Clue Number 1

Dear Charles,

if you are having difficulty cracking the last section you should probably take note of the underscore - I didn't think anything of it at first, especially given all the formatting, but it does stand out as unusual. For some reason this cipher uses 27 characters. The formatting is actually a red herring apart from that and is best removed, though it is correct in that it does encode the original word shapes, so it is worth keeping it around somewhere as a sanity check!

Harry

Clue Number 2

Dear Charles,

Still stuck? As I said before this is based on the idea of the ADFGVX cipher, but instead of starting with a 6x6 grid it starts with a 3x3x3 grid in which the 26 letters of the alphabet and the underscore, representing a space, are entered in some order. This encodes those twenty seven characters as triples AAA, AAB, AAC, ABA, ABB, ABC, ACA ACB, ... and so on.

Harry

Clue Number 3

Dear Charles

As with the ADFGVX cipher, after encoding the characters as triples the string of As, Bs and Cs is broken up into short block and a column transposition applied to jumble them up.

Hope this helps,

Harry

Clue Number 4

Dear Charles,

I realise I didn't complete the description of the cipher for you! After the transposition stage the resulting text is read off as triples and recoded back (using the same 3x3x3 block encoding) into characters. Maybe to throw the reader off or perhaps to give us some kind of extra crib, Tiberius reformatted the string so it fitted into the original text shapes. I suppose that does give another way to try to break in!

All the best,

Harry

Clue Number 5

Dear Charles,

Just in case you didn't work it out yet this is a symmetric cipher! That means that the transposition stage is a permutation with the property that if you do it twice it is the same as not doing it at all! Devious old Tiberius has used a block length of 5 for that stage so that once it has done its work the first and second characters are entangled, the 3rd,4th,5th are entangled, the 6th 7th are entangled, the 8th, 9th, 10th are entangled and so on. I guess that means that working with blocks of fifteen makes a certain amount of sense!

All the best,

Harry

Clue Number 6 - last one!

Dear Charles,

In case you are really stuck the transposition stage just reverses the order of the block so if block 1 reads ABCCB then after the transposition it reads BCCBA. I guess once you know this then it should be possible to crack the cipher using some frequency analysis!

Harry

PS, just noticed a typo in my previous message. It should have read:

the first and second characters are entangled, the second, 3rd, and fourth are entangled, the 4th,5th are entangled, the 6th 7th are entangled, the seventh 8th, 9th are entangled , the ninth and 10th are entangled and so on

Sorry, Harry

(PL notes – I think the entanglement description w2as a red herring!)

8A:

This was, for me, a fairly easy challenge. The first thing I did was to look for repeated sequences in the text, and see what the distances between the positions of the repetitions was. (I already had a program that could do this.) It turned out that this was almost always a multiple of 6, so I guessed that it might be a Vigenère cipher with a key of length 6. I then did frequency analysis on each of the six alphabets separately - the distribution of frequencies looked about right, so in three of the alphabets, where the most common letter was much more common than the next most common letter, I set the key so these letter went to 'e', and so that the rest of those alphabets were Caesar shifts. Having done so, there was an obvious crib at the beginning ("Dear Charles"), so I chose the rest of the key so that I would get these letters out. This made the whole message readable, so I was finished. (Incidentally, with the interactive analysis and decoding program I wrote, this took around five minutes to download, solve and submit.)

8B:

This was considerably harder. I won't say how long it took me to do it, but it was more than a day and less than a week.

Firstly, I found several specific cribs to be useful at first, primarily "Tiberius", "Valentine and Proteus" and "don't". To work out the possibilities for some of the other words I also tried using a crossword clue solver. (Put in some letters and some gaps, and it will tell you all the words that fit.) However, this turned out to be of limited use, although it did allow me to confirm that there was no letter that could go in both of two places, while I was still working with the wrong algorithm.

After several hours of wrong ideas, it occurred to me to try looking for repeated sequences in the ciphertext with all characters except a-z and underscores removed. Most of the distances between these repetitions were multiples of 5, so I decided to try solving it as if it was a Vigenère cipher (knowing that it wouldn't be). I got a bit of plain text by doing this, but there were obvious problems, such as the two 'e's in "Valentine" corresponding to different letters. However, I did notice that the frequencies of the third alphabet looked like normal English frequencies, and that a simple substitution seemed to be getting the right letters here

My next step was to look at exactly where the repeated sequences I found had appeared. Looking at the sequences of length 3 that appeared exactly twice, I noticed that most of them started on the first or third character of a block or five, while almost none of them started on the second character. This suggested that it might be a mixture of digraph substitution and single character substitution, with the first two characters being encoded together, the last two together, and the third by itself. Then most of the repeated sequences would consist of an encoded digraph and a single encoded character, while almost none of them consisted of half of a digraph, a single character then half of another digraph.

Using this algorithm, I again went through trying to work out as much of the plain text as possible. There were still a few problems - there were some ciphertext digraphs that needed to represent two different digraphs in the plaintext, and several ciphertext digraphs had to map to the same plaintext digraph. In particular, "WGX", "WIX" and "WJX" all corresponded to the word "the", in the same position in each block. However, I managed to guess about half of the plaintext, with perhaps 90% accuracy, before I was unable to spot any further words. (This did take me a very long time - I think I was doing

The next thing I did was to compare the mixed plaintext/ciphertext I had generated, and the original ciphertext, and to list all pairs of corresponding ciphertext and plaintext digraphs (separately for the first digraphs and the last digraphs in the blocks of 5). I then sorted these lists. For the first list, I noticed that for ciphertext digraphs beginning with the same letter, the corresponding plaintext digraphs would end with one of about three letters (sometimes slightly more due to mistakes in my guessed cribs). I then tried to group letters (including the underscore) together into 9 groups of 3, so that when the ciphertext digraphs began with the same letter, the corresponding plaintext digraphs mostly ended with letters from the same group. ("Mostly", because some of the cribs were wrong.) This gave me the groups "CYP", "HER", "TAS", "KLM", "NOQ", "UVW", "XZ\_", "BDF", "GIJ". (Note that this almost corresponds to the groups in the keystring - I have sorted them so that they looked like a hypothetical keystring.) This suggested that a keystring of "Cypher task" might have been used. However, when I tried a similar approach to the work out how the columns related to each other, it looked like 'A' ought to be in the third column, so I abandoned this latter line of thought and eventually came up with a simpler idea.

This final idea was fairly simple to implement as a program. Using the groups I found above (and similar groups for the other set of digraphs), and knowing which ones corresponded to which, I was able to work out that each letter in the cipher text corresponded to one of three plaintext letters. I thus wrote a program that split the text into blocks of five, and swapped the first two characters and the last two characters. Then it generated four strings. The first one consisted of spaces except for the third character in each block of five, which I could work out as a monoalphabetic substitution. The remaining three strings gave the possibities for each letter. It then took the orignal ciphertext, with spaces, and used this to add spaces and punctuation to the first string, and spaces to the corresponding positions in the rest of the strings. These four strings were then written to a file, the beginning of which looks like this (with the first line repeatead below as it appeared once I had 'solved' the cipher):

e e d o y a s d ... g , t i .

th umt u mpnn lvhk plb b fxh b nukg fo hx ... hh mmbt vnhh uo

se gas v awoo eyel wed d the d bgli td ez ... ee aads ybee gd

ar cra w rjqq zirm jzf f qkr f \_cmj qs r\_ ... rr rrfa i\_rr cs

the gas weapon developed by the fabulists had ... regards, tiberius.

I then merely has to work out each word given the choice of letters for it. This was mostly not too hard, but there were one or two words I struggled to identify (such as "forged" and "fitting"), and it turned out that I also had a small mistake in my groups. Thus it was very common for me to find places where I had a choice of "x", "z" or "\_", in which I actually used an "a" instead (such as the last word in the initial section shown here). Eventually, though, I had written out the entire plaintext. The final step before submission was to proof-read it (I remember losing several points a few years back when we didn't proof-read the plaintext for the hieroglypic cipher).

My story doesn't quite end there. Several days later (after a break to clear my mind and get some more sleep), I decided to work out how it was actually encoded. I did so while travelling to an airport by train on the 27th, so I had clue two to help me (which was useful). I was able to correct my earlier mistake in the keystring fairly easily, and then I looked for correlation between characters in the plaintext and characters in the ciphertext. Apart from the strong correlations I already knew about, I also got weaker correlation between the first letter in a plaintext block of five and the first ciphertext letter, the second and fourth letters (and vice versa) and the fifth and the fifth. Knowing this information, I wrote a program that took a list of fifteen integers (representing the transposition) and spat out the corresponding output. It took me a few attempts to get the correct sequence of numbers, due to a few silly errors, but when I found it I knew it had to be the right one, given how simple it was (of course, I ran the program one last time to check). This stage took me just over an hour (slightly less than the length of a train journey from Dover Priory to St Pancras International).

Keyword for 8A = VERONA

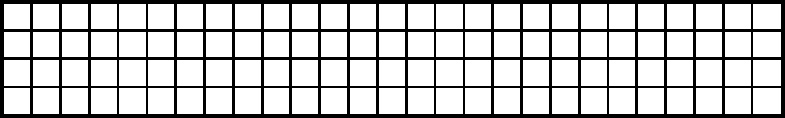
Keyword for 8B = CYPHER

How SaMTHG solved it:

<http://www.scribd.com/doc/119029182/8b>

How I did 8b:

I’m just going to do this quite quickly because it’s late.I started out by guessing that NWXVVWMC = TIBERIUSand JLZEGQWLP QXD S\_IOXMC = VALENTINE AND PROTEUSI had performed a frequency analysis on trigraphs and saw that all the distances between the mostcommon trigraph were 0 mod 5, so I guessed it was something to do with blocks of 5 (later confirmed bythe hint)I thought to myself it will probably end with no stray characters and so used the last 5 letters of  NWXVVWMC and TIBERIUS and came up with this: (bearing in mind I tried in columns first as in thetrifid cipher but with no luck). I didn’t use any programming yet, this was all done by hand. Also note thatI had the last hint by this point, I reckon I could have still done it without it, it just made it a lot easier.E1 E2 E3 R1 R2 R2 R1 E3 E2 E1 V1 V2 V3 V1 V2R3 I1 I2 I3 U1 == U1 I3 I2 I1 R3 == V3 W1 W2 W3 M1U2 U3 S1 S2 S3 S3 S2 S1 U3 U2 M2 M3 C1 C2 C3I then set up sets of equivalents:R2=V1=E2R1=V2=E1E3=V3=UIEtc. I then did this with VALENTINE AND PROTEUS and got more equivalences..And after an hour or so after getting on the right lines, ended up with:I1=W3=M3=S2=B2=O2=A3=L1=J3=Q3=T2=G1=N2=P3=Q1=\_2=M1=O1=R3=T3AndI3=W1=X2=E2=Z3=L3=V1=R2=S3=M2=O3=A1AndI2=W2=X3=B3=E1=P1=V2=R1Then I thought “although I don’t have all the letter sequences I can probably have a good bash at it, and Isaid the first list would all be 1s, the second list would all be 2s and the third all 3s, and drew up a table:ABCDEFGHIJKLMNOPQRSTUVWXYZ\_ 231111313221232112113321132121211121132



From this I already saw I had the placement of 3 letters:Box1:MIBox 2:Box 3:R I then used a lot of process of elimination such as, I knew E3 couldn’t be 1 because R would be in thesame place.. It was all about trying to determine the numbers so two letters didn’t occupy the same cell,anyway, after a long while and process of elimination I got here (you can figure out how yourselves):ABCDEFGHIJKLMNOPQRSTUVWXYZ\_ 2

232

31111

11

313

332

22

2232

1

1

2

3

3

2

211

31

211332

2

113

3223

2121

3

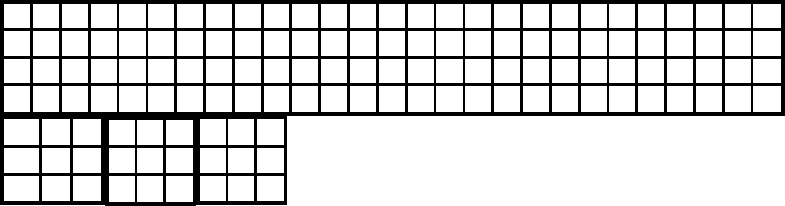
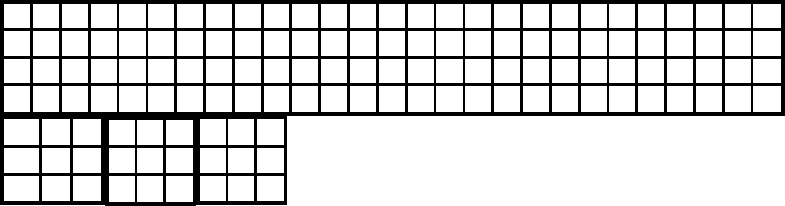
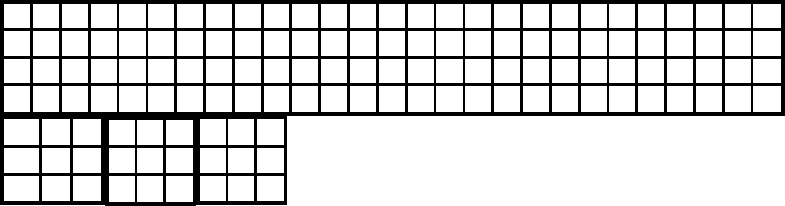
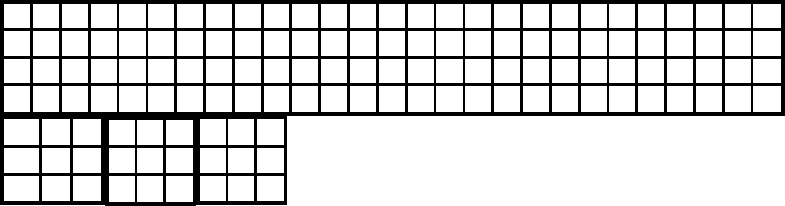
211121

2

132

3

So although I didn’t have all the letters I thought I had substantial amount, and carried to fill in the boxes:



Box1:

QON

M

L

I

G

Box 2:

DBAZXW

Box 3:

TS\_

R

EP

I noticed there was a pattern (going backwards) at LMNO and also saw that the box numbers of certainletters would make sense for the alphabet to be going backwards so reasoned that K was probably next toL, .. Made a few more guesses simply working by going backwards in the alphabetWX\*ZAB\*D\*G\*I\*\*LMNO\*Q so I also thought “y” is probably in the keyword, “c” is probably in thekeyword and “h” is probably in the key word.. Saw the \*\*P\*ER at the end, and filled in CYPHER  backwards, then the rest of the alphabet. Solved the cube.Then I went back to the cipher text, and with some programming help, changed every letter into it’scorresponding coordinate, got rid of all punctuation and spacing etc. broke up the 1000+ long string into blocks of 5 and reversed each one then made it into a long string again and broke it up again into 3s,finally changed each of the coordinates with the corresponding letter, and I was done. Tedious, but rather fun too :P And quite satisfying when done.(Note I had all the clues by the time I finished it)

This is the method my team used to do 8B:

We brainstormed for a bit, and eventually noticed that 27 was 3^3 and that it might be a key-cube of sorts. After a bit of research on the internet, we noticed the trifid cipher: en.wikipedia.org/wiki/Trifid\_cipher

This was quite similar to what we thought the cipher was (especially in that it re-encodes the trigrams back at the end), but slightly simpler in the middle stage (column transposition and all that). We also noted that Harry said the cipher was new, so it probably wasn't the trifid cipher exactly. He wouldn't have said the cipher was based on the ADFGVX cipher in 8A if it wasn't, so we combined the Trifid & ADFGVX to form a pretty accurate impression of the cipher.

The next step was cryptanalysing it. We figured out the key length was 5 because of repeated strings and suchlike (roughly the same way you'd find out the key length for a vinegar cipher). Once we'd got that, I hazarded a guess that the cipher ended "with(fondest/kindest/warmest)regardstiberius". With this information, I constructed a program to generate constraints based on the ciphertext/plaintext combo at the end, for each of the 120 (5!) ways to permutate the 5 separate columns.

I considered each letter as having a unique 3 character string constructed of a combination of a, b and c.

With each of these constraints (they were all of the form "the third character of "k" must be the same as the second character of "\_" or so), I plugged them into a second program I wrote using the glorious labix.org/python-constraint library. There were 0 solutions found for the first 119 different permutations (constructed using the itertools.permutations python function), but 5760 solutions found for the 120th permutation, which happened to be [4,3,2,1,0] or completely reversed.

For each of these 5760 solutions (which I realised later was actually multiplied by a factor of 6, because if you just replace all instances of a with b and vice versa (etc..), you get the same solution due to the triplets being turned back at the end), I solved the ciphertext with them and analysed the final solution. Using a function I wrote to determine how 'english' a text is based on its frequency analysis, I took the most 'english' of these solutions, and it was the perfectly decrypted plaintext.

Here's the final code I used (python 2.6): - bear in mind it's gone through a couple of iterations.

pastie.org/5624492

This is the contents of ciphertext.txt:

pastie.org/5624498

This is the contents of LetterFreqs.txt:

pastie.org/5624502

The code runs in just under 40 seconds with the full length crib.

Some interesting things to note: there were 960 possibilities with the crib "ithfondestregardstiberius" (having a crib length being a multiple of 5 made things easier for my code), crib length 25.

The number of possiblities increased dramatically as I reduced the length of the crib: "ndestregardstiberius" w/ length 20, yielded a mighty 80640 possible solutions, but no solutions for other permutations other than simply reversed.

I had to go all the way down to "dstiberius" in order to get possible solutions for column transpositions other than [4,3,2,1,0].

Feel free to ask any questions or comment on our code/methodology, if anyone wants to use any of the functions in it, feel free!

Hm :\ After reading how others did A I feel a bit idiotic, however, I'll say how I did it, because I didn't even realise there was a key, so for those who want to know how to do it without a key, here goes:

1) Perform Kasiski analysis on cipher text

2) Realise that highly probably keylength is 6

3) Split up cipher text into different strings of every 6th character, i.e. cipher text: 123456123456 becomes 11,22,33,44,55, and 66 using a list in Python and doing string[::6] string[1::6] etc.

4) Performed frequency analysis on each string to find most frequent letter used, assumed it was e and did a caesar shift accordingly but also cross referenced this with the assumption that it'd start "Dear...". For example, in the first set of 6th letters you have YCVCZ...blah, the most common letter is Z, so if that becomes e then Y should be one letter before, d and so on.

Smaug - would you recommend Mathematica for classical cryptography analysis and cracking. How does it compare to other high-level languages?I'd recommend it for almost everything, if you can get hold of it. Its two cons are that it is expensive (£96 including VAT for the student edition) and that it is much slower than most other languages (although it is getting steadily faster, and functions such as Compile allow it to be nearly as fast as C in some places).

It has built-in functions for almost everything, and they're intuitively named. Its documentation is very good. Once you've learned a few principles behind how things are named and how arguments to them are presented, you can usually guess what a function will be called and how its arguments will be given; it has an autocompleter for function names and arguments, too. It does loads of stuff, from image analysis to graph theory to statistical distributions to algebra… code written in it is short and readable. It's especially good at functional programming (that is, list-based, applying functions to lists).

Cryptography-wise, I think it might be a bit slow for many applications. However, I have written a cipher suite in it, and it cracks various ciphers perfectly adequately. You can test out ideas really quickly because you have to write so little code in it.

If you can afford it, get it; I used it in my A-level physics coursework and I'm currently using it to help get me through my first year in maths at uni. (It can do all the maths, but because it can't walk you through how to solve things, I use it only for checking my work, or for deciding whether a particular line of working will end up right or be a waste of time.) It's not very widely used in industry, and it gives you annoyingly high expectations for how easy a language should be to code in - Matlab is a disgusting language in comparison, but I will have to use it for my computational project.

If you're nearing university age and looking at universities it's worth bearing in mind that some universities (including, for example, Southampton) provide packages such as Mathematica and Maple free to students.

Thanks for the reccomendation Smaug, I'm downloading a free trial as we speak.. Its worth noting that they do have a year-long student licence for £30, so if you're not sure you'll use it a lot, it could be worth getting that instead of the full version.

It took me a little while to get used to functional programming, but it's actually much more intuitive than the procedural/imperative paradigms.

If you want inspiration, the Wolfram Demonstrations site contains many examples of smallish Mathematica programs. Also, Wolfram|Alpha contains some basic information scraped from the documentation - for example, here's the entry on the function Map. All the (interactive and really good) documentation which comes bundled with Mathematica can be accessed in less useful (ie. static) form on the Reference site.