Written problems

1. Prove that breaking Diffie-Hellman is at least as hard as breaking ElGamal (with the same parameters p, g). More precisely, show that if Eve has access to an oracle function `DH(A,B)` which takes two numbers $g^a\%p$ and $g^b\%p$ and instantly returns the number $g^{ab}\%p$, then she can (using this oracle) efficiently decipher any messages encrypted with ElGamal.

   Solution.

   The information that Eve has, in the notation of p. 72 of the textbook, consists of the numbers p, g, A, c_1, c_2, where she knows that $A = g^a\%p, c_1 = g^k\%p$ and $c_2 \equiv mA^k \pmod p$, for two (unknown) exponents a and k. It follows that $A^k \equiv g^{ak} \pmod p$, so Eve can make the function call $\mathtt{DH}(A, c_1)$ to instantly obtain the number $A^k\%p$. She can find the inverse modulo p of this number using the extended Euclidean algorithm, multiply this inverse by c_2 and reduce modulo p to obtain the plaintext m.

2. Textbook exercise 2.16 (same in the first edition).

   Solution.

   (a) For all $x \geq 1$, $\sqrt{x} \leq x^2$. It follows that $x^2 + \sqrt{x} \leq 2x^2$ for all $x \geq 1$. Taking $C = 1, c = 2$ in the textbook's definition of big-$\mathcal{O}$ notation, it follows that $x^2 + \sqrt{x} = \mathcal{O}(x^2)$.
   Alternatively, compute that $\lim_{x\to\infty} \frac{x^2+\sqrt{x}}{x^2} = 1$.

   (b) For all $x \geq 1$, $1 \leq x^5$ and $x^2 \leq x^5$, so

   $$\begin{aligned} \left|5 + 6x^2 - 37x^5\right| &\leq |5| + |6x^2| + |37x^5| \\ &\leq 5x^5 + 6x^5 + 37x^5 \\ &\leq 48x^5 \end{aligned}$$

   So taking $C = 1, c = 48$ in the definition gives $5 + 6x^2 - 37x^5 = \mathcal{O}(x^5)$.
   Alternatively, compute that $\lim_{x\to\infty} \frac{5+6x^2-37x^5}{x^5} = -37$.

   (c) Observe that $\frac{(k+1)^{300}}{k^{300}} = \left(\frac{k+1}{k}\right)^{300} = \left(1 + \frac{1}{k}\right)^{300}$. Suppose that k is large enough that $\left(1 + \frac{1}{k}\right)^{300} \leq 2$; for example, this will hold if $k \geq \left(2^{1/300} - 1\right)^{-1}$. Let C be the number $\left(2^{1/300} - 1\right)^{-1}$, rounded up to the nearest integer. Then it follows by induction that for all $k \geq C$, $k^{300} \leq C^{300} \cdot 2^{k-C} = \frac{C^{300}}{2^C} \cdot 2^k$. In particular, taking $c = \frac{C^{300}}{2^C}$, it follows that $(k+1)^{300} \leq c \cdot 2^k$ for all $k \geq C$. Therefore $k^{300} = \mathcal{O}(2^k)$.
   Alternatively, compute that $\lim_{k\to\infty} \frac{k^{300}}{2^k} = 0$. This follows, for example, by observing that if l'Hôpital's rule is applied 200 times, then the resulting denominator will be a $(\ln 2)^{200} 2^k$, while the resulting numerator will be the constant 300!, hence the resulting limit is equal to 0.

   (d) Consider the limit $\lim_{k\to\infty} \frac{(\ln k)^{375}}{k^{1/1000}}$. There are various ways to verify that it is equal to 0; the following is one approach that obtains the result with one application of l'Hôpital's rule (in the second line).

$$\begin{aligned}
\lim_{k\to\infty} \frac{(\ln k)^{375}}{k^{1/1000}} &= \left[\lim_{k\to\infty} \frac{\ln k}{k^{1/375000}}\right]^{375} \\
&= \left[\lim_{k\to\infty} \frac{1/k}{\frac{1}{375000}k^{1/375000-1}}\right]^{375} \\
&= \left[\lim_{k\to\infty} \frac{375000}{k^{1/375000}}\right]^{375} \\
&= \left[\frac{375000}{\infty}\right]^{375} \\
&= 0
\end{aligned}$$

It takes a while for the numerator to outpace the denominator, but it does so so eventually. Therefore $(\ln k)^{375} = \mathcal{O}(k^{1/1000})$.

(e) Consider the limit of the ratio of these functions, and rearrange as follows.

$$\lim_{k\to\infty} \frac{k^2 2^k}{e^{2k}} = \lim_{k\to\infty} \frac{k^2}{(e^2/2)^k}$$

The key facts are: the numerator is a polynomial, and the denominator is an exponential function, where the base exceeds 1 (since $2 < e^2$) . If l'Hôpital's rule is now applied twice is a row, the numerator will be a constant (specifically 2), while the denominator will be a constant times an exponential function (with base exceeding 1); the resulting limit therefore is equal to 0. It follows that $k^2 2^k = \mathcal{O}(e^{2k})$.

(f) Similar logic works as is the previous problem.

$$\lim_{N\to\infty} \frac{N^{10} 2^N}{e^N} = \lim_{N\to\infty} \frac{N^{10}}{(e/2)^N}$$

Now the numerator is a polynomial, while the denominator is an exponential, with base exceeding 1. By the same logic as the previous problem, one could apply l'Hôpital's rule 10 times to obtain a limit in which the numerator is constant and the denominator is exponential; this limit must therefore be 0, as must the original limit. Thus $N^{10}2^N = \mathcal{O}(e^N)$.

3. We discussed in class why Bob should create a new ephemeral key (denoted k on page 72 of the textbook) each time he enciphers a message to Alice using her public key, otherwise he exposes himself to a known-plaintext attack from Eve. In this problem, we will see why he also should not just perform "small variations" on a previously-used ephemeral key.

(a) Suppose that Bob previously used an ephemeral key k_1 to send Alice a message m_1, and that Eve knows what this message is (as well as the enciphered version that Bob sent to Alice). Suppose Bob now enciphers another message m_2 to Alice, using the ephemeral key $k_2 = k_1 + 1$. Explain how Eve can efficiently detect that this is how Bob has obtained k_2 from k_1 (without necessarily determining what k_1 is), and efficiently extract the message m_2.

Solution.

Suppose that (c_{11}, c_{12}) is the first cipher text, m_1 is the first plaintext, and (c_{21}, c_{22}) is the second cipher text. Notice that since Eve knows m_1, and she knows that $c_{12} \equiv A^{k_1} m_1 \pmod{p}$, she can deduce that $A^{k_1} \equiv c_{12} m_1^{-1} \pmod{p}$ (where she computes m_1^{-1} with the extended Euclidean algorithm). So throughout, Eve knows the value of $A^{k_1} \% p$.

If Bob has chosen $k_2 = k_1 + 1$ then it will follow that $c_{21} \equiv g^{k_1+1} \equiv g^{k_1} \cdot g \equiv c_{11} g \pmod{p}$. Therefore Eve can detect this fact (without learning k_1) by noticing that $c_{21} = (c_{11} g) \% p$.

Once Eve knows that $k_2 = k_1 + 1$, she can make the following inferences.

$$\begin{aligned} A^{k_2} &\equiv A^{k_1} \cdot A \pmod{p} \\ \Rightarrow A^{k_2} &\equiv c_{12} m_1^{-1} A \pmod{p} \\ \Rightarrow c_{22} &\equiv c_{12} m_1^{-1} A m_2 \pmod{p} \\ \Rightarrow c_{22} c_{12}^{-1} m_1 A^{-1} &\equiv m_2 \pmod{p} \end{aligned}$$

Since Eve knows all of the terms on the left side (some immediately, some by inverting using the Euclidean algorithm), she can efficiently compute the value of m_2.

To summarize:

$$\begin{aligned} \text{If Eve observes that} \quad & c_{21} \equiv c_{11} g \pmod{p} \\ \text{Then she can quickly compute } m_2 \text{ as} \quad & c_{22} c_{12}^{-1} m_1 A^{-1} \% p. \end{aligned}$$

(b) Suppose instead that Bob obtains k_2 as $42k_1$. Explain how Eve can efficiently detect this, and extract the message m_2.

Solution.

If Bob chooses $k_2 = 42k_1$, then it follows that $c_{21} \equiv (c_{11})^{42} \pmod{p}$. Eve can detect this relationship between k_1 and k_2 by observing this congruence.

If Eve observes that $c_{21} \equiv (c_{11})^{42}$, she may deduce (by raising both sides to the power a) that $A^{k_2} \equiv (A^{k_1})^{42} \pmod{p}$. Therefore she may deduce that

$$\begin{aligned} c_{22} &\equiv A^{k_2} m_2 \pmod{p} \\ &\equiv \left(A^{k_1}\right)^{42} m_2 \pmod{p} \\ &\equiv \left(c_{12} m_1^{-1}\right)^{42} m_2 \pmod{p} \\ \Rightarrow c_{22} \left(c_{12}^{-1} m_1\right)^{42} &\equiv m_2 \pmod{p}. \end{aligned}$$

Summarizing:

$$\begin{aligned} \text{If Eve observes that} \quad & c_{21} \equiv c_{11}^{42} \pmod{p} \\ \text{Then she can quickly compute } m_2 \text{ as} \quad & c_{22} \left(c_{12}^{-1} m_1\right)^{42} \% p. \end{aligned}$$

(She should use the fast-powering algorithm for the exponentiation, as usual.)

(c) Suppose instead that Bob obtains k_2 as $k_1 - 5$. Explain how Eve can efficiently detect this, and extract the message m_2.

Solution.

Bob's choice will leave a fingerprint in the fact that $c_{21} \equiv c_{11} g^{-5} \pmod{p}$ (where as usual, g^{-5} denotes the fifth power of $g^{-1} \pmod{p}$). Eve can detect this fact about Bob's choice by noticing that this congruence holds.

If Eve detects this congruence, she may make the following inferences. The first one follows by raising both sides of $c_{21} \equiv c_{11} g^{-5} \pmod{p}$ to the power a.

$$\begin{aligned}
A^{k_2} &\equiv A^{k_1} A^{-5} \pmod{p} \\
&\equiv c_{12} m_1^{-1} A^{-5} \pmod{p} \\
\Rightarrow c_{22} &\equiv (A^{k_1} A^{-5}) m_2 \pmod{p} \\
&\equiv c_{12} m_1^{-1} A^{-5} m_2 \pmod{p} \\
\Rightarrow c_{22} c_{12}^{-1} m_1 A^5 &\equiv m_2 \pmod{p}.
\end{aligned}$$

Summarizing:

$$\begin{array}{rl}
\text{If Eve observes that} & c_{21} \equiv c_{11} g^{-5} \pmod{p} \\
\text{Then she can quickly compute } m_2 \text{ as} & c_{22} c_{12}^{-1} m_1 A^5 \% p.
\end{array}$$

(d) Suppose instead that Bob obtains k_2 as $13k_1 + 2$. Explain how Eve can efficiently detect this, and extract the message m_2.

Solution.

Bob's choice will leave a fingerprint in the congruence $c_{21} \equiv (c_{11})^{13} g^2 \pmod{p}$. Eve can detect this choice by noticing that this congruence holds.

Once Eve detects this fact, she can raise both sides to the power a (even though she does not know what a is, she still knows that the resulting congruence must hold!) and make the following inferences.

$$\begin{aligned}
A^{k_2} &\equiv \left(A^{k_1}\right)^{13} A^2 \pmod{p} \\
&\equiv \left(c_{12} m_1^{-1}\right)^{13} A^2 \pmod{p} \\
\Rightarrow c_{22} &\equiv \left(c_{12} m_1^{-1}\right)^{13} A^2 m_2 \pmod{p} \\
\Rightarrow c_{22} \left(c_{12}^{-1} m_1\right)^{13} A^{-2} &\equiv m_2 \pmod{p}
\end{aligned}$$

Summarizing:

$$\begin{array}{rl}
\text{If Eve observes that} & c_{21} \equiv (c_{11})^{13} g^2 \pmod{p} \\
\text{Then she can quickly compute } m_2 \text{ as} & c_{22} \left(c_{12}^{-1} m_1\right)^{13} A^{-2} \% p.
\end{array}$$

4. Recall that the *order of g modulo p* is defined to be the minimum positive number d such that $g^d \equiv 1 \pmod{p}$. It is often denoted $\mathrm{ord}_p(g)$. Prove that if e is any integer, then the orders of g and of g^e are related by the following formula.

$$\mathrm{ord}_p(g^e) \cdot \gcd(e, \mathrm{ord}_p(g)) = \mathrm{ord}_p(g)$$

Solution.

For brevity, denote the number $\mathrm{ord}_p(g)$ by L. Then we know that the sequence of powers of g modulo p is periodic with period exactly L; in particular, $g^d \equiv 1 \pmod{p}$ if and only if $L|d$.

Now, we wish to determine the smallest number x such that $(g^e)^x \equiv 1 \pmod{p}$. We can use the characterization above, as follows.

$$\begin{aligned} (g^e)^x \equiv 1 \pmod{p} \quad &\text{if and only if} \quad g^{ex} \equiv 1 \pmod{p} \\ &\text{if and only if} \quad L|(ex) \\ &\text{if and only if} \quad ex \equiv 0 \pmod{L}. \end{aligned}$$

Now, by the reasoning in problem 10 on the previous problem set, the congruence $ex \equiv 0 \pmod{L}$ is equivalent to the congruence $x \equiv 0 \pmod{\frac{L}{\gcd(e,L)}}$, which is equivalent to saying that $\frac{L}{\gcd(e,L)}$ divides x. In particular, the minimum positive such value of x is $\frac{L}{\gcd(e,L)}$ itself; this must therefore be the order of g^e.

We have therefore determined that

$$\mathrm{ord}_p(g^e) = \mathrm{ord}_p(g)/\gcd(e, \mathrm{ord}_p(g)),$$

which is equivalent to the desired equation.

5. Let p be any prime, and let g be an element of $(\mathbf{Z}/p)^\times$. Let q be another prime number. Show that $\mathrm{ord}_p(g) = q$ if and only if $g \not\equiv 1 \pmod{p}$ and $g^q \equiv 1 \pmod{p}$. Deduce that if g is a primitive root modulo p, then $g^{(p-1)/q} \pmod{p}$ is an element of order q.

Solution.

If the order of g is q, then by definition this means that $g^q \equiv 1 \pmod{p}$, but $g^e \equiv 1 \pmod{p}$ for all positive $e < q$. This proves one direction.

For the other direction, we use the fact that q is a prime number. If $g^q \equiv 1 \pmod{p}$, then it follows that q is a multiple of the order of g, i.e. $\mathrm{ord}_p(g)$ is a factor of q. Since we additionally know that $g^1 \not\equiv 1 \pmod{p}$, we know that this factor is not 1. Since q is prime, it follows that the order g is q on the nose.

For the last part of the problem, notice that if $h = g^{(p-1)/q}$, then Fermat's little theorem implies that $h^q \equiv 1 \pmod{p}$ (since it is congruent to g^{p-1}). The definition of primitive root implies that if g is a primitive root, then $g^{(p-1)/q}$ cannot be congruent to 1, since $0 < \frac{p-1}{q} < p - 1$. Therefore the order of h is exactly q.

Alternatively, we can deduce the last statement from the previous problem: if g is a primitive root, $\mathrm{ord}_p(g) = p - 1$, thus $\gcd(\frac{p-1}{q}, \mathrm{ord}_p(g)) = \frac{p-1}{q}$ and it follows, taking $e = \frac{p-1}{q}$, that the order of h is q.

6. Prove that If p, q are two primes such that $p \equiv 1 \pmod{q}$, then there exists an element $g \in \mathbf{Z}/p$ such that $\mathrm{ord}_p(g) = q$.

   Note. The importance of this fact comes from the fact that Diffie-Hellman (and related systems) are most secure when the order of the element g chosen is divisible by a large prime. One way to ensure this is to choose that large prime (q) in advance, then generate a prime p with $p \equiv 1 \pmod{q}$, and then identify the element g (which is now guaranteed to exist, by the choice of p).

   Solution.

   Since $p \equiv 1 \pmod{q}$, it follows that q divides $p-1$. We know that a primitive root always exists, so we can apply the previous problem to deduce that for any primitive root g, the element $g^{(p-1)/q} \pmod{p}$ has order exactly q.

7. Use the Babystep-Giantstep algorithm to compute each of the following discrete logarithms. Show your calculations in a table as on page 83 of the textbook.

   (a) $\log_{10}(13)$ for the prime $p = 17$.
   (b) $\log_{15}(16)$ for the prime $p = 37$.
   (c) $\log_5(72)$ for the prime $p = 97$.

   Solution.

   (a) Since $\sqrt{p-1} = 4$, we can work is "base 4." The first four powers of 10 modulo 17 are $10, 15, 14, 4$, so can build the first row of the necessary table as follows.

e	0	1	2	3
$10^e\%17$	1	10	15	14

   The second row of the table should contain the numbers $13 \cdot 10^{-4e}$. Since $10^4 \equiv 4 \pmod{17}$, inverting 4 yields $10^{-4} \equiv 13 \pmod{17}$, we the second row is obtained by first writing 13 and then multiplying by $13 \pmod{17}$ repeatedly, giving $13, 16, 4, 1$. Thus the full table is as follows.

e	0	1	2	3
$10^e\%17$	1	10	15	14
$13 \cdot 10^{-4e}\%17$	13	16	4	1

   The common element of these two rows is 1, which occurs as 10^0 and $13 \cdot 10^{-3\cdot 4}$. Thus $13 \equiv 10^{0+3\cdot 4} \pmod{17}$, i.e. $\log_{10}(13) = 12$.

   (b) In this case, $\sqrt{p-1} = 6$, so we can work "base 6." The first row is obtained by first writing 1 and multiplying by $15 \pmod{37}$ for each new box. Similarly, the second row begins with 16, and each new entry is obtained by multiplying by $15^{-6} \pmod{37}$. By calculating $10^6\%37$ and then inverting, we see that $15^{-6} \equiv 11 \pmod{37}$; so each subsequent entry in the second row is obtained by multiplying the previous entry by 11 and reducing modulo 37. The resulting table is as follows.

e	0	1	2	3	4	5
$15^e\%37$	1	15	3	8	9	24
$16 \cdot 15^{-6e}\%37$	16	28	12	21	9	25

   The common entry is 9, in column 4 of both rows. Therefore $15^4 \equiv 16 \cdot 15^{-6\cdot 4} \pmod{37}$, which implies that $16 \equiv 15^{4+6\cdot 4} \equiv 15^{28} \pmod{37}$. So $\log_{15}(16) = 28$.

(c) Proceed similarly as before. In this case, we work "base 10." In this case, the relevant multiplier on the bottom row is $5^{-10} \pmod{97}$, which is $11 \pmod{97}$. The chart is as follows.

e	0	1	2	3	4	5	6	7	8	9
$5^e\%97$	1	5	25	28	43	21	8	40	6	30
$72 \cdot 5^{-10e}\%97$	72	16	79	93	53	1	11	24	70	91

The common entry is 1, which occurs in columns 0 and 5 respectively. Thus $5^{0+10\cdot5} = 5^{50} \equiv 72 \pmod{97}$, i.e. $\log_5(72) = 50$.

Programming problems

8. Write a program to solve the discrete logarithm problem for 16-bit primes. This problem serves as a warm-up (and a way to test more efficient algorithms); a fairly naive method should work.

Solution.

Here is one implementation, which operates by initializing variable `ge` to 1, and repeatedly multiplies by g and reduces modulo p until a is found, then returns the number of multiplications that were performed.

```
def dlp_naive(p,g,a):
        e = 0
        ge = 1
        while ge != a:
                e += 1
                ge = ge*g % p
        return e

p,g,a = map(int,raw_input().split())
print dlp_naive(p,g,a)
```

9. Write a program to solve the discrete logarithm problem for 36-bit primes.

Solution.

The babystep-giantstep algorithm comfortably solves the discrete logarithm problem for 36-bit primes in the time limits set by hackerrank. Here is one possible implementation.

```
import math

# Computes modular inverse, using the Euclidean algorithm
def inverse(a,p):
        pre = (p,0)
        cur = (a,1)
        while cur[0] > 0:
                k = pre[0]/cur[0]
                nex = (pre[0]-k*cur[0],pre[1]-k*cur[1])
                pre = cur
                cur = nex
        assert(pre[0] == 1) # Make sure the inverse was found
        return pre[1]%p

# Finds indices of a common element of two lists
def coll_ind(l1, l2):
        ind_of = dict()
        for i in xrange(len(l1)):
                ind_of[ l1[i] ] = i
        for j in xrange(len(l2)):
                if l2[j] in ind_of:
                        return (ind_of[l2[j]],j)
        # Return none if no collision found
        return None

# Babystep-Giantstep algorithm
def dlp_bsgs(p,g,a):
        # Choose a "base"
        B = int(math.sqrt(p-1))+1

        # Make the list of babysteps (powers of g mod p)
        ge = 1 #Current power of g
        bs = []
        for e in xrange(B):
                bs += [ge]
                ge = ge*g % p

        # Compute g^(-B) modulo p.
        # Note that ge is currenlty set to g^B%p, so we can just invert it.
        h = inverse(ge,p)

        # Make the giantstep list
        nex = a
        gs = []
        for e in xrange(B):
                gs += [nex]
                nex = nex*h % p

        # Find the collision and compute the result
        (i,j) = coll_ind(bs,gs)
        return i + B*j

p,g,a = map(int,raw_input().split())
print dlp_bsgs(p,g,a)
```

10. Write a program which deciphers a message enciphered with ElGamal, given the parameters p, g and your private key.

    Solution.

    The necessary ingredients are a fast-powering function and a modular inverse function. Here is one possible implementation.

```
# Computes modular inverse, using the Euclidean algorithm
def inverse(a,p):
        pre = (p,0)
        cur = (a,1)
        while cur[0] > 0:
                k = pre[0]/cur[0]
                nex = (pre[0]-k*cur[0],pre[1]-k*cur[1])
                pre = cur
                cur = nex
        assert(pre[0] == 1) # Make sure the inverse was found
        return pre[1]%p

# Fast-powering algorithm
def pow_mod(a,e,p):
        res = 1
        while e>0:
                if e%2 == 1: res = res*a % p
                a = a*a % p
                e /= 2
        return res

def elg_decipher(c1,c2,p,g,a):
        # Find the "shared secret"
        S = pow_mod(c1,a,p)
        # Compute the plaintext
        return c2 * inverse(S,p) % p

# I/O code
p,g = map(int,raw_input().split())
a,A = map(int, raw_input().split())
c1,c2 = map(int,raw_input().split())
print elg_decipher(c1,c2,p,g,a)
```

11. Write a program that deciphers an message from Bob to Alice using ElGamal encryption, given the system parameters p, g, Alice's public key, knowledge of one previous plaintext

that Bob has sent to Alice, as well as information about the nature of Bob's poorly chosen "random" number generator. See the problem statement on hackerrank for details.

Solution.

The insight needed to solve this problem is similar to what we saw in problem 3. Eve does *not* have access to Alice's private key a, nor Bob's first ephemeral key k_1, and it is not practical for her to obtain either of these numbers (due to the difficulty of the discrete logarithm problem). She does, however, know $g^{k_1} \pmod{p}$, as well as $A^{k_1} \pmod{p}$ (she can extract the latter since she knows the first plaintext m_1, as $c_{12}m_1^{-1} \pmod{p}$).

Now, consider the behavior of Bob's random number generator. If we define

$$\begin{aligned} C &= 2^{2203} - 1 \\ D &= 42 \end{aligned}$$

Then the generator successively replaces k by $Ck - D$. Although Eve doesn't know k, she nonetheless can simulate this replacement. All she ultimately needs to know is the *effect of this replacement* on the values of the type numbers g^k and $A^k \pmod{p}$. And this is a calculation that she can do, since

$$\begin{aligned} g^{Ck-D} &\equiv (g^k)^C \cdot g^{-D} \pmod{p} \\ A^{Ck-D} &\equiv (A^k)^C \cdot A^{-D} \pmod{p} \end{aligned}$$

Eve knows that if she performs this replacement repeatedly, she will eventually (in less than 100 steps, in fact) reach a situation where the two numbers she has computed are (congruent to) g^{k_2} and A^{k_2}. She will know that she has reached this situation because the first number in her pair (the one that is $g^k \pmod{p}$ for some number k from Bob's generator) is equal to part 1 of the second cipher text, c_{21}. Once the reaches this situation, she will know (without knowing k_2 itself!) what the value of $A^{k_2} \pmod{p}$ is; that information is enough to extract m_2.

To summarize, Eve's strategy is to first compute the "shared secret" S (congruent to $A^{k_1} \pmod{p}$), then repeatedly "revise" the pair (c_{11}, S) (which she knows is initially $(g^{k_1}, A^{k_1}) \pmod{p}$) to transform is into a new pair (g^k, A^k), where k is the next number from Bob's generator. She will need to revise it at most 100 times before she'll find c_{21} as the first number in the pair. At this stage, she can invert the second number, multiply it by c_{22}, and obtain Bob's second plaintext.

Here is an implementation.

```
### Include the functions inverse and pow_mod from the last problem.

# Constants from Bob's generator
C = 2**2203 - 1
D = 42
```

```
# Function to break Bob's second ciphertext
def analyze(p,g,A,c11,c12,m1,c21,c22):
        # Find the first pair g^k, A^k
        gk = c11
        Ak = c12 * inverse(m1,p) % p

        # Invert g and A for future use
        gi = inverse(g,p)
        Ai = inverse(A,p)

        # Revise the pair (gk,Ak) until gk is equal to c21
        while gk != c21:
                gk = pow_mod(gk,C)*pow_mod(gi,D) % p
                Ak = pow_mod(Ak,C)*pow_mod(Ai,D) % p

        # Now gk and Ak are g^(k_2) and A^(k_2)
        return c22 * inverse(Ak,p) % p


# I/O
p,g = map(int,raw_input().split())
A = int(raw_input())
c11,c12,m1 = map(int,raw_input().split())
c21,c22 = map(int,raw_input().split())

print analyze(p,g,A,c11,c12,m1,c21,c22)
```