 cards with the characters of the alphabet on each of them:



Encryption

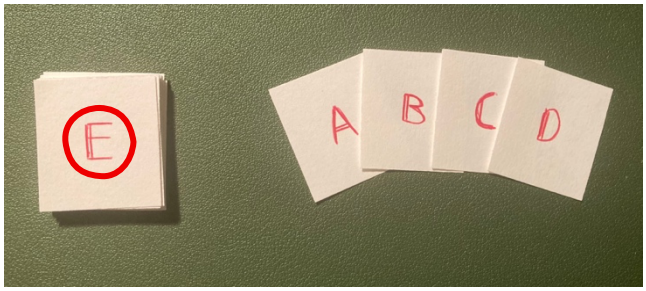
The initial state/the order of characters drastically determines the outputted ciphertext, and it is standard in this specification that the initial state is in the original A-Z alphabetical order. Throughout the encryption the keystream is determined by the current state of these letters and, after each character is encoded, the state of the series of characters changes.

The process involves taking the numerical value of the first/next letter in the plaintext, we'll call x so that x=n (mod25) and A=1 (omit Z), then moving n characters from the start of the series into a temporary sequence. Before returning them to the end in reverse order, so they appear flipped at the bottom of the 'deck of cards'. Then the letter now at the start of the series becomes the encoded character. This then moves onto the next letter in the plaintext, where the previous order of the characters in the series becomes our new current state.

The reason Z is omitted from the plaintext alphabet is due to modulo wraparound because, if n=26, all the letters are moved to the temporary series, resulting in identical encryption for both Y and Z. In which case, after decryption the letters are interchangeable, to reinclude words that use the letter Z. For example, encrypting and then decrypting the word ZEBRA results in YEBRA, meaning context is required to recover the original Z as plaintext is only an A-Y alphabet.

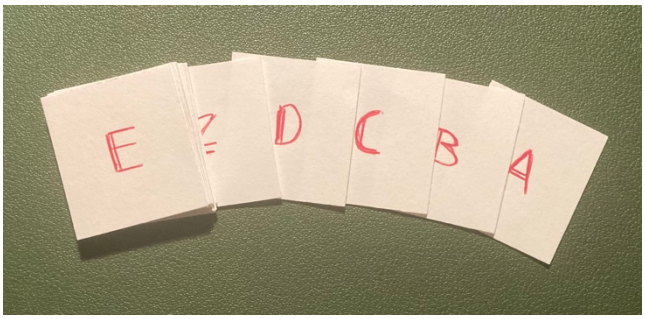
The letter shifts are not constant, as they rely on the numerical value of that character and the state of the series, which is completely reliant on the previous letter values. This means that frequency analysis is significantly weakened, and identical plaintext letters do not always map to identical ciphertext letters and in rare cases map to the same character as in the plaintext.

To put this into context, I will now encrypt the word "DOG" using the Cascade cipher:

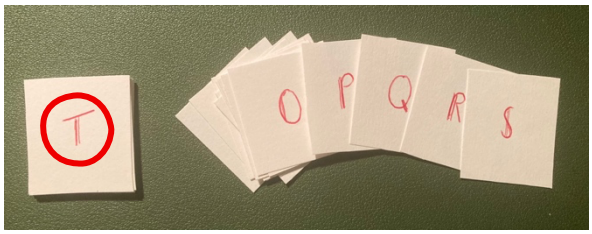


The letter D has a value of 4 so, taking the unshuffled deck of characters, I placed four of my cards down onto the table (one at a time, on top of the previous). The remaining card left on top of the deck is the ciphertext letter, in this case E.

Then to end the letter encryption and to create the new deck order for the next letter, I placed the cards I put onto the table at the bottom of the deck. This will resemble a reversed order from their appearance at the top of the deck.

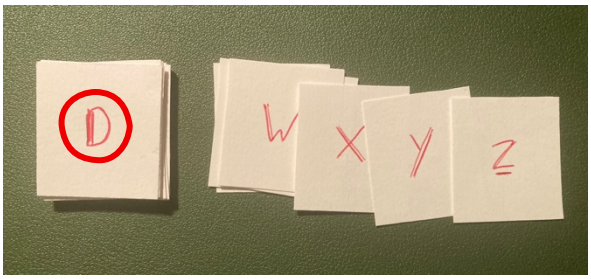


It is important to note that with an initial state that is not transposed from alphabetical order, the first letter in the ciphertext will always be +1 the character in the plaintext. This is only a property of the default initial state and not a general property of the cipher, otherwise this disappears.



Now doing the exact same with the letter O, which has a value of 15, I stacked 15 cards onto the table. Notice how I passed the letter O by four letters, which was the value of the previous letter. Leaving me with the letter T, as I placed the cards to the bottom.

Now moving onto the letter G, with a value of 7, I went through the entire deck and returned to D (because the total value of the letters was 4+15+7=26), but what remains is a current deck order that has 3 displacements. As you can see, the word DOG encrypts to ETD, but that is just a small number of n letters.



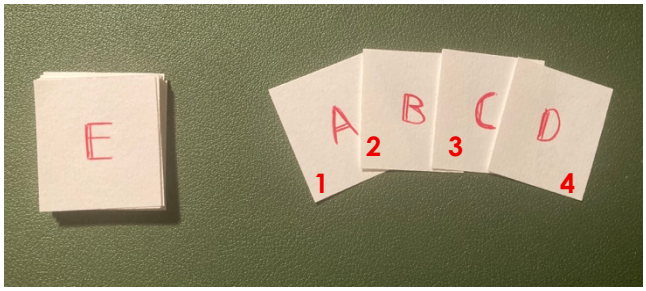
The plaintext alphabet is restricted to A–Y, but Z remains part of the internal permutation state, and therefore ciphertext. One notable property is the reliance on correct count, if someone were to miscount the value of the character, or (using the physical method) drag an extra card into the temporary series on the table, the human error propagates and affects every character afterward. There is also no automatic resync, so during decryption if one were to stumble upon an impossible character in the supposed plaintext, that results in the entire rest of the text to be nonsensical, it could be an error on either side. This means the Cascade cipher, if done by hand, requires careful bookkeeping. The closest classical cipher is the progressive Caesar cipher, where characters advance through a fixed alphabet sequence. However, unlike such encryptions, the Cascade cipher mutates the ordering of the alphabet itself and shifts by the value of the plaintext rather than merely advancing an index.

Decryption

In order to even begin decrypting the ciphertext of Cascade, we MUST know the initial state of the series, which means without a given key there are 26! possible initial states each producing a distinct decrypted outcome, though not all subsequent states may be reachable with any given plaintext. With that in mind, beginning with correct initial state, we move one letter at a time to the temporary series until we reach the character displayed in the ciphertext. As we did so, we'll have made note of the number of letters moved into the secondary series, this number is the numerical value of our plaintext character.

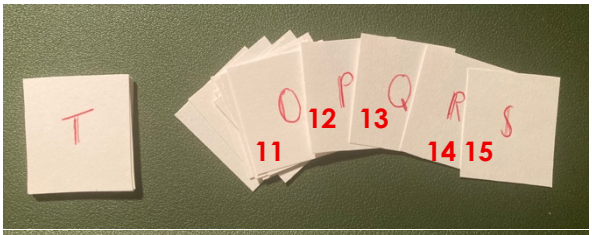
Then, like with encryption, the temporary series is returned to the end of the original series and is made the current state for the next letter to be decrypted. This is repeated until the whole text is decoded, unless an error occurs due to improbable characters, where it is advised as an operator error corrector to shift the value of the character by + or -1. This correction is not part of the cipher itself, but a rule of thumb associated with the manual operation. If that fails to work, it is advised to repeat from the beginning while double checking count or looking to use a different initial state (as that might be keyed but allow previous correct letters, such as swapping I's and J's positions).

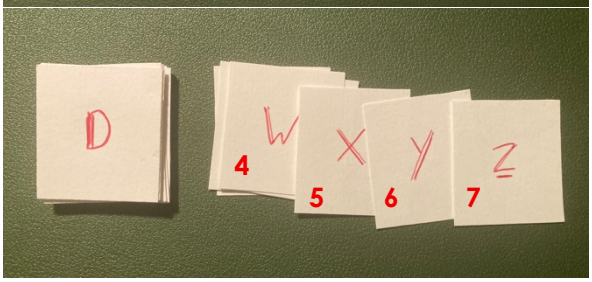
Again, to recontextualise what I have just stated, I will now attempt to decrypt the ciphertext "ETD" using this method:



Now starting with the standard A-Z initial deck order, I counted the number of cards I placed onto the table until I saw the first letter E. This was 4 which is the value of the letter D.

As for the other letters, the same process applies, making sure to count the number of cards placed down and not looking for the letter featuring on the card placed at the top of the pile on the table. In the case presented, it took 15 cards to get down to the letter T and 7 cards to get to the letter D, therefore the plaintext has the values 4 15 7, which decrypts back into DOG.





Larger Samples and Timed Decryption

These next sections provide evidence using larger samples and exacting an experiment on how long it takes to decode one character and the whole text for my peers using the previous sections of the text. I will use the standard initial state and will tell my peers I have done so, but I will not provide them with the cards I have made in order to see their choice of decryption.

Before encoding the text, I will attempt to write python code that executes the Cascade cipher on given plaintext. This should be easily reversible to then write the Decrypt code.

```
import string

def cascade_encrypt(plaintext):
    A_Z = string.ascii_uppercase
    x_to_n = {c: i + 1 for i, c in enumerate(A_Z)}

    deck = list(A_Z)
    ciphertext = []

    for ch in plaintext.upper():
        n = x_to_n[ch]
        temp_pile = []

        for _ in range(n):
            temp_pile.append(deck.pop(0))

        temp_pile.reverse()
        deck.extend(temp_pile)
        ciphertext.append(deck[0])

    return "".join(ciphertext)
plaintext = ""
ciphertext = cascade_encrypt(plaintext)
print(ciphertext)
```

To check the code I wrote worked correctly, I encrypted the text "THISISATESTOFTHEENCRYPTCODE" Using both the manual method, which I repeated to avoid potential human errors, and this python code. I am pleased to say this works as both resulted in the ciphertext: "URILURQYRPBLWXSDIYRQRVGTKXH", and in such a way I can reverse the method later to decrypt.

With the code confirmed to provide correct encryption, I will now input the plaintext: "THIS/IS/A/SECRET/MESSAGE" (without the dashes) due to the fact it is 20 characters long. While my peers read the previous pages of this spec, they may realise the first letter of ciphertext is +1 the value in the plaintext, so when taking measurements of the mean time to decrypt one character I will be omitting the first one regardless of if they make the connection. Using the code, I produced the ciphertext: "URILURQZUSVTSHMZRUGB."

Also, with the fact that each letter only ever shifts the series by a set amount, the ciphertext can produce isomorphs where repeating words or phrases produce similar letters within the ciphertext phrases. For example, the encoded phrase "INCREDIBLE WORK ONCE INCREDIBLE WORK TWICE INCREDIBLE WORK" produces the ciphertext "JXINZPBDXV ZNFJ ICRM DTEYRGLJTV RYOD TAHLN IYTVLDROYF LVJI." Where you can notice the pattern:

Encryptions of the plaintext "INCREDIBLE WORK" are isomorphic, the top line annotates repeated ciphertext letters.

```
ab·cd···b· dc·a
JXINZPBDXV ZNFJ
DTEYRGLJTV RYOD
IYTVLDROYF LVJI
```

Sebastian M ended up using a physical method, and using scrap pieces of paper, resulted in these times.

Letter	Value	Time taken
T	20	23.58s
H	8	18.20s
I	9	17.73s
S	19	37.65s
I	9	19.28s
S	19	29.63s
A	1	5.31s
S	19	28.00s
E	5	12.23s
C	3	13.96s
R	18	32.01s
E	5	16.68s
T	20	25.87s
M	13	19.13s
E	5	16.38s
S	19	25.70s
S	19	18.75s
A	1	5.75s
G	7	11.15s
E	5	9.50s
	Total time	6mins 26.46s
	Average time	19.101s

Ryan O opted to write his own Decrypt code in C++, which without extra direction took ~50mins. He did begin by writing an encrypt to confirm he understood the cipher, then successfully decrypted all the code in a single run of the program.

```cpp
#include <iostream>
#include <print>
#include <algorithm>

std::string_view alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

[[nodiscard]]
size_t alphaVal (const char ch) {
    return ch - 'A' + 1;
}

[[nodiscard]]
std::string cascadeEncrypt (std::string_view&& pt)
{
    //can change this to the initial state
    std::string deck = alpha.data();

    std::string ct = "";

    for (const char& ch : pt)
    {
        std::reverse(deck.begin(), deck.begin() + alphaVal(ch));
        std::rotate(deck.begin(), deck.begin() + alphaVal(ch), deck.end());

        ct += deck.front();
    }

    return ct;
}

[[nodiscard]]
std::string cascadeDecrypt (std::string_view&& ct)
{
    //can change this to the initial state
    std::string deck = alpha.data();
    std::string pt = "";

    size_t idx;

    for (const char& ch : ct)
    {
        idx = deck.find(ch);
        std::reverse(deck.begin(), deck.begin() + idx);
        std::rotate(deck.begin(), deck.begin() + idx, deck.end());
        pt += alpha[idx - 1];
    }

    return pt;
}

int main ()
{
    std::string ct = "URILURQZUSVTSHMZRUGB";

    std::println("{} -> {}", ct, cascadeDecrypt(std::string_view(ct)));

    std::cin.get();
}
```

The results have shown that it takes ~20s to decrypt each character using a physical method, and compared to a programmed decrypt, takes drastically longer the larger the ciphertext is. However, even with the knowledge of how the cipher works, the code takes a while to program. Which makes this all the harder if one didn't already understand how to decode the ciphertext.

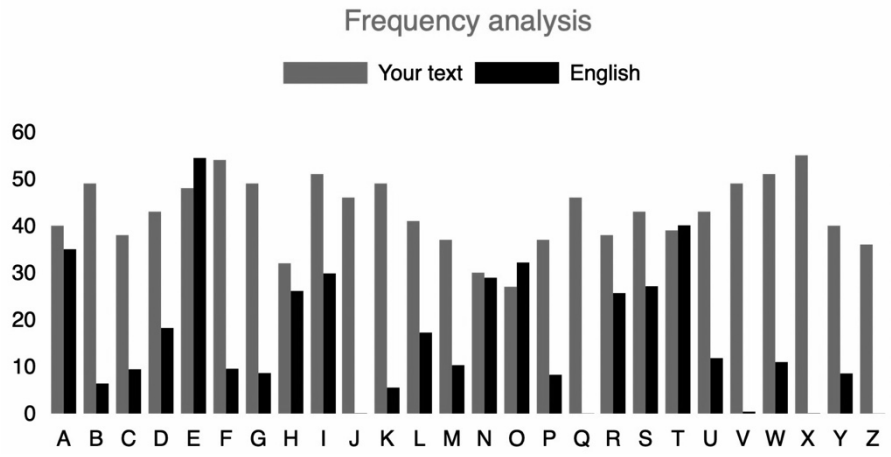
To finish off, I will also include my Python decrypt code, which uses similar ideas to how my peer, Ryan O, solved it.

```python
import string

def cascade_decrypt(ciphertext):
    A_Z = string.ascii_uppercase
    n_to_x = {i + 1: c for i, c in enumerate(A_Z)}

    deck = list(A_Z)
    plaintext = []

    for ch in ciphertext.upper():
        temp_pile = []
        count = 0

        while deck[0] != ch:
            temp_pile.append(deck.pop(0))
            count += 1

        plaintext.append(n_to_x[count])
        temp_pile.reverse()
        deck.extend(temp_pile)

    return "".join(plaintext)
ciphertext = ""
plaintext = cascade_decrypt(ciphertext)
print(plaintext)
```

The key differences with this code from the encrypt is that instead of a function turning a letter to number, only numbers need to be turned to letters. As well as, the inclusion of a count to mark how many 'cards' are placed in the temp pile, which stops when the top card of the deck is the character of the ciphertext. Just like Ryan's code, this Cascade decrypt code turns "URILURQZUSVTSHMZRUGB" into "THISISASECRETMESSAGE."

Strengths and Weaknesses

Some of the known strengths of the Cascade cipher lie in its stateful diffusion, meaning each character influences not only its own output but also the evolving internal series used for all subsequent characters. This prevents the cipher from behaving like a fixed substitution, ensuring there is no static plaintext-to-ciphertext mapping and greatly reducing the effect of basic frequency analysis. Because the state continually shifts, identical plaintext letters can encrypt to different ciphertext letters depending on position, increasing unpredictability and reducing pattern repetition. The cipher is also highly sensitive to error: even a single miscount or incorrect state update will propagate forward and produce errors in all later outputted plaintext. This raises resistance to casual tampering or guesswork, as small mistakes rapidly compound. Additionally, if the initial state is keyed, the effective keyspace becomes substantially larger, since different starting permutations produce entirely different encryption behaviours. Finally, despite the fact the cipher is prone to isomorphs with a plaintext of repeated words/phrases, the way the series is modified after each letter encryption means that the cipher is immune to common alphabet chaining. So, with knowledge of repeated words some of the plaintext can be found, however, no information could be gained as to the composition of the initial state.



The following shows the results of basic frequency analysis, using a long encoded form of the poem "Waterfall" by Wayne Anderson. Though similar even spread can be seen when measuring the distance between characters.

Alongside its strengths, known weaknesses of the Cascade cipher arise primarily from its reliance on a recoverable evolving state. In some circumstances, sufficiently long and contiguous known-plaintext or chosen-plaintext sequences could allow an attacker to reconstruct the current state at any given point and predict future outputs, especially if large ciphertext samples are available. The cipher also lacks built-in integrity checking, meaning altered ciphertext cannot be reliably detected, and its strong error propagation makes recovery impossible once mistakes occur, a drawback for both attackers and legitimate users. Furthermore, manual operation becomes impractical for long messages due to the cognitive load and cumulative risk of human error. Additionally, the ciphertext does feature isomorphs within repeated words/phrases no matter what the initial state is, due to the fact that each letter has a set value to modify the series by, making it perfectly isomorphic. Finally, the current design exhibits a structural collision where Y and Z produce parallel encryption outcomes, introducing a mathematical imperfection that should be addressed in future refinement.