Written problems

1. Textbook exercise 7.1.

Solution.

(a)

$$\begin{aligned} h &\equiv f^{-1}g \pmod q \\ &\equiv 19928^{-1}18643 \pmod{918293817} \\ &\equiv 767748560 \end{aligned}$$

(b) Following the steps on page 375:

$$\begin{aligned} a &= f \cdot e\%q \\ &= 19928 \cdot 619168806 \pmod{918293817} \\ &= 600240756 \\ b &= a \cdot f^{-1}_{\pmod g}\%g \\ &= 600240756 \cdot 9764\%18643 \\ &= 11818 \end{aligned}$$

So the plaintext is 11818.

(c) Bob sends $(rh+m)\%q = (19564 \cdot 767748560)\%918293817 = 619167208$.

2. Suppose that the congruential cryptosystem on page 375 (which I was calling "1Tru" in class) is modified as follows.

- Four parameters F, G, M, R are chosen at the beginning, in addition to the modulus q.
- When Alice makes her private and public keys, she chooses f, g so that $0 \le f < F$ and $M \le g < G$. She then computes her public key h as on page 375.
- When Bob sends a message, he must choose his message m so that $0 \le m < M$, and he must choose his random number r so that $0 \le r < R$. Then he computes e as on page 375.

(a) What are the values of F, G, M, R (in terms of q) used in the system described on page 375?

(b) Suppose Alice computes b from e as on page 375. Write an inequality in terms of F, G, M, R, and q that will guarantee that the number b she computes is definitely equal to m.

(c) Explain why decryption could fail if the requirement $M \le g$ were dropped in Alice's procedure for generating her public key.

Solution.

(a) The congruential cryptosystem in section 7.1 uses the following bounds.

$$\begin{aligned} F &= \sqrt{q/2} \\ G &= \sqrt{q/2} \\ M &= \sqrt{q/4} \\ R &= \sqrt{q/2} \end{aligned}$$

(b) Alice needs to know that the congruence

$$a \equiv gr + fm \pmod{q}$$

implies an exact equality

$$a = gr + fm,$$

as this equation will allow decryption to proceed. A sufficient condition for this to occur is that $0 \leq gr + fm < q$. The lower bound is automatic since all the numbers are nonnegative. The upper bound will be guaranteed if the parameters are set so that

$$GR + FM < q.$$

(c) If the requirement $M \leq g$ is dropped, then encryption may become impossible, because the remainder $m\%g$ no longer uniquely determines m, so the last step in the decryption algorithm may not recover the actual original plaintext.

3. Suppose that Alice has a plaintext the form of an integer D. Describe a procedure by which Alice can convert this message into an element $\mathbf{m}$ that she can encrypt using NTru (in the notation of page 419). Such a procedure is called an *encoding scheme.*

   Solution.
   An NTru plaintext has the form $\mathbf{m} = \sum_{i=0}^{N-1} m_i X^i$, where $-\frac{p}{2} < m_i \leq \frac{p}{2}$ (or, depending on conventions, $0 \leq m_i < p$. One simple way to express an integer D in this form is to first express D in base p, i.e. write
   $$D = d_0 + p \cdot d_1 + p^2 \cdot d_2 + \cdots + p^{N-1} d_{N-1}$$
   (this will be possible with only N "digits" if and only if $D < p^N$), and then let m_i be the centerlift of d_i modulo p (or, depending on conventions, just set $m_i = d_i$ directly).

4. Textbook exercise 7.22.

   Solution.

   (a)
   $$\begin{aligned}(-1+4x+5x^2) \star (-1-3x-2x^2) &= 1+3x+2x^2 \\ &\quad -4x-12x^2-8 \\ &\quad -5x^2-15-10x \\ &= -22-11x-15x^2\end{aligned}$$

   (b)
   $$\begin{aligned}(2-x+3x^3-3x^4) \star (1-3x^2-3x^3-x^4) &= 2-6x^2-6x^3-2x^4 \\ &\quad -x+3x^3+3x^4+1 \\ &\quad +3x^3-9-9x-3x^2 \\ &\quad -3x^4+9x+9x^2+3x^3 \\ &= -6-x+3x^3-2x^4\end{aligned}$$

   (c)
   $$\begin{aligned}(x+x^2+x^3) \star (1+x+x^5) &= x+x^2+1 \\ &\quad +x^2+x^3+x \\ &\quad +x^3+x^4+x^2 \\ &= 1+2x+3x^2+2x^3+x^4\end{aligned}$$

(d)

$$\begin{aligned}(x+x^2+x^3+x^4+x^6+x^7+x^9)\star(x^2+x^3+x^6+x^8) &= x^3+x^4+x^7+x^9\\ &\quad +x^4+x^5+x^8+1\\ &\quad +x^5+x^6+x^9+x\\ &\quad +x^6+x^7+1+x^2\\ &\quad +x^8+x^9+x^2+x^4\\ &\quad x^9+1+x^3+x^5\\ &\quad x+x^2+x^5+x^7\\ &= 3+2x+3x^2+2x^3+3x^4\\ &\quad +4x^5+2x^6+3x^7+2x^8+4x^9\end{aligned}$$

5. Textbook exercise 7.23.

Solution.

(a)

$$\begin{aligned}(1+x)\star(-5+4x+2x^3) &= -5+4x+2x^2\\ &\quad -5x+4x^2+2\\ &= -3-x+6x^2\\ &\equiv 4+6x+6x^2 \quad \text{(mod 7) (usual reduction)}\\ &\equiv -3-x-x^2 \quad \text{(mod 7) (centerlift)}\end{aligned}$$

(b)

$$\begin{aligned}&(2+2x-2x^2+x^3-2x^4)\\ &\star(-1+3x-3x^2-3x^3-3x^4) \equiv (2+2x+2x^2+x^3+2x^4)\\ &\qquad \star(-1-x+x^2+x^3+x^4) \quad \text{(mod 4)}\\ &\equiv -2-2x+2x^2+2x^3+2x^4\\ &\qquad -2x-2x^2+2x^3+2x^4+2\\ &\qquad -2x^2-2x^3+2x^4+2+2x\\ &\qquad -x^3-x^4+1+x+x^2\\ &\qquad -2x^4-2+2x+2x^2+2x^3 \quad \text{(mod 4)}\\ &\equiv 1+x+x^2+3x^3+3x^4 \quad \text{(mod 4) (usual reduction)}\\ &\equiv 1+x+x^2-x^3-x^4 \quad \text{(mod 4) (centerlift)}\end{aligned}$$

(c)

$$\begin{aligned}(x+x^3)\star(x+x^2+x^4+x^6) &= x^2+x^3+x^5+1\\ &\quad +x^4+x^5+1+x^2\\ &= 2+2x^2+x^3+x^4+2x^5 \text{ (usual reduction)}\\ &\equiv -1-x^2+x^3+x^4-x^5 \quad \text{(mod 3) (centerlift)}\end{aligned}$$

(d)

$$\begin{aligned}
(x^2+x^5+x^7+x^8+x^9) & \\
\star(1+x+x^3+x^4+x^5+x^7+x^8+x^9) &= x^2+x^3+x^5+x^6+x^7+x^9+1+x \\
&\quad +x^5+x^6+x^8+x^9+1+x^2+x^3+x^4 \\
&\quad +x^7+x^8+1+x+x^2+x^4+x^5+x^6 \\
&\quad +x^8+x^9+x+x^2+x^3+x^5+x^6+x^7 \\
&\quad +x^9+1+x^2+x^3+x^4+x^6+x^7+x^8 \\
&= 4+3x+5x^2+4x^3+3x^4+4x^5+5x^6+4x^7+4x^8+4x^9 \\
&\equiv x+x^2+x^4+x^6 \pmod 2
\end{aligned}$$

6. Textbook exercise 7.25.

Solution.

We can learn whether an inverse exists (and find it if it does) manually by attempting to solve for its coefficients. You can find more general (and more efficient) methods in parts of the book that we aren't covering, but it is instructive to do a few examples by hand.

Suppose that $\mathbf{c}(x)$ is an inverse for $\mathbf{a}(x)$. Then:

$$(1+x^2+x^3)\star(c_0+c_1x+c_2x^2+c_3x^3+c_4x^4) \equiv 1 \pmod 3$$

Expanding the left side gives:

$$\begin{aligned}
c_0+c_1x+c_2x^2+c_3x^3+c_4x^4 \\
+c_0x^2+c_1x^3+c_2x^4+c_3+c_4x \\
+c_0x^3+c_1x^4+c_2+c_3x+c_4x^2
\end{aligned}$$

or, collecting like terms:

$$(c_0+c_2+c_3)+(c_1+c_3+c_4)x+(c_2+c_4+c_0)x^2+(c_3+c_0+c_1)x^3+(c_4+c_1+c_2)x^4$$

Therefore, for $\mathbf{c}(x)$ to be an inverse, we must solve the following system.

$$\begin{aligned}
c_0+c_2+c_3 &\equiv 1 \pmod 3 \\
c_1+c_3+c_4 &\equiv 0 \pmod 3 \\
c_2+c_4+c_0 &\equiv 0 \pmod 3 \\
c_3+c_0+c_1 &\equiv 0 \pmod 3 \\
c_4+c_1+c_2 &\equiv 0 \pmod 3
\end{aligned}$$

We can achieve this using a sequence of substitutions, as follows.

$$
\begin{aligned}
\text{second congruence: } c_1 &\equiv -c_3 - c_4 \pmod 3 \\
\text{third congruence: } c_2 &\equiv -c_4 - c_0 \pmod 3 \\
\text{fourth congruence: } c_3 + c_0 + (-c_3 - c_4) &\equiv 0 \pmod 3 \\
\Rightarrow c_0 &\equiv c_4 \pmod 3 \\
\text{fifth congruence: } c_4 + (-c_3 - c_4) + (-c_4 - c_0) &\equiv 0 \pmod 3 \\
\Rightarrow c_4 + (-c_3 - c_4) + (-c_4 - c_4) &\equiv 0 \pmod 3 \\
\Rightarrow -2c_4 - c_3 &\equiv 0 \pmod 3 \\
\Rightarrow c_3 &\equiv -2c_4 \pmod 3 \\
\text{first congruence: } c_0 + c_2 + c_3 &\equiv 1 \pmod 3 \\
\Rightarrow c_4 + (-c_4 - c_0) + (-2c_4) &\equiv 1 \pmod 3 \\
\Rightarrow c_4 - 2c_4 - 2c_4 &\equiv 1 \pmod 3 \\
\Rightarrow -3c_4 &\equiv 1 \pmod 3 \\
\Rightarrow 0 &\equiv 1 \pmod 3
\end{aligned}
$$

Since this analysis led to a contradiction, we can conclude that $\mathbf{a}(x)$ does not have an inverse modulo 3.

Alternatively, there is a very short argument for why $\mathbf{a}(x)$ doesn't have an inverse: $\mathbf{a}(1) \equiv 0 \pmod 3$; as we discussed in class, this prohibits $\mathbf{a}(x)$ from having an inverse (see exercise 7.24 in the textbook).

Aside. It is worth noting that we only got into trouble in the very end because 3 is not invertible modulo 3. If we were looking for a solution with *rational* coefficients, or a solution modulo any other prime, we would succeed, because we could invert 3 and then back-substitute to obtain all of the other coefficients. Indeed, the solution in rational numbers can be obtained by solving $c_4 = -1/3$ and back-substituting to obtain $-\frac{1}{3} - \frac{1}{3}x + \frac{2}{3}x^2 + \frac{2}{3}x^3 - \frac{1}{3}x^4$; the inverse modulo any other prime would be obtained by replacing $\frac{1}{3}$ by the inverse of 3 in that modulus.

Now we turn to $\mathbf{b}(x)$. We proceed as before. Let $\mathbf{d}(x)$ be a purported inverse. Then:

$$(1 + x^2 - x^3) \star (d_0 + d_1x + d_2x^2 + d_3x^3 + d_4x^4) \equiv 1 \pmod 3$$

Expanding and collecting like terms as before gives the following system.

$$
\begin{aligned}
d_0 + d_3 - d_2 &\equiv 1 \pmod 3 \\
d_1 + d_4 - d_3 &\equiv 0 \pmod 3 \\
d_2 + d_0 - d_4 &\equiv 0 \pmod 3 \\
d_3 + d_1 - d_0 &\equiv 0 \pmod 3 \\
d_4 + d_2 - d_1 &\equiv 0 \pmod 3
\end{aligned}
$$

which we can solve by successively substituting as follows.

$$\begin{aligned}
\text{second congruence: } d_1 &\equiv d_3 - d_4 \pmod 3\\
\text{third congruence: } d_2 &\equiv d_4 - d_0 \pmod 3\\
\text{fourth congruence: } d_3 + (d_3 - d_4) - d_0 &\equiv 0 \pmod 3\\
\Rightarrow d_0 &\equiv 2d_3 - d_4 \pmod 3\\
\text{fifth congruence: } d_4 + (d_4 - d_0) - (d_3 - d_4) &\equiv 0 \pmod 3\\
\Rightarrow d_4 + d_4 - 2d_3 + d_4 - d_3 + d_4 &\equiv 0 \pmod 3\\
\Rightarrow 4d_4 &\equiv 3d_3 \pmod 3\\
\Rightarrow d_4 &\equiv 0 \pmod 3\\
\text{first congruence: } d_0 + d_3 - d_2 &\equiv 1 \pmod 3\\
\Rightarrow (2d_3 - d_4) + d_3 - (d_4 - d_0) &\equiv 1 \pmod 3\\
2d_3 - 0 + d_3 - 0 + (2d_3 - d_4) &\equiv 1 \pmod 3\\
5d_3 &\equiv 1 \pmod 3\\
d_3 &\equiv -1 \pmod 3\\
\text{now back-substitute for the rest: } d_0 &\equiv 2(-1) - 0 \pmod 3\\
&\equiv 1 \pmod 3\\
d_2 &\equiv 0 - 1 \pmod 3\\
&\equiv -1 \pmod 3\\
d_1 &\equiv -1 - 0 \pmod 3\\
&\equiv -1 \pmod 3
\end{aligned}$$

Hence the inverse of $(1 + x^2 - x^3)$ is $1 - x - x^2 - x^3$ modulo 3.

7. Textbook exercise 7.35 (the second part should be labeled part (b)).

   Solution.

   (a) We know that modulo q, $\mathbf{f}(x) \star \mathbf{h}(x) \equiv \mathbf{g}(x)$, so $\mathbf{a}(x) \equiv \mathbf{f}(x) \star \mathbf{e}(x) \equiv \mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x) \pmod q$. As long as q is sufficiently large, the truncated polynomial $\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)$ will have all coefficients between $-q/2$ and $q/2$, i.e. it will already be centerlifted modulo q, from which it will follow that $\mathbf{a}(x) = \mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)$ exactly.

   Now the fact that $\mathbf{g}(x) \equiv 0 \pmod p$ and $\mathbf{f}(x) \equiv 1 \pmod p$ (by construction) will guaranteed that $\mathbf{a}(x) \equiv \mathbf{m}(x) \pmod p$. Since $\mathbf{m}(x)$ is chosen to already be center lifted, it follows the the centerlift of $\mathbf{a}(x)$ modulo p will be $\mathbf{m}(x)$ exactly.

   (b) As in class, let $|\mathbf{a}(x)|_\infty$ denote the largest absolute value of a coefficient of $\mathbf{a}(x)$. We must show that $\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)$ is its own center lift modulo q, i.e. that $|\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)|_\infty < \frac{q}{2}$. Consider the maximum coefficients of each summand individually. The first summand is equal to $p\mathbf{g}_0(x) \star \mathbf{r}$, and by the same reasoning as in the usual NTru, $|\mathbf{g}_0(x) \star \mathbf{r}(x)|_\infty \le 2d$ (since this is a product of two elements of $\mathcal{T}(d,d)$, so $|\mathbf{g}(x) \star \mathbf{r}(x)|_\infty \le 2pd$.

   Now consider the second summary, $\mathbf{f}(x) \star \mathbf{m}(x)$. We can write this as $\mathbf{m}(x) + p\mathbf{f}_0(x) \star \mathbf{m}(x)$. Since $\mathbf{f}_0(x) \in \mathcal{T}(d,d)$, every term in $\mathbf{f}_0(x) \star \mathbf{m}$ is equal to a sum of d terms of $\mathbf{m}(x)$ minus a sum of d other terms of $\mathbf{m}(x)$. We are assuming that $\mathbf{m}(x)$ is ternary, so each such term is ± 1 or 0, and in the most extreme case a sum of $2d$ such numbers could have absolute value $2d$. Thus the largest possible absolute value of a term of $p\mathbf{f}_0(x) \star \mathbf{m}(x)$ is $2dp$. Adding $\mathbf{m}(x)$ to this can increase the absolute value of this largest coefficient by at most 1 (since $\mathbf{m}(x)$ is ternary). So we conclude that $|\mathbf{f}(x) \star \mathbf{m}(x)|_\infty \le 2dp + 1$.

Adding these together, the largest coefficient (in absolute value) of $\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)$ must have absolute value at most $2dp + 2dp + 1 = 4dp + 2$. Therefore, as long as $4dp + 2 < \frac{q}{2}$, the polynomial $\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x)$ is already centerlifted modulo q and decryption is guaranteed to succeed. This inequality is equivalent to $8dp + 2 < q$, giving the desired result.

8. A network of users set up a basic cryptocurrency as follows. They begin by agreeing on a digital signature scheme and a hash function. There are always exactly 1000 "coins" in existence. Each coin is said to "belong" to a *different* public key (in the beginning, some initial 1000 public keys are agreed upon as the initial owners). There is a public "transaction list," to which anyone can post anonymously at any time. The list records the order that posts are received, and it is impossible to erase a post. The users observe the following rule: a post is considered a "valid transaction" if and only if all of the following hold.

   - It has the format "k_A^{pub} k_B^{pub} S", where k_A^{pub} and k_B^{pub} are public keys, and S is a digital signature.
   - The signature is a valid signature, when k_A^{pub} is used as the verification key, for a document equal to the hash value of "k_A^{pub} k_B^{pub}".
   - k_A^{pub} is currently an owner, and k_B^{pub} is currently not an owner.

   When a valid transaction is posted, ownership of the coin transfers from k_A^{pub} to k_B^{pub}.

   Alice controls all of the coins for which she knows the private key for the "owner" public key. If Alice wishes to pay Bob one coin, Bob should first *create a new key-pair,* safely store the private key k_B^{priv}, and tell Alice the public key k_B^{pub}. Alice then computes a signature S as in the second bullet point (taking as k_A^{pub} *any* of the public keys for coins she currently controls), and sends it to Bob. The signature S is called Alice's "payment;" knowledge of S enables Bob to post a transaction and assume control of the coin in question.

   (a) Suppose that Bob accidentally tells Alice the wrong value of k_B^{pub} (telling her a random number instead), and proceeds to receive "payment" and post it to the transaction list. Explain why the coin is now essentially lost from circulation forever.

   (b) Suppose that Bob wishes to pay Carol a coin, but he will not own any coins until Alice pays him a coin next week. Describe a procedure Bob can use to send Carol "payment" now, such that the payment will become valid (and allow Carol to assume ownership of a coin) immediately after Alice pays Bob next week.

   (c) Suppose Alice owns a coin using a public key k_A^{pub} (i.e. this public key is an "owner," and Alice knows the corresponding private key). At some time, Alice transfers this coin to Bob. Later, Carol wishes to pay Alice a coin. To save effort, Alice doesn't bother creating a new public key, and sends Carol the same key k_A^{pub} that she used before. Explain how Bob can now steal this coin from Alice.

   (d) Suppose that a charity creates and publicizes a public key k_C^{pub} (storing the private key in a safe place), and both Alice and Bob submit "payment" (i.e. valid signatures to transfer coins to k_C^{pub}). Describe a procedure the charity can use to assume ownership of both Alice's coin and Bob's coin, despite the fact that each public key can only own one coin at a time.

   Remark. This system is a cartoon of some of the basic aspects of Bitcoin, and has several serious flaws as it is described. I have also left out the most innovative aspect of the Bitcoin design, namely the blockchain and the proof-of-work system. The original Bitcoin paper (under the pseudonym Satoshi Nakamoto) is mostly nontechnical and explains these features. You might also be interested to read about B-Money (`http://www.weidai.com/bmoney.txt`), a precursor to Bitcoin that is somewhat simpler.

   Solution.

(a) Suppose that R is the random "public key" that Bob sends to Alice. Then once Bob posts "$k_A^{\text{pub}}\ R\ S$" to the transaction list, the community will regard the coin as belonging to the public key R. However, no one in the world knows a private signing key corresponding to R. The private key is necessary in order for this coin to ever change hands again, so it is now impossible for it to ever change owners.

(b) Bob can choose a public key k_B^{pub} (recording the private key) and send Carol this key and a signature S for "$k_B^{\text{pub}} k_C^{\text{pub}}$" (where k_C^{pub} is a public key for which Carol knows the private key). Now, Carol can watch the transaction list and wait until she sees a transaction transferring a coin to k_B^{pub}, which will happen once Alice pays Bob. Once she sees this transaction, she knows that k_B^{pub} owns a coin; *at that time* she posts the transaction "$k_B^{\text{pub}} k_C^{\text{pub}} S$," immediately transferring the coin from Bob to Carol. Notice that Bob does not need to be involved at all once he has sent the "payment" to Carol and informed Alice which public key he would like his payment sent to.

Alternatively, Bob can promise Carol that he will send the public key k_C^{pub} to Alice as his preferred recipient for the upcoming payment. Then Bob can post the transaction himself when Alice pays him, and the coin will go immediately from Alice to Carol.

Remark. Both options, of course, require good faith on Bob's part in one way or another: that Alice really will pay him, and that Alice will pay the coin to the public key that Bob claims she will. It is conceivable that an additional layer could be added to the protocol to accommodate and enforce contracts (such as deferred payment), decreasing the need for this good faith, but there would be many practical difficulties (especially if anonymity is to be preserved to a significant extent).

(c) Bob already knows a valid signature S (with the private key corresponding to k_A^{pub}) for "$k_A^{\text{pub}} k_B^{\text{pub}}$." He can post the *exact same transaction* "$k_A^{\text{pub}} k_B^{\text{pub}} S$" to the list as many times as he wishes. In particular, *any time k_A^{pub} controls a coin*, Bob can post the same transaction again and transfer it to himself. In particular, if Alice requests payment at this same public key ever again, Bob can immediately take the coin at the moment it is transferred to Alice.

In other words: once Alice pays Bob a coin from k_A^{pub}, then for all practical purposes any coins given to k_A^{pub} belong to Bob. For this reason, it is prudent for Alice to completely delete a public key (and its private key) immediately after she pays someone from it.

(d) Carol can create two new public keys (saving the corresponding private keys in a safe place); call these $k_{C'}^{\text{pub}}$ and $k_{C''}^{\text{pub}}$. Then she can simply "pay herself" the coins payed to k_C^{pub} one at time: she can post transactions in the following order.

$$\begin{gathered} k_A^{\text{pub}}\ k_C^{\text{pub}}\ S_A \\ k_C^{\text{pub}} k_{C'}^{\text{pub}} S_{C,1} \\ k_B^{\text{pub}}\ k_C^{\text{pub}}\ S_B \\ k_C^{\text{pub}} k_{C''}^{\text{pub}} S_{C,2} \end{gathered}$$

Here, $S_{C,1}$ and $S_{C,2}$ are signatures that the charity creates with the private key corresponding to k_C^{pub}. The last transition is not actually necessary, since Carol has already assumed control of both coins, but it may be a smart thing to do to "free up" k_C^{pub} to receive subsequent payments.

Remark. Parts (c) and (d) suggest that a smart way to handle transactions efficiently in this system would be the following: create one public key k_A^{pub} that you publish widely, and always use it to receive payment, but *never pay directly from it.* Instead, every time you receive a payment, you can immediately create a new key-pair and pay the coin to yourself at your newly created public key. This way, those paying you only ever have to receive one public key from you, which eliminates the need for a two-way communication whenever you are paid, but you can simultaneously avoid the problem described in (c).

Programming problems

9. Write a program to encrypt an NTru message using a given random element $\mathbf{r}$, as well as a public key and parameters.

Solution.

We need three helper functions: one to add truncated polynomials, one to convolve them, and one to center lift them. After these functions are written, the enciphering function just puts them together.

One solution is below. Note that I represent truncated polynomials as lists in this implementation, where `a[i]` denotes the x^i term of the polynomial.

```
def add_tru(a,b):
        N = len(a)
        return [a[i]+b[i] for i in range(N)]

def convolve(a,b):
        N = len(a)
        c = [0]*N
        for i in range(N):
                for j in range(N):
                        c[(i+j)%N] += a[i]*b[j]
        return c

def centerlift(a,m):
        res = a[:]
        for i in range(len(a)):
                res[i] %= m
                if res[i] > m/2: res[i] -= m
        return res

def encipher(N,p,q,d,h,m,r):
        hr = convolve(h,r)
        phr = [p*a for a in hr]
        return centerlift(add_tru(phr,m),q)

# I/O
N,p,q,d = map(int,raw_input().split())
h = map(int,raw_input().split())
m = map(int,raw_input().split())
r = map(int,raw_input().split())

e = encipher(N,p,q,d,h,m,r)
print ' '.join(map(str,e))
```

10. Write a program to decrypt an NTru ciphertext $\mathbf{e}$, given both the public and private key. The input will include the inverses $\mathbf{F}_p$ and $\mathbf{F}_q$ as part of the private key, so you do not need to write an algorithm to compute these inverses.

Solution. We can use the same helper functions written in the previous problem; the deciphering process that puts them together as prescribed in the textbook.

```
### Omitted: code for add_tru, convolve, and centerlift (see previous problem).

def decipher(N,p,q,f,Fp,e):
        a = centerlift(convolve(f,e),q)
        return centerlift(convolve(a,Fp),p)

# I/O
N,p,q,d = map(int,raw_input().split())
h = map(int,raw_input().split())
f = map(int,raw_input().split())
g = map(int,raw_input().split())
Fp = map(int,raw_input().split())
Fq = map(int,raw_input().split())
e = map(int,raw_input().split())

m = decipher(N,p,q,f,Fp,e)
print ' '.join(map(str,m))
```

11. Write a program that is given the list of initial “owners” in the currency from problem 8 and a sequence of posts of the form “$k_A^{\text{pub}} k_B^{\text{pub}} S$” to the transaction list, either accepts or rejects each transaction, and determines the current list of owners after all transactions are completed. The signature system will be ECDSA with Bitcoin’s curve (Secp256k1), and the hash function will be SHA-256 (details and some sample code in the online statement).

    Solution.

    We can use the same source code for ECDSA as last week, although we will need to enter the new `Seckp256k1` parameters. Then we must insert the hash computation function that was provided by the 0-point “hashing for payment” problem. From here, it is just a matter of checking, for each transaction, whether the tree given conditions are satisfied. If so, we update the list `owners`. Finally, we print the whole `owners` list at the end. Here is an implementation.

```
### Omitted: source code from last week's ECDSA problem. In particular, the classes
### 'curve' and 'ec_pt', and the function 'valid.'

# Cives the system-wide parameters for elliptic curve secp256k1.
# Copied from p. 489 of the textbook.
def secp256k1():
        p = 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6 - 2**4 - 1
        a = 0
        b = 7
        Gx = 55066263022277343669578718895168534326250603453777594175500187360389
116729240
        Gy = 32670510020758816978083085130507043184471273380659243275938904335757
337482424
        C = curve(a,b,p)
        G = ec_pt(C,(Gx,Gy))
        q = p + 1 - 432420386565659656852420866390673177327
        return C,p,q,G
```

```
# Function to create the hash value (provided online)
import hashlib
def hashval(A,B,C,D):
   m = hashlib.sha256()
   m.update('%d %d %d %d'%(A,B,C,D))
   return int(m.hexdigest(),16)


# Read the input
N,T = map(int,raw_input().split())
owners = []
for n in range(N):
        x,y = map(int,raw_input().split())
        owners += [(x,y)]
transactions = []
for t in range(T):
        x1,y1,x2,y2,s1,s2 = map(int,raw_input().split())
        transactions += [(x1,y1,x2,y2,s1,s2)]

# Store the system parameters
C,p,q,G = secp256k1()

# Process each transaction, update owners as necessary
for x1,y1,x2,y2,s1,s2 in transactions:
        hv = hashval(x1,y1,x2,y2)
        #Three conditions for validity:
        val1 = valid(C,p,q,G,ec_pt(C,(x1,y1)),(s1,s2),hv) #Valid signature
        val2 = (x1,y1) in owners
        val3 = (x2,y2) not in owners
        if val1 and val2 and val3:
                # Change the owner
                i = owners.index( (x1,y1) )
                owners[i] = (x2,y2)

#Print the final list of owners
for i in xrange(N):
        print owners[i][0],owners[i][1]
```