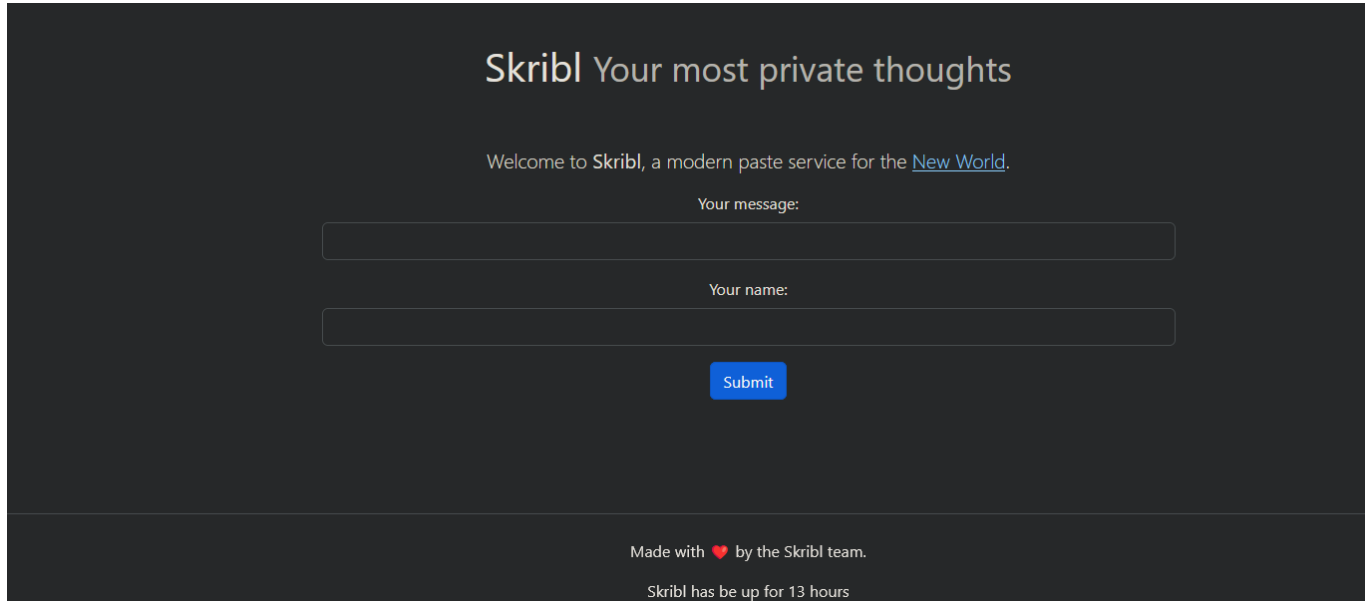


## Rev 3 (Skribl)

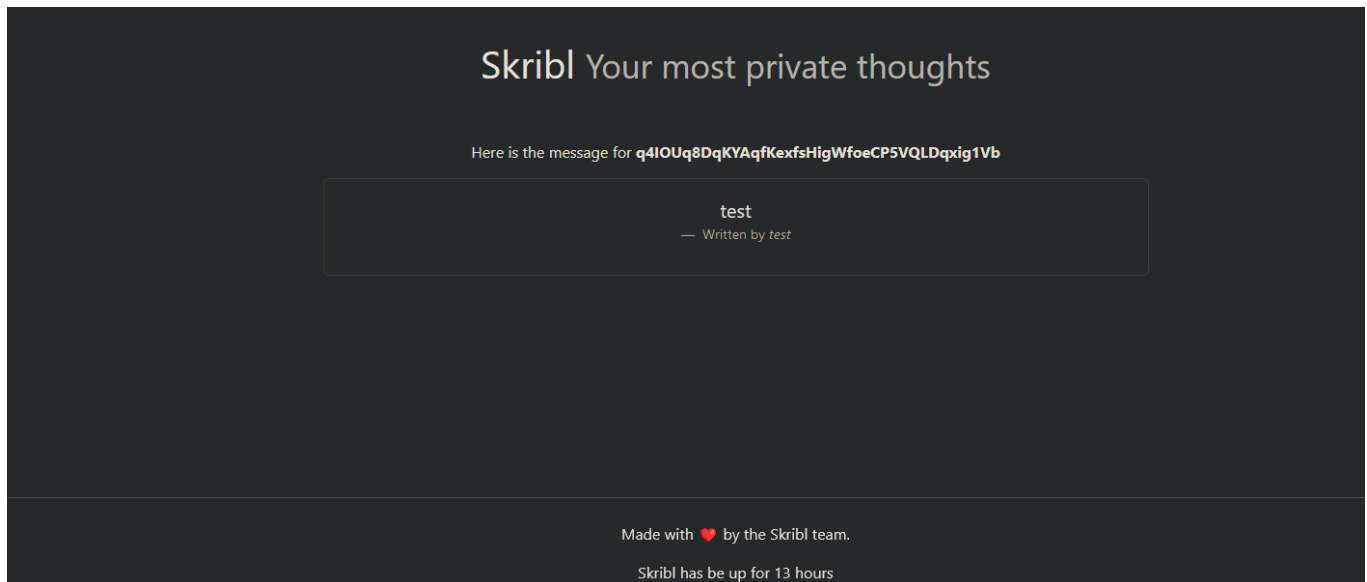
 dist.zip

<https://skribl.chall.pwnoh.io>

Given for the challenge is a .zip file and a link to a website. The website allows you to write a note and attach it to a name. From this, it appends `/view/<key>` to the URL, where `<key>` is a randomly generated string of letters and numbers:



The screenshot shows the Skribl website's home page. At the top, the title "Skribl Your most private thoughts" is displayed. Below it, a welcome message reads "Welcome to Skribl, a modern paste service for the [New World](#)." There are two input fields: "Your message:" and "Your name:". A blue "Submit" button is positioned below the "Your name:" field. At the bottom of the page, it says "Made with ❤️ by the Skribl team." and "Skribl has be up for 13 hours".



The screenshot shows the Skribl website's view page. At the top, the title "Skribl Your most private thoughts" is displayed. Below it, a message reads "Here is the message for **q4IOUq8DqKYAqfKexfsHigWfoeCP5VQLDqxig1Vb**". There is a text box containing the word "test" and the text "— Written by test" below it. At the bottom of the page, it says "Made with ❤️ by the Skribl team." and "Skribl has be up for 13 hours".

Given in the .zip archive is the source code of the website, contained within the file skribl.py:

```
import math

import time

from flask import Flask, render_template, redirect, url_for, request
```

```

from flask_bootstrap import Bootstrap5

from flask_wtf import FlaskForm, CSRFProtect

from wtforms import StringField, SubmitField

from wtforms.validators import DataRequired, Length

# Don't try this at home, kids

try:

    from backend import create_skribl, init_backend

except:

    from .backend import create_skribl, init_backend

app = Flask(__name__)

app.secret_key = 'tO$&!|0wkamvVia0?n$NqIRVWOG'

bootstrap = Bootstrap5(app)

csrf = CSRFProtect(app)

skribls = {}

stime = math.floor(time.time())

init_backend(skribls)

class SkriblForm(FlaskForm):

    skribl = StringField('Your message: ', validators=[DataRequired(), Length(1, 250)])

    author = StringField("Your name:", validators=[Length(0, 40)])

```

```

submit = SubmitField('Submit')

@app.route('/', methods=['GET', 'POST'])
def index():
    form = SkriblForm()

    message = ""

    if form.validate_on_submit():
        message = form.skribl.data

        author = form.author.data

        key = create_skribl(skribls, message, author)

        return redirect(url_for('view', key=key))

    return render_template('index.html', form=form, error_msg=request.args.get("error_msg",
''))

@app.route('/view/<key>', methods=['GET'])
def view(key):
    print(f"Viewing with key {key}")

    if key in skribls:
        message, author = skribls[key]

        return render_template("view.html", message=message, author=author, key=key)

    else:
        return redirect(url_for('index', error_msg=f"Skribl not found: {key}"))

@app.route('/about', methods=["GET"])
def about():
    return render_template('about.html')

```

```
@app.context_processor

def inject_stime():

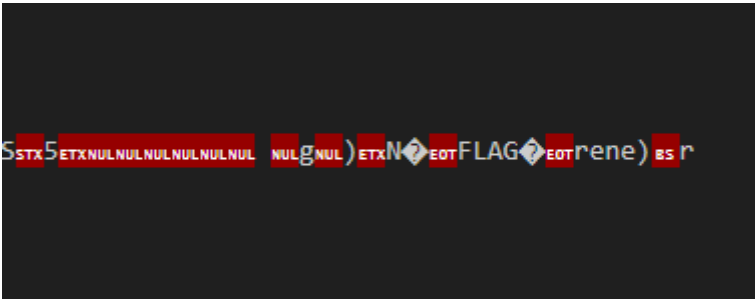
    return dict(stime=math.floor(time.time()) - stime)
```

My first instinct was to try pasting the "Secret Key" at the end of the sire view URL, which didn't work. The second notable thing in this file is that a library called `backend` is imported.

Going back to the .zip archive, inside a folder called `__pycache__` is the backend library:

backend.cpython-313.pyc

Very promising, as it includes the string "FLAG" in it.



However, the bad news is that the library exists as a compiled file. Usually, decompiling Python is a simple task using something like `pydcd` or `Uncompyle`. However, there's a catch: the file was compiled with Python version 3.13, which was at the time of writing this the experimental branch of Python. Because of this, `pydcd` and `Uncompyle` didn't support 3.13 decompilation yet.

The first action I took was to actually install cPython to get access to the 3.13 branch of Python. After this, I unsuccessfully tried to modify pydcd to support 3.13. During this time, I stumbled across the `dis` library for Python, which allows disassembly of Python bytecode, and is able to run on the experimental branch. With my lack of success modifying pydcd, I figured I'd give `dis` a shot.

```
dis.dis(x=None, *, file=None, depth=None, show_caches=False, adaptive=False)
```

Disassemble the `x` object. `x` can denote either a module, a class, a method, a function, a generator, an asynchronous generator, a coroutine, a code object, a string of source code or a byte sequence of raw bytecode. For a module, it disassembles all functions. For a class, it disassembles all methods (including class and static methods). For a code object or sequence of raw bytecode, it prints one line per bytecode instruction. It also recursively disassembles nested code objects. These can include generator expressions, nested functions, the bodies of nested classes, and the code objects used for [annotation scopes](#). Strings are first compiled to code objects with the `compile()` built-in function before being disassembled. If no object is provided, this function disassembles the last traceback.

The disassembly is written as text to the supplied `file` argument if provided and to `sys.stdout` otherwise.

The maximal depth of recursion is limited by `depth` unless it is `None`. `depth=0` means no recursion.

If `show_caches` is `True`, this function will display inline cache entries used by the interpreter to specialize the bytecode.

If `adaptive` is `True`, this function will display specialized bytecode that may be different from the original bytecode.

<https://docs.python.org/3.13/library/dis.html>

The `dis.dis()` function is the focus here, as it should show what the bytecodes in the `.pyc` file correspond with in a semi-readable format. As a file can't be passed directly to it, I needed to convert it to a code object. Luckily, this is just the contents of the `.pyc` file I had, minus the header. To disassemble the file from there, I ran the following commands in a Python 3.13 session:

```
import dis, marshal
pyc = open("PATH_TO_BACKEND_PYC", "rb")
pyc.seek(16)
obj = marshal.load(pyc)
dis.dis(obj)
```

The `seek` is necessary to bypass the header of the `.pyc` file.

This gave me the bytecode operations for the `.pyc`:

1	2	LOAD_CONST	0 (0)
	4	LOAD_CONST	1 (None)
	6	IMPORT_NAME	0 (string)
	8	STORE_NAME	0 (string)
2	10	LOAD_CONST	0 (0)
	12	LOAD_CONST	1 (None)
	14	IMPORT_NAME	1 (random)
	16	STORE_NAME	1 (random)
3	18	LOAD_CONST	0 (0)
	20	LOAD_CONST	1 (None)
	22	IMPORT_NAME	2 (time)
	24	STORE_NAME	2 (time)
4	26	LOAD_CONST	0 (0)

	28	LOAD_CONST	1 (None)
	30	IMPORT_NAME	3 (math)
	32	STORE_NAME	3 (math)
5	34	LOAD_CONST	0 (0)
	36	LOAD_CONST	1 (None)
	38	IMPORT_NAME	4 (os)
	40	STORE_NAME	4 (os)
8	42	LOAD_CONST	2 ('return')
	44	LOAD_NAME	5 (str)
	46	BUILD_TUPLE	2
	48	LOAD_CONST	3 (<code object create_skribl at 0x7f1a1bf78d50, file "/home/rene/Documents/Java/OSUCyberSecurityClub/buckeyectf23/buckeyectf-challenges/chals/rev- pycache/dist/chal/backend.py", line 8>)
	50	MAKE_FUNCTION	
	52	SET_FUNCTION_ATTRIBUTE	4 (annotations)
	54	STORE_NAME	6 (create_skribl)
18	56	LOAD_CONST	4 (<code object init_backend at 0x7f1a1c005f70, file "/home/rene/Documents/Java/OSUCyberSecurityClub/buckeyectf23/buckeyectf-challenges/chals/rev- pycache/dist/chal/backend.py", line 18>)
	58	MAKE_FUNCTION	
	60	STORE_NAME	7 (init_backend)
	62	RETURN_CONST	1 (None)

Disassembly of <code object create\_skribl at 0x7f1a1bf78d50, file  
"/home/rene/Documents/Java/OSUCyberSecurityClub/buckeyectf23/buckeyectf-challenges/chals/rev-  
pycache/dist/chal/backend.py", line 8>:

8	0	RESUME	0
9	2	LOAD_GLOBAL	1 (print + NULL)
	12	LOAD_CONST	1 ('Creating skribl ')
	14	LOAD_FAST	1 (message)
	16	FORMAT_SIMPLE	
	18	BUILD_STRING	2
	20	CALL	1
	28	POP_TOP	
11	30	LOAD_GLOBAL	2 (string)
	40	LOAD_ATTR	4 (ascii_lowercase)
	60	LOAD_GLOBAL	2 (string)
	70	LOAD_ATTR	6 (ascii_uppercase)
	90	BINARY_OP	0 (+)
	94	LOAD_GLOBAL	2 (string)
	104	LOAD_ATTR	8 (digits)
	124	BINARY_OP	0 (+)
	128	STORE_FAST	3 (alphabet)
12	130	LOAD_GLOBAL	11 (range + NULL)
	140	LOAD_CONST	2 (40)
	142	CALL	1
	150	GET_ITER	

	152	LOAD_FAST_AND_CLEAR	4 (i)
	154	SWAP	2
	156	BUILD_LIST	0
	158	SWAP	2
>>	160	FOR_ITER	25 (to 214)
	164	STORE_FAST	4 (i)
	166	LOAD_GLOBAL	12 (random)
	176	LOAD_ATTR	14 (choice)
	196	PUSH_NULL	
	198	LOAD_FAST	3 (alphabet)
	200	CALL	1
	208	LIST_APPEND	2
	210	JUMP_BACKWARD	27 (to 160)
>>	214	END_FOR	
	216	STORE_FAST	5 (key_list)
	218	STORE_FAST	4 (i)
14	220	LOAD_CONST	3 (')
	222	LOAD_ATTR	17 (join + NULL self)
	242	LOAD_FAST	5 (key_list)
	244	CALL	1
	252	STORE_FAST	6 (key)
15	254	LOAD_FAST_LOAD_FAST	18 (message, author)
	256	BUILD_TUPLE	2
	258	LOAD_FAST_LOAD_FAST	6 (skribls, key)
	260	STORE_SUBSCR	
16	264	LOAD_FAST	6 (key)
	266	RETURN_VALUE	
None	>>	268 SWAP	2
		270 POP_TOP	
12	272	SWAP	2
	274	STORE_FAST	4 (i)
	276	RERAISE	0

ExceptionTable:

156 to 214 -> 268 [2]

Disassembly of <code object init\_backend at 0x7f1a1c005f70, file

"/home/rene/Documents/Java/OSUCyberSecurityClub/buckeyectf23/buckeyectf-challenges/chals/rev-pycache/dist/chal/backend.py", line 18>:

18	0	RESUME	0
19	2	LOAD_GLOBAL	0 (random)
	12	LOAD_ATTR	2 (seed)
	32	PUSH_NULL	
	34	LOAD_GLOBAL	4 (math)
	44	LOAD_ATTR	6 (floor)
	64	PUSH_NULL	
	66	LOAD_GLOBAL	8 (time)
	76	LOAD_ATTR	8 (time)

	96	PUSH_NULL	
	98	CALL	0
	106	CALL	1
	114	CALL	1
	122	POP_TOP	
21	124	LOAD_GLOBAL	11 (create_skribl + NULL)
	134	LOAD_FAST	0 (skribls)
	136	LOAD_GLOBAL	12 (os)
	146	LOAD_ATTR	14 (environ)
	166	LOAD_CONST	1 ('FLAG')
	168	BINARY_SUBSCR	
	172	LOAD_CONST	2 ('rene')
	174	CALL	3
	182	POP_TOP	
	184	RETURN_CONST	0 (None)

The first important part is the block of disassembly at 0x7f1a1c005f70. This can be rewritten as something like:

```
random.seed(math.floor(time.time()))
create_skribbl("FLAG", "rene")
```

`time.time` returns the time in UNIX epoch format, `math.floor` rounds it down to the nearest whole number, and `random.seed` sets the passed parameter to the seed to be used to generate pseudorandom numbers. This means that the time whenever this function was called can be used to generate the same key that was generated for the flag message.

The second important part is at 0x7f1a1bf78d50, this is the `create_skribbl` function from the last snippet, of which the relevant part can be rewritten as something like:

```
def create_skribbl(message, name):
    print('Creating skribl: ')
    alphabet = string.ascii_lowercase + string.ascii_uppercase + string.digits
    key = ""
    for i in range(40):
        key+=random.choice(alphabet)
```

We know from this now how to generate a key using the seed. The last piece is actually on the website.

There is a line of code showing how long it has been up in epoch time:

```
stime = moment.duration(95736, 'seconds'); stime_text = document.getElementById("stime"); stime_text.innerHTML = stime.humanize()
```

By taking this time and setting up a timer to count the amount of time since I copied the time to then add, I was able to calculate how long ago the website was brought online to the precision of about a few seconds. After that, the only thing left is to make a key generator.

```
import random
import time
import math
```



```
import string
alphabet = string.ascii_lowercase + string.ascii_uppercase + string.digits
random.seed(1696023787)
full_url = "https://skribl.chall.pwnoh.io/view/"
for i in range(40):
    full_url+=random.choice(alphabet)
print(full_url)
```

The seed is the exact epoch time the site was last brought online. The program generated the URL <https://skribl.chall.pwnoh.io/view/ByiOmilUKsYQ60fMaWPq8S6X22aEC4pyS82NSp3y>, and visiting this URL gave me the flag:

Here is the message for **ByiOmilUKsYQ60fMaWPq8S6X22aEC4pyS82NSp3y**

bctf{wHy\_d0\_w3\_Ne3d\_s0\_m@ny\_N0T3\$\_aNyW@y}

— Written by *rene*