6 Powers and Roots in $\mathbb{Z}_n$

6.1 Successive Squaring and k^{th} Roots

In this chapter, we flesh out two contrasting ideas: Powers are *easy*, roots (and factorization) are *hard*.

Example 6.1. To compute $14^{217} \pmod{67}$, we currently have a couple of options:

1. Reduce the problem via Euler/Fermat $14^{217} \equiv 14^{3\cdot 66+19} \equiv 14^{19} \pmod{67}$.
2. Hunt for a small power of 14 which has small remainder modulo 67. This could take a while!

For large moduli, these options become markedly less attractive! Instead we try a more systematic approach where we repeatedly compute squares:

$$\begin{aligned} 14^2 \equiv 196 \equiv -5 &\implies 14^4 \equiv (-5)^2 \equiv 25 \\ &\implies 14^8 \equiv 25^2 \equiv 625 \equiv 22 \\ &\implies 14^{16} \equiv 22^2 \equiv 484 \equiv 15 \pmod{67} \end{aligned}$$

Each squaring was easy, and by considering the binary decomposition $19 = 16+2+1 = 2^4+2^1+2^0$ of the exponent, we now have enough information to compute the answer:

$$14^{19} \equiv 14^{16+2+1} \equiv 15 \cdot (-5) \cdot 14 \equiv 22 \pmod{67}$$

Successive Squaring Algorithm to compute $a^k \pmod{m}$

1. Take the binary decomposition $k = 2^r + 2^{r-1}\mu_{r-1} + \cdots + 2\mu_1 + \mu_0$ where each $\mu_j = 0, 1$.
2. Repeatedly square modulo m: compute $A_j \equiv a^{2^j} \equiv A_{j-1}^2 \pmod{m}$
3. Compute $a^k \equiv A_r A_{r-1}^{\mu_{r-1}} \cdots A_0^{\mu_0} \cdots \pmod{m}$.

Example 6.2. We compute $6^{73} \pmod{25}$ using the successive squaring algorithm:

1. $73 = 64+8+1 = 2^6+2^3+2^0$.
2. Starting with $A_0 = a = 6$ we square:
$$\begin{aligned} A_1 \equiv 6^2 \equiv 35 \equiv 11 &\implies A_2 \equiv 11^2 \equiv 121 \equiv -4 \\ &\implies A_3 \equiv (-4)^2 \equiv 16 \equiv -9 \\ &\implies A_4 \equiv 16^2 \equiv 256 \equiv 6 \equiv A_0 \\ &\implies A_5 \equiv A_1 \equiv 11, \qquad A_6 \equiv A_1 \equiv -4 \end{aligned}$$
Notice how the pattern repeats once we reach $A_4 \equiv A_0$.
3. $6^{73} \equiv A_6 A_3 A_0 \equiv (-4)\cdot(-9)\cdot 6 \equiv 16 \pmod{25}$.

For more speed, we could have started with Euler's Theorem: $\varphi(25) = 5 \cdot 4 = 20$, whence

$$6^{73} \equiv (6^{20})^3 \cdot 6^{13} \equiv 6^{2^3+2^2+2^0} \equiv A_3 A_2 A_0 \equiv (-9)\cdot(-4)\cdot 6 \equiv 16 \pmod{25}$$

However, considering how the original list started repeating, this didn't save us much time.

Efficiency While tedious to perform by hand, the algorithm is very efficient for a computer: this is what we mean by *powers are easy.*

- The binary expansion of $k = \mu_0 + 2\mu_1 + 2^2\mu_2 + \cdots + 2^r$ has $r+1$ terms if and only if

  $$2^r \le k < 2^{r+1} \iff r \le \log_2 k < r+1$$

  This is likely the form in which the computer stores k already!
- Squaring and computing each A_j and the product $A_0^{\mu_0} \cdots A_r$ requires $r+1$ *take the remainder* calculations.
- The algorithm therefore requires approximately $\log_2 k$ remainder steps to complete; roughly 3.3 times number of digits of k.
- There are many algorithms available for taking the remainder: these are roughly about as efficient as multiplying, so the full algorithm is very efficient indeed!

Slightly faster algorithms even than this are available; even when x, k, m are 100's of digits long, a modern computer can evaluate $x^k \pmod b$ in *micro*seconds. To really stress a computer, we need much larger exponents! Here are a few benchmarking times[1] where $x = 13^{89}+1$ and $m = 17^{81}+3$ are 100-digit numbers.

27 μs:	$x^{10^{100}} \equiv$ 2811368376719703263528063091846551559031253759668873958264247724126725739585183812656683304446721416
17 ms:	$x^{10^{100,000}} \equiv$ 4488975456548368803859052207045919909802116591225720977576091772560693617724591244737588457285087356
174 ms:	$x^{10^{1,000,000}} \equiv$ 892615982890619665980610534874493547494556817553720182073417194047142550414087154521448828227415676
1.78 s:	$x^{10^{10,000,000}} \equiv$ 2225414932073741734978750203003783867698573600388509903995387020623239040547081286262846393211045316
17.7 s:	$x^{10^{100,000,000}} \equiv$ 3349869250081483676357258995295278747886045025380645413804988720714016105105145445830489542782366876

Note how the computing time is roughly proportional to the number of digits in the exponent: each calculation ($100000 \to 1$ million $\to 10$ million $\to 100$ million digits) takes approximately 10 times as long as the previous.

Computing kth roots modulo m

The contrasting problem of finding kth roots is much *harder,* in that computers cannot do it efficiently. Again we motivate via an example.

Example 6.3. Solve the congruence $x^5 \equiv 7 \pmod{26}$: that is, find the 5th roots of 7 modulo 26.

- First note that $\gcd(7, 26) = 1$ and that any solution x must therefore be a unit:

  $$d \mid x \text{ and } d \mid 26 \implies d \mid 7 \implies d = 1 \implies \gcd(x, 26) = 1$$
- By Euler's Theorem: $x^{\varphi(26)} \equiv x^{12} \equiv 1 \pmod{26}$.
- Now hunt for a multiple of 5 which is congruent to 1 modulo $\varphi(m) = 12$. In this case

  $$5^2 = 25 = 2\varphi(26) + 1 \implies x \equiv x^{1+2\varphi(26)} \equiv x^{25} \equiv 7^5 \equiv 11 \pmod{26}$$

In the last step we may appeal to successive squaring to compute 7^5 or simply hack at it…

[1]These times were obtained running Sage on a single core of an Intel i5-9600K desktop CPU, clocked at 4.4 GHz.

We lucked out in the example: the final step relied on being able to solve the congruence

$$5u \equiv 1 \pmod{12}$$

which we know we can do because $\gcd(5,12) = 1$. When trying to take k^{th} roots in general, this step may not be possible. At least we have identified the critical ingredient necessary for being able to find a *unique* k^{th} root.

Theorem 6.4. *Suppose that $\gcd(b,m) = \gcd\big(k,\varphi(m)\big) = 1$. Then the congruence equation $x^k \equiv b \pmod{m}$ has a unique solution, which can be found as follows:*

1. *Compute $\varphi(m)$.*
2. *Find $u \in \mathbb{N}$ such that $ku \equiv 1 \pmod{\varphi(m)}$.*
3. *Evaluate $x \equiv b^u \pmod{m}$.*

Proof. First observe that any purported solution x must be a unit ($\gcd(x,m) = 1$): if not, then $b \equiv x^k$ and m would have a common divisor greater than 1, contradicting our assumptions.
Step 2 is possible since $\gcd\big(k,\varphi(m)\big) = 1$; we can therefore write $ku = 1 + \lambda\varphi(m)$ for some $\lambda \in \mathbb{N}$. Since any suitable x is a unit, we can now apply Euler's Theorem

$$b^u \equiv (x^k)^u \equiv x^{1+\lambda\varphi(m)} \equiv x \pmod{m}$$

Uniqueness is clear since we found $x \equiv b^u$ by doing the same thing (raising to the power u) to both sides of the congruence $x^k \equiv b \pmod{m}$. ∎

Example 6.5. Find the unique solution to $x^{283} \equiv 29 \pmod{42}$
We trivially verify that $\gcd(29,42) = 1$ and $\varphi(42) = 12$. We therefore need to solve

$$283u \equiv 1 \pmod{12}$$

This is straightforward, since $283 \equiv 7 \pmod{12}$, we easily spot that $u \equiv 7$ is a solution.[a] Now compute

$$\begin{aligned} x^{283} \equiv 29 &\implies x^{283\cdot 7} \equiv 29^7 \pmod{42} \\ &\implies x \equiv x^{1+165\varphi(42)} \equiv 29^7 \pmod{42} \end{aligned}$$

It remains to compute the final power: applying the successive squaring algorithm, we have $7 = 2^0 + 2^1 + 2^2$, and

$$A_0 = 29, \qquad A_1 = 29^2 = 169 = 1, \qquad A_2 = 1$$

whence

$$x \equiv 29^7 \equiv 29 \cdot 1 \cdot 1 \equiv 29 \pmod{42}$$

[a]If this makes you nervous, use the Euclidean algorithm to solve $7u = 1 + 12\lambda$, or indeed $283 = 1 + 12\lambda$:

$$\left.\begin{aligned} \mathbf{283} &= \mathbf{12}\cdot 23 + \mathbf{7} \\ \mathbf{12} &= \mathbf{7}\cdot 1 + \mathbf{5} \\ \mathbf{7} &= \mathbf{5}\cdot 1 + \mathbf{2} \\ \mathbf{5} &= \mathbf{2}\cdot 2 + \mathbf{1} \end{aligned}\right\} \implies \gcd(283,12) = \mathbf{1} = \mathbf{12}\cdot 118 - \mathbf{283}\cdot 5 \implies \mathbf{283}\cdot 7 = \mathbf{1} + \mathbf{12}\cdot 165$$

where we reversed the algorithm to obtain the final result.

Efficiency Even when a unique k^{th} root exists, finding it is typically much slower than computing a k^{th} power. Comparing the steps in Theorem 6.4:

1. Computing $\varphi(m)$ is *very, very* slow; you essentially need to factorize m.
2. The Euclidean algorithm is fast to implement.
3. This can be done using successive squaring; also fast.

When m is large the discrepancy in computing speeds becomes *enormous.*

The same modern desktop considered earlier took 216 seconds to factorize the 100-digit base discussed previously:

$$17^{81}+3 = 2^2 \times 5 \times 107 \times 20381297 \times 5040257978377 \times 10487165161371821332685555297973 7 \times 201189896476403174943819047900047481422801171$$

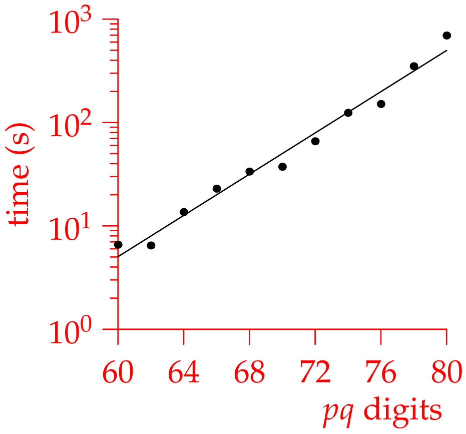
This example isn't ideal since if a large number is lucky enough to be divisible by several small primes, it can often be factorized very quickly. For a more sensible benchmark, here are the times taken by the computer to factorize several *semiprimes* pq, where the primes p, q were of comparable size.

pq digits	60	62	64	66	68	70	72	74	76	78	80
Factorization time (s)	6.58	6.46	13.6	22.9	33.5	37.3	65.8	124	151	350	694

When the factorization time is graphed logarithmically, the data appears linear. The best-fitting straight line therefore represents an *exponential* model:

$$T(n) \approx \exp(0.2297n - 12.1665)$$

By this metric, we might expect that factoring a 100-digit semiprime would require $13\frac{1}{2}$ *hours*, a 150-digit semiprime 150 *years*!



Non-unique k^{th} roots It is reasonable to ask what can happen in when either or both of the conditions $\gcd(b,m) = 1 = \gcd\left(k, \varphi(m)\right) = 1$ fails. The short answer is that anything is possible; you could have no k^{th} root, a unique root, or several roots. Some of the details are in the exercises.

Example 6.6. Modulo 6, we have $\gcd(\varphi(6), 4) = 2 \neq 1$. By computing fourth powers modulo 6:

x	0	1	2	3	4	5
x^4	0	1	4	3	4	1

we see that the congruence $x^4 \equiv b \pmod 6$ has a unique solution if $b = 0, 3$, two solutions if $b = 1, 4$, and no solutions if $b = 2, 5$.

Exercises 6.1 1. Use the method of successive squaring to compute each of the following:

(a) $5^{13} \pmod{23}$ (b) $12^{260} \pmod{1000}$ (c) $28^{749} \pmod{1147}$

(*Use a calculator*!)

2. For each congruence, verify the hypotheses of the Theorem 6.4 and solve the congruences:

(a) $x^{83} \equiv 15 \pmod{322}$ (b) $x^{329} \equiv 452 \pmod{1147}$

3. We search for k^{th} roots of b modulo m in situations where at least one of the standard conditions fails:

$$\gcd(b, m) \neq 1 \quad \text{or} \quad \gcd\big(k, \varphi(m)\big) \neq 1 \tag{$*$}$$

(a) If p is an odd prime, show that 1 has exactly two square-roots modulo p. Which of the conditions $(*)$ fails in this case?

(b) Investigate the cube-roots of b modulo 8 for each remainder b (see, e.g., Example 6.6). How many such b have a cube-root? Are they unique? What do the gcd conditions $(*)$ say in each case? When could we have used the theorem on unique k^{th} roots?

(c) Repeat for the fourth-roots of b modulo 8.

(d) Repeat for the cube-roots of b modulo 10.

6.2 The RSA Cryptosystem

Perhaps the modern-world's most utilised cryptosystem, it is likely that you (indirectly) use some version of RSA[2] every day, when your phone or computer connects securely to another, for instance using https. Here is how the method works.

Encoding

1. Start with distinct primes p, q and build the semiprime $m = pq$.
2. Calculate $\varphi(m) = (p-1)(q-1)$.
3. Choose an integer s such that $1 < s < \varphi(m)$ and $\gcd\big(s, \varphi(m)\big) = 1$.
4. Encode a numerical message by mapping $x \mapsto x^s \pmod{m}$.

Decoding This is based on the following.

Theorem 6.7. *Let $u \in \mathbb{N}$ satisfy $us \equiv 1 \pmod{\varphi(m)}$. Then, for all x, $(x^s)^u \equiv x \pmod{m}$.*

Proof. Since $m = pq$ is a semiprime, we have

$$x^{su} \equiv x \pmod{m} \iff \begin{cases} x^{su} \equiv x \pmod{p}, \text{ and} \\ x^{su} \equiv x \pmod{q} \end{cases}$$

Since $su \equiv 1 \pmod{\varphi(m)}$ we see that $su = 1 + j(p-1)(q-1)$ for some $j \in \mathbb{Z}$. If $x \not\equiv 0 \pmod{p}$, then Fermat's little theorem tells us that

$$x^{su} \equiv x \cdot (x^{p-1})^{j(q-1)} \equiv x \pmod{p}$$

The result is plainly trivial if $x \equiv 0$, and the calculation is similar for the other modulus q. ∎

The process is very simple: think of s for 'scramble' and u for 'unscramble.'

$$x \xmapsto{\text{encode}} x^s \pmod{m} \xmapsto{\text{decode}} (x^s)^u \equiv x \pmod{m}$$

As we saw in the previous section, even if m, s, u are 100+ digits long, these calculations are very fast for modern computers.

The values m, s are known as the *public key*: these are all you need to encode messages. Indeed these can be made freely available so that anyone can encode.

To decode messages, one also requires the *private key* u. Provided you keep this number secret, only you can decode messages sent to you.

One implementation involves a group of friends each of whom have different keys. They keep secret their private keys u, but share their public keys s, m with the group. Then all friends can send messages to each other but, once encoded, each can only be decoded by the intended recipient.

[2]The acronym is formed from the initials of Rivest, Shamir and Adleman who discovered the system while working at MIT in 1977. It was in fact first described in 1973 by Clifford Cocks while working for GCHQ, the British equivalent of the US National Security Agency. Cocks' discovery was classified, even though, due to the lack of available computing power, it was deemed to have no practical application.

Examples 6.8. 1. For a very simple example, we start by encoding a message via the obvious substitution $A \mapsto 1$, $B \mapsto 2$, etc.:

I	T	S	A	L	L	G	R	E	E	K	T	O	M	E
9	20	19	1	12	12	7	18	5	5	11	20	15	13	5

If we choose the semiprime $m = 5 \times 7 = 35$, then $\varphi(m) = 4 \times 6 = 24$, and we can choose, say, $s = 5$. We then encode by mapping $x \mapsto x^5 \pmod{35}$:

$$9 \mapsto 9^5 \equiv 81^2 \cdot 9 \equiv 11^2 \cdot 9 \equiv 16 \cdot 9 \equiv 4 \pmod{35}$$
$$20 \mapsto 20^5 \equiv 20 \qquad 19 \mapsto 19^5 \equiv 24, \qquad 1 \mapsto 1^5 \equiv 1, \ldots$$

resulting in the string of numbers

4, 20, 24, 1, 17, 17, 7, 23, 10, 10, 16, 20, 15, 13, 10

We could also translate the encoded message back into letters:

D, T, X, A, Q, Q, G, W, J, J, P, T, O, M, J

To decode, we require u such that $5u \equiv 1 \pmod{24}$; that is $u = 5$ (it doesn't matter for us that this equals s!). Again compute

$$4^5 \equiv 4^3 \cdot 4^2 \equiv 64 \cdot 4^2 \equiv -6 \cdot 4^2 \equiv -24 \cdot 4 \equiv 44 \equiv 9 \pmod{35}$$
$$20^5 \equiv 20, \qquad 24^5 \equiv 19, \ldots$$

to recover the original string of numbers and message ITSALLGREEKTOME.

2. Suppose you intercept the message

59, 4, 57, 2, 82, 4, 86, 43, 4, 43, 57, 4

which you know has been encoded using the public key $s = 11$, $m = 119$. You also know that the message may be read via the translation

$$11 \leftrightarrow A,\ 12 \leftrightarrow B, \ldots, 36 \leftrightarrow Z$$

To crack the code, our first job is computing the totient: $m = 7 \times 17 \implies \varphi(m) = 6 \cdot 16 = 96$.

We now need to find the private key, which satisfies $11u \equiv 1 \pmod{96}$. A relatively short application of the Euclidean algorithm says that

$$1 = 11 \cdot 35 - 4 \cdot 96 \implies u = 35$$

We now compute:[a]

$$59 \mapsto 59^{35} \equiv 19 \mod 119$$

etc. The full decode is

19, 29, 19, 30, 25, 24, 30, 18, 15, 30, 15, 29, 30

which you're welcome to translate into letters if you're so inclined...

[a]If you don't want to beg the help of a calculator, use the successive squaring algorithm: the binary decomposition is $35 = 2^5 + 2^1 + 2^0$, which yields

$$A_0 = 59, \quad A_1 = 30, \quad A_2 = -52, \quad A_3 = -33, \quad A_4 = 18, \quad A_5 = -33$$
$$\implies 59^{35} \equiv A_5 A_1 A_0 \equiv -33 \cdot 30 \cdot 59 \equiv 19 \pmod{119}$$

Don't knock it: it's what your computer has to do for *every* element of the code!

Speed and Security of the RSA system

Encoding and decoding (once in possession of the private key) require only the computation of powers modulo m. While our examples used very small moduli, modern applications use semiprimes with 300 or more digits. While unfeasible in 1973, for modern computers such work is trivial.

Now suppose that you are in possession of the public key m, s and want to crack an encoded message. You need to do two things:

1. Find $\varphi(m)$; equivalently factorize $m = pq$. As we saw in the previous section, for moduli in the 300 digit range this is essentially impossible in any reasonable time-frame.[3]
2. Find $u \in \mathbb{N}$ such that $us \equiv 1 \pmod{\varphi(m)}$. Employing the Euclidean algorithm requires no more than $2\log_2 \varphi(m)$ applications of the division algorithm and some back substitution. Even for 300 digit numbers, this can be completed in microseconds *provided* $\varphi(m)$ *is known.*

While resilient against general attack, RSA is not foolproof. Its main drawback is that it is a *table cipher*: if $18 \mapsto 11$ during encoding, then 18 is always mapped to 11. If a decoded message is intercepted, a hacker then knows how to decode any future messages without calculation. Long messages reduce security since common combinations such as 'e' and 'the' might be guessable if they appear frequently. Correctly guessing even a few letters makes decoding a full message much easier. Of course, with very large moduli perhaps the entire message can be transmitted using only one digit! RSA can also easily be combined with other cryptographic methods for greater security.

Exercises 6.2 1. For each message and public key m, s, find the private key u and decode the message, using $1 \mapsto A,\ 2 \mapsto B$, etc. to translate back to letters.

(a) When $m = 35$ and $s = 11$ you receive $28, 4, 18, 18, 10, 2, 4, 14, 28, 28, 4, 2, 1, 6, 6, 10, 24$

(b) When $m = 143$ and $s = 103$ you receive $63, 1, 63, 63, 12, 113, 27, 123, 63, 1, 63, 141, 141, 27, 72$

2. (a) Let $m = p_1 \cdots p_n$ be a product of *distinct* primes, and assume that $\gcd(k, \varphi(m)) = 1$ so that $\exists u$ with $ku \equiv 1 \pmod{\varphi(m)}$. Prove that $x^k \equiv b \pmod m$ has unique solution $x \equiv b^u \pmod m$, regardless of whether $\gcd(b, m) = 1$.

   (*Hint: Carefully read the proof of Theorem 6.7*)

   (b) Consider the congruence $x^5 \equiv 6 \pmod 9$. Show that you can find u satisfying $5u \equiv 1 \pmod{\varphi(9)}$, but that $x \equiv 6^u$ is not a solution to the required congruence. Can you identify where the *distinct prime* condition was needed in part (a)?

   (c) Solve the congruence $x^{49} \equiv 3 \pmod{1155}$

3. In Example 6.8, we saw that the public and private keys s, u were equal (both being 5). Relative to the semiprime modulus $m = 35$, show that this is *always* the case; regardless of which s you choose, you will always have $u = s$.

4. Modern implementations typically replace Euler's totient function $\varphi(m) = (p-1)(q-1)$ with $\Lambda(m) := \operatorname{lcm}(p-1, q-1)$.

   Given a public key m, s where $\gcd(s, \Lambda(m)) = 1$, show that decoding may be accomplished by finding the private key satisfying $us \equiv 1 \pmod{\Lambda(m)}$.

[3]RSA Labs used to offer cash prizes for factoring large semiprimes (the *RSA-numbers*). As of 2020, the largest yet factorized has 250 digits, requiring supercomputer resources equivalent to over 1000 years on a single desktop core.