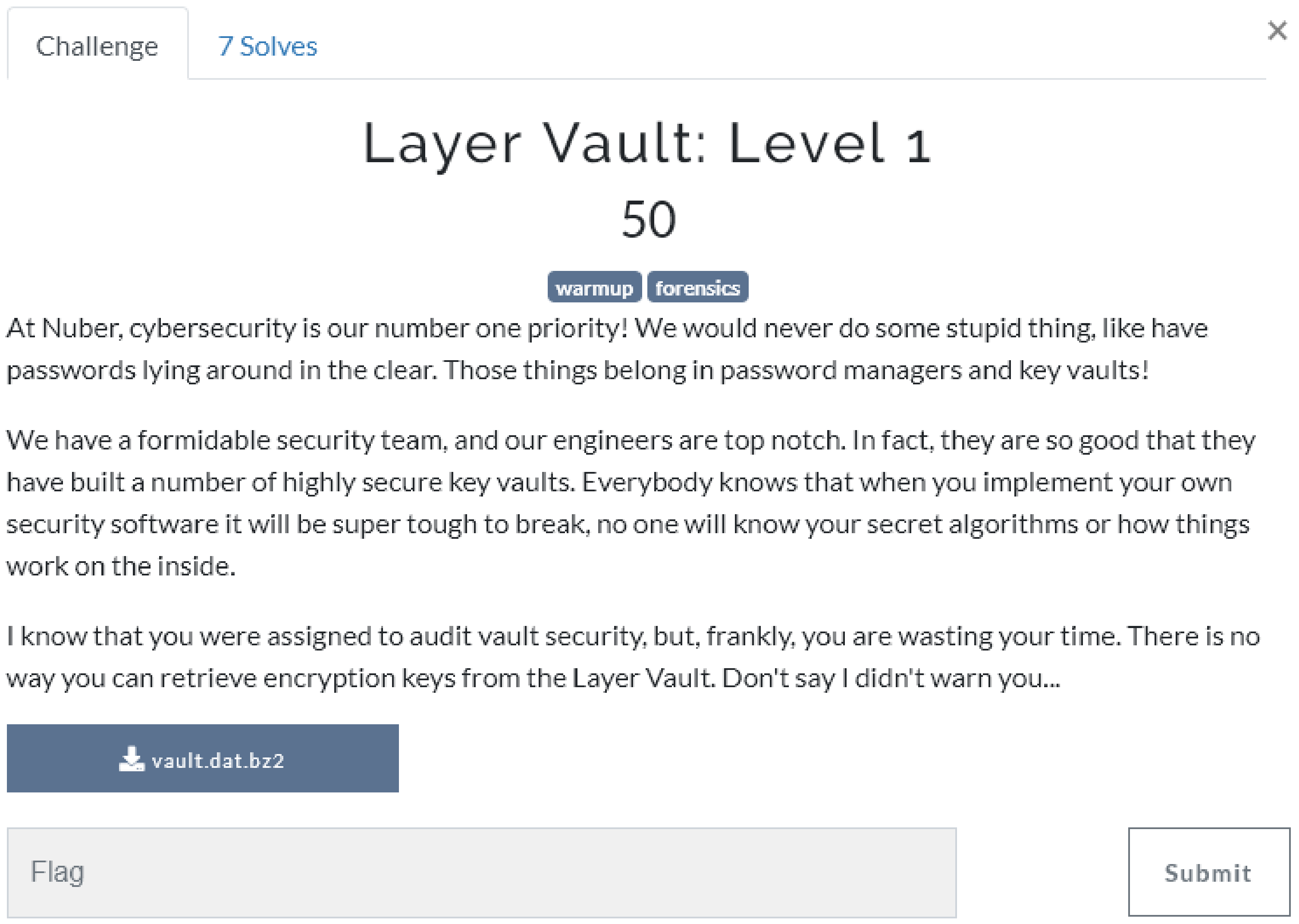
Cyber Sci

Nüber Edition

November 5, 2022

Vault Challenges Walkthrough

Dmitriy Beryoza (0xd13a)



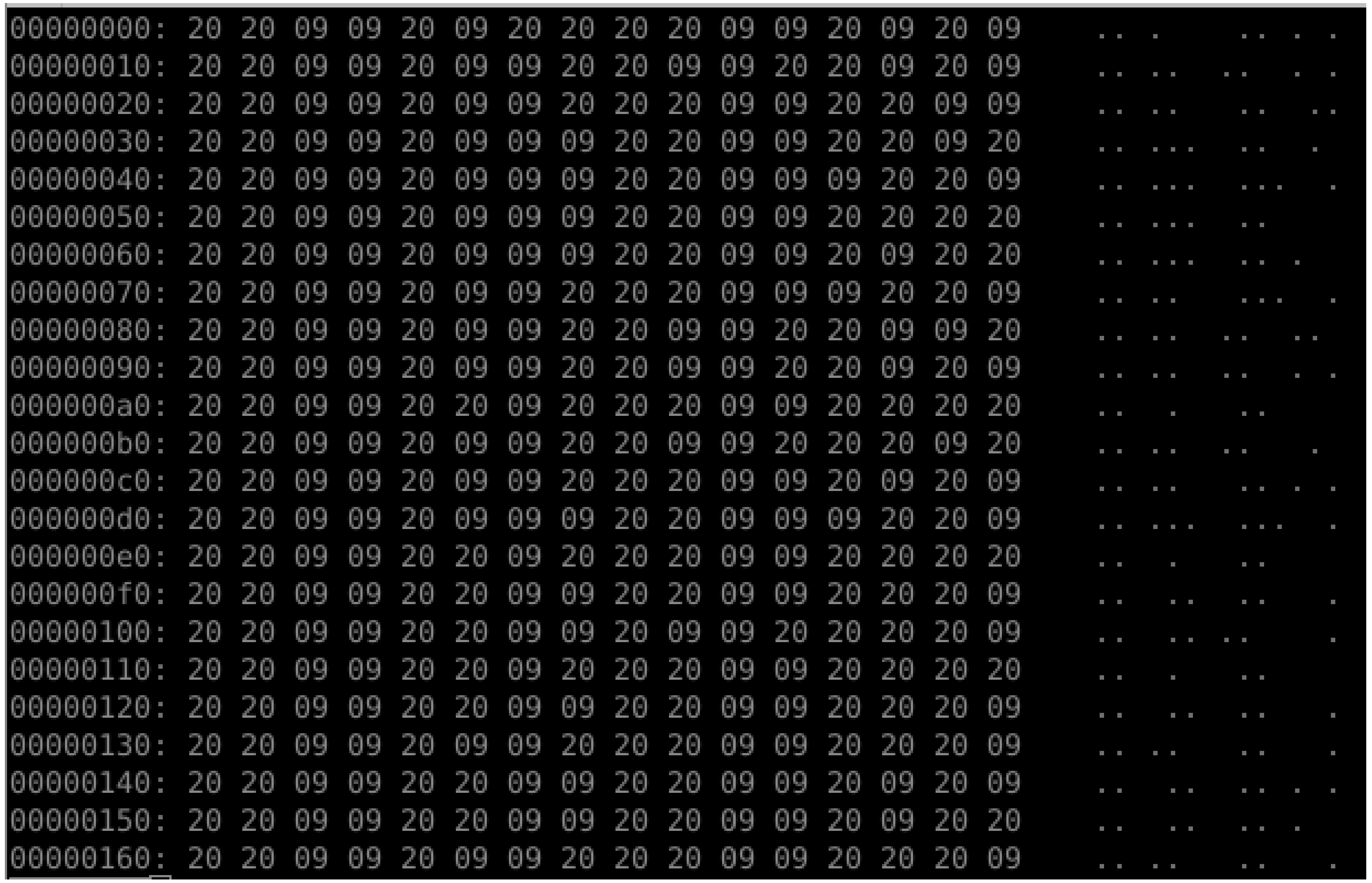
Layer Vault: Level 1

50

At Nuber, cybersecurity is our number one priority! We would never do some stupid thing, like have passwords lying around in the clear. Those things belong in password managers and key vaults!

We have a formidable security team, and our engineers are top notch. In fact, they are so good that they have built a number of highly secure key vaults. Everybody knows that when you implement your own security software it will be super tough to break, no one will know your secret algorithms or how things work on the inside.

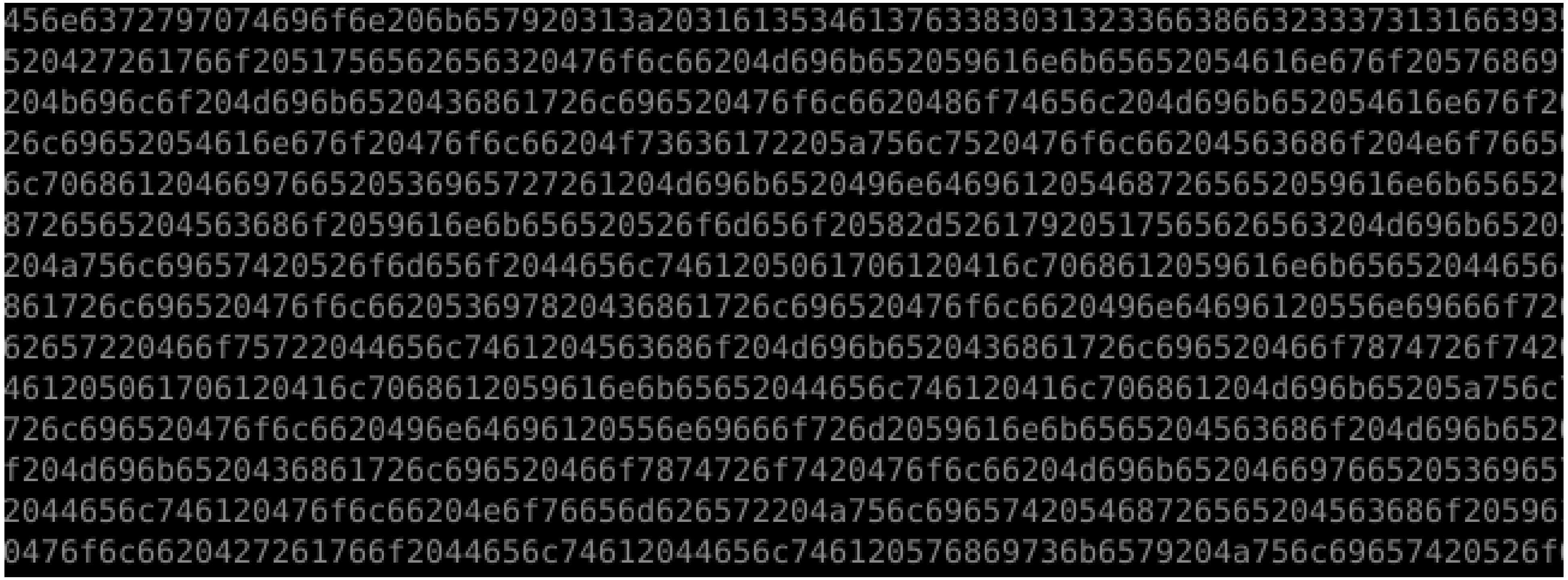
I know that you were assigned to audit vault security, but, frankly, you are wasting your time. There is no way you can retrieve encryption keys from the Layer Vault. Don't say I didn't warn you...



```
00000000: 20 20 09 09 20 09 20 20 20 20 09 09 20 09 20 09      .. .    .. . .
00000010: 20 20 09 09 20 09 09 20 20 09 09 20 20 09 20 09      .. ..  ..  . .
00000020: 20 20 09 09 20 09 09 20 20 20 09 09 20 20 09 09      .. ..   ..  ..
00000030: 20 20 09 09 20 09 09 09 20 20 09 09 20 20 09 20      .. ...  ..  . 
00000040: 20 20 09 09 20 09 09 09 20 20 09 09 09 20 20 09      .. ...  ...  .
00000050: 20 20 09 09 20 09 09 09 20 20 09 09 20 20 20 20      .. ...  ..    
00000060: 20 20 09 09 20 09 09 09 20 20 09 09 20 09 20 20      .. ...  .. .  
00000070: 20 20 09 09 20 09 09 20 20 20 09 09 09 20 20 09      .. ..   ...  .
00000080: 20 20 09 09 20 09 09 20 20 09 09 20 20 09 09 20      .. ..  ..  .. 
00000090: 20 20 09 09 20 09 09 20 20 09 09 20 20 09 20 09      .. ..  ..  . .
000000a0: 20 20 09 09 20 20 09 20 20 20 09 09 20 20 20 20      ..  .   ..    
000000b0: 20 20 09 09 20 09 09 20 20 09 09 20 20 20 09 20      .. ..  ..   . 
000000c0: 20 20 09 09 20 09 09 20 20 20 09 09 20 09 20 09      .. ..   .. . .
000000d0: 20 20 09 09 20 09 09 09 20 20 09 09 09 20 20 09      .. ...  ...  .
000000e0: 20 20 09 09 20 20 09 20 20 20 09 09 20 20 20 20      ..  .   ..    
000000f0: 20 20 09 09 20 20 09 09 20 20 09 09 20 20 20 09      ..  ..  ..   .
00000100: 20 20 09 09 20 20 09 09 20 09 09 20 20 20 20 09      ..  .. ..    .
00000110: 20 20 09 09 20 20 09 20 20 20 09 09 20 20 20 20      ..  .   ..    
00000120: 20 20 09 09 20 20 09 09 20 20 09 09 20 20 20 09      ..  ..  ..   .
00000130: 20 20 09 09 20 09 09 20 20 20 09 09 20 20 20 09      .. ..   ..   .
00000140: 20 20 09 09 20 20 09 09 20 20 09 09 20 09 20 09      ..  ..  .. . .
00000150: 20 20 09 09 20 20 09 09 20 20 09 09 20 09 20 20      ..  ..  .. .  
00000160: 20 20 09 09 20 09 09 20 20 20 09 09 20 20 20 09      .. ..   ..   .
```

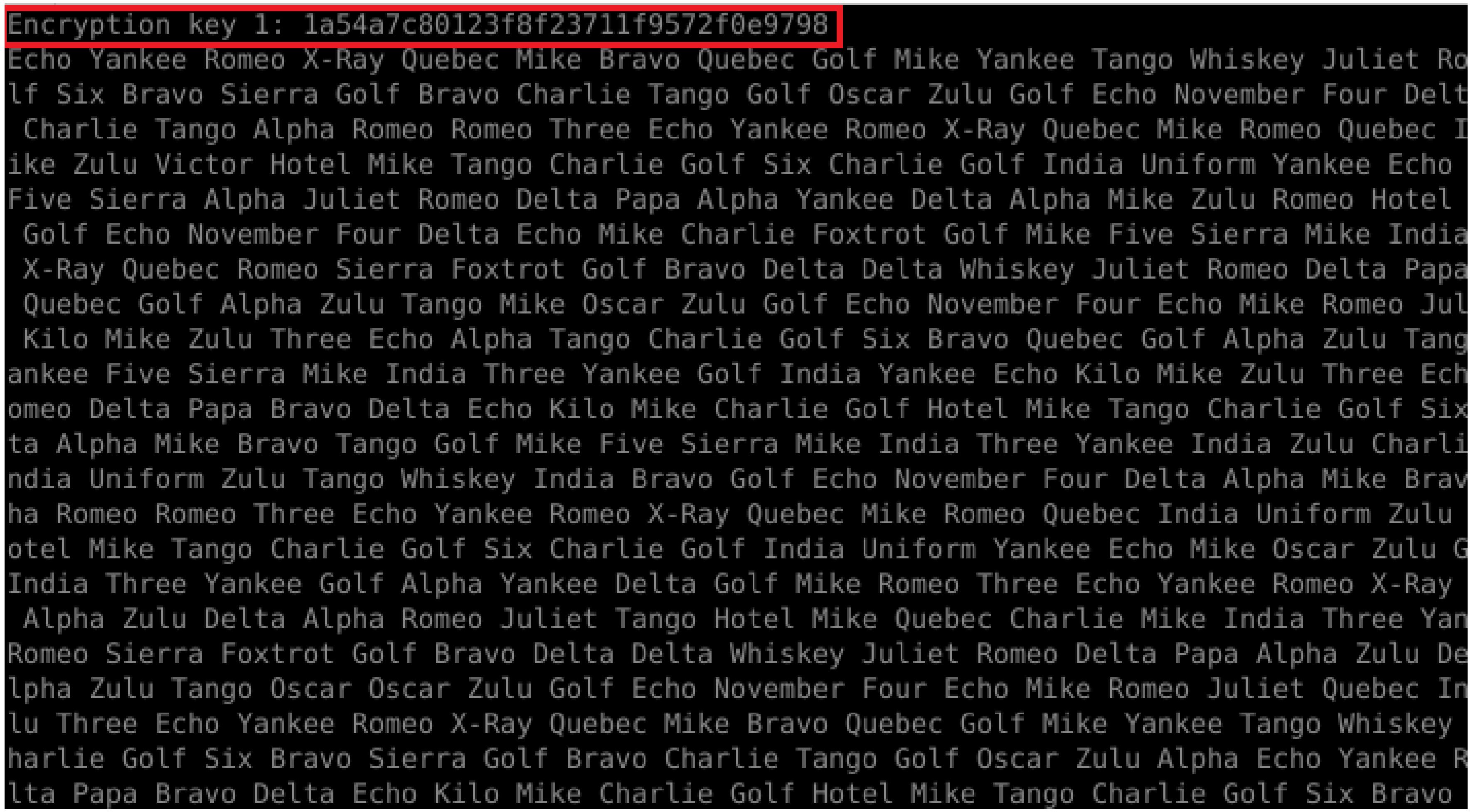
- Binary encoding

0x20 - space - Bit 0
0x09 - tab - Bit 1



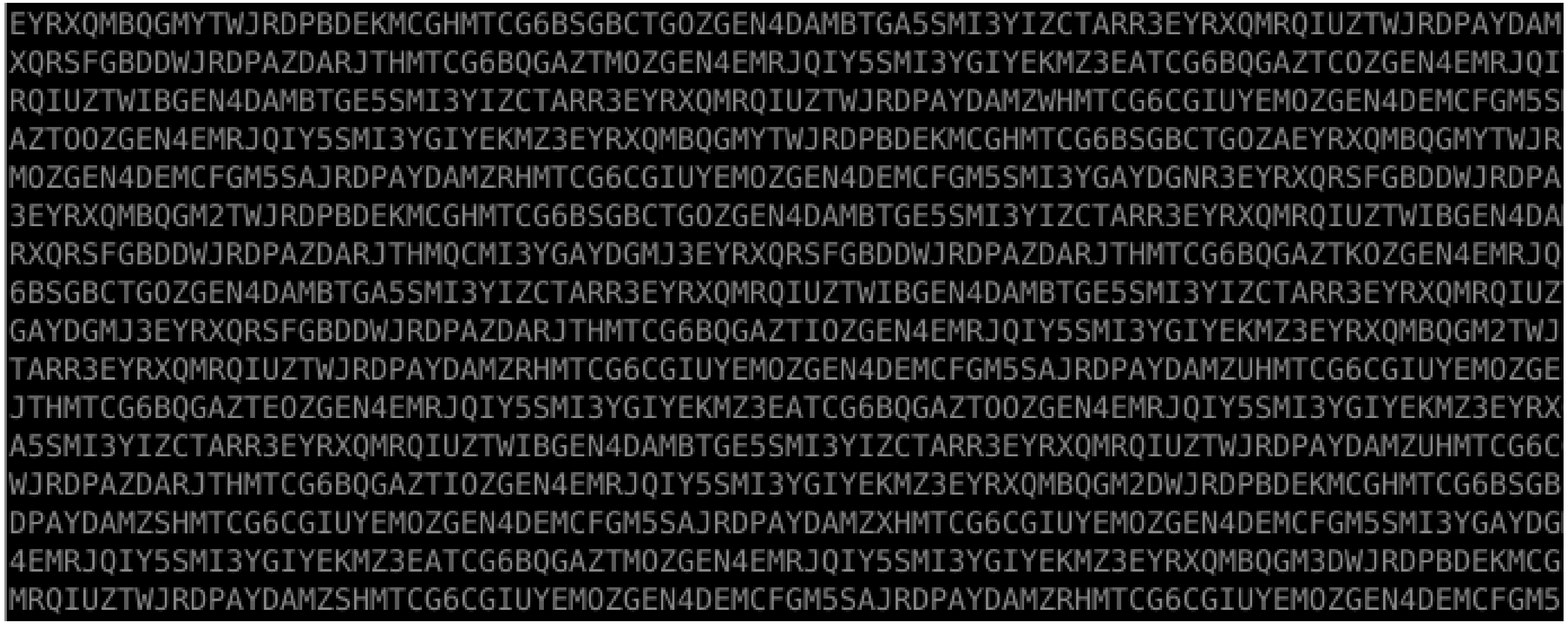
```
456e6372797074696f6e206b657920313a20316135346137633830313233663866323337313166393
520427261766f20517565626563320476f6c66204d696b652059616e6b65652054616e676f2057686
204b696c6f204d696b6520436861726c696520476f6c6620486f74656c204d696b652054616e676f2
26c69652054616e676f20476f6c66204f73636172205a756c7520476f6c66204563686f204e6f7665
6c706861204669766520536965727261204d696b6520496e646961205468726565205961e6b656520
8726565204563686f2059616e6b656520526f6d656f20582d52617920517565626563204d696b6520
204a756c69657420526f6d656f2044656c7461205061706120416c7068612059616e6b65652044656
861726c696520476f6c662053697820436861726c696520476f6c6620496e64696120556e69666f72
62657220466f75722044656c7461204563686f204d696b6520436861726c696520466f7874726f742
461205061706120416c7068612059616e6b65652044656c746120416c706861204d696b65205a756c
726c696520476f6c6620496e64696120556e69666f726d2059616e6b6565204563686f204d696b652
f204d696b6520436861726c696520466f7874726f7420476f6c66204d696b65204669766520536965
2044656c746120476f6c66204e6f76656d626572204a756c696574205468726565204563686f20596
0476f6c6620427261766f2044656c74612044656c746120576869736b6579204a756c69657420526f
```

- Hex encoding



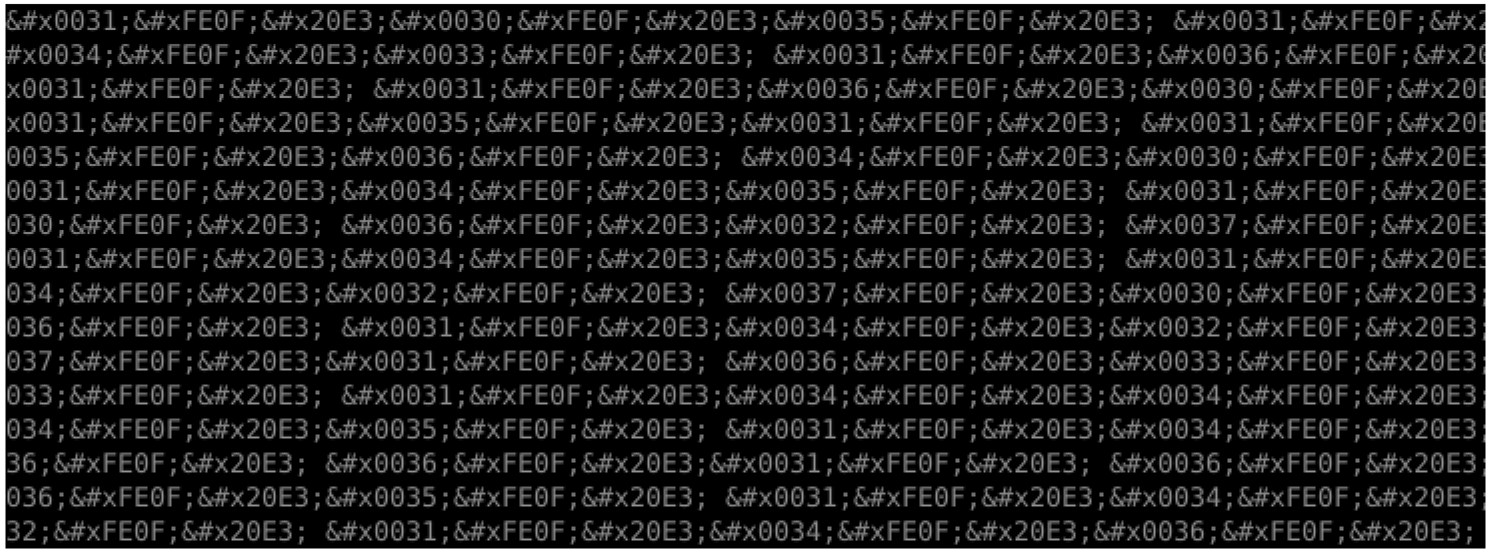
```
Encryption key 1: 1a54a7c80123f8f23711f9572f0e9798
Echo Yankee Romeo X-Ray Quebec Mike Bravo Quebec Golf Mike Yankee Tango Whiskey Juliet Ro
lf Six Bravo Sierra Golf Bravo Charlie Tango Golf Oscar Zulu Golf Echo November Four Delt
 Charlie Tango Alpha Romeo Romeo Three Echo Yankee Romeo X-Ray Quebec Mike Romeo Quebec I
ike Zulu Victor Hotel Mike Tango Charlie Golf Six Charlie Golf India Uniform Yankee Echo
Five Sierra Alpha Juliet Romeo Delta Papa Alpha Yankee Delta Alpha Mike Zulu Romeo Hotel
 Golf Echo November Four Delta Echo Mike Charlie Foxtrot Golf Mike Five Sierra Mike India
 X-Ray Quebec Romeo Sierra Foxtrot Golf Bravo Delta Delta Whiskey Juliet Romeo Delta Papa
 Quebec Golf Alpha Zulu Tango Mike Oscar Zulu Golf Echo November Four Echo Mike Romeo Jul
 Kilo Mike Zulu Three Echo Alpha Tango Charlie Golf Six Bravo Quebec Golf Alpha Zulu Tang
ankee Five Sierra Mike India Three Yankee Golf India Yankee Echo Kilo Mike Zulu Three Ech
omeo Delta Papa Bravo Delta Echo Kilo Mike Charlie Golf Hotel Mike Tango Charlie Golf Six
ta Alpha Mike Bravo Tango Golf Mike Five Sierra Mike India Three Yankee India Zulu Charli
ndia Uniform Zulu Tango Whiskey India Bravo Golf Echo November Four Delta Alpha Mike Brav
ha Romeo Romeo Three Echo Yankee Romeo X-Ray Quebec Mike Romeo Quebec India Uniform Zulu
otel Mike Tango Charlie Golf Six Charlie Golf India Uniform Yankee Echo Mike Oscar Zulu G
India Three Yankee Golf Alpha Yankee Delta Golf Mike Romeo Three Echo Yankee Romeo X-Ray
 Alpha Zulu Delta Alpha Romeo Juliet Tango Hotel Mike Quebec Charlie Mike India Three Yan
Romeo Sierra Foxtrot Golf Bravo Delta Delta Whiskey Juliet Romeo Delta Papa Alpha Zulu De
lpha Zulu Tango Oscar Oscar Zulu Golf Echo November Four Echo Mike Romeo Juliet Quebec In
lu Three Echo Yankee Romeo X-Ray Quebec Mike Bravo Quebec Golf Mike Yankee Tango Whiskey
harlie Golf Six Bravo Sierra Golf Bravo Charlie Tango Golf Oscar Zulu Alpha Echo Yankee R
lta Papa Bravo Delta Echo Kilo Mike Charlie Golf Hotel Mike Tango Charlie Golf Six Bravo
```

- NATO phonetic alphabet

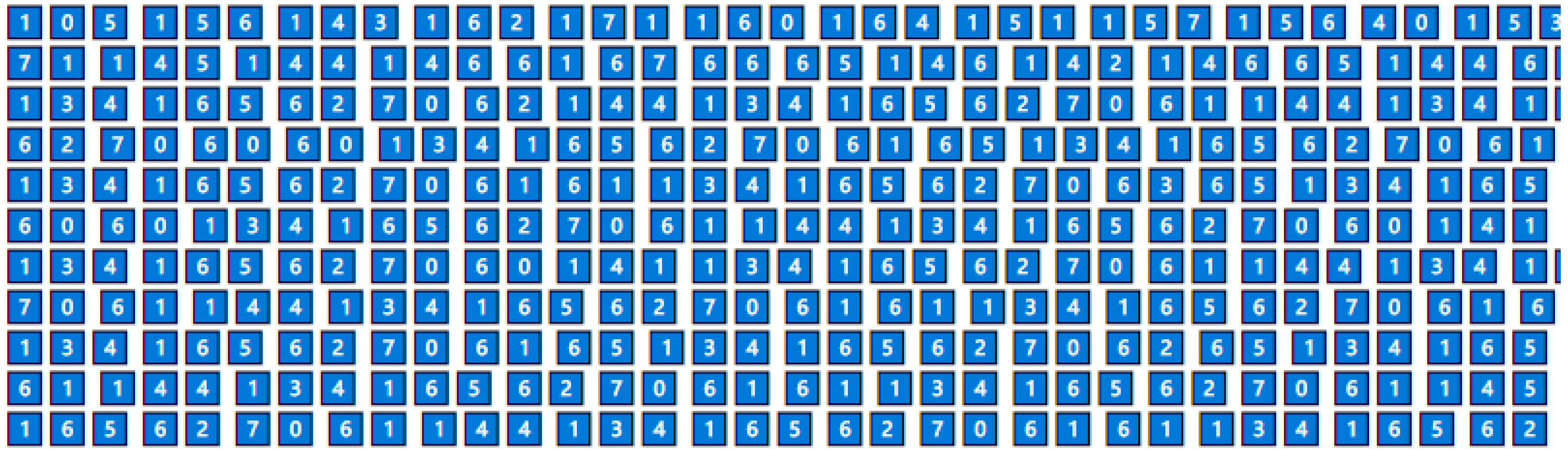


```
EYRXQMBQGMYTWJRDPBDEKMCGHMTCG6BSGBCTGOZGEN4DAMBTGA5SMI3YIZCTARR3EYRXQMRQIUZTWJRDPAYDAM
XQRSFGBDDWJRDPAZDARJTHMTCG6BQGAZTMOZGEN4EMRJQIY5SMI3YGIYEKMZ3EATCG6BQGAZTCOZGEN4EMRJQI
RQIUZTWIBGEN4DAMBTGE5SMI3YIZCTARR3EYRXQMRQIUZTWJRDPAYDAMZWHMTCG6CGIUYEMOZGEN4DEMCFGM5S
AZTOOZGEN4EMRJQIY5SMI3YGIYEKMZ3EYRXQMBQGMYTWJRDPBDEKMCGHMTCG6BSGBCTGOZAEYRXQMBQGMYTWJR
MOZGEN4DEMCFGM5SAJRDPAYDAMZRHMTCG6CGIUYEMOZGEN4DEMCFGM5SMI3YGAYDGNR3EYRXQRSFGBDDWJRDPA
3EYRXQMBQGM2TWJRDPBDEKMCGHMTCG6BSGBCTGOZGEN4DAMBTGE5SMI3YIZCTARR3EYRXQMRQIUZTWIBGEN4DA
RXQRSFGBDDWJRDPAZDARJTHMQCMI3YGAYDGMJ3EYRXQRSFGBDDWJRDPAZDARJTHMTCG6BQGAZTKOZGEN4EMRJQ
6BSGBCTGOZGEN4DAMBTGA5SMI3YIZCTARR3EYRXQMRQIUZTWIBGEN4DAMBTGE5SMI3YIZCTARR3EYRXQMRQIUZ
GAYDGMJ3EYRXQRSFGBDDWJRDPAZDARJTHMTCG6BQGAZTIOZGEN4EMRJQIY5SMI3YGIYEKMZ3EYRXQMBQGM2TWJ
TARR3EYRXQMRQIUZTWJRDPAYDAMZRHMTCG6CGIUYEMOZGEN4DEMCFGM5SAJRDPAYDAMZUHMTCG6CGIUYEMOZGE
JTHMTCG6BQGAZTEOZGEN4EMRJQIY5SMI3YGIYEKMZ3EATCG6BQGAZTOOZGEN4EMRJQIY5SMI3YGIYEKMZ3EYRX
A5SMI3YIZCTARR3EYRXQMRQIUZTWIBGEN4DAMBTGE5SMI3YIZCTARR3EYRXQMRQIUZTWJRDPAYDAMZUHMTCG6C
WJRDPAZDARJTHMTCG6BQGAZTIOZGEN4EMRJQIY5SMI3YGIYEKMZ3EYRXQMBQGM2DWJRDPBDEKMCGHMTCG6BSGB
DPAYDAMZSHMTCG6CGIUYEMOZGEN4DEMCFGM5SAJRDPAYDAMZXHMTCG6CGIUYEMOZGEN4DEMCFGM5SMI3YGAYDG
4EMRJQIY5SMI3YGIYEKMZ3EATCG6BQGAZTMOZGEN4EMRJQIY5SMI3YGIYEKMZ3EYRXQMBQGM3DWJRDPBDEKMCG
MRQIUZTWJRDPAYDAMZSHMTCG6CGIUYEMOZGEN4DEMCFGM5SAJRDPAYDAMZRHMTCG6CGIUYEMOZGEN4DEMCFGM5
```

- Base32 encoding

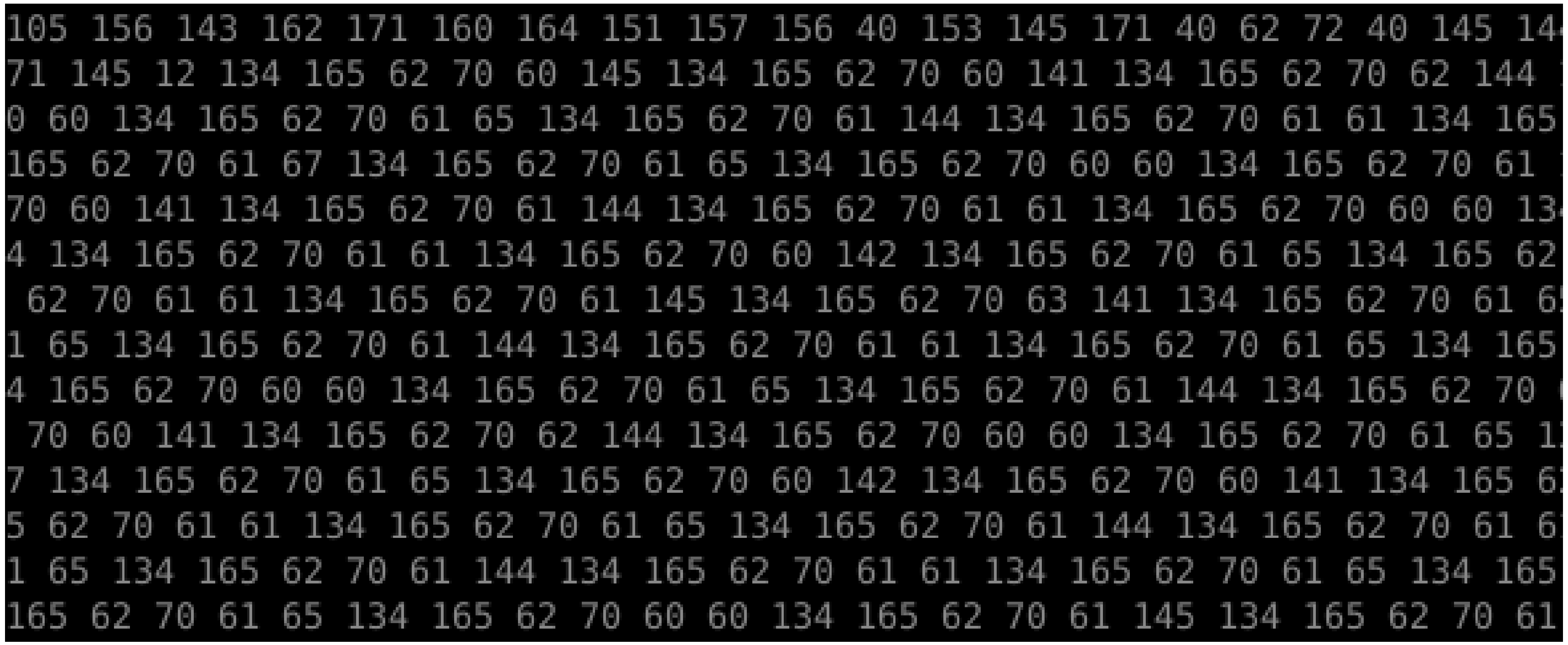


```
&#x0031;&#xFE0F;&#x20E3;&#x0030;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x2
#x0034;&#xFE0F;&#x20E3;&#x0033;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0036;&#xFE0F;&#x20
x0031;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0036;&#xFE0F;&#x20E3;&#x0030;&#xFE0F;&#x20E
x0031;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3;&#x0031;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E
0035;&#xFE0F;&#x20E3;&#x0036;&#xFE0F;&#x20E3; &#x0034;&#xFE0F;&#x20E3;&#x0030;&#xFE0F;&#x20E3
0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3
030;&#xFE0F;&#x20E3; &#x0036;&#xFE0F;&#x20E3;&#x0032;&#xFE0F;&#x20E3; &#x0037;&#xFE0F;&#x20E3
0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3
034;&#xFE0F;&#x20E3;&#x0032;&#xFE0F;&#x20E3; &#x0037;&#xFE0F;&#x20E3;&#x0030;&#xFE0F;&#x20E3;
036;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;&#x0032;&#xFE0F;&#x20E3;
037;&#xFE0F;&#x20E3;&#x0031;&#xFE0F;&#x20E3; &#x0036;&#xFE0F;&#x20E3;&#x0033;&#xFE0F;&#x20E3;
033;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;
034;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;
36;&#xFE0F;&#x20E3; &#x0036;&#xFE0F;&#x20E3;&#x0031;&#xFE0F;&#x20E3; &#x0036;&#xFE0F;&#x20E3;
036;&#xFE0F;&#x20E3;&#x0035;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;
32;&#xFE0F;&#x20E3; &#x0031;&#xFE0F;&#x20E3;&#x0034;&#xFE0F;&#x20E3;&#x0036;&#xFE0F;&#x20E3;
```



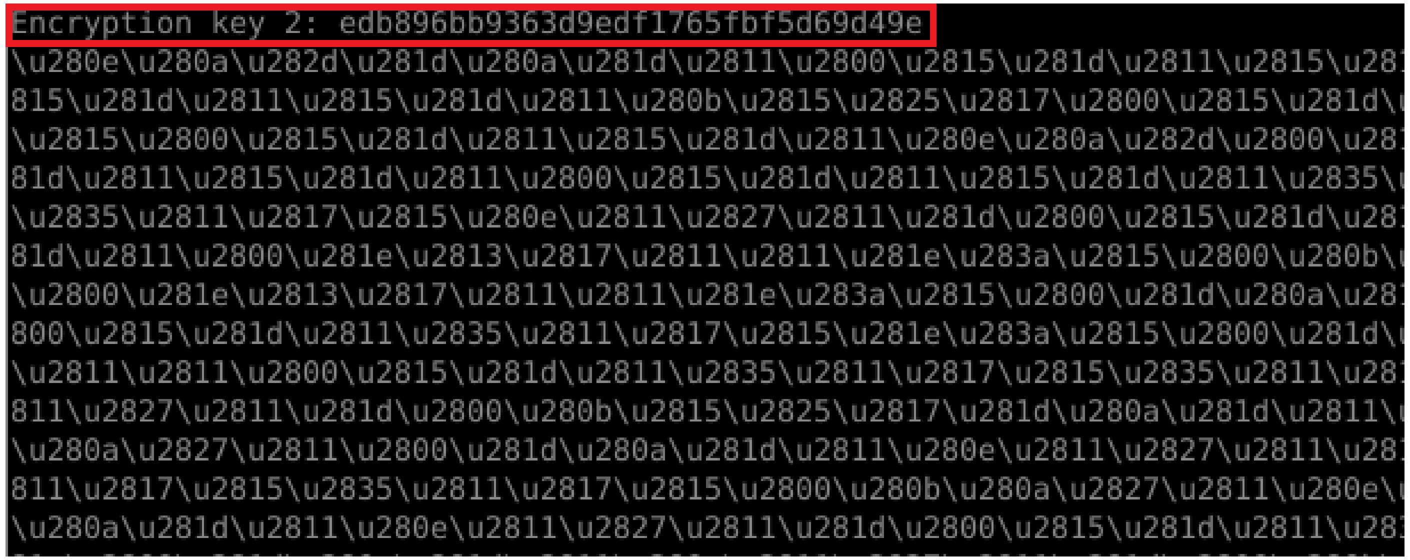
```
105 156 143 162 171 160 164 151 157 156 40 153
71 145 144 146 61 67 66 65 146 142 146 65 144 6
134 165 62 70 62 144 134 165 62 70 61 144 134 1
62 70 60 60 134 165 62 70 61 65 134 165 62 70 61
134 165 62 70 61 61 134 165 62 70 63 65 134 165
60 60 134 165 62 70 61 144 134 165 62 70 60 141
134 165 62 70 60 141 134 165 62 70 61 144 134 1
70 61 144 134 165 62 70 61 61 134 165 62 70 61 6
134 165 62 70 61 65 134 165 62 70 62 65 134 165
61 144 134 165 62 70 61 61 134 165 62 70 61 145
165 62 70 61 144 134 165 62 70 61 61 134 165 62
```

- HTML entities of emojis

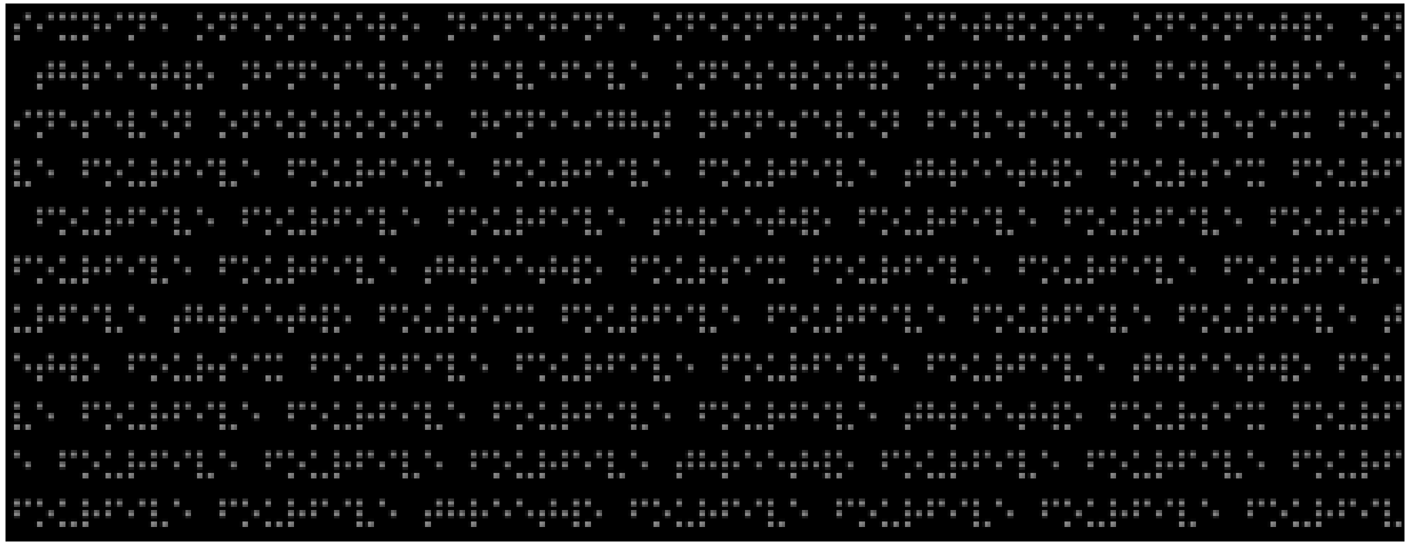


```
105 156 143 162 171 160 164 151 157 156 40 153 145 171 40 62 72 40 145 14
71 145 12 134 165 62 70 60 145 134 165 62 70 60 141 134 165 62 70 62 144
0 60 134 165 62 70 61 65 134 165 62 70 61 144 134 165 62 70 61 61 134 165
165 62 70 61 67 134 165 62 70 61 65 134 165 62 70 60 60 134 165 62 70 61
70 60 141 134 165 62 70 61 144 134 165 62 70 61 61 134 165 62 70 60 60 13
4 134 165 62 70 61 61 134 165 62 70 60 142 134 165 62 70 61 65 134 165 62
 62 70 61 61 134 165 62 70 61 145 134 165 62 70 63 141 134 165 62 70 61 6
1 65 134 165 62 70 61 144 134 165 62 70 61 61 134 165 62 70 61 65 134 165
4 165 62 70 60 60 134 165 62 70 61 65 134 165 62 70 61 144 134 165 62 70
 70 60 141 134 165 62 70 62 144 134 165 62 70 60 60 134 165 62 70 61 65 1
7 134 165 62 70 61 65 134 165 62 70 60 142 134 165 62 70 60 141 134 165 6
5 62 70 61 61 134 165 62 70 61 65 134 165 62 70 61 144 134 165 62 70 61 6
1 65 134 165 62 70 61 144 134 165 62 70 61 61 134 165 62 70 61 65 134 165
165 62 70 61 65 134 165 62 70 60 60 134 165 62 70 61 145 134 165 62 70 61
```

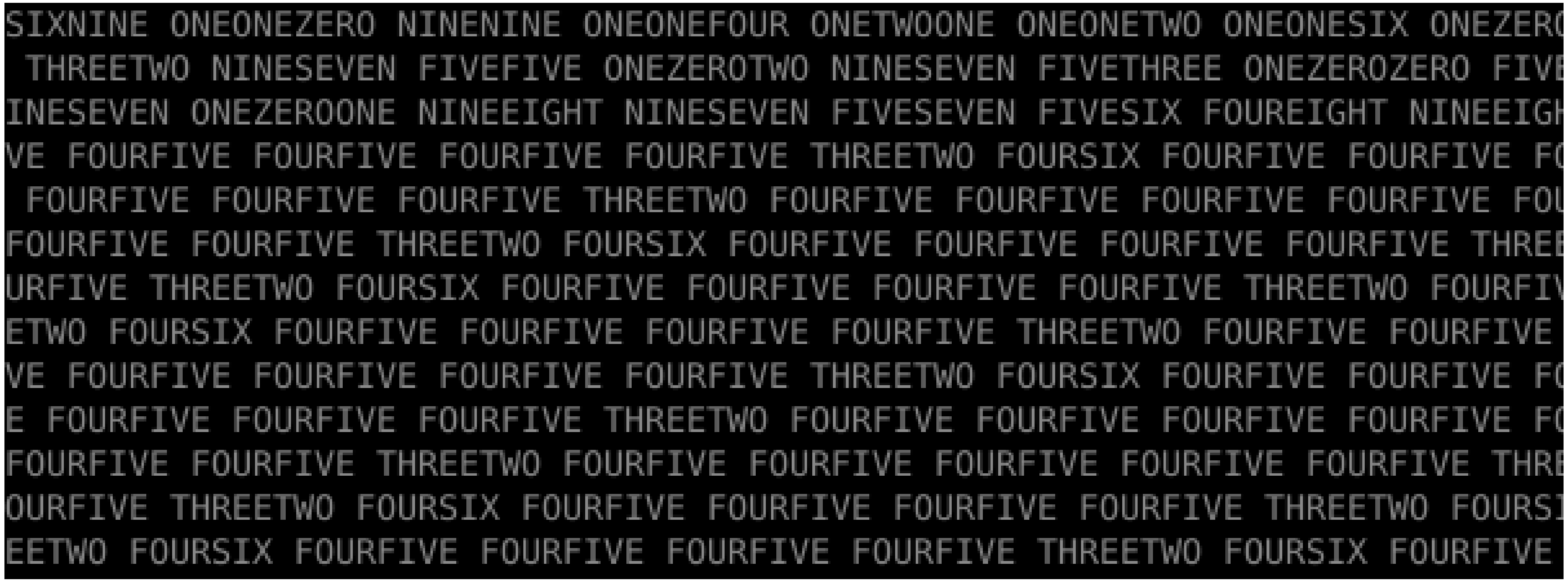
- Octal numbers



```
Encryption key 2: edb896bb9363d9edf1765fbf5d69d49e
\u280e\u280a\u282d\u281d\u280a\u281d\u2811\u2800\u2815\u281d\u2811\u2815\u281
815\u281d\u2811\u2815\u281d\u2811\u280b\u2815\u2825\u2817\u2800\u2815\u281d\u
\u2815\u2800\u2815\u281d\u2811\u2815\u281d\u2811\u280e\u280a\u282d\u2800\u281
81d\u2811\u2815\u281d\u2811\u2800\u2815\u281d\u2811\u2815\u281d\u2811\u2835\u
\u2835\u2811\u2817\u2815\u280e\u2811\u2827\u2811\u281d\u2800\u2815\u281d\u281
81d\u2811\u2800\u281e\u2813\u2817\u2811\u2811\u281e\u283a\u2815\u2800\u280b\u
\u2800\u281e\u2813\u2817\u2811\u2811\u281e\u283a\u2815\u2800\u281d\u280a\u281
800\u2815\u281d\u2811\u2835\u2811\u2817\u2815\u281e\u283a\u2815\u2800\u281d\u
\u2811\u2811\u2800\u2815\u281d\u2811\u2835\u2811\u2817\u2815\u2835\u2811\u281
811\u2827\u2811\u281d\u2800\u280b\u2815\u2825\u2817\u281d\u280a\u281d\u2811\u
\u280a\u2827\u2811\u2800\u281d\u280a\u281d\u2811\u280e\u2811\u2827\u2811\u281
811\u2817\u2815\u2835\u2811\u2817\u2815\u2800\u280b\u280a\u2827\u2811\u280e\u
\u280a\u281d\u2811\u280e\u2811\u2827\u2811\u281d\u2800\u2815\u281d\u2811\u281
```

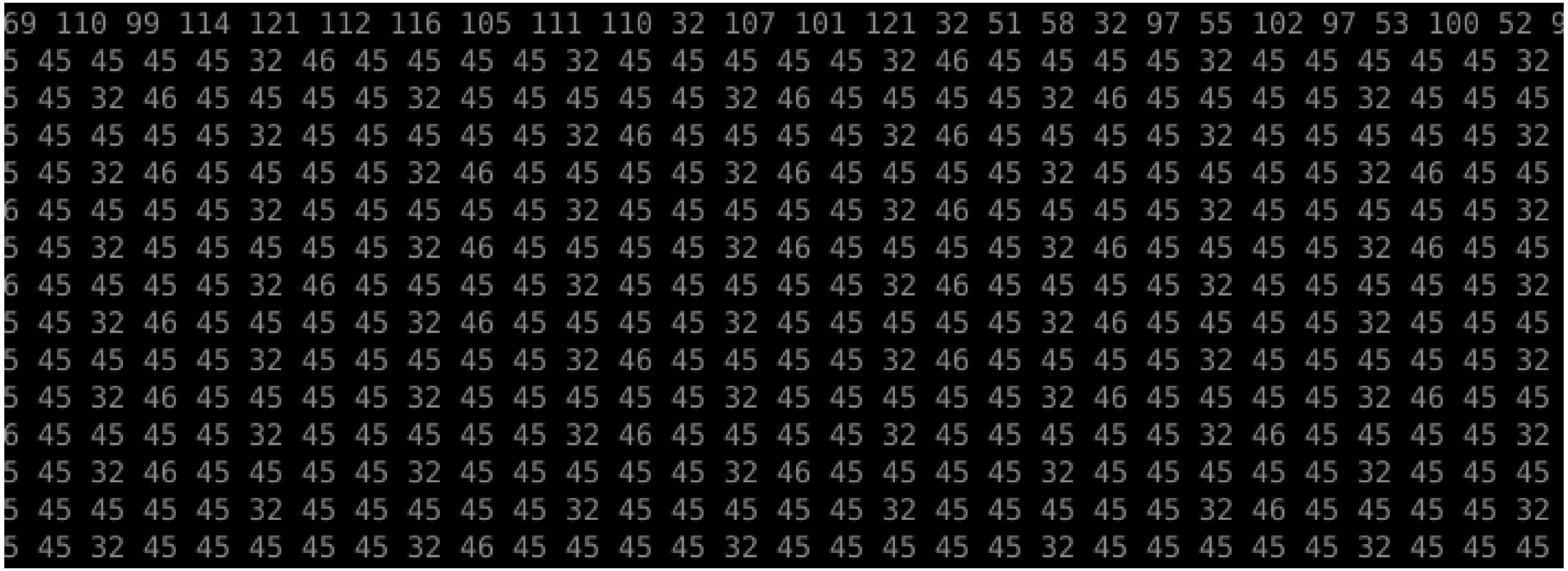


- Braille



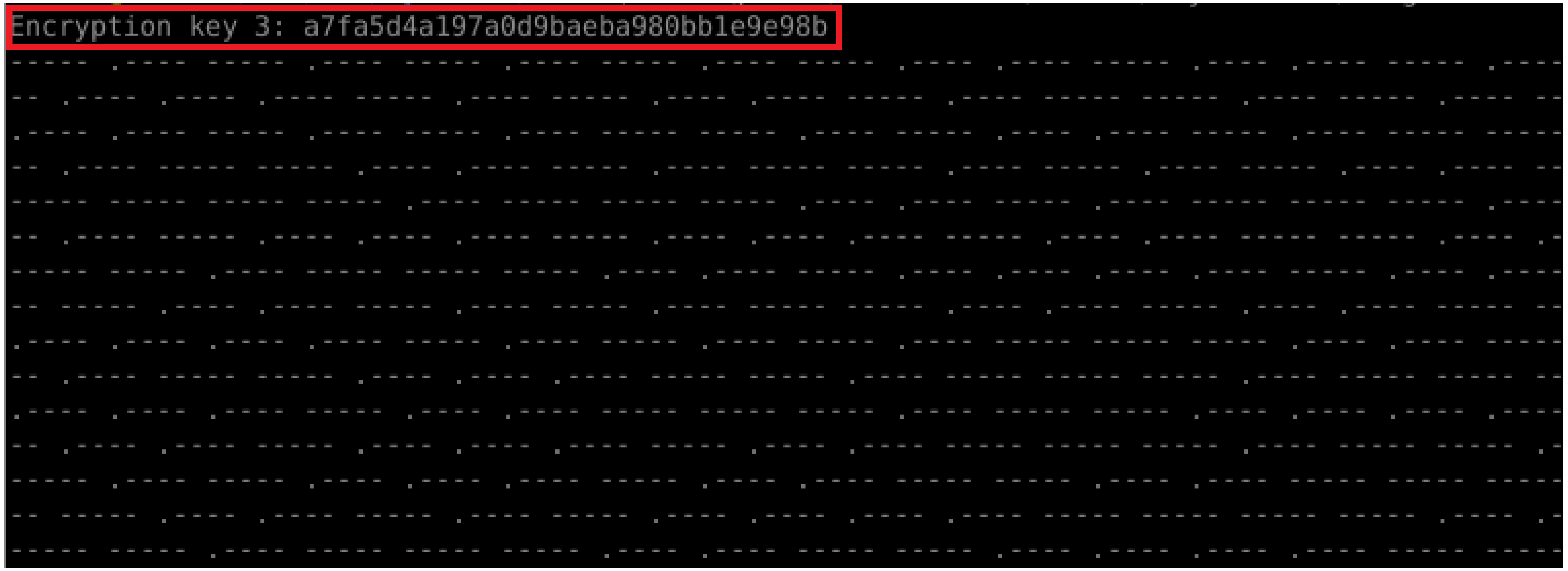
```
SIXNINE ONEONEZERO NINENINE ONEONEFOUR ONETWOONE ONEONETWO ONEONESIX ONEZER
 THREETWO NINESEVEN FIVEFIVE ONEZEROTWO NINESEVEN FIVETHREE ONEZEROZERO FIVE
INESEVEN ONEZEROONE NINEEIGHT NINESEVEN FIVESEVEN FIVESIX FOUREIGHT NINEEIGH
VE FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURSIX FOURFIVE FOURFIVE FO
 FOURFIVE FOURFIVE FOURFIVE THREETWO FOURFIVE FOURFIVE FOURFIVE FOURFIVE FOU
FOURFIVE FOURFIVE THREETWO FOURSIX FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREE
URFIVE THREETWO FOURSIX FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURFIV
ETWO FOURSIX FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURFIVE FOURFIVE
VE FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURSIX FOURFIVE FOURFIVE FO
E FOURFIVE FOURFIVE FOURFIVE THREETWO FOURFIVE FOURFIVE FOURFIVE FOURFIVE FO
FOURFIVE FOURFIVE THREETWO FOURFIVE FOURFIVE FOURFIVE FOURFIVE FOURFIVE THRE
OURFIVE THREETWO FOURSIX FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURSI
EETWO FOURSIX FOURFIVE FOURFIVE FOURFIVE FOURFIVE THREETWO FOURSIX FOURFIVE
```

- Numbers as words



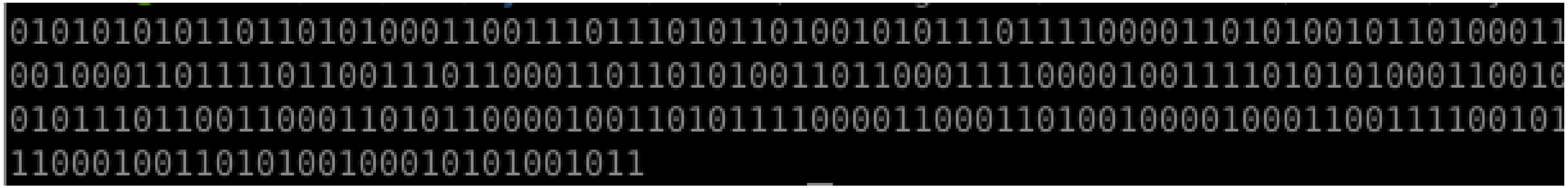
```
69 110 99 114 121 112 116 105 111 110 32 107 101 121 32 51 58 32 97 55 102 97 53 100 52 9
5 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32
5 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 45 45 45
5 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32
5 45 32 46 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45
6 45 45 45 45 32 45 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32
5 45 32 45 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 46 45 45
6 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32
5 45 32 46 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45
5 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32
5 45 32 46 45 45 45 45 32 45 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 46 45 45
6 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32
5 45 32 46 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 45 45 45
5 45 45 45 45 32 45 45 45 45 45 32 45 45 45 45 45 32 45 45 45 45 45 32 46 45 45 45 45 32
5 45 32 45 45 45 45 45 32 46 45 45 45 45 32 45 45 45 45 45 32 45 45 45 45 45 32 45 45 45
```

- Decimal



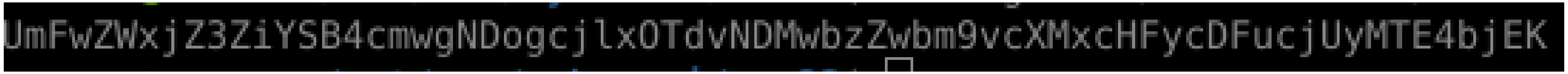
```
Encryption key 3: a7fa5d4a197a0d9baeba980bb1e9e98b
----- .---- ----- .---- ----- .---- ----- .---- ----- .---- .---- ----- .---- .---- ----- .----
-- .---- .---- .---- ----- .---- ----- .---- .---- ----- .---- ----- ----- .---- ----- .---- --
.---- .---- ----- .---- ----- .---- ----- ----- .---- ----- .---- .---- ----- .---- ----- -----
-- .---- ----- ----- .---- .---- ----- .---- ----- ----- .---- ----- .---- ----- .---- .---- --
----- ----- ----- ----- .---- ----- ----- ----- .---- .---- ----- .---- ----- ----- ----- .----
-- .---- ----- .---- .---- .---- ----- .---- .---- .---- ----- .---- .---- ----- ----- .---- .-
----- ----- .---- ----- ----- ----- .---- .---- ----- .---- .---- .---- .---- ----- .---- .----
-- ----- .---- .---- ----- .---- ----- .---- ----- ----- .---- .---- ----- .---- .---- ----- --
.---- .---- .---- .---- ----- .---- ----- .---- ----- .---- ----- ----- ----- .---- .---- -----
-- .---- ----- ----- .---- .---- .---- ----- ----- .---- ----- ----- ----- .---- ----- ----- --
.---- .---- .---- ----- .---- .---- ----- ----- ----- .---- ----- ----- .---- .---- .---- .----
-- .---- .---- ----- .---- .---- .---- ----- .---- .---- ----- ----- ----- .---- ----- ----- .-
----- .---- ----- .---- .---- .---- ----- .---- .---- ----- ----- .---- .---- ----- ----- -----
-- ----- .---- .---- ----- .---- ----- .---- .---- .---- .---- ----- ----- ----- ----- .---- .-
----- ----- .---- ----- ----- ----- .---- .---- ----- ----- .---- .---- .---- .---- ----- -----
```

- Morse code



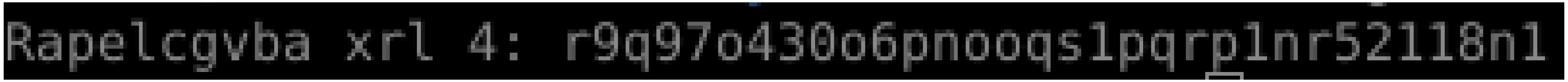
```
0101010101101101010001100111011101011010010101110111100001101010010110100011
0010001101111011001110110001101101010011011000111100001001111010101000110010
0101110110011000110101100001001101011110000110001101001000010001100111100101
1100010011010100100010101001011
```

- Binary



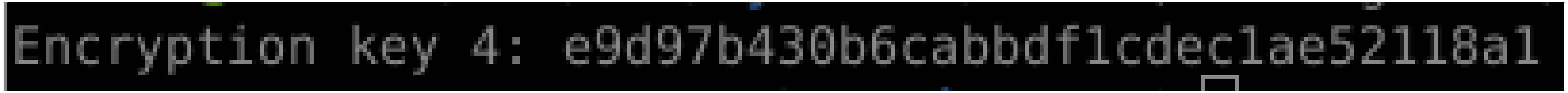
UmFwZWxjZ3ZiYSB4cmwgNDogcjlxOTdvNDMwbzZwbm9vcXMxcHFycDFucjUyMTE4bjEK

- Base64 encoding

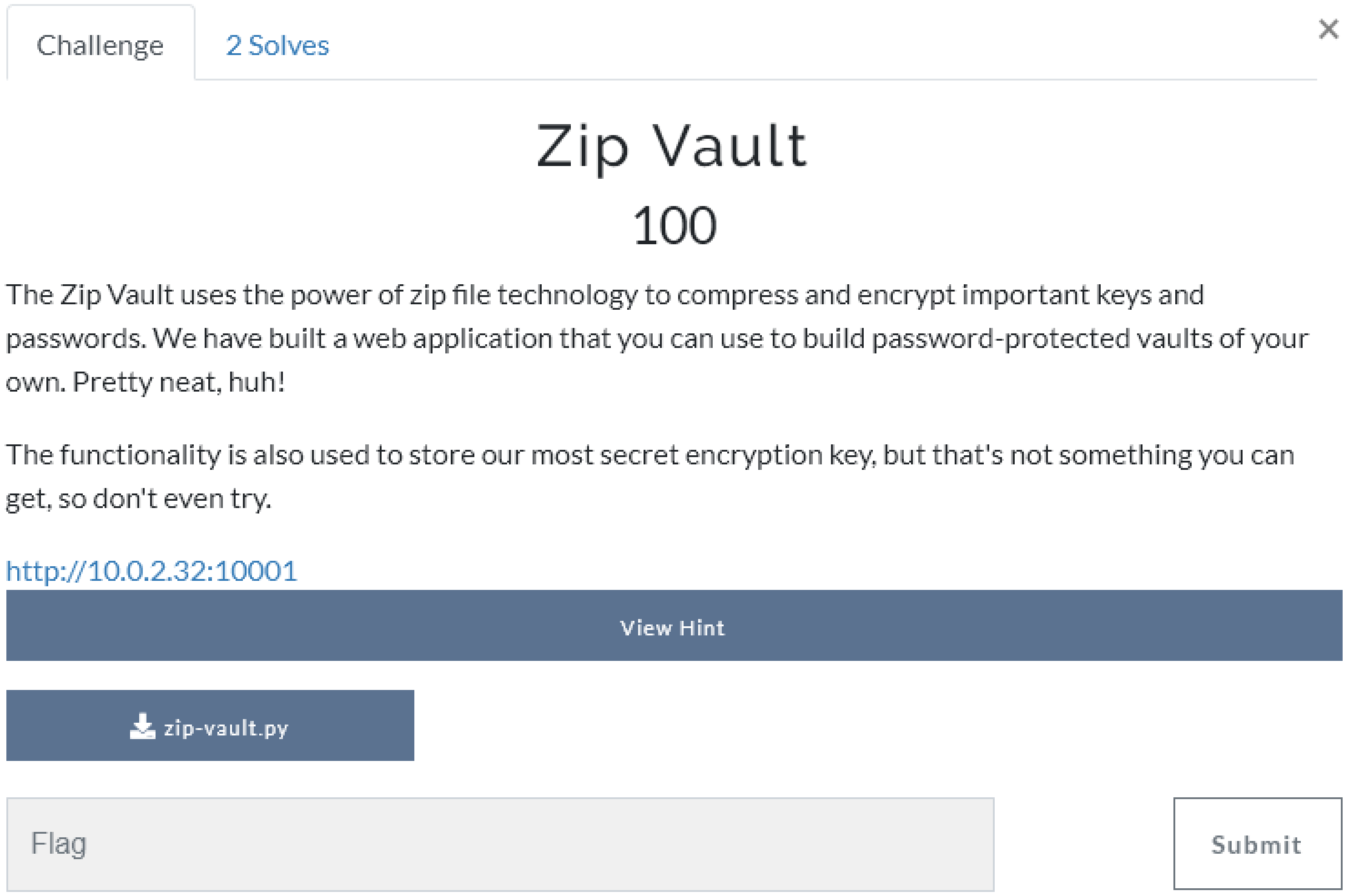


```
Rapelcgvba xrl 4: r9q97o430o6pnooqs1pqrp1nr52118n1
```

- Rot13



```
Encryption key 4: e9d97b430b6cabbdf1cdec1ae52118a1
```



Challenge 2 Solves

Zip Vault

100

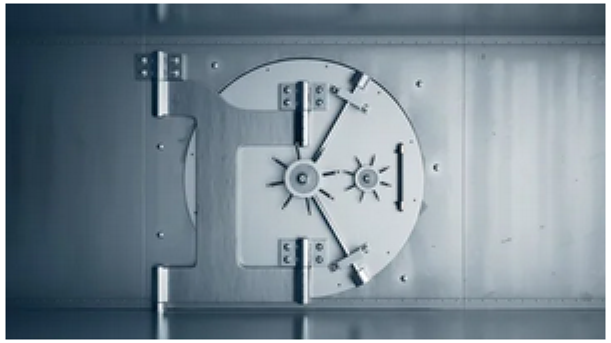
The Zip Vault uses the power of zip file technology to compress and encrypt important keys and passwords. We have built a web application that you can use to build password-protected vaults of your own. Pretty neat, huh!

The functionality is also used to store our most secret encryption key, but that's not something you can get, so don't even try.

http://10.0.2.32:10001

Zip Vault

Just zip it...



Create Vault

Vault key:

Vault contents:

Create

Open Vault

Vault key:

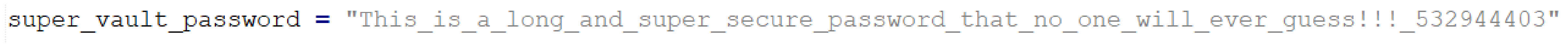
Vault: Browse... No file selected.

Open

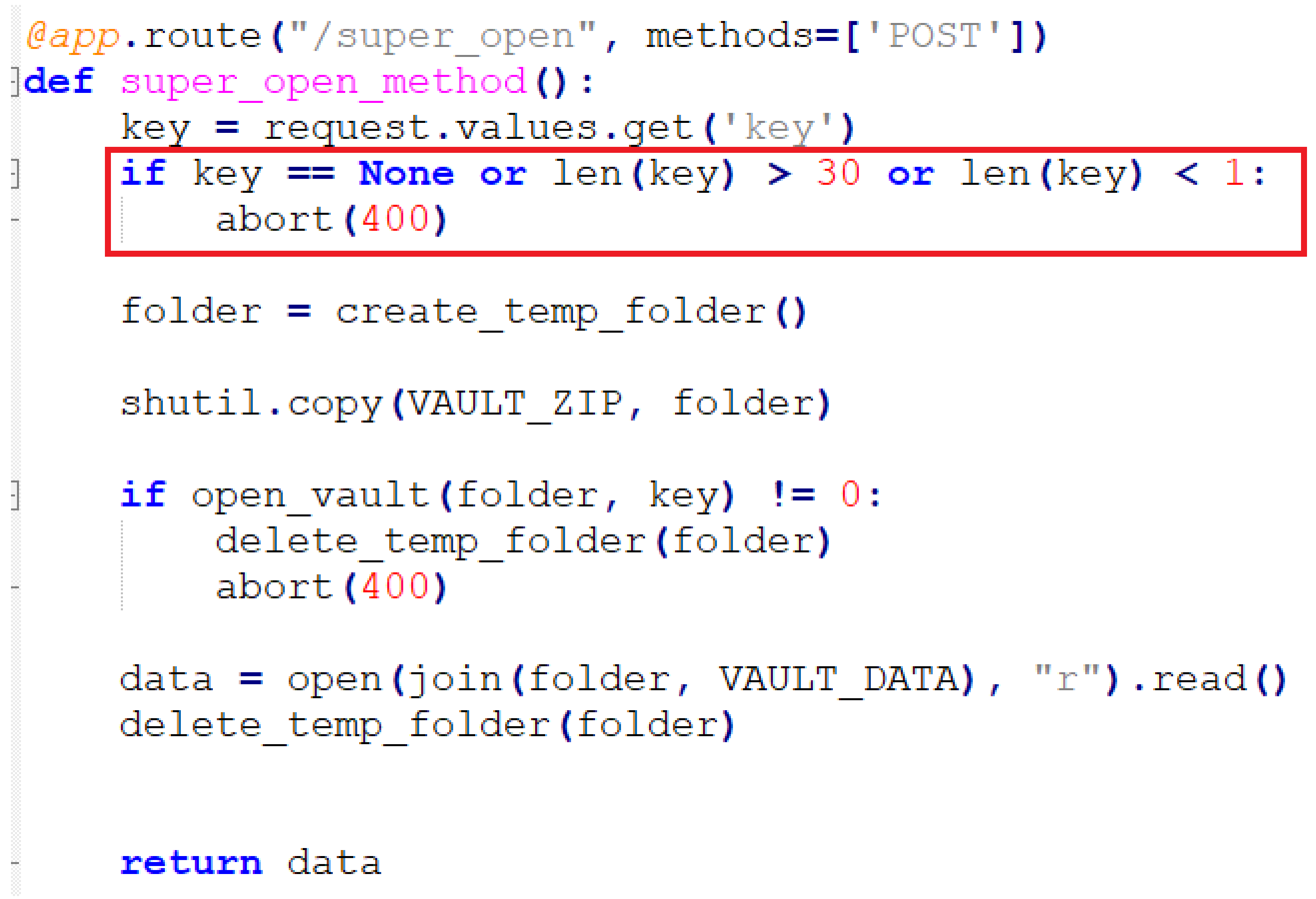
Open Super Vault

Vault key:

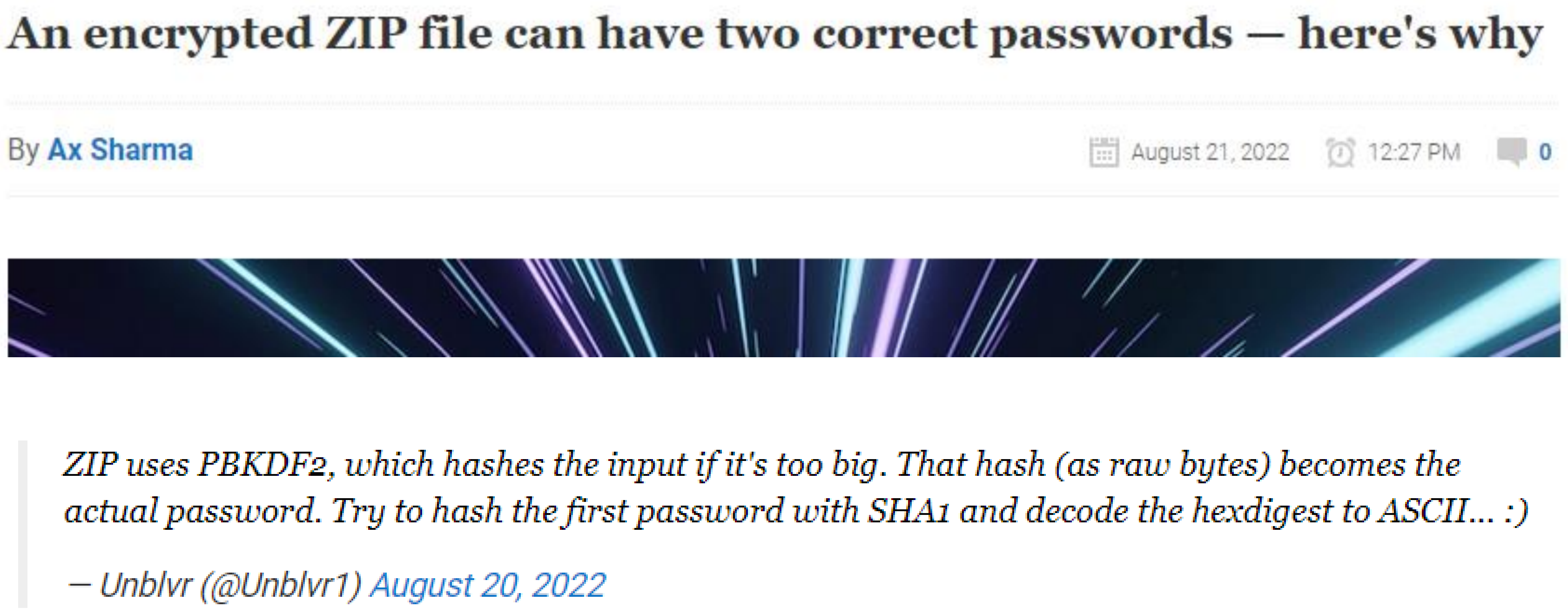
Open



```
super_vault_password = "This_is_a_long_and_super_secure_password_that_no_one_will_ever_guess!!!_532944403"
```



```
@app.route("/super_open", methods=['POST'])
def super_open_method():
    key = request.values.get('key')
    if key == None or len(key) > 30 or len(key) < 1:
        abort(400)

    folder = create_temp_folder()

    shutil.copy(VAULT_ZIP, folder)

    if open_vault(folder, key) != 0:
        delete_temp_folder(folder)
        abort(400)

    data = open(join(folder, VAULT_DATA), "r").read()
    delete_temp_folder(folder)


    return data
```



An encrypted ZIP file can have two correct passwords — here's why

By Ax Sharma August 21, 2022 12:27 PM 0

> *ZIP uses PBKDF2, which hashes the input if it's too big. That hash (as raw bytes) becomes the actual password. Try to hash the first password with SHA1 and decode the hexdigest to ASCII... :)*
>
> — Unblvr (@Unblvr1) August 20, 2022

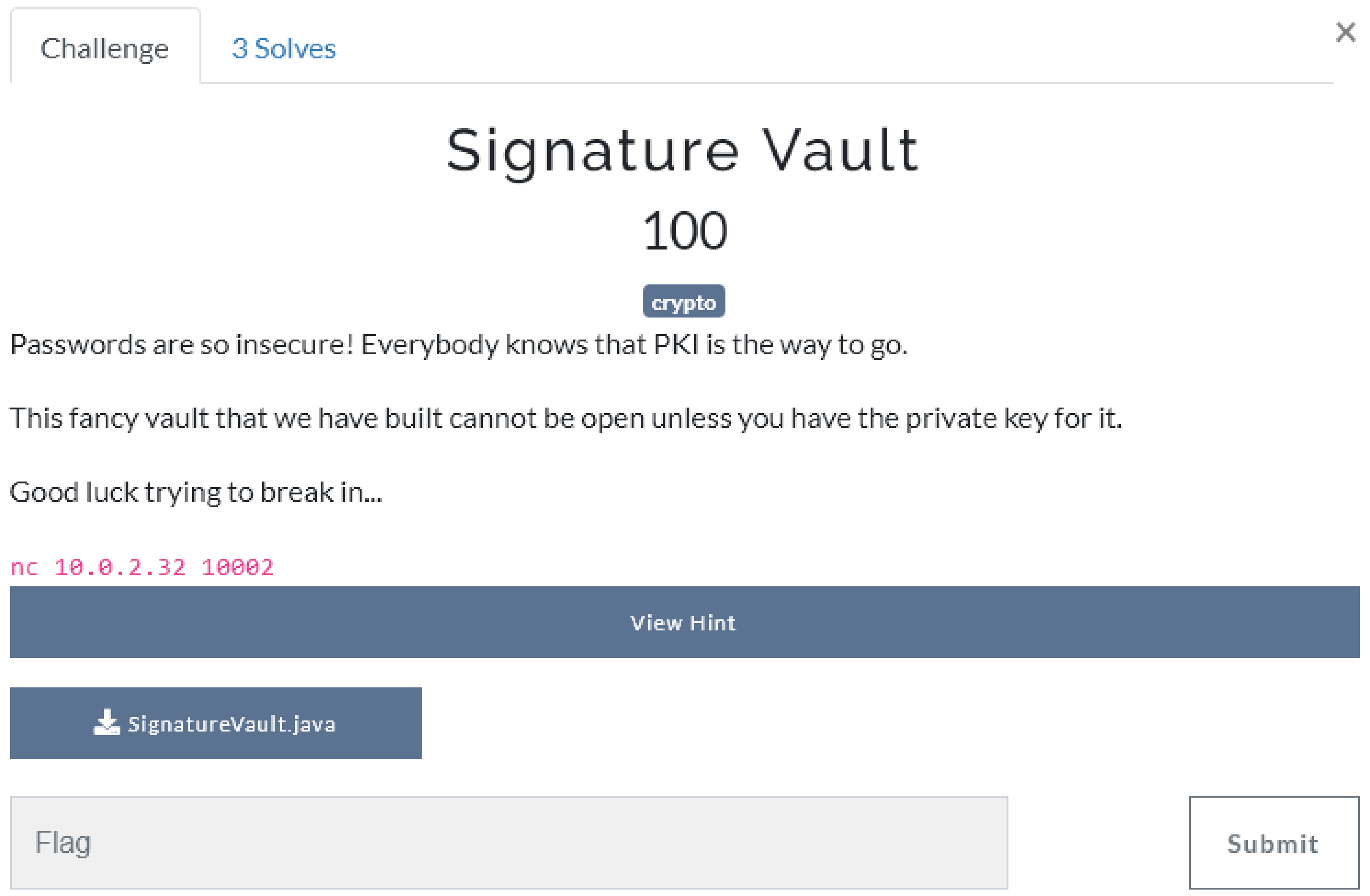
SHA1(**"This_is_a_long_and_super_secure_password_that_no_one_will_ever_guess!!!_532944403"**) =

0x7037603064505b4f49253e4354363d6966484c5b =

"p7`0dP[OI%>CT6=ifHL["

- Unintended bug - found by **@SteakEnthusiast | Will**

- Can be solved by uploading a zip file with **vault.dat** that is a soft link to **"/vault.dat"**



Challenge 3 Solves

Signature Vault

100

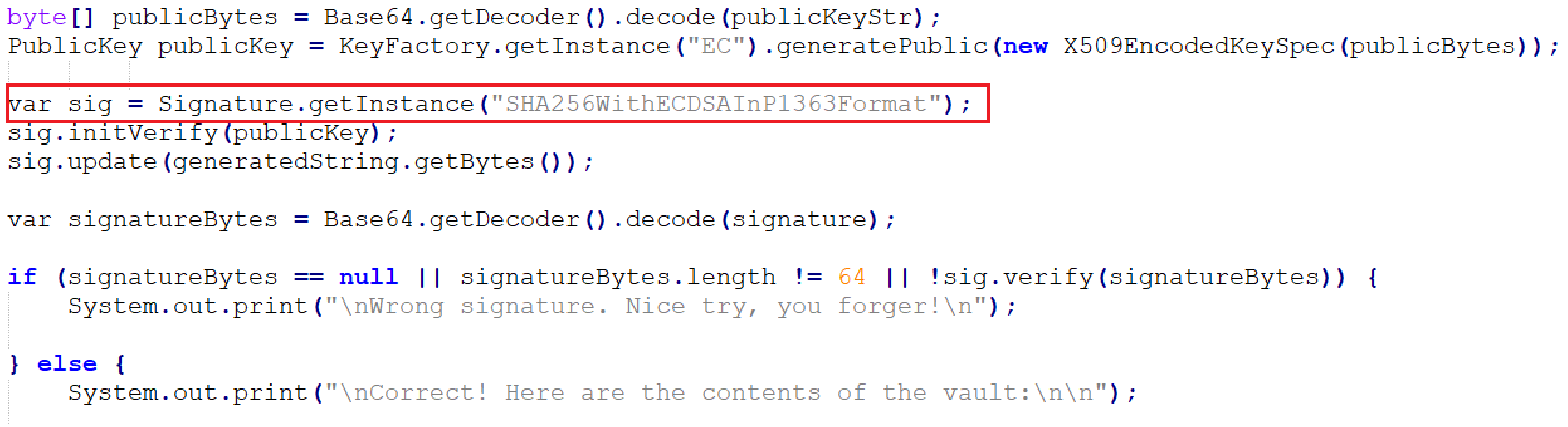
crypto

Passwords are so insecure! Everybody knows that PKI is the way to go.

This fancy vault that we have built cannot be open unless you have the private key for it.

Good luck trying to break in...

```
nc 10.0.2.32 10002
```



```java
byte[] publicBytes = Base64.getDecoder().decode(publicKeyStr);
PublicKey publicKey = KeyFactory.getInstance("EC").generatePublic(new X509EncodedKeySpec(publicBytes));

var sig = Signature.getInstance("SHA256WithECDSAInP1363Format");
sig.initVerify(publicKey);
sig.update(generatedString.getBytes());

var signatureBytes = Base64.getDecoder().decode(signature);

if (signatureBytes == null || signatureBytes.length != 64 || !sig.verify(signatureBytes)) {
    System.out.print("\nWrong signature. Nice try, you forger!\n");

} else {
    System.out.print("\nCorrect! Here are the contents of the vault:\n\n");
```

CVE-2022-21449: Psychic Signatures in Java

Neil Madden

19 April, 2022

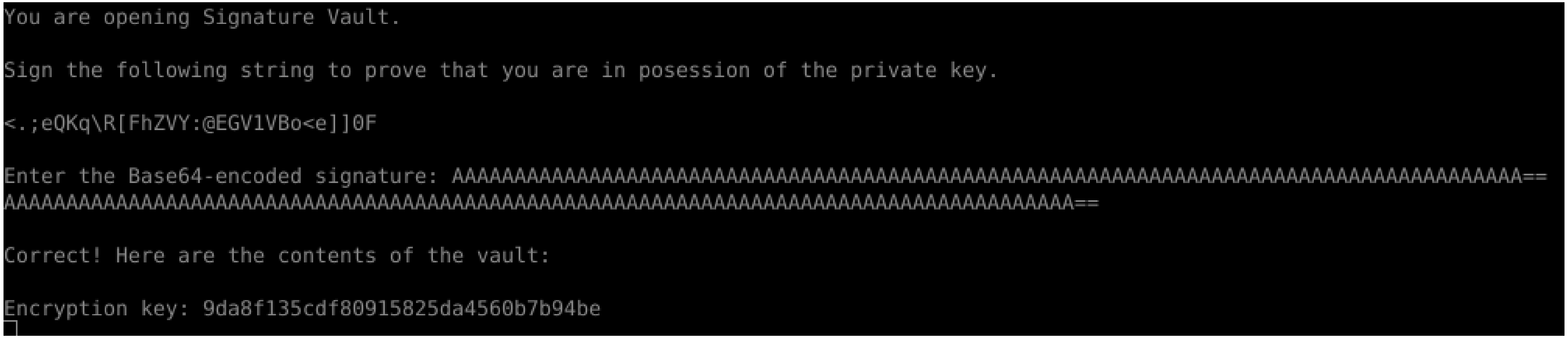
cryptography, Security

API security, cryptography, Java, jose, jwt, web-security

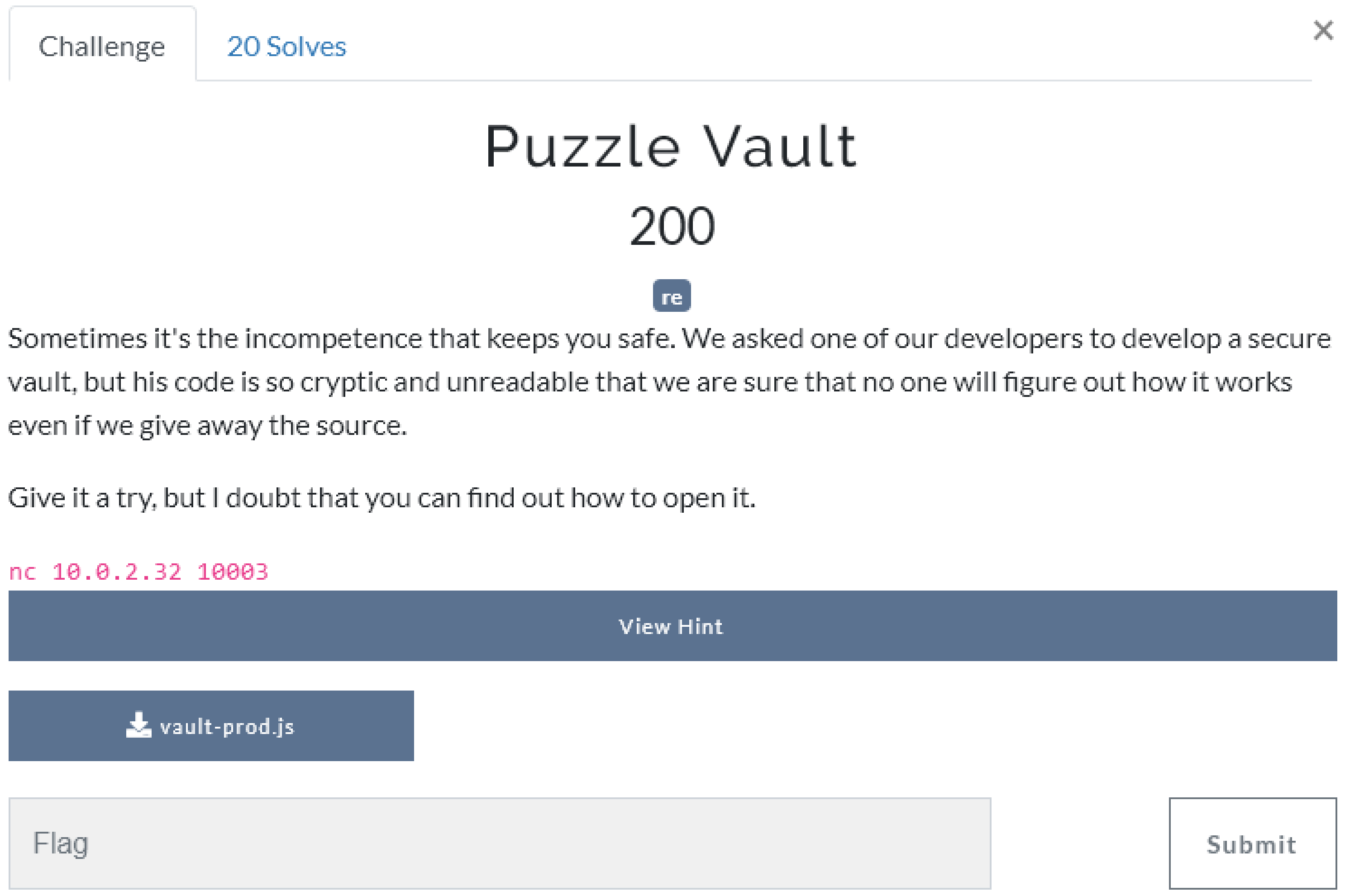
The long-running BBC sci-fi show *Doctor Who* has a recurring plot device where the Doctor manages to get out of trouble by showing an identity card which is actually completely blank. Of course, this being Doctor Who, the card is really made out of a special "psychic paper", which causes the person looking at it to see whatever the Doctor wants them to see: a security pass, a warrant, or whatever.

"Looks legit to me. Hic!"

```
|  Welcome to JShell -- Version 17.0.1
|  For an introduction type: /help intro
jshell> import java.security.*
jshell> var keys =
KeyPairGenerator.getInstance("EC").generateKeyPair()
keys ==> java.security.KeyPair@626b2d4a
jshell> var blankSignature = new byte[64]
blankSignature ==> byte[64] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... ,
0, 0, 0, 0, 0, 0, 0, 0 }
jshell> var sig =
Signature.getInstance("SHA256WithECDSAInP1363Format")
sig ==> Signature object: SHA256WithECDSAInP1363Format<not initialized>
jshell> sig.initVerify(keys.getPublic())
jshell> sig.update("Hello, World".getBytes())
jshell> sig.verify(blankSignature)
$8 ==> true
// Oops, that shouldn't have verified...
```



```
You are opening Signature Vault.

Sign the following string to prove that you are in posession of the private key.

<.;eQKq\R[FhZVY:@EGV1VBo<e]]0F

Enter the Base64-encoded signature: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==

Correct! Here are the contents of the vault:

Encryption key: 9da8f135cdf80915825da4560b7b94be
```



Puzzle Vault

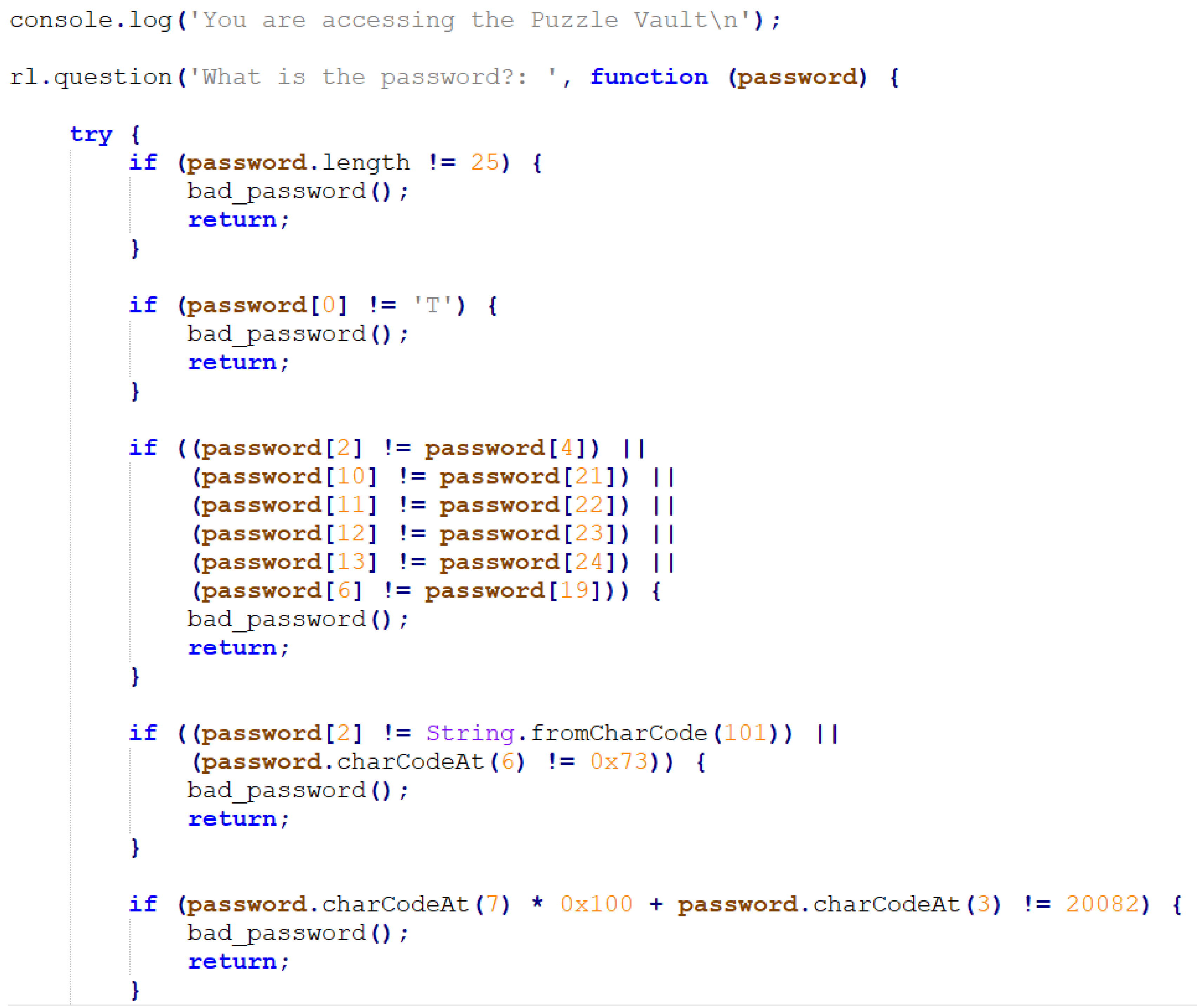
200

Sometimes it's the incompetence that keeps you safe. We asked one of our developers to develop a secure vault, but his code is so cryptic and unreadable that we are sure that no one will figure out how it works even if we give away the source.

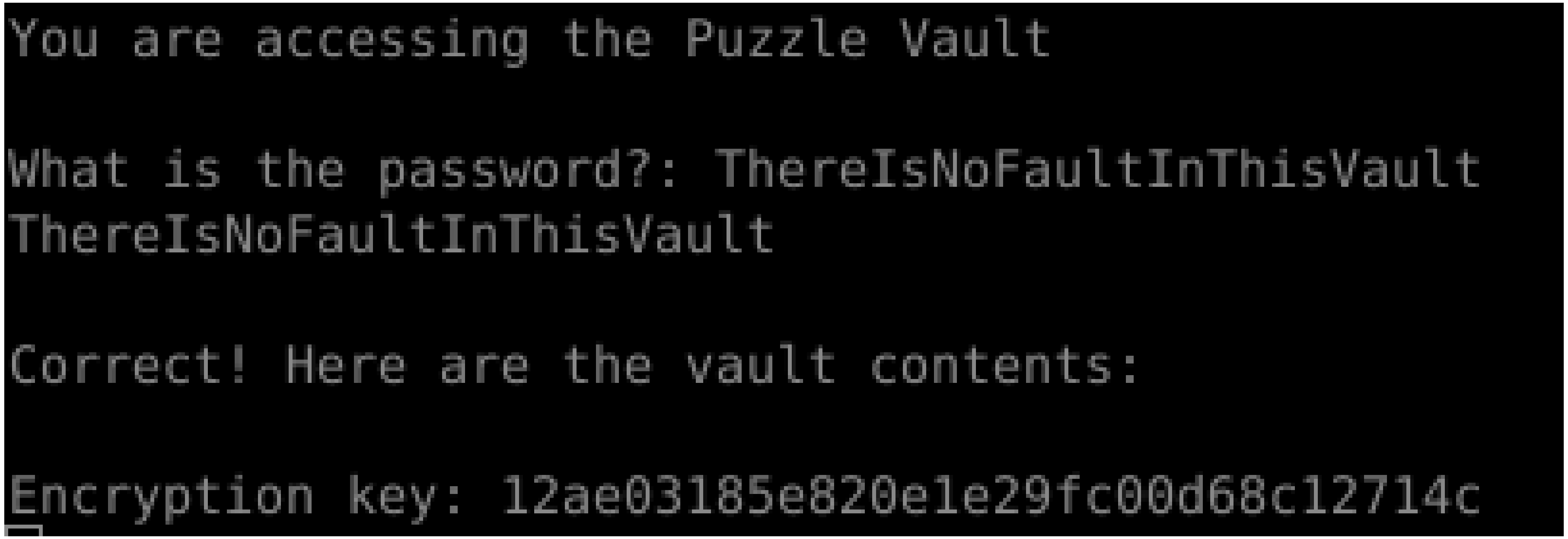
Give it a try, but I doubt that you can find out how to open it.

```
nc 10.0.2.32 10003
```

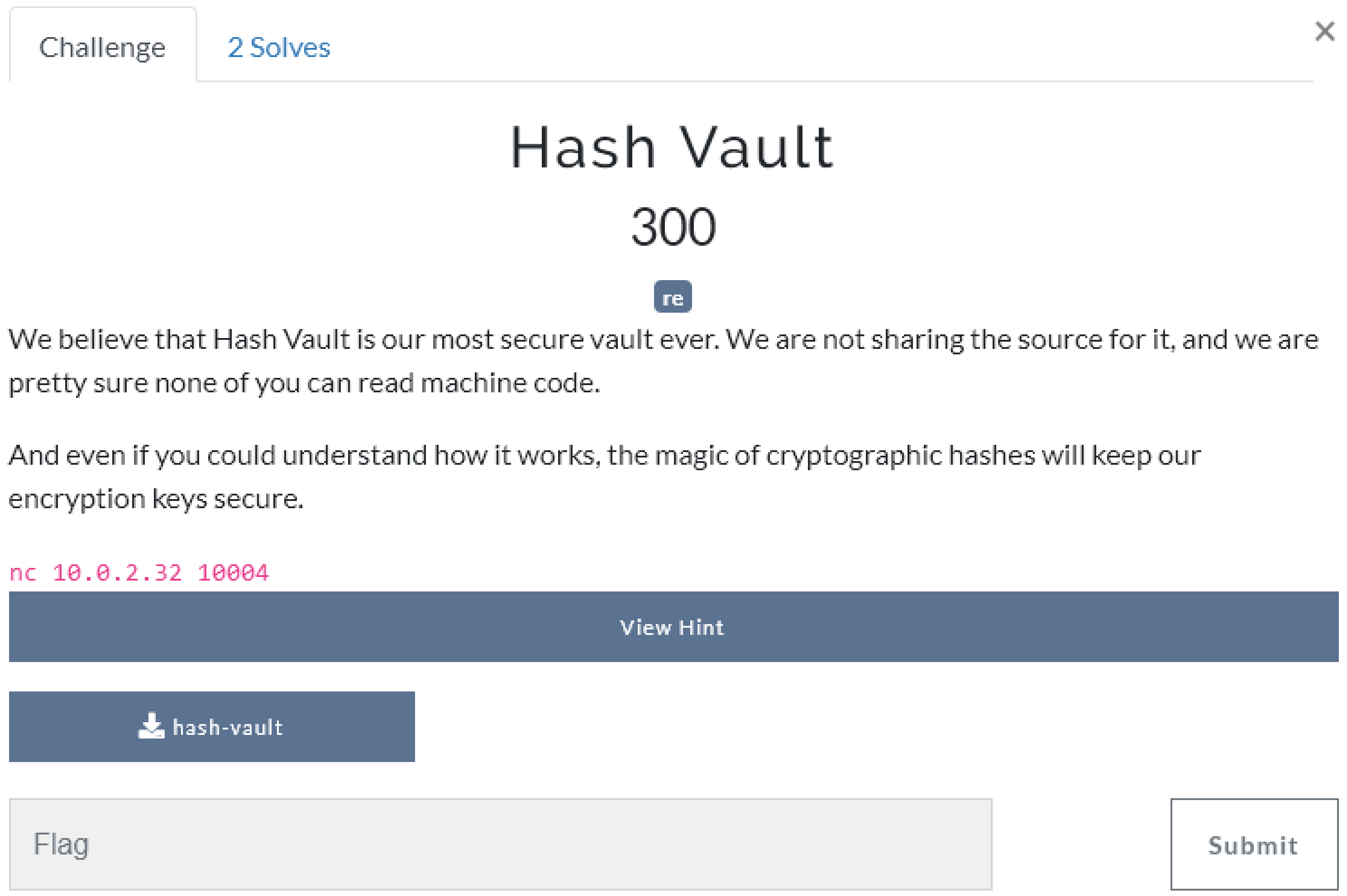
```
#!/usr/bin/env node

var fs=require('fs');const readline=require('readline'),rl=readline['cr
'stdout']});function bad_password(){console['log']('\x0aWrong\x20passw
'You\x20are\x20accessing\x20the\x20Puzzle\x20Vault\x0a'),rl['question']
try{if(_0x2fa26d['length']!=0x19){bad_password();return;}if(_0x2fa26d[0
_0x2fa26d[0x4]||_0x2fa26d[0xa]!=_0x2fa26d[0x15]||_0x2fa26d[0xb]!=_0x2fa
]!=_0x2fa26d[0x18]||_0x2fa26d[0x6]!=_0x2fa26d[0x13]){bad_password();ret
_0x2fa26d['charCodeAt'](0x6)!=0x73){bad_password();return;}if(_0x2fa26d
0x4e72){bad_password();return;}if(_0x2fa26d['charCodeAt'](0x5)*0x539+0x
'charCodeAt'](0x9);if(String['fromCharCode']((val<<0x3|val>>0x5)&0xff)!
,0x19)['split']('')['reverse']()['join']('')!=='tlua'){bad_password();r
;if(nopass[nopass['length']-0x17]!=_0x2fa26d[0x1]||nopass[0x19]!=_0x2fa
,0x74,0x6,0x6,0x7,0x76];for(i=0x0;i<_0x1a21fd['length'];i++){if(String[
_0x2fa26d[0xe+i]){bad_password();return;}}console['log']('\x0aCorrect!\
'readFile']('./vault.dat',{'encoding':'utf-8'},function(_0x19e5c7,_0x1e
_0x19e5c7);});}finally{rl['close']();}});
```



```
console.log('You are accessing the Puzzle Vault\n');

rl.question('What is the password?: ', function (password) {

    try {
        if (password.length != 25) {
            bad_password();
            return;
        }

        if (password[0] != 'T') {
            bad_password();
            return;
        }

        if ((password[2] != password[4]) ||
            (password[10] != password[21]) ||
            (password[11] != password[22]) ||
            (password[12] != password[23]) ||
            (password[13] != password[24]) ||
            (password[6] != password[19])) {
            bad_password();
            return;
        }

        if ((password[2] != String.fromCharCode(101)) ||
            (password.charCodeAt(6) != 0x73)) {
            bad_password();
            return;
        }

        if (password.charCodeAt(7) * 0x100 + password.charCodeAt(3) != 20082) {
            bad_password();
            return;
        }
```



```
You are accessing the Puzzle Vault

What is the password?: ThereIsNoFaultInThisVault
ThereIsNoFaultInThisVault

Correct! Here are the vault contents:

Encryption key: 12ae03185e820e1e29fc00d68c12714c
```



Challenge 2 Solves

Hash Vault

300

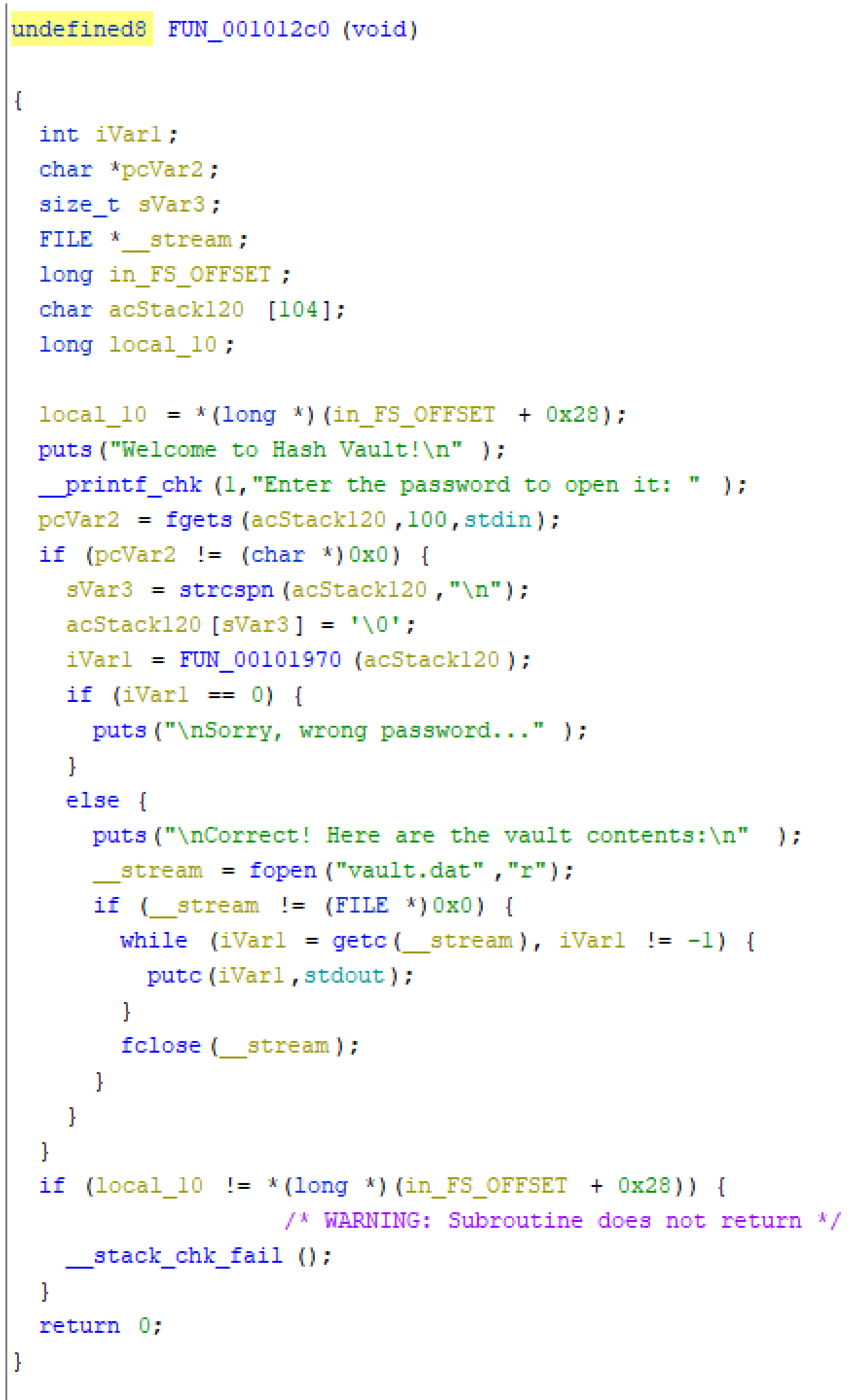
re

We believe that Hash Vault is our most secure vault ever. We are not sharing the source for it, and we are pretty sure none of you can read machine code.

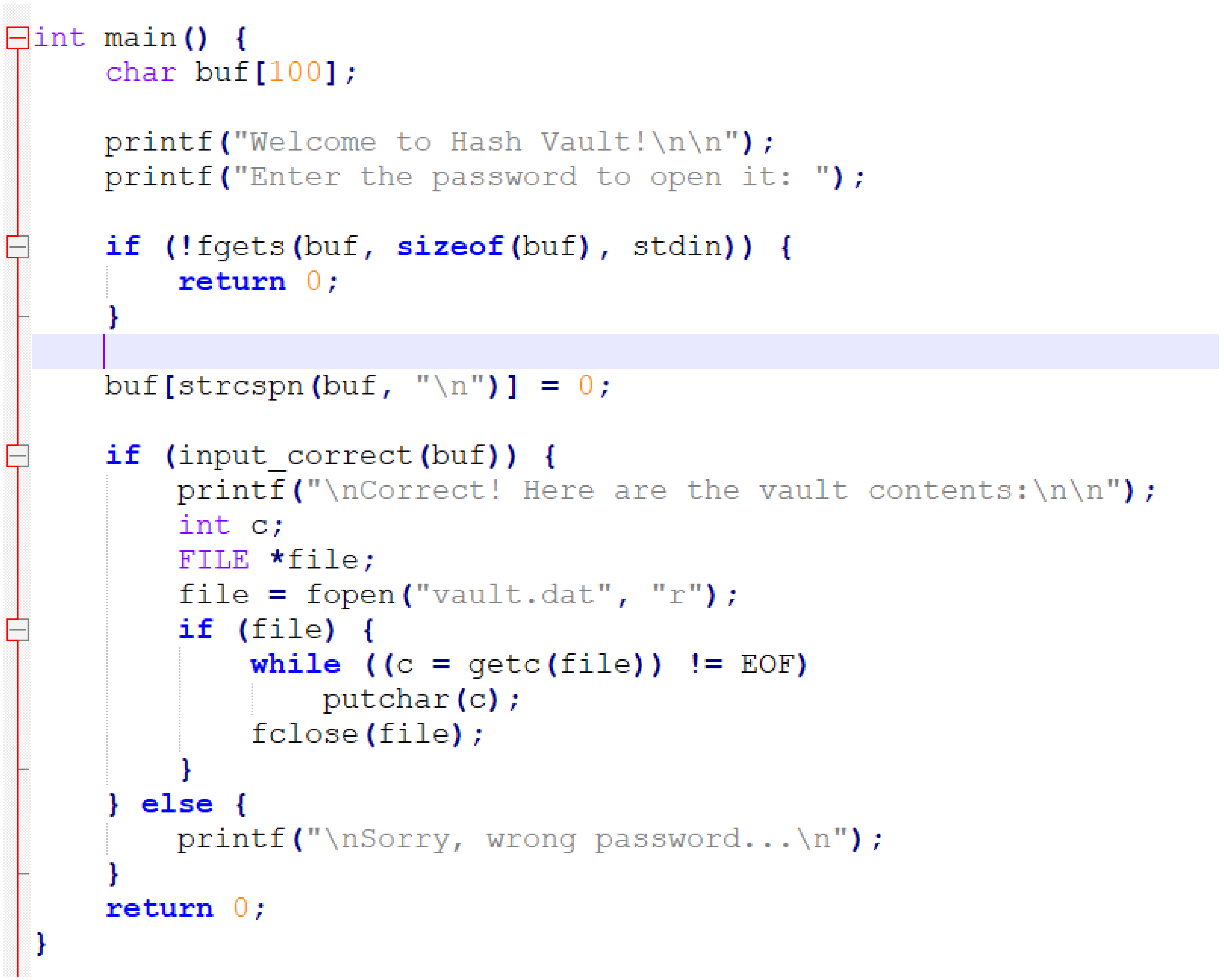
And even if you could understand how it works, the magic of cryptographic hashes will keep our encryption keys secure.

```
nc 10.0.2.32 10004
```

- Open in Ghidra (or IDA Pro, etc.)

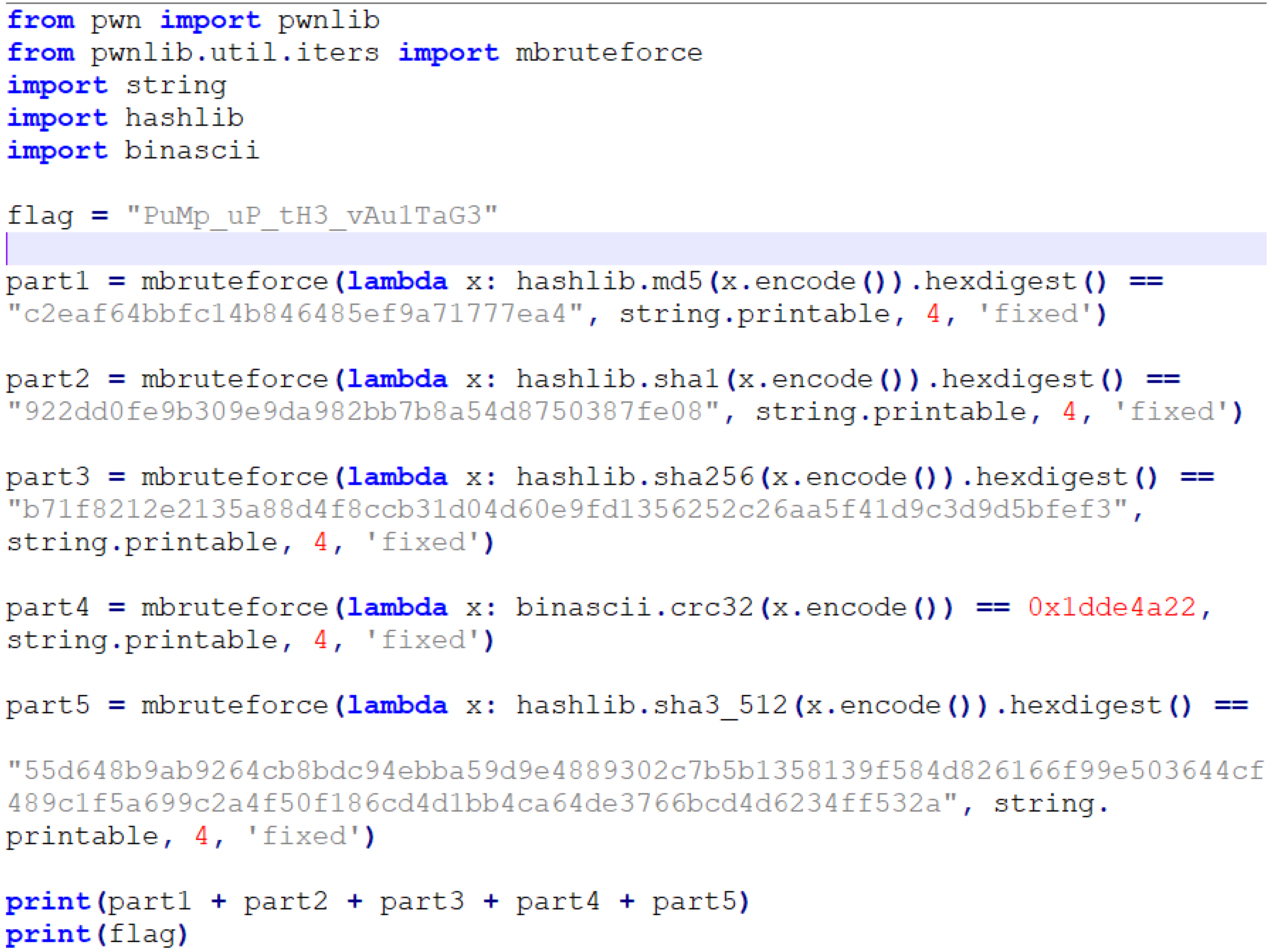


```
undefined8 FUN_001012c0 (void)

{
  int iVar1;
  char *pcVar2;
  size_t sVar3;
  FILE *__stream;
  long in_FS_OFFSET;
  char acStack120 [104];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  puts("Welcome to Hash Vault!\n");
  __printf_chk(1,"Enter the password to open it: ");
  pcVar2 = fgets(acStack120,100,stdin);
  if (pcVar2 != (char *)0x0) {
    sVar3 = strcspn(acStack120,"\n");
    acStack120[sVar3] = '\0';
    iVar1 = FUN_00101970(acStack120);
    if (iVar1 == 0) {
      puts("\nSorry, wrong password...");
    }
    else {
      puts("\nCorrect! Here are the vault contents:\n");
      __stream = fopen("vault.dat","r");
      if (__stream != (FILE *)0x0) {
        while (iVar1 = getc(__stream), iVar1 != -1) {
          putc(iVar1,stdout);
        }
        fclose(__stream);
      }
    }
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

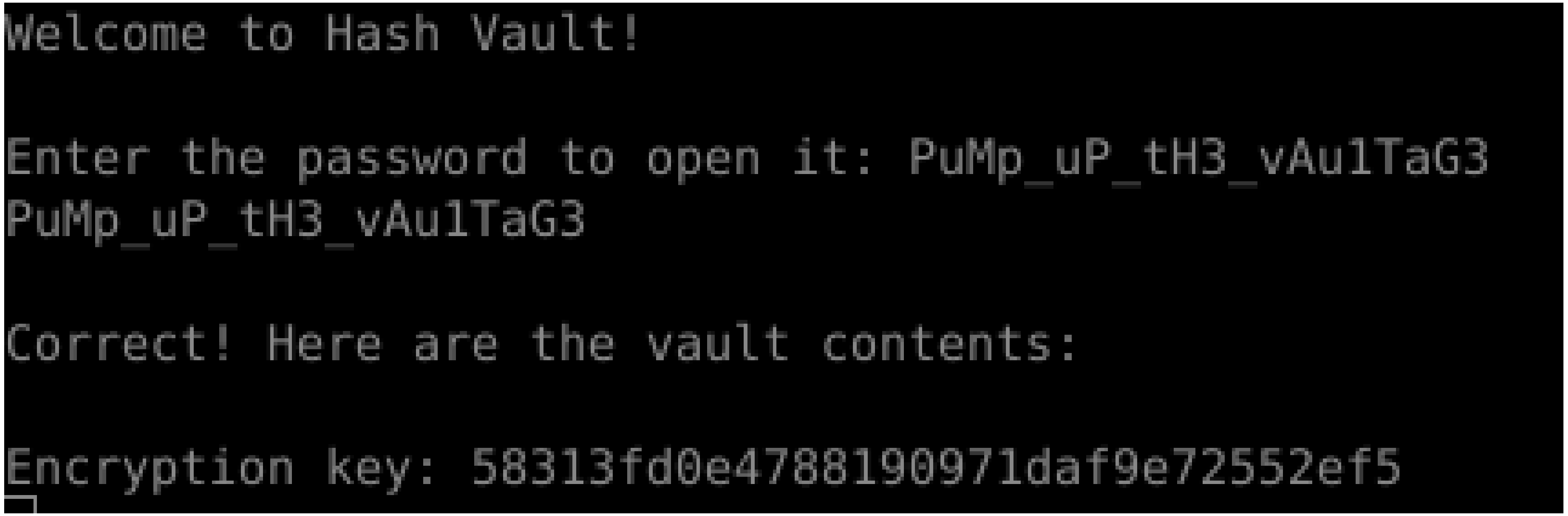

```
int main() {
    char buf[100];

    printf("Welcome to Hash Vault!\n\n");
    printf("Enter the password to open it: ");

    if (!fgets(buf, sizeof(buf), stdin)) {
        return 0;
    }

    buf[strcspn(buf, "\n")] = 0;

    if (input_correct(buf)) {
        printf("\nCorrect! Here are the vault contents:\n\n");
        int c;
        FILE *file;
        file = fopen("vault.dat", "r");
        if (file) {
            while ((c = getc(file)) != EOF)
                putchar(c);
            fclose(file);
        }
    } else {
        printf("\nSorry, wrong password...\n");
    }
    return 0;
}
```

```c
// Check length and 5 parts
int input_correct(char* buf) {
    if (strlen(buf) == 20 && part_1_correct(buf) && part_2_correct(buf) &&
        part_3_correct(buf) && part_4_correct(buf) && part_5_correct(buf)) {
        return 1;
    }
    return 0;
}
```

```c
// Check part 1 of password
int part_1_correct(char* buf) {
    char right_hash[] =
        {0xc2, 0xea, 0xf6, 0x4b, 0xbf, 0xc1, 0x4b, 0x84,
         0x64, 0x85, 0xef, 0x9a, 0x71, 0x77, 0x7e, 0xa4};

    int success = 0;
    unsigned char hash[EVP_MAX_MD_SIZE];
    EVP_MD_CTX* context = EVP_MD_CTX_new();

    if(context != NULL) {
        if(EVP_DigestInit_ex(context, EVP_md5(), NULL)) {
            if(EVP_DigestUpdate(context, buf, 4)) {
                unsigned int lengthOfHash = 0;

                if(EVP_DigestFinal_ex(context, hash, &lengthOfHash)) {
                    if (lengthOfHash == sizeof(right_hash)) {
                        if (!memcmp((void*)right_hash, hash, sizeof(right_hash))) {
                            success = 1;
                        }
                    }
                }
            }
        }

        EVP_MD_CTX_free(context);
    }

    return success;
}
```



```
from pwn import pwnlib
from pwnlib.util.iters import mbruteforce
import string
import hashlib
import binascii

flag = "PuMp_uP_tH3_vAulTaG3"

part1 = mbruteforce(lambda x: hashlib.md5(x.encode()).hexdigest() ==
"c2eaf64bbfc14b846485ef9a71777ea4", string.printable, 4, 'fixed')

part2 = mbruteforce(lambda x: hashlib.sha1(x.encode()).hexdigest() ==
"922dd0fe9b309e9da982bb7b8a54d8750387fe08", string.printable, 4, 'fixed')

part3 = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest() ==
"b71f8212e2135a88d4f8ccb31d04d60e9fd1356252c26aa5f41d9c3d9d5bfef3",
string.printable, 4, 'fixed')

part4 = mbruteforce(lambda x: binascii.crc32(x.encode()) == 0x1dde4a22,
string.printable, 4, 'fixed')

part5 = mbruteforce(lambda x: hashlib.sha3_512(x.encode()).hexdigest() ==

"55d648b9ab9264cb8bdc94ebba59d9e4889302c7b5b1358139f584d826166f99e503644cf
489c1f5a699c2a4f50f186cd4d1bb4ca64de3766bcd4d6234ff532a", string.
printable, 4, 'fixed')

print(part1 + part2 + part3 + part4 + part5)
print(flag)
```



```
Welcome to Hash Vault!

Enter the password to open it: PuMp_uP_tH3_vAu1TaG3
PuMp_uP_tH3_vAu1TaG3

Correct! Here are the vault contents:

Encryption key: 58313fd0e4788190971daf9e72552ef5
```

Thank you!