ECSC Estonia Prequalifier - DDCLE

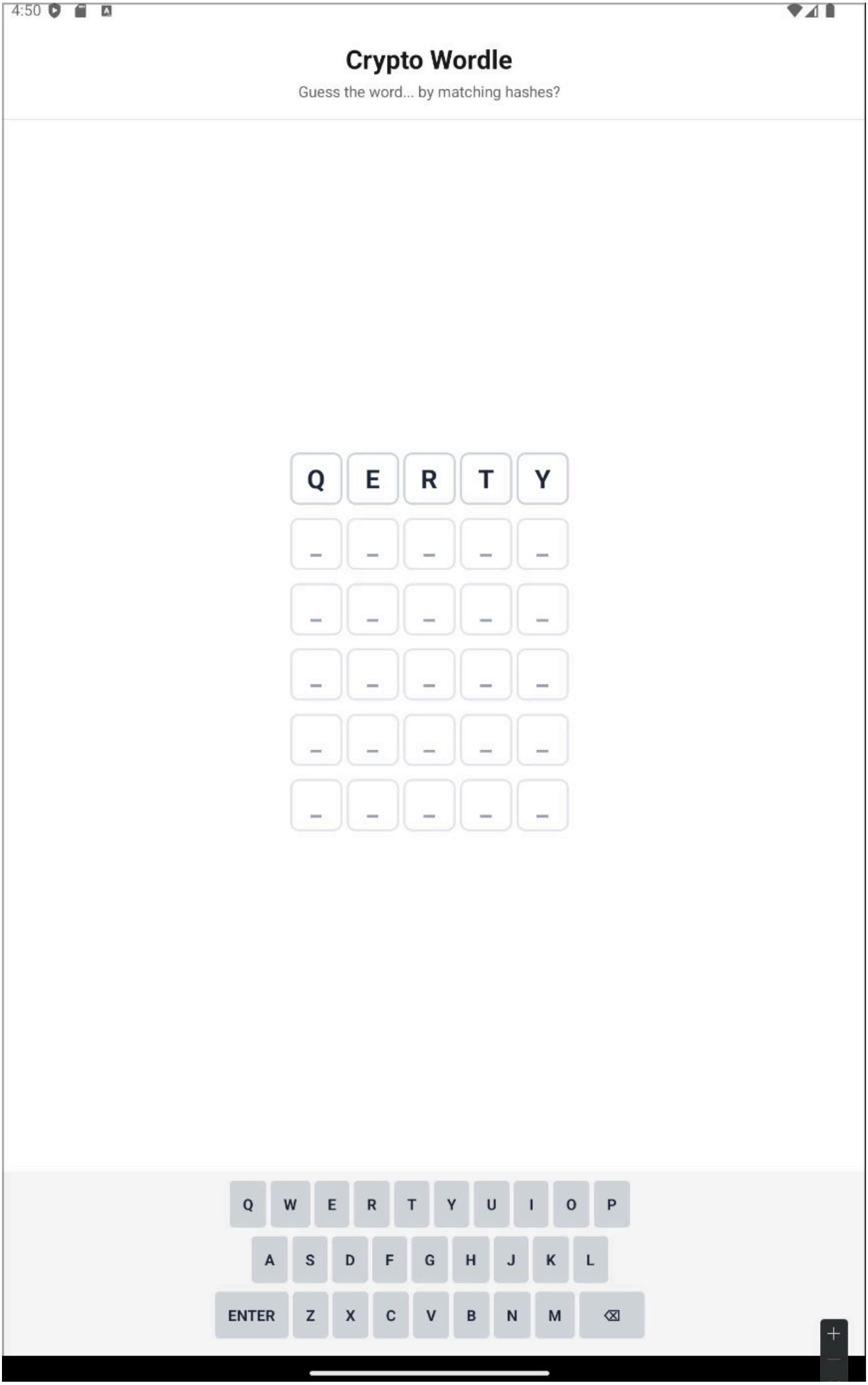
We are given an apk file.

First off just dump the source code of it with `jadx`. Searching for the `MainActivity.java` file we find this:

```
package com.willbenem.DDCLE;

import android.os.Build;
import android.os.Bundle;
import com.facebook.react.ReactActivity;
import com.facebook.react.ReactActivityDelegate;
import com.facebook.react.defaults.DefaultNewArchitectureEntryPoint;
import com.facebook.react.defaults.DefaultReactActivityDelegate;
import expo.modules.ReactActivityDelegateWrapper;
import expo.modules.splashscreen.SplashScreenManager;
import kotlin.Metadata;

/* compiled from: MainActivity.kt */
@Metadata(d1 =
{"\u0000&\n\u0002\u0018\u0002\n\u0002\u0018\u0002\n\u0002\b\u0002\n\u0002\
u0018\u0002\n\u0000\n\u0002\u0010\u000e\n\u0000\n\u0002\u0010\u0002\n\u000
2\b\u0002\n\u0002\u0018\u0002\n\u0000\u0018\u00002\u00020\u0001B\u0005¢
\u0006\u0002\u0010\u0002J\b\u0010\u0003\u001a\u00020\u0004H\u0014J\b\u0010
\u0005\u001a\u00020\u0006H\u0014J\b\u0010\u0007\u001a\u00020\bH\u0016J\u00
12\u0010\t\u001a\u00020\b2\b\u0010\n\u001a\u0004\u0018\u00010\u000bH\u0014
¨\u0006\f"}, d2 = {"Lcom/willbenem/DDCLE/MainActivity;",
"Lcom/facebook/react/ReactActivity;", "()V",
"createReactActivityDelegate",
"Lcom/facebook/react/ReactActivityDelegate;", "getMainComponentName", "",
"invokeDefaultOnBackPressed", "", "onCreate", "savedInstanceState",
"Landroid/os/Bundle;", "app_release"}, k = 1, mv = {1, 9, 0}, xi = 48)
/* loaded from: classes2.dex */
public final class MainActivity extends ReactActivity {
    @Override // com.facebook.react.ReactActivity,
androidx.fragment.app.FragmentActivity,
androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity,
android.app.Activity
    protected void onCreate(Bundle savedInstanceState) {
        SplashScreenManager.INSTANCE.registerOnActivity(this);
        super.onCreate(null);
    }

    @Override // com.facebook.react.ReactActivity
    protected String getMainComponentName() {
```

```
        return "main";
    }

    @Override // com.facebook.react.ReactActivity
    protected ReactActivityDelegate createReactActivityDelegate() {
        final String mainComponentName = getMainComponentName();
        final boolean fabricEnabled =
DefaultNewArchitectureEntryPoint.getFabricEnabled();
        return new ReactActivityDelegateWrapper(this, true, new
DefaultReactActivityDelegate(this, mainComponentName, fabricEnabled) { //
from class: com.willbenem.DDCLE.MainActivity$createReactActivityDelegate$1
            {
                MainActivity mainActivity = this;
            }
        });
    }

    @Override // com.facebook.react.ReactActivity,
com.facebook.react.modules.core.DefaultHardwareBackBtnHandler
    public void invokeDefaultOnBackPressed() {
        if (Build.VERSION.SDK_INT <= 30) {
            if (moveTaskToBack(false)) {
                return;
            }
            super.invokeDefaultOnBackPressed();
            return;
        }
        super.invokeDefaultOnBackPressed();
    }
}
```

The rest of the decompiled source code also looked as empty. At first I was confused, so I decided to launch the android emulator to run the app there and was met with a wordle like app.


4:50
Crypto Wordle
Guess the word... by matching hashes?
Q E R T Y
Q W E R T Y U I O P
A S D F G H J K L
ENTER Z X C V B N M ⌫

I tried inputting words, but nothing happened, I could only write to the first row and as such was confused as to what was going on. I even uploaded this to a physical android mobile phone to check if maybe my emulator was broken, but got the same behaviour.

I thought that maybe this was part of the challenge (but not before I submitted a ticket asking if this was intended behaviour, but quickly realized it was before I got a reply so I just marked it as resolved), so I started looking around, since it felt weird that I was met with an app even though there seemed to be no source code.
I tried looking around the Java libraries, but they seemed to be imported libraries, so I just ran a grep command to search for the string `crypto wordle` in the source files folder of the app. I was met with an interesting result.

```
❯ grep -Ri 'crypto wordle'
grep: src/main/assets/index.android.bundle: binary file matches
```

Upon googling this filename I stumbled across this thing called React Native. Which is a popular thing but since I am not an android developer I was not aware that it was this. It seems this application was made with react native and the react source code is compiled into a binary file and that is what the `index.android.bundle` is. What's more, this seems to be using Hermes as it's javascript engine, meaning the react code is compiled to Hermes bytecode and is not plain javascript.

Googling on the internet I found this Hermes bytecode decompiler. I dumped the hermes bytecode both in the disassembly format and the javascript format. Both of them are very confusing and unreadable, but the javascript one is a bit less so, so I performed my reversing based on the javacsript file.

The decompiled output was over 300 000 lines, which is massive, so I just Ctrl+F for the keyword "crypto wordle" again and found it in this snippet of decompiled pseudo-javascript:

```
        r8 = r9.defineProperty;
        r7 = {};
        r1 = true;
        r7['value'] = r1;
        r1 = '__esModule';
        r1 = r8.bind(r9)(r3, r1, r7);
        r1 = function() { // Original name: GameScreen, environment: r2
            r0 = _closure1_slot1;
            r0 = r0.useGameStore;
            r3 = undefined;
            r0 = r0.bind(r3)();
            r1 = r0.P;
            r0 = r0.R;
            r0 = _closure1_slot4;
            r2 = r0.jsxs;
            r0 = _closure1_slot0;
```

```
            r1 = r0.SafeAreaView;
            r0 = {};
            r4 = _closure1_slot5;
            r4 = r4.container;
            r0['style'] = r4;
            r4 = _closure1_slot4;
            r7 = r4.jsxs;
            r4 = _closure1_slot0;
            r6 = r4.View;
            r4 = {};
            r8 = _closure1_slot5;
            r8 = r8.header;
            r4['style'] = r8;
            r8 = _closure1_slot4;
            r10 = r8.jsx;
            r8 = _closure1_slot0;
            r9 = r8.Text;
            r8 = {};
            r11 = _closure1_slot5;
            r11 = r11.title;
            r8['style'] = r11;
            r11 = 'Crypto Wordle';
            r8['children'] = r11;
            r9 = r10.bind(r3)(r9, r8);
            r8 = new Array(2);
            r8[0] = r9;
            r9 = _closure1_slot4;
            r11 = r9.jsx;
            r9 = _closure1_slot0;
            r10 = r9.Text;
            r9 = {};
            r12 = _closure1_slot5;
            r12 = r12.subtitle;
            r9['style'] = r12;
            r12 = 'Guess the word... by matching hashes?';
            r9['children'] = r12;
            r9 = r11.bind(r3)(r10, r9);
            r8[1] = r9;
```

The second thing I searched for was `flag`. I found this code snippet:

```
case 293: // catch_target0
                        CatchBlockStart(arg_register=10);
                        r3 = global;
                        r9 = r3.console;
                        r6 = r9.error;
                        r3 = 'Failed to decrypt flag:';
                        r3 = r6.bind(r9)(r3, r10);
                        r4 = 'Error decrypting flag';
```

```
case 322:
                        r3 = {};
                        r15 = r3;
                        r14 = r1;
                        r6 = copyDataProperties(r15, r14);
                        r6 = 'L';
                        r3[r6] = r8;
                        r6 = 'M';
                        r3[r6] = r7;
                        r6 = r5;
                        r5 = 'P';
                        r3[r5] = r6;
                        r5 = r4;
                        r4 = 'R';
```

After scrolling down a bit from there, I stumbled upon this array:

```
 r6 = 5;
        r2['WORD_LENGTH'] = r6;
        r1 = {'INVALID_LENGTH': null, 'NOT_IN_WORDLIST': 'Not in word
list', 'GAME_OVER': 'Game is already over'};
        r3 = r3.HermesInternal;
        r5 = r3.concat;
        r4 = 'Word must be ';
        r3 = ' letters';
        r3 = r5.bind(r4)(r6, r3);
        r1['INVALID_LENGTH'] = r3;
        r2['GAME_MESSAGES'] = r1;
        r1 = ['ABORT', 'AKTIE', 'AKTIV', ...];
        r2['VALID_WORDS'] = r1;
        r2['TARGET_WORDS'] = r1;
```

It seems the words were in danish and that was why I could not get any feedback no matter what word I put in.
It seems that these words do indeed work

Crypto Wordle

Guess the word... by matching hashes?

A B O R T

fba8a6ce5f66683c7505f56fcf8513864e99a2a6d3006535b92660eb9496566bc2211c9826ffcf3777c21de343da48ddc7f9aa8a554f250ee160a8103b5572ac

_	_	_	_	_
_	_	_	_	_
_	_	_	_	_
_	_	_	_	_
_	_	_	_	_

Q W E R T Y U I O P

A S D F G H J K L

ENTER Z X C V B N M ⌫

It outputs a long hexstring, which, considering the text about matching hashes, is probably a hash of some kind. This made the challenge a lot clearer. Most likely we need to find a suitable word or combination of words to obtain the flag. Let's look around some more.

Upon scrolling down a bit more I found this:

```
 r1 = r5[r1];
        r1 = r4.bind(r0)(r1);
        var _closure1_slot8 = r1;
        r1 =
'7y7q06tmjufJu3UucuB/xg==:pYCr5aNWDhx1hkqWS4+erLkzDb3bF4P+/BhoD9SPsbXZWYpe
3hx0yaFZBJaAu7nr';
        r3['F'] = r1;
        var LONG_MAYBE_AES_KEY_STR = r1;
        r1 =
'0d2462ac7a20b9814032653168ef0238a4eb0f1191a33c2d305356728de16556dd5b8586b
0905ce244cb029a3f95a3727d74de879adde0ad76763c097b31e3fe';
        r3['T'] = r1;
        var LONG_BYTESTRING_MAYBE_HASH = r1;
        return r0;
```

(`LONG_BYTESTRING_MAYBE_HASH` is a text replaced by me, was not originally named like that)

What is intersting is the hexstring and the base64 blob of text. Taking an educated guess, this hexstring is the hexstring we want to obtain, i.e it is the hash of the word that will give us the flag.
The second string I assumed to be AES key or something like that, as the length is 16 bytes and the text after the `:` is probably a ciphertext encrypted with AES.

Now that we have found our goal, let's try to find how this hash is calculated.

Scrolling back up to the array of allowed words, I find the definition of this function:

```
r1 = function(a0) { // Original name: A, environment: r2
            r1 = a0;
            r0 = r1.toLowerCase;
            r4 = r0.bind(r1)();
            r1 = _closure1_slot5;
            r3 = r1.bytesToHex;
            r1 = _closure1_slot2;
            r1 = r1.blake3;
            r2 = undefined;
            r1 = r1.bind(r2)(r4);
            r4 = r3.bind(r2)(r1);
            r1 = _closure1_slot5;
            r3 = r1.bytesToHex;
            r1 = _closure1_slot4;
            r1 = r1.sha3_384;
```

```
            r1 = r1.bind(r2)(r4); // r4 = bytestoHex(blake3(inpt.lower()))
            r3 = r3.bind(r2)(r1);
            r1 = _closure1_slot5;
            r1 = r1.bytesToHex;
            r0 = _closure1_slot3;
            r0 = r0.sha512;
            r0 = r0.bind(r2)(r3);
            r0 = r1.bind(r2)(r0);
            return r0;
        };
```

By looking at the code, it soon dawned on me what it was doing. Due to this being a decompilation of the bytecode, it is quite messy, but what it is doing in its essence is

```
rX = 'some-value';
rA = some_func;
rB = rA.bind()(rX); // This just calls the some_func function with 'some-value' as its argument
```

Let's look at a more practical example from the start of this function A (as given by the decompilation's `original name: A` comment).

```
 r1 = function(a0) { // Original name: A, environment: r2
            r1 = a0; // Put the input value to r1 variable. The input value is probably some string
            r0 = r1.toLowerCase; // Load the toLowerCase method of the input string to the r0 variable
            r4 = r0.bind(r1)(); // Call the toLowerCase method and assign it to r4 variable. The first parentheses is the `self` (instance) of the class and the second is its arguments. This `self` is passed to the first parentheses since we are calling a method of the string's class. If this were a normal function then there would be no value inside the first parentheses and the value would be in the second
```

What the above code just did is that it just converted the input string to lowercase. In python the equivalent would be

```
def A(a0):
    r1 = a0
    r4 = r1.lower()
```

Let's look at a bigger example with the `A` function and what it does:

```
r1 = function(a0) { // Original name: A, environment: r2
            r1 = a0;
            r0 = r1.toLowerCase;
```

```
            r4 = r0.bind(r1)(); // r4 = a0.lower()
            r1 = _closure1_slot5; // I do not what r1 is currently, but
these _closure_slots seem to be global variables/libraries
            r3 = r1.bytesToHex; // Seems like the _closure1_slot5 was an
imported strings library that holds the bytesToHex function
            r1 = _closure1_slot2;
            r1 = r1.blake3; // Seems like _closure1_slot2 was a
cryptography library that holds the blake3 function
            r2 = undefined;
            r1 = r1.bind(r2)(r4); // This calls blake3(a0.lower())
            r4 = r3.bind(r2)(r1); // this calls bytesToHex on the output
of blake3
            r1 = _closure1_slot5;
            r3 = r1.bytesToHex;
            r1 = _closure1_slot4;
            r1 = r1.sha3_384;
            r1 = r1.bind(r2)(r4); // Calls sha3_384 on the bytestring of
blake3 output
            r3 = r3.bind(r2)(r1); // Converts the previous result to a
hexstring
            r1 = _closure1_slot5;
            r1 = r1.bytesToHex;
            r0 = _closure1_slot3;
            r0 = r0.sha512;
            r0 = r0.bind(r2)(r3); // Hashes the previous result with
sha512
            r0 = r1.bind(r2)(r0); // Converts the sha512 result to
hexstring
            return r0;
        };
```

Let's rewrite this in python:

```
import hashlib
from blake3 import blake3

def A_aka_hash_input(inpt: str):
    inpt = inpt.lower().encode()
    hashed = blake3(inpt).hexdigest().encode()
    hashed = hashlib.sha3_384(hashed).hexdigest().encode()
    hashed = hashlib.sha512(hashed).hexdigest()
    return hashed
```

This was function A, looking around in that same area we also find functions D and C:

```
r0 = function(a0) { // Original name: D, environment: r2
            r1 = a0;
            r0 = r1.toLowerCase;
```

```
            r3 = r0.bind(r1)();
            r1 = _closure1_slot6;
            r5 = r1.pbkdf2;
            r0 = _closure1_slot3;
            r4 = r0.sha512;
            r0 = global;
            r1 = r0.TextEncoder;
            r2 = r1.prototype;
            r2 = Object.create(r2, {constructor: {value: r1}});
            r10 = r2;
            r1 = new r10[r1](r9);
            r2 = r1 instanceof Object ? r1 : r2;
            r1 = r2.encode;
            r8 = r1.bind(r2)(r3);
            r0 = r0.TextEncoder;
            r1 = r0.prototype;
            r1 = Object.create(r1, {constructor: {value: r0}});
            r10 = r1;
            r0 = new r10[r0](r9);
            r2 = r0 instanceof Object ? r0 : r1;
            r1 = r2.encode;
            r0 = 'wordle-ctf-salt';
            r7 = r1.bind(r2)(r0);
            r10 = undefined;
            r6 = {'c': 10000, 'dkLen': 32};
            r9 = r4;
            r0 = r10[r5](r9, r8, r7, r6, r5);
            return r0;
        };
```

```
r0 = function(a0) { // Original name: C, environment: r2
            r0 = _closure1_slot8;
            r2 = r0.base64;
            r1 = r2.decode;
            r0 = a0;
            r0 = r1.bind(r2)(r0);
            return r0;
        };
```

Converting these to python we get these functions:

```
from base64 import b64decode
import hashlib

def C_aka_b64decode(inpt: bytes):
    return b64decode(inpt)

def D_aka_get_key(inpt: str):
```

```
    inpt = inpt.lower().encode()
    salt = 'wordle-ctf-salt'.encode()
    iterations = 10000
    dkLen = 32
    h = hashlib.pbkdf2_hmac('sha512', inpt, salt, 10000, 32)
    return h
```

A bit further we also find the `B` function:

```
r1 = function(a0) { // Original name: B, environment: r2
            r3 = call_A_aka_blake3_hash_func;
            r2 = undefined;
            r1 = a0;
            r1 = r3.bind(r2)(r1);
            r0 = LONG_BYTESTRING_MAYBE_HASH;
            r0 = r1 === r0;
            return r0;
        };
```

Which converted to python is:

```
def B(inpt: str):
    hashed = A_aka_hash_input(inpt)
    return hashed == long_hashstring
```

Note that the `call_A_aka_blake3_hash_func` comes from the fact that after the function `A` is defined, the value is assigned to a closure slot, which I renamed so I could read the code better:

```
r1 = function(a0) { // Original name: A, environment: r2
            r1 = a0;
            r0 = r1.toLowerCase;
...
            return r0;
        };
        var call_A_aka_blake3_hash_func = r1;
```

A similar thing is done to all the other functions as well.

Now since this function A seems to be producing the hashes we saw earlier in the app and this function B seems to be comparing them, I took an educated guess that this function B checks whether the inputted word is the correct one.

```
for i in VALID_WORDS:
    if B(i):
```

```
        print(i)
```

This script outputs the word `MOBIL`, which indeed gives us the flag when put into the app.

I however, had reversed the decryption function as well before this so I got the flag in copy-pasteable form on my console, instead of in the mobile app.

Btw the decryption function looks like this in python (was called anon_func or sth in the decompiled code):

```
def _anon_func_aka_decrypt(inpt: str):
    aes_key = D_aka_get_key(inpt)
    a, b = aes_iv_and_encrypted_ciphertext.split(':')
    iv = C_aka_b64decode(a)
    ciphertext = C_aka_b64decode(b)

    cipher = AES.new(aes_key, AES.MODE_CBC, iv=iv)
    decrypted = cipher.decrypt(ciphertext)

    return decrypted.decode()
```

Final code to get flag is then:

```
from base64 import b64decode
import hashlib
from blake3 import blake3
from Crypto.Cipher import AES

VALID_WORDS = TARGET_WORDS = ['ABORT', 'AKTIE', 'AKTIV', 'ALARM', 'ALTAN',
'ANDEN', 'ANGST', 'APPEL', 'AREAL', 'AFTEN', 'BANAN', 'BASIS', 'BEDST',
'BEGGE', 'BEHOV', 'BETAL', 'BINDE', 'BLANK', 'BLIND', 'BLODT', 'BOLIG',
'BOMBE', 'BORDE', 'BREVE', 'BRAGE', 'BRAND', 'BRUGT', 'BRUGE', 'BÆLTE',
'BÅDEN', 'CREME', 'CYBER', 'CITAT', 'CELLE', 'DANSK', 'DIGIT', 'DIRKE',
'DRAGE', 'DRAMA', 'DRENG', 'DRIKKE', 'DRONE', 'ENHED', 'ETHER', 'ELLER',
'FAKTA', 'FALDE', 'FARVE', 'FATAL', 'FERIE', 'FJORD', 'FLADE', 'FLEST',
'FLØDE', 'FOKUS', 'FORME', 'FRISK', 'FUGLE', 'FÆLDE', 'GAMLE', 'GERNE',
'GIVER', 'GREJE', 'GRUND', 'GRØNT', 'GUIDE', 'HASTE', 'HAVDE', 'HAVEN',
'HENTE', 'HJÆLP', 'HOBBY', 'HOLDE', 'HOTEL', 'HUNDE', 'HUSET', 'HYGGE',
'HØLOF', 'IDEEL', 'INDRE', 'INPUT', 'IRONI', 'JORDE', 'KAFFE', 'KALDE',
'KASTE', 'KLOGE', 'KNUDE', 'KOBLE', 'KODER', 'KONGE', 'KRAFT', 'KRIGE',
'KUNST', 'KURVE', 'KVÆLE', 'KÆMPE', 'KØBER', 'LANDE', 'LANGE', 'LASER',
'LEVEN', 'LISTE', 'LOGIK', 'LUNGE', 'LYGTE', 'LÆBER', 'LÆGER', 'LØBER',
'LÅGER', 'MADEN', 'MAGER', 'MANGE', 'MAPPE', 'MARTS', 'METAL', 'MILJØ',
'MINDE', 'MIXER', 'MOBIL', 'MÅNED', 'MÅSKE', 'NABOR', 'NAKKE', 'NATUR',
'NISSE', 'NORDE', 'NOTAT', 'NYHED', 'NÆVNE', 'NØGLE', 'OFFER', 'OLIER',
'OPTAG', 'ORDEN', 'ORDRE', 'ORGAN', 'OXIDE', 'PAKKE', 'PANDE', 'PAPIR',
'PAUSE', 'PENGE', 'PIXEL', 'PLADS', 'PLEJE', 'POINT', 'PRINS', 'PRØVE',
```

```
'PUNKT', 'PÅSKE', 'RADAR', 'RAMME', 'RATIO', 'REGEL', 'REGNE', 'REJSE',
'RETUR', 'ROBOT', 'ROLLE', 'ROLIG', 'RULLE', 'RYTME', 'RÆKKE', 'RÅBER',
'SALAT', 'SAMLE', 'SENDE', 'SERIE', 'SIKRE', 'SKABE', 'SKADE', 'SKIFT',
'SKOLE', 'SKRÅT', 'SKÆRM', 'SLOTS', 'SMIDE', 'SMUKK', 'SOBER', 'SOGNE',
'SOLID', 'SPARE', 'SPIDS', 'SPILL', 'SPROG', 'SPØGE', 'STANG', 'STAVE',
'STEMT', 'STILE', 'STOLE', 'STORM', 'STRAM', 'STRIK', 'STRØM', 'STYRE',
'STORE', 'STØJT', 'SVAMP', 'SVARE', 'SVÆRt', 'SYNES', 'SOGNE', 'TABEL',
'TAKKE', 'TALES', 'TAVLE', 'TEAMS', 'TERMA', 'TEMPO', 'TIGER', 'TIMER',
'TJENE', 'TONER', 'TRACK', 'TRINE', 'TRYGE', 'TRÆNE', 'TRÆTE', 'TUNGE',
'TVIVL', 'TÆNKE', 'TØJET', 'UDLØB', 'UDRYK', 'UDVEJ', 'RÆKKE', 'VANDE',
'VARME', 'VENDE', 'VENNE', 'VIDEO', 'VILJE', 'VIRKE', 'VIRUS', 'VITAL',
'VÆRDI', 'VÆSKE', 'VÅBEN', 'YNGRE', 'YNDIG'];
GAME_MESSAGES = {'INVALID_LENGTH': 'Word must be X letters',
'NOT_IN_WORDLIST': 'Not in word list', 'GAME_OVER': 'Game is already
over'}
MAX_GUESSES = 6
WORD_LENGTH = 5

aes_iv_and_encrypted_ciphertext =
'7y7q06tmjufJu3UucuB/xg==:pYCr5aNWDhx1hkqWS4+erLkzDb3bF4P+/BhoD9SPsbXZWYpe
3hx0yaFZBJaAu7nr'

long_hashstring =
'0d2462ac7a20b9814032653168ef0238a4eb0f1191a33c2d305356728de16556dd5b8586b
0905ce244cb029a3f95a3727d74de879adde0ad76763c097b31e3fe'


def A_aka_hash_input(inpt: str):
    inpt = inpt.lower().encode()
    hashed = blake3(inpt).hexdigest().encode()
    hashed = hashlib.sha3_384(hashed).hexdigest().encode()
    hashed = hashlib.sha512(hashed).hexdigest()
    return hashed


def B(inpt: str):
    hashed = A_aka_hash_input(inpt)
    return hashed == long_hashstring

def C_aka_b64decode(inpt: bytes):
    return b64decode(inpt)


def D_aka_get_key(inpt: str):
    inpt = inpt.lower().encode()
    salt = 'wordle-ctf-salt'.encode()
    iterations = 10000
    dkLen = 32
    h = hashlib.pbkdf2_hmac('sha512', inpt, salt, 10000, 32)
    return h
```

```
def _anon_func_aka_decrypt(inpt: str):
    aes_key = D_aka_get_key(inpt)
    a, b = aes_iv_and_encrypted_ciphertext.split(':')
    iv = C_aka_b64decode(a)
    ciphertext = C_aka_b64decode(b)

    cipher = AES.new(aes_key, AES.MODE_CBC, iv=iv)
    decrypted = cipher.decrypt(ciphertext)

    return decrypted.decode()

def main():
    inpt = 'Value'
    try:
        _anon_func_aka_decrypt(inpt)
    except:
        print('Flag decryption failed/Unknown error occured')


for i in VALID_WORDS:
    if B(i):
        print(i)

print(_anon_func_aka_decrypt("MOBIL"))
```

(Some other stuff is added which I did not talk about but they are not relevant to solving the challenge).