Problem Set 1

PAWS 2025

Nathaniel Hurst

1 Instead of using a shift of exactly 3 letters in Caesar's cipher, one could also use a secret shift depending on a key $k \in \{0, ..., 25\}$. Describe $\mathcal{M}, \mathcal{C}, K$, **Dec** and **Enc** for this new encryption method.

Let $m \in \mathcal{M} = \{0, ..., 25\}$ and $k \in \{0, ..., 25\}$. Then $\mathbf{Enc}(m) = c$ where $c = m + k \pmod{26}$ and $\mathbf{Dec}(c) = c - k \pmod{26}$. In this way we have **Dec**(**Enc**(m))=**Dec**$(m + k) = m + k - k = m$ for all $m \in \mathcal{M}$

2 To increase the number of keys for the Caesar cipher, one may also choose a key of the form $k = (a, b)$ with $a \in (\mathbb{Z}/26\mathbb{Z})^*, b \in (\mathbb{Z}/26\mathbb{Z})$, and

$$\mathrm{Enc}_{k(m_1,...,m_n)} = (c_1, ..., c_n) \text{ with } c_i = am_i + b$$

The corresponding scheme is known as an affine cipher.

(a) Describe the corresponding decryption function Dec_k. Why is it necessary that a is a unit in $(\mathbb{Z}/26\mathbb{Z})$?

(b) You (Eve) read a cipher text starting with BMVVK and you think that it means HELLO. Is it possible that Alice and Bob used an affine cipher in their communication? Can you recover their secret key?

(c) Alice and Bob noticed that you found their secret, and chose a new private key. This time you intercept the cipher text:

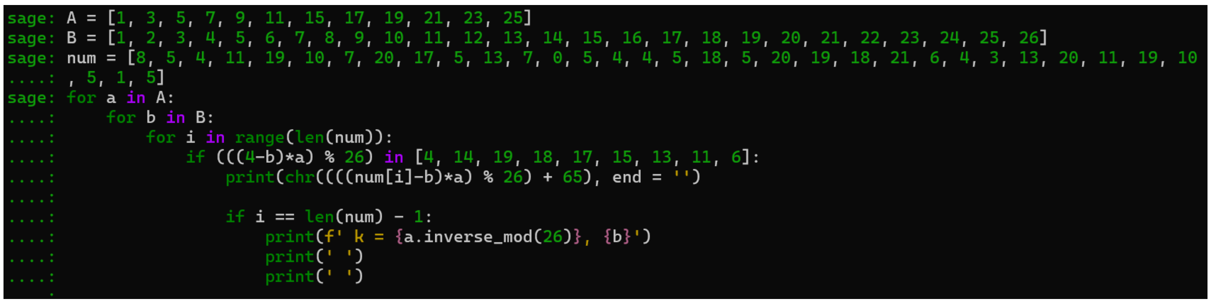
IFELTKHURFENHAFEEFSFUTSVGEDNULTKFBF

Can you find the plain text message?

(a) $\mathrm{Dec}_{(a,b)(c_1,...,c_n)} = (m_1, ..., m_n)$ where $m_i = (c_i - b) \cdot a^{-1} \pmod{26}$. It is necessary that a is a unit in $(\mathbb{Z}/26\mathbb{Z})$ for if it was not then it would be possible that there would be no way to find m_i given $c_i - b = am_i$ (take for example $a \equiv 0 \pmod{26}$).

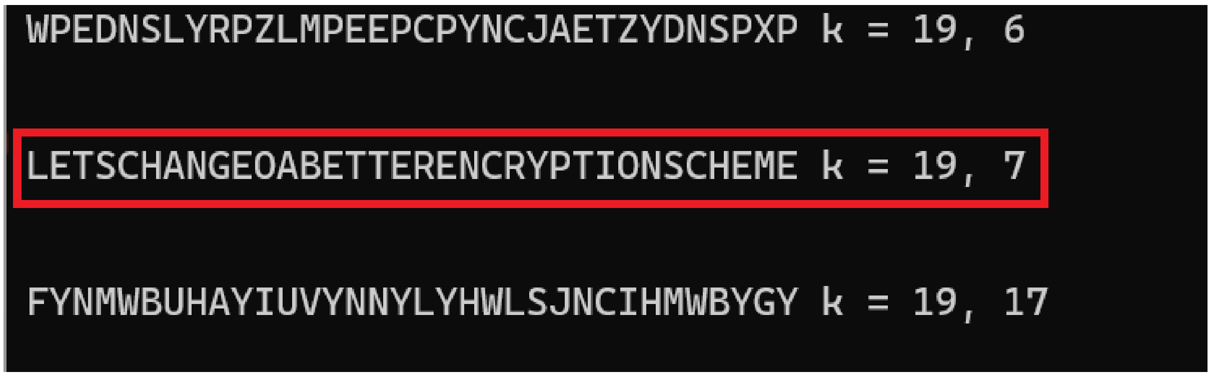
(b) Alice and Bob are using an affine cipher with the secret key $k = (5, 14)$ as HELLO $=$ $(8, 5, 12, 12, 15)$ and $\mathrm{Enc}_{(5,14)}((8, 5, 12, 12, 15)) = (2, 13, 22, 22, 11)$ which corresponds to BMVVK. This secret key was found through guess and check, as we only need to check each $a \in (\mathbb{Z}/26\mathbb{Z})^*$ and the b which solves $a(12) + b \equiv 22 \pmod{26}$ (which corresponds to encrypting $L = 12$ into $V = 22$) in this way we can check 12 possible key pairs $k = (a, b)$ and deduce the key $k = (5, 14)$.

(c) For this problem we employ SageMath. We can brute force this, but reduce the computations by only checking certain keys in which a double letter (i.e. tt) is encrypted as EE (there are only 9 such letters). Using this we find the private key $k = (a, b) = (19, 7)$ and the plaintext is LETSCHANGEOABETTERENCRYPTIONSCHEME



```
sage: A = [1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25]
sage: B = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
sage: num = [8, 5, 4, 11, 19, 10, 7, 20, 17, 5, 13, 7, 0, 5, 4, 4, 5, 18, 5, 20, 19, 18, 21, 6, 4, 3, 13, 20, 11, 19, 10
....: , 5, 1, 5]
sage: for a in A:
....:     for b in B:
....:         for i in range(len(num)):
....:             if (((4-b)*a) % 26) in [4, 14, 19, 18, 17, 15, 13, 11, 6]:
....:                 print(chr((((num[i]-b)*a) % 26) + 65), end = '')
....:
....:                 if i == len(num) - 1:
....:                     print(f' k = {a.inverse_mod(26)}, {b}')
....:                     print(' ')
....:                     print(' ')
....:
```

Figure 1: “Code”



```
WPEDNSLYRPZLMPEEPCPYNCJAETZYDNSPXP k = 19, 6

LETSCHANGEOABETTERENCRYPTIONSCHEME k = 19, 7

FYNMWBUHAYIUVYNNYLYHWLSJNCIHMWBYGY k = 19, 17
```

Figure 2: Results

3 How many affine ciphers are possible using the 26-letter English alphabet? How many are possible if we also allow the symbols “?” , “.” , “,” and “!”?

To construct an affine cipher in a 26-character alphabet we pick $a \in (\mathbb{Z}/26\mathbb{Z})^*$ and $b \in \mathbb{Z}/26\mathbb{Z}$ to form a pair $k = (a, b)$. Thus the number of possible keys k is $\varphi(26) \cdot 26 = 12 \cdot 26 = 312$ (where φ is Eulers totient function). Adding four more characters to our alphabet increases the size from 26 to 30, and thus the number of possible keys in this case is $\varphi(30) \cdot 30 = 8 \cdot 30 = 240$. Which is less than a 26 character alphabet! This is due to $\varphi(26) > \varphi(30)$.

4 You intercepted the cipher text

JIVQOJIV LEALAVQO KGOONDTV QOAELONE OAINYNGJ SOBVQODB CLAVQOKG OONDTJIV QOJIVLEA EIBHTBLO YBLEQPIG AA from a conversation between Alice and Bob. You know that they used a substitution cipher. Can you recover the plain text m? Note that the spacing is only used for readability and does not coincide with the spacing of the original text.

For this problem we employ frequency analysis in order to make educated guesses at the plain text m. Since O is the most common ciphertext letter we correspond this to the most common english letter E, and since thew bigram QO is the most common bigram in the ciphertext we correspond this to the most common english bigram (satisfying $E \to 0$) HE. From here we can fill in the rest of the letters with educated guesses to recover the plaintext

$$\text{MATHEMATICS IS THE QUEEN OF THE SCIENCES AND}$$
$$\text{NUMBER THEORY IS THE QUEEN OF MATHEMATICS CARL FRIEDRICH GAUSS}$$

5

Alice and Bob want to create a shared Diffie-Hellman key. They use setups with varying security levels. In all of these, the public parameters are a prime $p = 2q + 1$, and the element $g = 4 \in \mathbb{F}_p^*$ with order q. You observe the following conversations. Can you find the shared keys?

(a) $q = 4294967681 \approx 2^{32}$, $A = 5104411285$, $B = 7620748646$.

(b) $q = 18446744073709552109 \approx 2^{64}$, $A = 17485644247020728566$, $B = 17485644247020728566$.

(c) $q = 340282366920938463463374607431768219863 \approx 2^{128}$, $A = 15855669586157245378211095347605706305$, $B = 643791185530305885858740134964520672205$.

In SageMath, you can use the **log** function to compute discrete logarithms, i.e., $a = A.\log(g)$ (provided that g, A are defined as elements over $\mathbb{F}_p$). Further, you can use **%time** to time your results. How does the runtime evolve for increasing values of q?

(a) First we compute $p = 2q + 1 = 8589935363$. Then we compute $a = A.\log(g) = 121029226$. Then we can compute the shared key $B^a \equiv 3122549640 \pmod{p}$. The runtime for the discrete logarithm to compute a was 7.1 ms.

(b) First we compute $p = 2q + 1 = 36893488147419104219$. Then we compute $a = A.\log(g) = 17913846143021880100$. Then we can compute the shared key $B^a \equiv 10647114428957721787 \pmod{p}$. The runtime for the discrete logarithm to compute a was 190 ms.

(c) First we compute $p = 2q + 1 = 680564733841876926926749214863536439727$. Then we compute $a = A.\log(g) = 63361436478061474206645201191086014926$. Then we can compute the shared key $B^a \equiv 214488715430712623062490308139976786604 \pmod{p}$. The runtime for the discrete logarithm to compute a was 54 seconds. As seen from the evolution of the time to compute the DLP, it is clear that the DLP can get very hard when numbers get large (with current methods).

6

We now try to set up the ElGamal public key encryption scheme. We will start by doing that in the group $G = \mathbb{F}_{29}^*$.

(a) First we need to fix a generator for G, say $g = 2$. Check that g is actually a generator of G. Could we have chosen $g = 3$ or $g = 5$?

(b) Bob chooses his secret key $b = 5$, compute his public key.

(c) Bob receives the encryption $(c_1, c_2) = (7, 9)$ from Alice, decrypt it and find the message. (Bonus: do you think Bob can also find k?)

(d) Suppose now that Alice receives the public key $B = 14$ from Bob. Encrypt the message $m = 23$.

(e) Suppose that Bob sets his public key to $B = 28$. Do you think this is secure? Suppose Eve sees the encryption $(c_1, c_2) = (14, 22)$. Can she say something about the message?

(a) We must check that ord(g) = 28. Now since the order of g divides the order of the multiplicative group $(\mathbb{Z}/29\mathbb{Z})^*$ we must have that ord(g)$|28$, or ord(g) is one of $2, 4, 7, 14, 28$. Now observe that $2^2 \equiv 4 \not\equiv 1 \pmod{29}$, $2^4 \equiv 16 \not\equiv 1 \pmod{29}$, $2^7 \equiv 12 \not\equiv 1 \pmod{29}$, $2^{14} \equiv 28 \not\equiv 1 \pmod{29}$. Thus we are left to conclude that ord(g) = 28, or g is a primitive element of $\mathbb{F}_{29}^*$. Similarly we can deduce that 3 is also a primitive element and thus a valid generator, but 5 is not a primitive element, as $5^{14} \equiv 1 \pmod{29}$.

(b) Bob's public key is $B \equiv g^b \equiv 2^5 \equiv 3 \pmod{29}$.

(c) Since $7 \equiv c_1 \equiv 2^k \pmod{29}$ and $9 \equiv c_2 \equiv m \cdot B^k \equiv m \cdot 3^k \pmod{29}$ we can decrypt the message as $m \equiv \left(mB^k\right)\left(g^{(k)^b}\right)^{-1} \equiv 9 \cdot \left(7^2\right)^{-1} \equiv 9 \cdot 16 \equiv 28 \pmod{29}$. Since Bob only ever sees k in the form B^k and g^k Bob cannot find k unless he solves the discrete logarithm problem.

(d) We can encrypt the message by choosing $k = 5$ and computing $(c_1, c_2) = \left(2^5, 23 \cdot 14^5\right) = (3, 2)$.

(e) Since $B \equiv 28 \equiv -1 \pmod{29}$ we have $B^k \equiv 1 \pmod{29}$ if k even and $B^k \equiv -1 \pmod{29}$ if k odd. But since $22 = m \cdot B^k$ we have that if $B^k \equiv 1 \pmod{29}$ then $m \equiv 22 \pmod{29}$ and if $B^k \equiv -1 \pmod{29}$ then $m \equiv -22 \equiv 7 \pmod{29}$. Now since $2^k \equiv 14 \pmod{29}$ we must have that since

2 is a primitive element that $(2^{14})^k \equiv (-1)^k \equiv 14^{14} \equiv 28 \equiv -1 \pmod{29}$, or $(-1)^k \equiv -1 \pmod{29}$ giving us that k must be odd, or $m \equiv 7 \pmod{29}$.

7

Try now to implement the ElGamal public key encryption scheme in SageMath using prime fields. You need to implement the following functionalities:

(i) **KeyGen**, taking as input a prime p and a generator g, returning a key pair (sk, pk).

(ii) **Enc**, taking as input a prime p, a generator g, a public key pk and a message m, returning an encryption (c_1, c_2).

(iii) **Dec**, taking as input a prime p, a generator g, the secret key sk and an encryption (c_1, c_2), returning a message m.

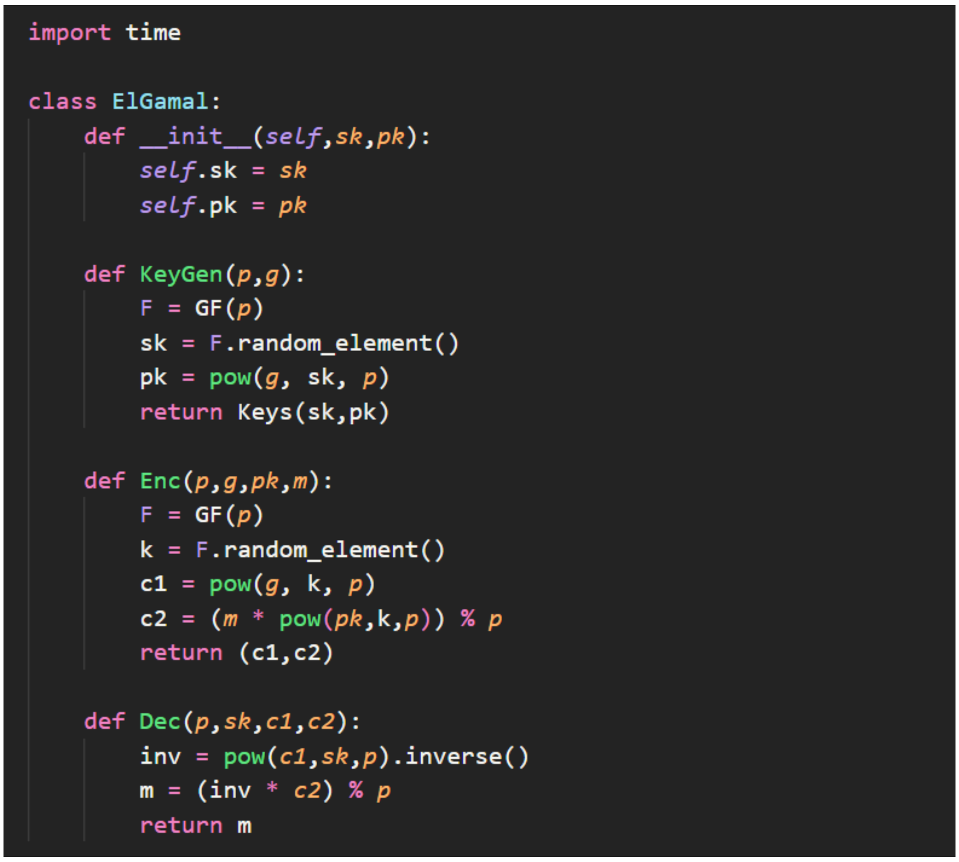
For the prime field and the generator use the parameter set **ffdhe3072**, in which

$$p = 2^{3072} - 2^{3008} + \left(\lfloor 2^{2942} \cdot e \rfloor + 2625351\right) \cdot 2^{64} - 1$$

and $g = 2$. You can find more information and the hexadecimal representation for the prime in the Appendix A and A.2 of the IETF standard Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). Try to time your implementations of **KeyGen**, **Enc** and **Dec**.

Some useful functions you can use in SageMath are the time library (load it with **import time**, then use **time.time()** to get the Unix time) the function **pow** (try to run **pow?** to see how to use it) or the constructor **GF(p)** to create the finite field $\mathbb{F}_p$.

You can also find this code in the Scripts folder on my github repo, in the file titled elgamal.ipynb



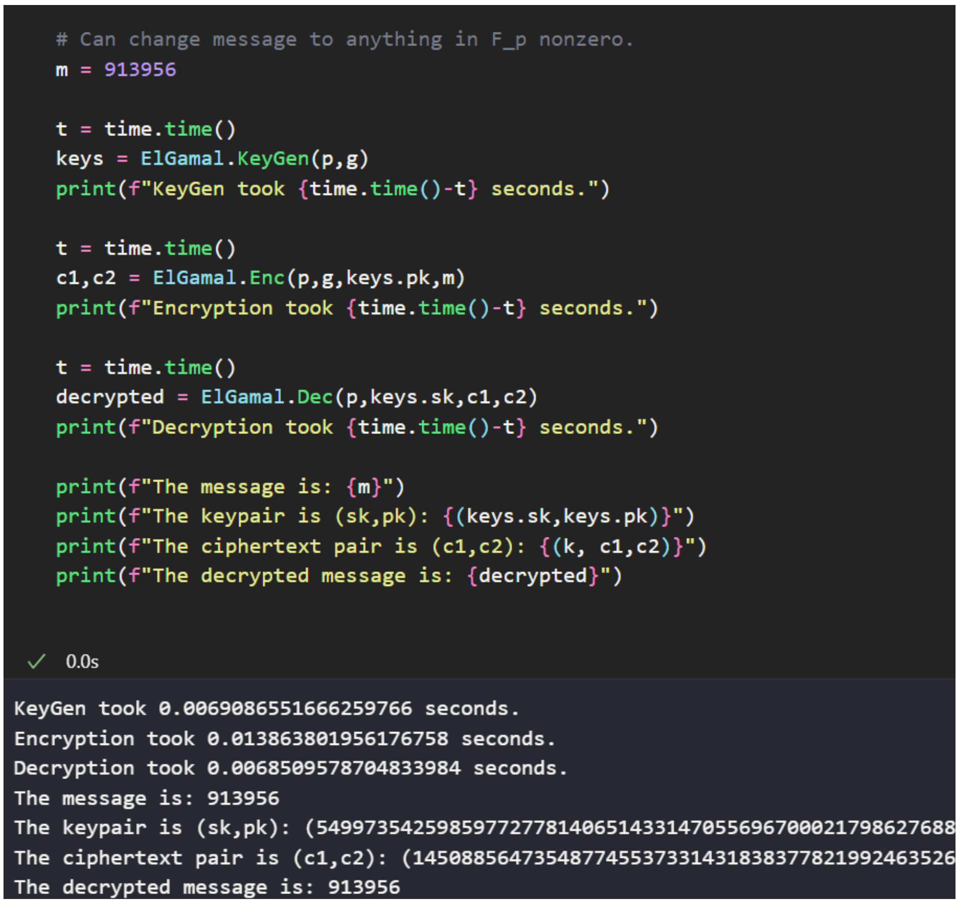
```python
import time

class ElGamal:
    def __init__(self,sk,pk):
        self.sk = sk
        self.pk = pk

    def KeyGen(p,g):
        F = GF(p)
        sk = F.random_element()
        pk = pow(g, sk, p)
        return Keys(sk,pk)

    def Enc(p,g,pk,m):
        F = GF(p)
        k = F.random_element()
        c1 = pow(g, k, p)
        c2 = (m * pow(pk,k,p)) % p
        return (c1,c2)

    def Dec(p,sk,c1,c2):
        inv = pow(c1,sk,p).inverse()
        m = (inv * c2) % p
        return m
```



```python
# Can change message to anything in F_p nonzero.
m = 913956

t = time.time()
keys = ElGamal.KeyGen(p,g)
print(f"KeyGen took {time.time()-t} seconds.")

t = time.time()
c1,c2 = ElGamal.Enc(p,g,keys.pk,m)
print(f"Encryption took {time.time()-t} seconds.")

t = time.time()
decrypted = ElGamal.Dec(p,keys.sk,c1,c2)
print(f"Decryption took {time.time()-t} seconds.")

print(f"The message is: {m}")
print(f"The keypair is (sk,pk): {(keys.sk,keys.pk)}")
print(f"The ciphertext pair is (c1,c2): {(k, c1,c2)}")
print(f"The decrypted message is: {decrypted}")
```

✓ 0.0s

```
KeyGen took 0.006908655166259766 seconds.
Encryption took 0.013863801956176758 seconds.
Decryption took 0.006850957870483398 seconds.
The message is: 913956
The keypair is (sk,pk): (549973542598597727781406514331470556967000217986276 88
The ciphertext pair is (c1,c2): (14508856473548774553733143183837782199246352 6
The decrypted message is: 913956
```

Figure 4: Note that in this screenshot I leave out initializing the prime as it is a long hex string.

8

Consider the following setup:

$$p = 8589935363, g = 4 \in \mathbb{F}_p^*,$$

and assume that Bob's public key is $B = 1865230978$.

(For readability, we chose a small prime for which the dlog can still be computed efficiently. For the sake of this exercise, assume however that you cannot compute $b = \log_g(B)$.)

Bob is asking some yes/no questions to Alice. Alice encrypts her answers ($Y = 25 \in \mathbb{F}_p^*$ for yes and $N = 14 \in \mathbb{F}_p^*$ for no) using Bob's public key and the ElGamal encryption scheme.

(a) Eve intercepts Alice's answers:

$$\text{Answer } 1 : (2456530342, 8487632028)$$

$$\text{Answer } 2 : (2456530342, 1660697205)$$

$$\text{Answer } 3 : (2456530342, 1660697205)$$

and immediately sees that Alice is reusing the random integer $k \in \mathbb{Z}$.

(i) Without doing any computations: What are the possible answers that Alice could have sent?

(ii) With some (computationally easy) computation: What are Alice's answers to Questions 1,2,3?

(b) Alice notices her mistake and uses different random exponents for the next answers. However she decides that it is easier to encode $Y = 1 \in \mathbb{F}_p^*$ and $N = -1 \in \mathbb{F}_p^*$. Now Eve intercepts the following messages:

$$\text{Answer } 4 : (63264669601, 8569725934)$$

$$\text{Answer } 5 : (5864877653, 1038689194)$$

$$\text{Answer } 6 : (1841857395, 573429127)$$

Can you recover Alice's answers to Question 4,5,6 as well?

(a)

(i) Immediately we see (since the parameter k was reused) that the possible answers were YNN or NYY.

(ii) It turns out we can find m using only modular inverses. We endevour to find m_1, m_2 given by $c_1 = m_1 B^k = 8487632028$ and $c_2 = m_2 B^k = 1660697205$. Since k was reused we can find $c_1 c_2 = m_1 m_2 B^{2k}$, and inverting $m_1 m_2$ (we know this is $25 \cdot 14$ as $m_1 \neq m_2$) and find $(m_1 m_2)^{-1}(c_1 c_2) = (m_1 m_2)^{-1}(m_1 m_2 B^{2k}) = B^{2k}$. Now we can invert B^{2k} and find $B^{-2k} c_1 = m_1 B^{-k}$ and $B^{-2k} c_2 = m_2 B^{-k}$. From here we can use process of elimination since we know that $m_1, m_2 \in (14, 25)$. We compute $25^{-1} m_1 B^{-k}$ and $14^{-1} m_1 B^{-k}$. Then we find $a_1 = c_1 25^{-1} m_1 B^{-k} = m_1 25^{-1} m_1$ and $a_2 = c_1 14^{-1} m_1 B^{-k} = m_1 14^{-1} m_1$. Now one of $a_1, a_2 \in (14, 25)$, and the other will be some other number, whichever is in the set gives us m_1. Doing this computation gives us that Alice's answers are YNN. (Note all computations are done modulo p)

(b) For this problem we can use results about quadratic residues. Observe first that $p \equiv 3 \pmod 4$ and so if a is a quadratic residue then $-a$ is not. Similarly we can compute if a is a quadratic residue by computing if $a^{\frac{p-1}{2}} \equiv 1 \pmod p$. In this way we see that B is a quadratic residue, and hence B^k is a quadratic residue as if $a^2 \equiv B \pmod p$ then $\left(a^k\right)^2 \equiv \left(a^2\right)^k \equiv B^k \pmod p$. Then to decode the message we need only to check if mB^k is a quadratic residue or not, for if it is then $m = 1$ and if not $m = -1$. Doing the computation gives us that Alice's answers were YYN.

9

The most widely used cryptosystem is RSA. The RSA algorithm works as follows: Bob chooses two secret large primes p and q; he computes $N = pq$, e such that $\gcd(e, (p-1)(q-1)) = 1$, and $d = e^{-1} \pmod{(p-1)(q-1)}$. Bob's public key is (N, e). Alice encrypts the message $m < N$ by computing $c = m^e \pmod N$. To decrypt, Bob computes $c^d \pmod N$.

(a) Suppose Bob's public key is $(55, 3)$. Decrypt the ciphertext $c = 12$.

(b) Why would an efficient algorithm for factoring make RSA insecure? (A fun fact: Shor's algorithm is an algorithm which enables quantum computers to factor efficiently; finding alternatives which are believed to be secure against quantum computers is an active area of research!)

(a) Since $N = 55$ is small we can factor it as $55 = 5 \cdot 11$ and hence we need only to compute $d = 3^{-1} \pmod{40}$. From here we find $d = 27$ and hence $m = c^d \equiv 23 \pmod{55}$.

(b) An efficient algorithm for factorization would make RSA insecure as an attacker could take Bob's public key and easily compute his private key by factoring $N = pq$.

10

Let p be a prime and $g \in \mathbb{F}_p^*$ a primitive element. We denote

$$p - 1 = p_1^{e_1} \ldots p_n^{e_n}$$

for the prime factorization of $p - 1$. The goal of this exercise is to show that solving the DLP in $\mathbb{F}_p^*$ is essentially as hard as solving the DLP in a subgroup $G \subset \mathbb{F}_p^*$ of prime order $\operatorname{ord}(G) = \max\{p_i \mid i \in (1, \ldots, n)\}$. To make this more formal, let us say that the DLP in a subgroup $G_i \subset \mathbb{F}_p^*$ of order p_i can be solved in time $O(S_i)$.

Let $g \in \mathbb{F}_p^*$ be a primitive element and $A \in \mathbb{F}_p^*$ the challenge for which we want to solve the DLP, i.e., we want to find $a \in \mathbb{Z}$ with $g^a = A$.

(a) Use the Chinese Remainder Theorem to translate the problem into solving n smaller instances of the DLP in subgroups of order $p_i^{e_i}$ with $i \in \{1, \ldots, n\}$, respectively.

(b) Fix $i \in \{1, ..., n\}$ and say you want to solve one of the small DLP instances on input $A_i, g_i \in \mathbb{F}_p^*$, where $\operatorname{ord}(g_i) = p_i^{e_i}$, i.e., you want to find $a_i \in \mathbb{Z}$ with

$$g_i^{a_i} \equiv A_i \pmod{p_i^{e_i}}$$

Show that this can be done in time $O(e_i S_i)$. Hint: Use a p_i-ary representation of $a_i = \alpha_0 + \alpha_1 p_i + ... + \alpha_{e_i - 1} p_i^{e_i - 1}$.

(c) Combine the results of (a) and (b) to show that the DLP can be solved in time

$$O(\operatorname{polylog}(p) \max\{S_i\})$$

(a) By the Chinese Remainder Theorem it suffices to find x_i such that $a \equiv x_i \pmod{p_1^{e_i}}$ for all $i \in \{1, ..., n\}$. To do this we can solve the DLP in subgroups of order p^{e_i}, i.e. we can find a such that $g^a \equiv A \pmod{p^{e_i}}$. Once we compute each x_i a solution to the original DLP is given by employing the Extended Euclidean algorithm and the Chinese Remainder Theorem. It is given by $a = \sum_{i=1}^n x_i M_i N_i$, where $N_i = \prod_{j \neq i} p_j^{e_j}$ and M_i satisfies $M_i N_i + m_i n_i = 1$ (given by the Extended Euclidean Algorithm). Now this solution a is actually given modulo $p - 1$ (as that is the integer we are factoring). However solutions to the DLP are only given in the range $\{0, ..., p - 1\}$, and 0 is a trivial solution only given when $g^a \equiv 1 \pmod{p}$ hence we still have the right solution.

(b) We want to solve $g_i^{a_i} \equiv A_i \pmod{p_i^{e_i}}$ given that the DLP in a subgroup of order p_i can be solved in time $O(S_i)$. Using the hint, we write $a_i = \alpha_0 + \alpha_1 p_i + ... + \alpha_{e_i - 1} p_i^{e_i - 1}$. Then we only need to solve the DLP in e_i subgroups of order p_i. More specifically we can solve for α_0 by writing $(g^a)^{p_i^{e_i - 1}} \equiv A^{p_i^{e_i - 1}} \pmod{p - 1}$, which gives us $g^{\alpha_0 p_i^{e_i - 1}}$, and solving the DLP for this gives us $\alpha_0 p_i^{e_i - 1}$, in which α_0 can be easily extracted using modular inverses. Now since $(g^a)^{p_i^{e_i - 1}}$ has order p_i (because $(g^a)^{\left(p_i^{e_i - 1}\right)^{p_i}} \equiv (g^a)^0 \equiv 1 \pmod{p_i^{e_i}}$ we know we can actually solve the DLP in $O(S_i)$ time, giving us α_0. Then we can solve $\left(\frac{g^a}{g^{\alpha_0}}\right)^{p_i^{e_i - 2}} \equiv \left(\frac{A}{g_0^{\alpha}}\right)^{p_i^{e_i - 2}} \pmod{p_i^{e_i}}$ in $O(S_i)$ time again since $\left(\frac{g^a}{g^{\alpha_0}}\right)^{p_i^{e_i - 2}}$ has order p_i since $\left(\frac{g^a}{g^{\alpha_0}}\right)^{\left(p_i^{e_i - 2}\right)^{p_i}} \equiv \left(\frac{g^a}{g^{\alpha_0}}\right)^{p_i^{e_i - 1}} \equiv g^0 \equiv 1 \pmod{p_i^{e_i}}$ Then solving the DLP in this subgroup gives $(a - \alpha_0) \cdot \left(p_i^{e_i - 2}\right) \equiv \alpha_1 p_i^{e_i - 1} \pmod{p_i^{e_i}}$ in which we can easily extract α_1 using modular inverses. Continuing in this fashion (dividing by g^{α_j} and raising this to the $p^{e_i - (j - 1)}$ to solve the DLP in a subgroup of order p_i) we find $\alpha_0, ..., \alpha_{e_i - 1}$, where we go through e_i iterations in order to contruct a the solution to the original DLP problem. Thus the time complexity of solving one of the small DLP instances $\left(\operatorname{mod} p_i^{e_i}\right)$ is $O(e_i S_i)$.

(c) Since in (a) we found that if $p - 1 = p_1^{e_1} ... p_n^{e_n}$ then we can translate the problem into solving the DLP in subgroups of order $p_i^{e_i}$, we can use our result in (b) to say that solving the DLP for such p takes $O(\operatorname{polylog}(p) \max\{S_i\})$, for $\max\{S_i\}$ gives us the worst case for solving the DLP in each subgroup, and the number of such solutions we will have to compute is $\prod_{i=1}^n e_i$, in which each e_i is given by taking logarithms of $p = p_1^{e_1} ... p_n^{e_n}$.

11

To avoid the attack described above in implementations we use **safe primes**, i.e., primes p so that $p - 1 = 2q$ with q also prime (primes as q are also known as Sophie Germain primes). Try to understand if the following are safe primes or not:

(a) $p_1 = 140970312865529183$

(b) $p_2 = 282481865344496003$

(c) $p_3 = 289942627069958089$

(d) $p_4 =$ 0x57f6fbca67731551934e21ad7372f2402eb9cce3f77dbe8e4fcce5052bee98efb

(e) $p_5 =$ 0x7404cc9709ed7d6a4e7551e85465df1c5bd4855274bff5d392da63732baa65

(f) the prime from the parameter set ffdhe4096 (see the IETF standard Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS))

(a) This is not a safe prime as $\frac{p-1}{2}$ is divisible by 7 (factored with Sage) and hence not prime.

(b) This prime is also not safe as $\frac{p-1}{2}$ is divisible by 1783 (factored with Sage), and hence not prime.

(c) This prime is not safe, as $\frac{p-1}{2}$ is even and greater than 2, and hence cannot be a prime.

(d) This is not a safe prime as $\frac{p-1}{2}$ is divisible by 1061815659967 (factored with Sage), and hence not prime.

(e) This is not a safe prime as $\frac{p-1}{2}$ is even and greater than 2, and hence cannot be prime.

(f) Judging by the fact that Sage is unable to factor $\frac{p-1}{2}$ without timing out we can guess that this is a safe prime. A quick look at Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS confirms that it is such a prime.

12

Alice and Bob create a symmetric cipher in the following way: Their private key k is a large integer and their messages are d-digit integers, so

$$\mathcal{M} = \{m \in \mathbb{Z} : 0 \leq m < 10^d\}$$

To encrypt a message, Alice computes $\sqrt{k}$ to d decimal places and lets α be the d-digit number to the right of the decimal place. (For example, if $k = 87$ and $d = 6$ then $\sqrt{87} = 9.32737905...$ and $\alpha = 327379$.)

Alice encrypts m as $c \equiv m + \alpha \pmod{10^d}$. Bob decrypts c by computing $m \equiv c - \alpha \pmod{10^d}$.

(a) Alice and Bob choose the secret key $k = 11$ and use $d = 6$. Bob wants to send Alice the message $m = 328973$. What is the ciphertext that he sends?

(b) Alice and Bob choose the secret key $k = 23$ and use it to encrypt 8-digit integers. Alice receives $c = 78183903$. What is the plaintext m?

(c) Show that

$$\alpha = \left\lfloor 10^d \left(\sqrt{k} - \left\lfloor \sqrt{k} \right\rfloor\right)\right\rfloor$$

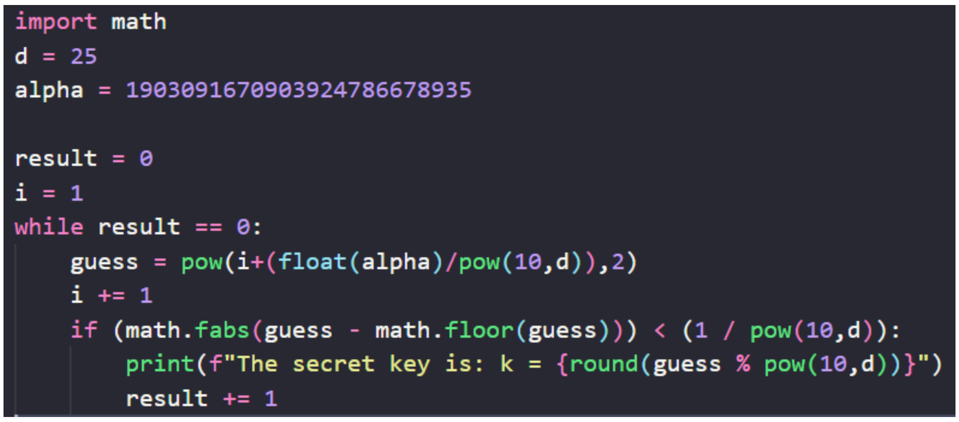
(d) (Challenge!) If Eve steals a plaintext/ciphertext pair (m, c), then she can easily compute $\alpha \equiv c - m \pmod{10^d}$. If 10^d is large compared to k, can she also recover k? (This would be useful if k is reused as a secret key, for greater values of d.)

(a) First we compute α as the first 6 digits of the decimal expansion of $\sqrt{11} = 3.31662479036...$ giving us $\alpha = 316624$. Then we encrypt $c \equiv m + \alpha \equiv 328973 + 316624 \equiv 645597 \pmod{10^6}$.

(b) First we compute α as the first 8 digits of the decimal expansion of $\sqrt{23} = 4.79583152331...$ giving us $\alpha = 79583152$. Then to decrypt we do $m \equiv c - \alpha \equiv 78183903 - 79583152 \equiv 98600751 \pmod{10^8}$.

(c) Observe that $\left\lfloor \sqrt{k} \right\rfloor$ gives the decimal expansion of $\sqrt{k}$ including only digits to the left of the decimal point. Then $\sqrt{k} - \left\lfloor \sqrt{k} \right\rfloor$ gives $0.D_1D_2...$, where $D_1D_2...$ is the decimal expansion of $\sqrt{k}$ only including digits to the right of the decimal point. Then $10^d\left(\sqrt{k} - \left\lfloor \sqrt{k} \right\rfloor\right)$ gives us $D_1D_2...D_d.D_{d+1}...$ (i.e. the previous number shifted D places to the left. Finally adding the floor function cuts off all digits to the right of the decimal point. Thus we are left with exactly the first d decimal places of the decimal expansion of $\sqrt{k}$, which is α.

(d) Since we have the stipulation that 10^d is large compared to k, we can brute force k given α with relative ease. More specifically we assume that $k \ll 10^d$ in order to say that $\sqrt{k}$'s decimal expansion is unique when considered against the other numbers up to k (so we will always find the right number - this tends to be true in practice, I tried a lot of numbers!). We employ the following algorithm (on the next page):



```python
import math
d = 25
alpha = 1903091670903924786678935

result = 0
i = 1
while result == 0:
    guess = pow(i+(float(alpha)/pow(10,d)),2)
    i += 1
    if (math.fabs(guess - math.floor(guess))) < (1 / pow(10,d)):
        print(f"The secret key is: k = {round(guess % pow(10,d))}")
        result += 1
```

Figure 5: This code can be found in the scripts folder on my github repo in the file named decimalexp.py. Try it out by square-rooting your favorite number and inputing the first d digit of its decimal expansion as alpha!