Contents

2 Public-Key Cryptography **1**

2.1 Symmetric and Asymmetric Cryptosystems . . . 2

2.2 Powers and Congruences Modulo m . . . 3

2.2.1 Orders of Elements Modulo m . . . 5

2.2.2 Fermat's Little Theorem . . . 6

2.2.3 The Chinese Remainder Theorem . . . 7

2.2.4 The Euler φ-Function and Euler's Theorem . . . 9

2.3 Rabin Encryption . . . 11

2.4 The RSA Encryption System . . . 13

2.4.1 Procedure for RSA . . . 14

2.4.2 Attacks on RSA . . . 15

2.5 Primality and Compositeness Testing . . . 19

2.5.1 The Fermat Compositeness Test . . . 20

2.5.2 The Miller-Rabin Compositeness Test . . . 21

2.5.3 The Lucas Primality Criterion . . . 22

2.5.4 The AKS Primality Test . . . 23

2.6 Factorization Algorithms . . . 24

2.6.1 The Fermat Factorization . . . 24

2.6.2 Pollard's $p-1$ Algorithm . . . 25

2.6.3 Pollard's ρ-Algorithm . . . 27

2.6.4 Sieving Methods . . . 28

2 Public-Key Cryptography

In this chapter, we discuss several modern public-key cryptosystems, which (unlike all of the classical cryptosystems we have previously discussed) are resistant to known-plaintext attacks. We will begin by discussing the general principles of asymmetric cryptography and mention the general ideas behind some modern symmetric cryptosystems. We next develop some necessary results from number theory on modular exponentiation, so that we may treat in sufficient detail a pair of public-key cryptosystems, including the famous RSA encryption system.

Effective implementation of these cryptosystems requires generating large primes that can be quickly proven to be prime, so we will then discuss primality testing and factorization algorithms, since these are important ingredients in implementing public-key cryptosystems.

2.1 Symmetric and Asymmetric Cryptosystems

- All of the classical cryptosystems we have previously discussed are examples of symmetric cryptosystems: the information required to encode a message is the same as the information required to decode a message.
    - For example, the key for encoding a Caesar shift is an integer k giving the number of letters the message is shifted forward. The knowledge of the integer k allows one both to encode and decode a message.
    - Similarly, a copy of the key text for a one-time pad is required to encode or to decode a message.
- We will mention that there are several symmetric cryptosystems that are in current use and considered to be strong: it is avowedly not the case that symmetric cryptosystems are inherently vulnerable to simple attacks the way the historical cryptosystems we discussed are.
- One such symmetric cryptosystem that was adopted as a national standard for unclassified data by the United States in 1977 is known as the Data Encryption Standard (DES).
    - The DES cryptosystem was a 64-bit block cipher (meaning that it operated on blocks of data 64 bits in length), with a key length of 56 bits. 8 bits were devoted to parity checks, in order to detect errors in data transmission.
    - To describe the system in detail would take a great deal of time and effort. Ultimately, our interest is not in the specific details of the system, but rather in the basic idea behind it: the system operates on a data block by applying 16 identical stages of processing called "rounds", each of which scrambles the block according to a particular nonlinear procedure dictated by the key and the algorithm.
    - Much is known about the security of DES, as it was the subject of significant research, but it turns out that there are few attacks that are much faster than a simple brute-force search.
    - Various procedures were developed to quicken a brute-force search, and a direct approach to breaking the cipher takes on the order of 2^{43} calculations. To give an idea of how feasible such an attack is, to store the result of 2^{43} single-bit calculations would require a mere 1 terabyte of data storage, less than a typical desktop computer hard drive.
    - Ultimately, owing to the small key size, DES was phased out in the 1980s, and in 1999 a single DES key was broken in less than 24 hours.
- In the 1980s, a sufficient number of security concerns about DES, as well as some concerns about the slowness of the algorithm itself when implemented in software (it was originally designed for hardware implementation), motivated the deployment of additional block cipher algorithms.
    - Many of these simply reused the basic structure of DES but increased the size of the data blocks, the size of the key, or the number of rounds.
    - Most such algorithms are proprietary, although there are some such as Blowfish that are open-source, and others that are proprietary but available royalty-free such as CAST-128.
- The generally-considered successor to DES is known as the Advanced Encryption System (AES) and also known as Rijndael, from a portmanteau of its creators' names (Joan Daemen and Vincent Rijmen).
    - The National Institute of Standards and Technology (NIST) held an open competition for the successor to DES in the late 1990s, and, following a lengthy evaluation, Rijndael was eventually selected from the 15 submissions as satisfying the constraints of security, efficiency, and portability.
    - Part of the motivation for the open and lengthy evaluation process were some suspicions (of varying legitimacy) about whether previous algorithms like DES had hidden "backdoors" built in.
    - The AES cipher is a 128-bit block cipher with possible key lengths of 128, 192, or 256 bits, and operates in a similar manner to DES, invoking a number of rounds of operations (10, 12, or 14) each of which rearranges and transforms the block according to the key.
    - It is generally believed that AES is resistant to most kinds of direct attacks, and it has been approved by the US government for use on classified information.

- Current estimates place the computational difficulty of breaking a single 128-bit AES key, using the best known attacks, at roughly 2^{96} operations in a worst-case scenario, with an expected number of operations typically closer to 2^{126}, and an estimated memory requirement of about 2^{56} bits (approximately 4 million terabytes).

- One of the fundamental drawbacks of symmetric cryptosystems is that, by definition, being able to encode a message is equivalent to being able to decode a message. But there are certain situations where we might prefer to have an asymmetric cryptosystem, one in which the encoding and decoding procedures are sufficiently different that being able to encode messages does not imply that one can decode them.

  - For example, a fundamental issue with symmetric cryptosystems is that of key exchange: if Alice and Bob want to communicate via a symmetric cryptosystem over a long distance, they must first share the key with one another, but they require a method that will not allow Eve to learn the key. They could do this by using a different cryptosystem, but again: how do they decide what key to use for the second cryptosystem, and how do they tell each other without letting Eve know?
  - With an asymmetric system, this is not a problem: Alice simply tells Bob how to send her an encrypted message, and Bob can send her the key they will use for subsequent communications.
  - As another example, if Alice wishes to digitally sign a document to indicate that it belongs to her, she wants it to be easy for anyone to verify that the signature is actually hers, yet also very difficult to decouple the signature from the document itself (because this would allow anyone to forge her signature on a new document).

- It turns out that, perhaps surprisingly, it is possible to create secure cryptosystems in which one can make the encryption method completely public: such systems are known as public-key cryptosystems.

  - Sending a message via public-key encryption is then very easy: Alice simply asks Bob for his public key (and the encryption procedure), and then follows the procedure.
  - Bob can feel free giving her this information even knowing that Eve might also be listening, because of the asymmetry in the cryptosystem: the fact that Eve knows how to encode a message does not mean that she can decode anything.
  - A good analogy for public-key encryption is a locked dropbox: anyone can place an envelope into the dropbox, but only the owner (or at least, the person who has the key) can retrieve the letters from the box.

- Ultimately, public-key cryptosystems revolve around the existence of so-called one-way functions: functions which are easy to evaluate ("forward") but very difficult to invert ("backward") on most outputs.

  - Many examples of one-way functions come from number theory.
  - As an example, consider the function $f(p, q) = pq$ that takes two prime numbers and outputs their product.
  - It is a trivial matter of arithmetic to compute the product pq given p and q, but if we are given pq and asked to find p and q, we would need to know how to factor an arbitrary integer: this is believed to be much, much harder.
  - Ultimately, the property that factorization is much harder than multiplication is the basis for many public-key cryptosystems, including the famous RSA cryptosystem.
  - In order to discuss public-key encryption, we must first cover the relevant results from number theory.

2.2 Powers and Congruences Modulo m

- We now turn our attention to discussing powers of elements modulo m, which are of central importance in the implementation of most known public-key cryptosystems.

- As an example to motivate the discussion in the rest of this section, suppose we want to find the remainder when we divide 2^{516} by 61.

- One way we could do this is simply by computing the actual integer 2^{516} (which has 156 digits in base 10), and then dividing it by 61. This is certainly feasible with a computer, but would be very unpleasant by hand.
- A faster way would be to compute successive powers of 2 and reduce modulo 61 at each stage: 2, 4, 8, 16, 32, $64 \equiv 3$, 6, 12, 24, 48, $96 \equiv 35$, $70 \equiv 9$, 18, 36, ... This is certainly faster and feasible to do by hand (in the sense of not requiring the computation of a 156-digit integer), but it would still require over 100 multiplications.
- We can speed up the process significantly if we instead only compute the powers 2^1, 2^2, 2^4, 2^8, 2^{16}, ... , 2^{512} and so forth (modulo 61) by successively squaring the previous values and reducing. Then we can find 2^{516} by observing that $2^{516} = 2^{512} \cdot 2^4$.
- Explicitly, we obtain the following:

$$\begin{array}{ccc} 2^2 = 4 & 2^{16} \equiv 12^2 = 144 \equiv 22 & 2^{128} \equiv 16^2 \equiv 12 \\ 2^4 = 16 & 2^{32} \equiv 22^2 = 484 \equiv -4 & 2^{256} \equiv 12^2 \equiv 22 \\ 2^8 = 16^2 \equiv 256 \equiv 12 & 2^{64} \equiv (-4)^2 = 16 & 2^{512} \equiv 22^2 \equiv -4 \end{array}$$

- Therefore we see that $2^{516} = 2^{512} \cdot 2^4 \equiv (-4) \cdot 16 = -64 \equiv \boxed{58}$ modulo 61.

- Observe that, in the computations we performed, the later entries started repeating earlier ones. This will in fact always be the case, as we will see soon.

- For posterity, we record this technique of successive squaring:

- Algorithm (Successive Squaring): To compute a^k modulo m, first find the binary expansion of $k = \underline{b_j b_{j-1} \cdots b_0}$. Then compute the powers a^2, a^4, $\cdots$, a^{2^d} by squaring the previous entry in the sequence and reducing modulo m. Finally, compute $a^k \equiv \prod_{\substack{0 \le i \le j \\ b_i = 1}} a^{2^{b_i}}$ modulo m.
  - Observe that the total number of multiplications and reductions mod m required is roughly $2\log_2(k)$, which is a vast improvement over the k multiplications and reductions required to compute a^k directly.

- It is possible to rearrange the computations in this procedure to require less storage, though the same numbers of multiplications and squarings are required.

- Algorithm (Power Chain Squaring): To compute a^k modulo m, first find the binary expansion of $k = \underline{c_1 c_2 \cdots c_d}$. Begin with the value $r_1 = 1$, and, for each $1 \le i \le d$, define $r_i = r_{i-1}^2 \pmod m$ if $c_i = 0$ and $r_i = (a r_{i-1})^2 \pmod m$ if $c_i = 1$. Finally, if $c_d = 1$ set $r_{d+1} = a r_d$, and otherwise set $r_{d+1} = r_d$. Then $r_{d+1} \equiv a^k \pmod m$.
  - To illustrate the idea, consider the problem of computing a^6 and note that $13 = \underline{1101}_2$, so $c_1 = c_2 = c_4 = 1$ and $c_3 = 0$.
  - We begin with $r_1 = 1$. Then $r_2 = (a r_1)^2 = a^2$ since $c_1 = 1$.
  - Similarly, $r_3 = (a r_2)^2 = (a^3)^2 = a^6$ since $c_2 = 1$.
  - Next, $r_4 = (r_3)^2 = (a^6)^2 = a^{12}$ since $c_3 = 0$.
  - Finally, $r_5 = a r_4 = a^{13}$ since $c_4 = 1$. We obtain the result a^{13}, as required.

- We will remark that for certain exponents, this exponentiation process can be streamlined further using more general "addition chains". However, the gains are generally small, and come at the expense of using additional memory during the computation.
  - As an example, the binary method we described above requires 6 steps to compute $a^{15} \pmod m$: it successively computes a^2, $a \cdot a^2$, $(a \cdot a^2)^2$, $a \cdot (a \cdot a^2)^2$, $(a \cdot (a \cdot a^2)^2)^2$, and $a \cdot (a \cdot (a \cdot a^2)^2)^2$.
  - However, if instead we compute the quantities a^2, $a \cdot a^2 = a^3$, $(a^3)^2$, $((a^3)^2)^2$, and $a^3 \cdot ((a^3)^2)^2$, we can compute a^{15} using only 5 steps, although it requires storing the value a^3 for later use.
  - In general, it is a difficult problem to search for optimized addition chains.

2.2.1 Orders of Elements Modulo m

- We would like to study the behavior of powers of units modulo m.
- A basic observation is that if u is any fixed unit, then u^k is also a unit for any k (since its inverse is $(u^{-1})^k$).
  - Since there are only finitely many possible different values for u^k modulo m, the values u, u^2, u^3, ... must eventually repeat. Indeed, this is true for the powers of any element modulo m, even nonunits.
  - But if $u^a \equiv u^b$ with $a < b$, multiplying by u^{-a} shows that $u^{b-a} \equiv 1$.
  - This means that some power of u is equal to 1 mod m. We give this situation a name:
- Definition: If u is a unit modulo m, the smallest $k > 0$ such that $u^k \equiv 1 \pmod{m}$ is called the order of u.
- Example: Find the order of 2 modulo 11.
  - We compute powers:

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	$\cdots$
2	4	8	5	10	9	7	3	6	1	2	4	$\cdots$

  - The earliest time we obtain 1 is with 2^{10}, so 2 has order $\boxed{10}$ modulo 11.
- Example: Find the order of 5 modulo 13.
  - We compute powers:

5^1	5^2	5^3	5^4	5^5	5^6	5^7	5^8	$\cdots$
5	12	8	1	5	12	8	1	$\cdots$

  - The earliest time we obtain 1 is with 5^4, so 5 has order $\boxed{4}$ modulo 13.
- We collect a few useful results about orders.
- Proposition: If u is a unit modulo m and $u^n \equiv 1 \pmod{m}$, then the order of u divides n.
  - Proof: Let k be the order of u.
  - Apply the division algorithm to write $n = qk + r$ with $0 \leq r < k$, and then observe that $u^r = u^n (u^k)^{-q} \equiv 1 \cdot 1^{-q} \equiv 1 \pmod{m}$.
  - If r were not zero, then we would have $u^r \equiv 1 \pmod{m}$ with $0 < r < k$, which contradicts the definition of order. Thus $r = 0$, meaning that k divides n.
- Proposition: If u has order k modulo m, then the order of u^n modulo m is $k/\gcd(n,k)$. In particular, if n and k are relatively prime, then u^n also has order k.
  - Proof: Let $d = \gcd(n,k)$ and suppose u^n has order r: then $u^{nr} \equiv 1 \pmod{m}$.
  - By the previous proposition, we see that k divides nr, which is equivalent to saying that k/d divides $(n/d)r$.
  - But since k/d and n/d are relatively prime, this implies k/d divides r.
  - On the other hand, $(u^n)^{k/d} = (u^k)^{n/d} \equiv 1^{n/d} = 1 \pmod{m}$, so the order of u^n cannot be larger than k/d.
  - Thus, we conclude $r = k/d$. The other statement is immediate, being simply the case with $d = 1$.
- The next result is useful in computing the order of a product of two elements.
- Proposition: If u has order k and w has order $l \pmod{m}$, where k and l are relatively prime, then uw has order kl.
  - Proof: Suppose that uw has order d, so that $(uw)^d \equiv 1 \pmod{m}$.
  - Raising to the kth power yields $w^{dk} \equiv 1 \pmod{m}$, so by the above proposition, we see that l divides dk.

- Since l and k are relatively prime, this implies l divides d.
- By a symmetric argument, k divides d. Since l and k are relatively prime, we see kl divides d.
- But clearly, $(uw)^{kl} \equiv 1 \pmod{m}$, so $d \leq kl$. Hence we obtain $d = kl$ as claimed.
- Remark: A weaker result also holds when the orders k and l are not relatively prime: in general, the argument above shows that the order of uw is a multiple of $kl/\gcd(k,l)^2$, and divides $kl/\gcd(k,l) = \text{lcm}(k,l)$. (A particularly bad case is if $u = w^{-1}$.)

- We might also like a method to verify that a unit u modulo m has a particular order, in a way that is more efficient than computing all of the lower powers of u.
- Proposition: If $u^d \equiv 1 \pmod{m}$, and $u^{d/p} \not\equiv 1 \pmod{m}$ for any prime divisor p of d, then u has order d modulo m.
  - Proof: Suppose u has order r modulo m. We know that r must divide d, by our results on orders.
  - If $r < d$, then there must be some prime p in the prime factorization of d that appears to a strictly lower power in the factorization of r: then r divides d/p.
  - But then $u^{d/p}$ would be an integral power of $u^r \equiv 1$, so that $u^{d/p} \equiv 1 \pmod{m}$, contrary to the given information.
  - Hence we conclude $r = d$, meaning that u has order d.

2.2.2 Fermat's Little Theorem

- From the examples we computed earlier, we can see that $2^{11} \equiv 2 \pmod{11}$, and also that $5^{13} \equiv 5 \pmod{13}$. The presence of these exponents is not an accident:
- Theorem (Fermat's Little Theorem): If p is a prime, then $a^p \equiv a \pmod{p}$.
  - Remark: If $p \nmid a$, we can multiply by a^{-1} to get the equivalent formulation $a^{p-1} \equiv 1 \pmod{p}$. Since the result is immediate if $p|a$, Fermat's Little Theorem is often also stated as "$a^{p-1} \equiv 1 \pmod{p}$ if $p \nmid a$".
  - Proof: We will show that $a^{p-1} \equiv 1 \pmod{p}$ if $p \nmid a$. By assumption, a is a unit.
  - Consider the elements $a \cdot 1,\ a \cdot 2,\ a \cdot 3, \cdots, a \cdot (p-1)$ modulo p: we claim that they are simply the nonzero residue classes modulo p, in some order.
  - Since there are $p-1$ elements listed and they are all nonzero, it is enough to verify that they are all distinct.
  - So suppose $a \cdot x \equiv a \cdot y \pmod{p}$. Since a is a unit, multiply by a^{-1}: this gives $x \equiv y \pmod{p}$, but this forces $x = y$.
  - Hence modulo p, the elements $a \cdot 1,\ a \cdot 2,\ a \cdot 3, \cdots, a \cdot (p-1)$ are simply 1, 2, $\cdots$, $(p-1)$ in some order. Therefore we have
    $$(a \cdot 1)(a \cdot 2) \cdots (a(p-1)) \equiv 1 \cdot 2 \cdot \cdots \cdot (p-1) \pmod{p},$$
    whence
    $$a^{p-1} \cdot (p-1)! \equiv (p-1)! \pmod{p}.$$
    Cancelling the $(p-1)!$ term yields $a^{p-1} \equiv 1 \pmod{p}$, as desired.

- Using Fermat's little theorem we can compute large powers modulo primes more efficiently than with successive squaring.
- Example: Calculate (as efficiently as possible) the remainder when 2^{3003} is divided by 61.
  - We could use successive squaring to compute this, but we would need to square 12 times (since $2^{12} = 2048$).
  - Since 61 is prime, we can do the computation much more quickly if we use Fermat's Little Theorem, which tells us that $2^{60} \equiv 1 \pmod{61}$.
  - Taking the 50th power of this yields $2^{3000} = (2^{60})^{50} \equiv 1^{50} = 1 \pmod{61}$.
  - Thus, $2^{3003} = 2^3 \cdot 2^{3000} \equiv \boxed{8 \pmod{61}}$.

2.2.3 The Chinese Remainder Theorem

- We will now divert our attention slightly and discuss the solutions to linear equations modulo m. We have already explained how to solve a single equation:
- Proposition: The equation $ax \equiv b \pmod{m}$ has a solution for x if and only if $d = \gcd(a, m)$ divides b. If $d|b$, then if we write $a' = a/d$, $b' = b/d$, and $m' = m/d$, the general solution to the equation $ax \equiv b \pmod{m}$ is $x \equiv (a')^{-1}b' \pmod{m'}$.
- Now suppose that we wish to solve a collection of simultaneous congruences in the variable x. The above proposition allows us to convert any single equation $cx \equiv d \pmod{m}$ to one of the form $x \equiv a \pmod{m'}$, or to see that such an equation has no solutions (in which case neither does the system!). Therefore, to solve general systems, all we must do is characterize those x which satisfy a system of the form

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \quad \vdots \quad \vdots \\ x &\equiv a_k \pmod{m_k}. \end{aligned}$$

- Of course, it is possible for the equations to be inconsistent: for example, the system

$$\begin{aligned} x &\equiv 1 \pmod{4} \\ x &\equiv 2 \pmod{6} \end{aligned}$$

  has no solution, because the first equation requires x to be odd and the second requires x to be even. The key problem here is that 4 and 6 are not relatively prime, and the equations give inconsistent requirements modulo $2 = \gcd(4, 6)$. It turns out that this is the only thing that can go awry.
- Theorem (Chinese Remainder Theorem): Let $m_1, m_2, \ldots, m_k$ be pairwise relatively prime positive integers (i.e., with $\gcd(m_i, m_j) = 1$ whenever $i \neq j$), and $a_1, a_2, \ldots, a_k$ be arbitrary integers. Then there exists an integer a such that the set of values of x satisfying the equations

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \quad \vdots \quad \vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

  is precisely those integers x congruent to a modulo $m_1 m_2 \cdots m_k$. In other words, the system has a unique solution modulo $m_1 m_2 \cdots m_k$.
  - Remark: This theorem is so named because it was known to Chinese mathematicians of antiquity.
  - Proof: First we prove the result for two congruences

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2}. \end{aligned}$$

  - For existence, the first congruence implies $x = a_1 + km_1$ for some integer k; plugging into the second then yields $a_1 + km_1 \equiv a_2 \pmod{m_2}$. Rearranging yields $km_1 \equiv (a_2 - a_1) \pmod{m_2}$. Since by hypothesis m_1 and m_2 are relatively prime, by our proposition above we see that this congruence has a unique solution for k modulo m_2, and hence a solution for x.
  - For uniqueness, suppose x and y are both solutions. Then $x - y$ is 0 modulo m_1 and 0 modulo m_2, meaning that $m_1|(x - y)$ and $m_2|(x - y)$. But since m_1 and m_2 are relatively prime, their product must therefore divide $x - y$, meaning that x is unique modulo $m_1 m_2$. It is also obvious that any other integer congruent to x modulo $m_1 m_2$ also satisfies the system.

- Finally, if we have more than two congruences, we can apply the result just proven to convert the last two congruences into a single congruence. Repeatedly applying this procedure shows that the solution set is a single residue class modulo $m_1 m_2 \cdots m_k$, as claimed.

- The proof is essentially constructive in that it also allows us to compute the solutions explicitly.

- Example: Find all integers n such that $n \equiv 1 \pmod 7$ and $n \equiv 3 \pmod 8$.
  - The Chinese Remainder Theorem says we only need to compute one solution; all others are congruent modulo $7 \cdot 8$.
  - The first congruence implies that $n = 3 + 8k$ for some integer k.
  - Plugging into the second congruence yields $3 + 8k \equiv 1 \pmod 7$, which reduces to $k \equiv -2 \pmod 7$.
  - Taking $k = -2$ yields $n = 3 + 8k = -13$. The set of all solutions are the integers of the form $\boxed{-13 + 56d}$ for $d \in \mathbb{Z}$.

- Although the Chinese Remainder Theorem is only stated for relatively prime moduli, it is easy to deal with the case where the moduli have common divisors.
  - The Theorem implies that if $d|m$ and $\gcd(d, m/d) = 1$, the single equation $x \equiv a \pmod m$ is equivalent to the two equations $x \equiv a \pmod d$ and $x \equiv a \pmod{m/d}$.
  - Thus, if the moduli have common divisors, we need only compute the common divisors (rapidly, using the Euclidean Algorithm), and then split the congruences apart so as to have no common divisors.
  - If we can factor the moduli, we could also just split them all into prime powers. (However, we can still split the moduli into relatively prime pieces in the manner described above, even if we cannot factor them.)
  - Alternatively, we could simply solve each congruence, plug it into the next one, and eliminate coefficients. This also works, though it requires a bit more care in dealing with the case where the coefficients and modulus have a common divisor. (It will also be less obvious precisely where a contradiction between the congruences occurs.)

- Example: Find all solutions to the congruences $n \equiv 34 \pmod{36}$, $n \equiv 7 \pmod{15}$, and $n \equiv 2 \pmod{40}$.
  - First observe that 36 and 15 have a common divisor of 3. Since 3^2 divides 36, we split the first congruence into two congruences modulo 4 and 9, and the second into congruences modulo 3 and 5.
  - This yields $n \equiv 2 \pmod 4$, $n \equiv 7 \pmod 9$, $n \equiv 1 \pmod 3$, and $n \equiv 2 \pmod 5$.
  - These congruences' moduli have common divisors with the last congruence, which we split modulo 5 and modulo 8 to obtain $n \equiv 2 \pmod 5$ and $n \equiv 2 \pmod 8$.
  - We then have $n \equiv 2 \pmod 4$, $n \equiv 7 \pmod 9$, $n \equiv 1 \pmod 3$, $n \equiv 2 \pmod 5$, $n \equiv 2 \pmod 5$, and $n \equiv 2 \pmod 8$.
  - Removing duplicates yields no contradictions, and we get $n \equiv 7 \pmod 9$, $n \equiv 2 \pmod 5$, and $n \equiv 2 \pmod 8$, whose moduli are now relatively prime.
  - The second two congruences visibly have the common solution $n \equiv 2 \pmod{40}$, giving $n = 2 + 40k$ for some k.
  - Plugging into the only remaining congruence yields $2 + 40k \equiv 7 \pmod 9$, whence $4k \equiv 5 \pmod 9$. The inverse of 4 modulo 9 is easily computed as -2. Multiplying by it yields $k \equiv -10 \equiv -1 \pmod 9$.
  - Hence the congruences have a solution $x = 2 + 40k = -38$, and the set of all solutions is $x = \boxed{-38 + 360d}$ for $d \in \mathbb{Z}$.

2.2.4 The Euler φ-Function and Euler's Theorem

- We would now like to generalize Fermat's little theorem to the case of composite moduli: that is, to find some exponent n such that $a^n \equiv a \pmod{m}$, or something like this. For motivation, we first try a few examples:
  - Consider the powers of 2 modulo 24:

$$\begin{array}{cccccccccc} 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 & 2^8 & \cdots \\ 2 & 4 & 8 & 16 & 8 & 16 & 8 & 16 & \cdots \end{array}$$

  - Here, we see that the powers do eventually start repeating, but they never return to 1 (nor even to 2). This should not be surprising, because 2 is not a unit modulo 24. In particular, we see that there is no exponent $n > 1$ such that $2^n \equiv 2 \pmod{24}$.
  - Instead, perhaps we should only consider cases where a is a unit modulo m. Consider the powers of 2 modulo 21:

$$\begin{array}{cccccccccccccc} 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 & 2^8 & 2^9 & 2^{10} & 2^{11} & 2^{12} & \cdots \\ 2 & 4 & 8 & 16 & 11 & 1 & 2 & 4 & 8 & 16 & 11 & 1 & \cdots \end{array}$$

    so we see that 2 has order 6 modulo 21. We also note that $2^{20} \equiv 4 \not\equiv 1 \pmod{21}$, so the proper exponent is not simply $m - 1$, like it is for primes.

- From the examples above, we see that in order to obtain some sort of reasonable generalization of Fermat's Little Theorem to composite moduli, we should restrict our attention to powers of units.
- We have already characterized which integers are units modulo m, but we might also like to know how many there are:
- Definition: If m is a positive integer, we define the Euler φ-function $\varphi(m)$, sometimes also called Euler's totient function, to be the number of units modulo m. Equivalently, $\varphi(m)$ is the number of integers between 1 and m inclusive that are relatively prime to m.
  - It is self-evident that $\varphi(1) = 1$, and that $\varphi(p) = p - 1$ if p is a prime.
  - Example: To compute $\varphi(30)$, we simply list the integers relatively prime to 30 in the proper range. It is not hard to see that 1, 7, 11, 13, 17, 19, 23, and 29 are the only ones, so $\varphi(30) = 8$.
- Proposition: If p is a prime and $k \geq 0$, $\varphi(p^k) = p^k - p^{k-1}$.
  - Proof: Observe that a has a common divisor with p^k if and only if p divides a.
  - Therefore, the integers between 1 and p^k which are *not* relatively prime to p^k are simply the multiples of p, of which there are p^{k-1}. The remaining $p^k - p^{k-1}$ integers are relatively prime to p.
- We can use the Chinese Remainder Theorem to great effect in analyzing the Euler φ-function:
- Theorem (Multiplicativity of φ): If m and n are relatively prime, then $\varphi(mn) = \varphi(m)\varphi(n)$.
  - Terminology: A function f with domain $\mathbb{Z}$ with the property that $f(mn) = f(m)f(n)$ whenever m and n are relatively prime is called a multiplicative function. This terminology is somewhat infelicitous since it would tend to suggest that $f(mn) = f(m)f(n)$ holds for *any* m and n, not just relatively prime ones.
  - Proof: We claim that if m and n are relatively prime, a is a unit modulo mn if and only if it is a unit mod m and mod n.
  - If a is a unit mod mn, then there exists a b such that $ab \equiv 1 \pmod{mn}$. Reducing mod m and n yields $ab \equiv 1 \pmod{m}$ and $ab \equiv 1 \pmod{n}$, so a is a unit mod m and mod n.
  - Conversely, if a is a unit mod m and mod n, then let $ab \equiv 1 \pmod{m}$ and $ac \equiv 1 \pmod{n}$. Then the Chinese Remainder Theorem implies there exists an integer d such that $d \equiv b \pmod{m}$ and $d \equiv c \pmod{n}$: then ad is congruent to 1 modulo m and modulo n, hence modulo mn.
  - Now we just count: there are $\varphi(mn)$ units modulo mn, and $\varphi(m)\varphi(n)$ pairs of units modulo m and modulo n. These are counting the same thing, so $\varphi(mn) = \varphi(m)\varphi(n)$.

- Corollary (Formula for φ): If $m = \prod_i p_i^{a_i}$ is factored into prime powers, then $\varphi(m) = \prod_i p_i^{a_i-1}(p_i - 1) = m\prod_i(1 - 1/p_i)$. In particular, the value of $\varphi(m)/m$ only depends on the primes dividing m.
  - Proof: Apply the relation $\varphi(mn) = \varphi(m)\varphi(n)$ repeatedly to the factorization of m into prime powers.
  - We obtain $\varphi(m) = \prod_i \varphi(p_i^{a_i}) = \prod_i(p_i^{a_i} - p_i^{a_i-1})$, which can be rearranged into the two given formulas.
- Example: Find $\varphi(1680)$.
  - First we factor $1680 = 2^4 \cdot 3 \cdot 5 \cdot 7$, and then we can write $\varphi(1680) = \varphi(2^4)\varphi(3)\varphi(5)\varphi(7) = 8 \cdot 2 \cdot 4 \cdot 6 = \boxed{384}$.
- Now that we understand the units a bit better, we can give the proper generalization of Fermat's little theorem to composite moduli:
- Theorem (Euler's Theorem): If $\gcd(a, m) = 1$, then $a^{\varphi(m)} \equiv 1 \pmod{m}$.
  - Proof: By assumption, a is a unit mod m.
  - Let the set of all units mod m be $u_1, u_2, \ldots, u_{\varphi(m)}$, and consider the elements $a \cdot u_1,\ a \cdot u_2, \cdots, a \cdot u_{\varphi(m)}$ modulo m: we claim that they are simply the elements $u_1, u_2, \ldots, u_{\varphi(m)}$ again (possibly in a different order).
  - Since there are $\varphi(m)$ elements listed and they are all still units, it is enough to verify that they are all distinct.
  - So suppose $a \cdot u_i \equiv a \cdot u_j \pmod{m}$. Since a is a unit, multiply by a^{-1}: this gives $u_i \equiv u_j \pmod{m}$, but this forces $i = j$.
  - Hence modulo m, the elements $a \cdot u_1,\ a \cdot u_2, \cdots, a \cdot u_{\varphi(m)}$ are simply $u_1, u_2, \ldots, u_{\varphi(m)}$ in some order. Therefore we have
$$(a \cdot u_1)(a \cdot u_2) \cdots (a \cdot u_{\varphi(m)}) \equiv u_1 \cdot u_2 \cdots u_{\varphi(m)} \pmod{m},$$
whence, upon cancelling $u_1 \cdot u_2 \cdots u_{\varphi(m)}$ from both sides, we obtain
$$a^{\varphi(m)} \equiv 1 \pmod{m},$$
as desired.
- We can use Euler's theorem to compute large powers of elements with a composite modulus.
- Example: Find the last two digits of 17^{2016} when written in base 10.
  - Equivalently, we want to compute $17^{2016} \pmod{100}$. We could do this by directly using successive squaring, but we would need to square 10 times (since $2^{10} = 1024$) and then do many multiplications.
  - Alternatively, we could use Euler's theorem. Since $100 = 2^2 5^2$ we have $\varphi(100) = \varphi(4)\varphi(25) = 2 \cdot 20 = 40$.
  - Then Euler's theorem says that $17^{40} \equiv 1 \pmod{100}$. Taking the 50th power yields $17^{2000} = (17^{40})^{50} \equiv 1^{50} = 1 \pmod{100}$.
  - Then $17^{2016} \equiv 17^{16} \pmod{100}$, and we can compute this with far fewer successive squarings:
$$\begin{aligned} 17^2 &= 289 \equiv -11 \pmod{100} \\ 17^4 &\equiv (-11)^2 = 121 \equiv 21 \pmod{100} \\ 17^8 &\equiv 21^2 = 441 \equiv 41 \pmod{100} \\ 17^{16} &\equiv 41^2 = 1681 \equiv 81 \pmod{100} \end{aligned}$$
Therefore, we see that $17^{2016} \equiv \boxed{81 \pmod{100}}$.

2.3 Rabin Encryption

- A simple example of a public-key cryptosystem is the Rabin public-key cryptosystem.
  - This procedure was first published in 1979 by Michael O. Rabin. It is one of the first non-classified public-key cryptosystems, and it is also one of the simplest.
- First, Bob must create his public key.
  - To do this, he simply computes two large primes p and q each congruent to 3 modulo 4.
  - Bob then publishes $N = pq$. This value N is his public key.
- Now suppose that Alice wants to send Bob a message.
  - First, Alice converts her message into an integer m modulo N in some agreed-upon manner.
  - For example, if N has 257 digits in base 2, then Alice could break her message into pieces that are each 256 base-2 digits long, and encode each one separately.
  - If Alice's message is text, she would of course convert it to a number using some fixed text encoding, and then break it into pieces as above.
  - Alice then computes m^2 modulo N and sends the result to Bob.
- If Bob receives a message m^2, then to decode the message Bob needs to compute the square root of m^2 modulo pq.
  - By the Chinese Remainder Theorem, Bob can equivalently find the solutions to $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$, where $a = m^2$.
  - Each of these congruences has two solutions, and finding one of them immediately gives the other: $x^2 \equiv m^2 \bmod p$ is equivalent to $p|(x-m)(x+m)$, meaning $x = \pm m \bmod p$.
  - The key observation is that $x = a^{(p+1)/4}$ has the property that $x^2 \equiv a \pmod{p}$: since $a = m^2 \pmod{p}$ and $m^{p-1} \equiv 1 \pmod{p}$ by Euler's theorem, we have
$$x^2 \equiv a^{(p+1)/2} \equiv m^{p+1} \equiv m^2 \equiv a \pmod{p}.$$
  - Therefore, to decrypt the message, Bob must solve the simultaneous congruences $x = \pm a^{(p+1)/4} \pmod{p}$ and $x = \pm a^{(q+1)/4} \pmod{q}$, which he can do easily with the Chinese Remainder Theorem.
- Note that once Bob decrypts the message, he will have four values each of which squares to m^2 modulo N: how does he know which one was actually Alice's original message?
  - Without additional information, Bob cannot determine which of these four values was actually Alice's message.
  - One way of fixing this problem is for Alice to append some particular string of digits to the beginning of her message m: it is then very unlikely that any of the other square roots of m^2 will also start with this string of digits.
- Example: Bob sets up a Rabin public-key cryptosystem with $N = 1817 = 23 \cdot 79$. Alice sends him the encrypted message 347, and tells Bob that the two-digit message was padded with starting digits "11". Decode the message.
  - Decoding requires solving $x^2 \equiv 347 \pmod{1817}$ for x.
  - By our analysis, the solutions satisfy $x \equiv \pm 347^{(23+1)/4} \pmod{23}$ and $x \equiv \pm 347^{(79+1)/4} \pmod{79}$.
  - Successive squaring yields $x \equiv \pm 18 \pmod{23}$ and $x \equiv \pm 49 \pmod{79}$.
  - Using the Chinese Remainder Theorem, we obtain the four solutions $x \equiv \pm 662, \pm 741 \pmod{1817}$.
  - Hence the original message was one of $x = 662, 741, 1076, 1155$. The only one of these that starts with "11" is $x = 1155$, so the original message was $\boxed{55}$.

- Now suppose that Eve intercepts the encrypted message m^2 and wants to decode it. In order to do this, Eve would need to be able to compute all the square roots of m^2 modulo N.
- We claim that computing these square roots is equivalent to factoring N when N is a product of two primes.
  - Explicitly, suppose that m is a unit modulo N, and we are looking for the solutions of $x^2 \equiv m^2 \pmod{N}$.
  - By the Chinese Remainder Theorem, solving $x^2 \equiv a \pmod{pq}$ is equivalent to solving $x^2 \equiv m^2 \pmod{p}$ and $x^2 \equiv m^2 \pmod{q}$.
  - Observe that $x^2 \equiv m^2 \pmod{p}$ is equivalent to $(x-m)(x+m) \equiv 0 \pmod{p}$, or $p|(x-m)(x+m)$, from which $x \equiv \pm m \pmod{p}$.
  - Similarly, $x \equiv m^2 \pmod{q}$ is equivalent to $x \equiv \pm m \pmod{q}$.
  - Thus, there are four solutions to the congruence $x^2 \equiv m^2 \pmod{n}$: they are $\pm m$ and $\pm w$, where $w \equiv m \pmod{p}$ and $w \equiv -m \pmod{q}$.
  - Now observe that $w + m \equiv 2m \pmod{p}$ and $w + m \equiv 0 \pmod{q}$, so q divides $w + m$ but p does not. Therefore, $\gcd(w+m, pq) = p$.
  - Therefore, if we are given the three values w, m, and $pq = N$, we can find the value of a prime factor of N, and thus its factorization because N is the product of two primes, by computing $\gcd(w+m, pq)$. (Computing the greatest common divisor is very fast using the Euclidean algorithm.)
  - What this means is: breaking Rabin encryption for a single message is equivalent to factoring N.
  - If p and q are both very large, then it is believed to be extremely difficult to factor N: thus, Eve will be unable to decode Alice's message.
- Rabin encryption is very simple, yet it is easy to prove that breaking it (in general) is equivalent to factoring the public key N. However, it does suffer from some weaknesses of varying severity, of which we will list a few.
- Attack 1 (Brute force): If the number of possible plaintexts is small and Eve wants to know how a message decodes, she could simply encrypt all possible plaintexts and compare them to the ciphertext.
  - This is not really a problem of Rabin encryption per se: the same problem exists for any cryptosystem with a small number of possible plaintexts.
  - To avoid this issue, Bob simply needs to choose his value of N to be sufficiently large that it is infeasible for Eve to test every possible plaintext, and then to pad each message with a random string at the beginning (or end), of sufficient length that makes it infeasible for Eve to test all of the possibilities.
  - Padding can also overcome the nonuniqueness of square roots, but (in this case) breaking the encryption is no longer provably equivalent to factorization.
- Attack 2 (Chosen-ciphertext): Eve chooses a random message m and asks Bob's decoding machine to decode m^2 for her. Eve then has a good chance of being able to use the result to determine Bob's key.
  - As we explained above, there are four square roots of m^2 modulo n: $\pm m$ and $\pm w$, where w is the solution to $w \equiv m \pmod{p}$ and $w \equiv -m \pmod{q}$.
  - When Bob's computer decodes Eve's message, it has a 50% chance of erroneously assuming that w or $-w$ was actually Eve's message.
  - Suppose it gives Eve the value of w: then by using the attack we described above, $\gcd(m+w, n) = q$ is one of the prime divisors of Bob's public key N.
  - Similarly, the computer gives Eve the value $-w$, then $\gcd(m-w, N) = p$.
  - Hence, there is a 50% chance that Eve would be able to factor Bob's public key and thus break the encryption.
  - If Even repeats this process a mere ten times, she will be overwhelmingly likely to obtain a factorization of Bob's public key.
  - Bob can attempt to prevent this by never revealing a decrypted message to anyone. But in a computerized implementation of the procedure, this is very hard to manage.

- The second attack is sufficiently serious that (in addition to the rather annoying issue of nonuniqueness of square roots) Rabin encryption, despite being provably equivalent to factorization, is not suitable for modern use.
    - One way to try to fix the problem is to pad each message to make them adhere to a particular format (for example, by encoding messages in blocks of 1024 bits, where the last 128 bits are duplicates of the previous 128) and then refuse to return a decoded message that does not decode to the correct format.
    - It would not be possible to use a chosen-ciphertext attack to get around such a procedure since the number of attempts required to find a ciphertext message whose corresponding plaintext adheres to the correct encoding is on the order of 2^{126} or so (each ciphertext has 4 associated plaintexts, and a random plaintext has a $1/2^{128}$ probability of having the right formatting).
    - However, making any alteration to the Rabin encryption scheme will yield something that is no longer provably equivalent to factorization.
- We will also remark that it is not necessary to restrict the primes to being congruent to 3 modulo 4.
    - This assumption is only made because it is much easier to compute square roots modulo such primes using successive squaring, because $(m^2)^{(p+1)/4} \equiv m \pmod{p}$.
    - There are other fast algorithms to compute square roots modulo primes congruent to 1 modulo 4, but they require some more results from abstract algebra (specifically, a finite field factorization algorithm known as Berlekamp's algorithm).
- There are also some other ways to attempt to fix the nonuniqueness of square roots modulo N.
    - One way is to take both primes congruent to 3 modulo 4 and then require that the message m be a perfect square modulo $N = pq$.
    - There is an efficient procedure for determining whether a given residue class a modulo a prime p is a perfect square (or "quadratic residue") involving the computation of something called the Legendre symbol. Specifically,
    - If $p \equiv q \equiv 3 \pmod{4}$ then it can be shown that among the four messages $\pm m$ and $\pm w$, only m will be a perfect square modulo p and modulo q.
    - It is then easy for Bob to determine which of the four possible square roots of m^2 was the original message by identifying the one that is a square modulo p and modulo q.
    - However, in practice this procedure is not really possible to implement: Alice would need to know that her message was a perfect square before she sends it, and determining whether a given element m modulo $N = pq$ is a square is in general a very difficult problem. (There are in fact other public-key cryptosystems based on the hardness of this problem.)
    - There is a fast method, called the Jacobi symbol, that 50% of the time will tell Alice that her message is not a square, and the other 50% of the time will tell her that there is a 1/2 probability her message is a square and a 1/2 probability it is not a square. Ultimately, however, it seems that the nonuniqueness is extremely difficult to remove.

2.4 The RSA Encryption System

- One of the practical issues with the Rabin cryptosystem is the nonuniqueness of square roots, since its encoding function (the squaring map modulo N) is not one-to-one.
- A way to get around this problem is to use a different power map, rather than the squaring map, chosen to be invertible mod N: this is the idea behind RSA encryption.
    - The RSA cryptosystem was first publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, from whose surnames the initialism "RSA" was formed.
    - It turns out that an essentially equivalent system had been developed by Clifford Cocks in 1973 while working for Britain's Government Communications Headquarters (GCHQ). However, his work was not declassified until 1997, and his system was marginally less general than RSA.

2.4.1 Procedure for RSA

- First, Bob must create his public key.
  - To do this, he first computes two large primes p and q and sets $N = pq$.
  - Bob also chooses an integer e which is relatively prime to $\varphi(N) = (p-1)(q-1)$.
    - Often, $e = 3$ is used. (This requires choosing p and q to be primes congruent to 2 modulo 3.) There are various reasons, which we discuss later, why $e = 3$ is not always a good choice.
    - Another popular choice is $e = 2^{16} + 1 = 65537$, which is prime and also allows for rapid successive squaring.
  - Bob then publishes the two values N and e, which serve as his public key.
- Now suppose that Alice wants to send Bob a message.
  - Alice converts her message into an integer m modulo N in some agreed-upon manner.
  - Alice then computes $c \equiv m^e$ modulo N (using successive squaring) and sends the result to Bob.
- If Bob has received a ciphertext block $c \equiv m^e \pmod{N}$, he wishes to recover the value of m.
  - We claim that Bob can recover m by computing c^d modulo N using successive squaring, where d is the inverse of e modulo $\varphi(N)$.
  - By choosing e to be relatively prime to $\varphi(N)$, such a d will always exist, and Bob can easily compute it via the Euclidean algorithm because he knows $\varphi(N) = (p-1)(q-1)$.
  - Most actual implementations of RSA use the Chinese Remainder Theorem to do the decoding modulo p and modulo q separately, and then combine the results. This is faster since the moduli are much smaller, but it is not strictly necessary.
- One way to show that the decryption procedure will work is via the Chinese Remainder Theorem and Fermat's Little Theorem.
  - Explicitly, since $N = pq$, by the Chinese Remainder Theorem it is enough to show that $c^d \equiv m \pmod{p}$ and $c^d \equiv m \pmod{q}$.
  - By assumption, $de \equiv 1 \pmod{\varphi(N)}$ and $\varphi(N) = (p-1)(q-1)$, so in particular $de \equiv 1 \pmod{p-1}$, so $de = 1 + k(p-1)$ for some integer k.
  - Now since $c \equiv m^e \pmod{p}$, we have $c^d \equiv m^{de} \equiv m^{1+k(p-1)} \equiv m \cdot (m^{p-1})^k \pmod{p}$.
  - If $m \equiv 0 \pmod{p}$ then $c^d \equiv 0 \equiv m \pmod{p}$ so the result holds.
  - Otherwise, if p does not divide m, by Fermat's Little Theorem we have $m^{p-1} \equiv 1 \pmod{p}$, so $c^d \equiv m \cdot 1^k \equiv m \pmod{p}$, as claimed.
  - We can use the same argument to see that $c^d \equiv m \pmod{q}$: thus, $c^d \equiv m \pmod{pq}$, as required.
- There is a slightly faster to see how the procedure works using Euler's theorem.
  - Again, since $c \equiv m^e \pmod{N}$, we obtain $c^d \equiv m^{de} \pmod{N}$.
  - Also, since $de \equiv 1 \pmod{\varphi(N)}$ we can write $de \equiv 1 + r\varphi(N)$ for some integer r.
  - Then by Euler's theorem, if m is relatively prime to N we have $m^{\varphi(N)} \equiv 1 \pmod{N}$, so we can write

    $$c^d \equiv m^{de} \equiv m^{1+r\varphi(N)} \equiv m \cdot (m^{\varphi(N)})^r \equiv m \cdot 1^r \equiv m \pmod{N}$$

    as required.
  - Note that technically, this explanation only applies when m is relatively prime to N.
  - In practice, however, this is essentially always the case, since the only time m is not relatively prime to N is when m is divisible by p or by q, which happens only with probability about $1/p + 1/q$.
- Example: Encode, and then decode, the message $m = 444724$ using RSA, with $N = 18\,212\,959$ and $e = 3$.

- To encode, we simply compute m^3 modulo N, which is $\boxed{12\,534\,939}$.
- To decode, we first factor $N = 3329 \cdot 5471$, and compute $\varphi(N) = 3328 \cdot 5470 = 18\,204\,160$.
- Next, we need to find the decryption exponent d, which is the inverse of 3 modulo $\varphi(N) = 18204160$.
- Applying the Euclidean algorithm will eventually produce the relation $18204160 - 6068053 \cdot 3 = 1$, from which we can see that the inverse is $-6068053 \equiv 12\,136\,107$.
- Hence $d = 12\,136\,107$.
- Now we simply compute $12\,534\,939^{12\,136\,107}$ modulo N via successive squaring. (Of course, this requires a computer.)
- We eventually obtain the decrypted message $\boxed{444\,724}$, which is, of course, what we should have gotten.

- It is clear from our description that RSA is fairly straightforward to implement, at least in principle.
  - The encoding and decoding procedures only require successive squaring, which is quite fast.
  - Bob's computation of the decryption exponent d requires the Euclidean algorithm, which is also quite fast.
  - It is not so obvious, however, that RSA is secure.

- Suppose Eve is spying on Alice and Bob.
  - Eve will have the values of N and e, since those are public, and she will also have the ciphertext $c \equiv m^e \pmod{N}$.
  - Thus, Eve's goal is to solve the congruence $m^e \equiv c \pmod{N}$ for m, given the values of e, c, N.

- One way for Eve to try to decode the message is for her to find the decryption exponent d.
  - Suppose the order of m modulo N is r: then Eve needs to find is a d such that r divides $ed - 1$.
  - To see this: if $m^{ed} \equiv m \pmod{N}$, then $m^{ed-1} \equiv 1 \pmod{N}$, hence r divides $ed - 1$ by properties of order.
  - Conversely, if d is such that $ed \equiv 1 \bmod r$, then $m^{ed} \equiv m \pmod{N}$, since $m^r \equiv 1 \pmod{N}$.

- In general, the expectation is that Eve would essentially need to factor N in order to compute a decryption exponent in a reasonable amount of time. Without knowledge of the exact value of $\varphi(N)$, there is no known way to construct such a d that also allows for efficient computation.
  - Furthermore, it can be shown that computing $\varphi(N)$ is equivalent to factoring N, if N is the product of two primes.
  - It can also be shown that the order of a unit modulo $N = pq$ divides $\operatorname{lcm}(p-1, q-1)$, and that there are always units whose order is exactly equal to this value.
  - If $p-1$ and $q-1$ have many factors in common (e.g., if p and q were chosen poorly) then the order could be much smaller than N. On the other hand, if p and q are chosen carefully with $\gcd(p-1, q-1)$ small, then the lcm is quite large, meaning there is little hope for Eve to construct a d without knowledge of $\varphi(N)$.

- To summarize, it is strongly suspected (but not proven) that there does not exist any algorithm that can compute decryption exponents for RSA that is particularly more efficient than factoring the public key N.

2.4.2 Attacks on RSA

- There are a number of attacks on RSA, particularly if the encryption exponent is small. We will list a few of them at varying levels of effectiveness.

- Attack 1 (Brute force): If the number of possible plaintexts is small and Eve wants to know how a message decodes, she could simply encrypt all possible plaintexts and compare them to the ciphertext.

- As in our earlier discussion, the same problem exists for any cryptosystem with a small number of possible plaintexts.
- To avoid this issue, Bob simply needs to choose his value of N to be sufficiently large, and then to pad each message with a random string at the beginning (or end) of sufficient length that makes it infeasible for Eve to test all of the possibilities.

- Attack 2 (Factoring): If Eve wants to break Bob's RSA key, one method that would certainly work is factoring N.
  - Once Eve has a factorization of N, she can compute the decryption exponent the same way Bob does.
  - In general, it is believed that factorization of large integers is difficult with a standard (i.e., non-quantum) computer, provided the primes in the factorization are sufficiently large and not of any particularly special form (e.g., not congruent to 1 modulo a large power of 2 and not such that $p-1$ has a large number of small divisors). We will describe some general-purpose and special-purpose factorization algorithms later.
  - If some extra information about the prime divisors is known to Eve, then there are more efficient factorization procedures.
  - For example, if we are trying to factor $N = pq$ where p and q are primes of approximately equal size, and the first half or the last half of the digits of p are known, then the factorization can be found using lattice reduction methods very quickly, in time polynomial in $\log_2 p$. For comparison, a brute-force attempt of all possible primes less than p whose digits agree with the known ones would take about $\sqrt{p}$ steps.

- We will remark that the current (publicly known) record for factorization of an RSA public key is 768 bits, which took approximately 2^{67} individual computations and a total computing time equivalent to roughly 2000 years on a single-core 2GHz desktop computer.
  - It is expected that an RSA key of length 1024 bits is probably factorable now in 2016, given sufficient computing power (e.g., on the order of a government agency). But 2048 bits seems very much out of reach with current technology.
  - A direct factorization attack using a standard computer appears computationally infeasible for sufficiently large public keys. However, there exist much faster factorization algorithms, such as Shor's algorithm, that could be run on a quantum computer, assuming a sufficiently large one can ever be built.

- Attack 3 (Håstad's attack): Suppose the same message m is encrypted using the encryption exponent $e = 3$ each time and sent to 3 recipients using 3 different public keys N_1, N_2, and N_3, which are assumed to be relatively prime.
  - Note that if the public keys are not relatively prime, taking the gcd of two keys would immediately give a factorization of both, so we are not making that much of an assumption above.
  - Suppose Eve intercepts the three encoded messages c_1, c_2, and c_3.
  - Using the Chinese Remainder Theorem, Eve solves the three congruences $C \equiv c_1 \pmod{N_1}$, $C \equiv c_2 \pmod{N_2}$, and $C \equiv c_3 \pmod{N_3}$, to obtain a residue class C modulo $N_1N_2N_3$, with $C \equiv m^3 \pmod{N_1N_2N_3}$.
  - But now, since $0 \leq m < N_i$ for each $i = 1, 2, 3$, it is the case that $0 \leq m^3 < N_1N_2N_3$. Since C also lies in this range and is congruent to m^3, in fact $C = m^3$ (as an integer).
  - But now Eve can compute the plaintext m by finding the cube root of C over the integers (which is easy to do numerically).

- Håstad's attack is one of the reasons it can be a poor idea to use a small encryption exponent.
  - In general, performing Håstad's attack with an encryption exponent of e requires e different encodings of the message with different public keys.

- Identical encodings of the same message with different public keys could happen in a variety of settings. A natural one would be a mass email that is sent to many different addresses: if each copy of the message is sent to a different recipient using RSA, then an eavesdropper could obtain tens, hundreds, or even thousands of encodings of the message with different public keys (certainly enough to decode the message unless the value of e is extremely large). In practice, email is not usually encoded with asymmetric encryption, but the principle still holds.

- Although RSA is comparatively fast, it is still much slower than modern symmetric cryptosystems. As such, a typical use of RSA is to send a key for a symmetric cryptosystem which is then used to encode future messages. However, if some care is not taken when encoding the message, this procedure can be attacked.

- Attack 4 (Short plaintext attack): Suppose it is 1983 and Alice wants to send Bob a 56-bit key for DES, of which there are about 2^{56} possibilities. Alice simply encodes the message as a 56-bit integer and sends it to Bob using Bob's 200-digit RSA key.

  - A direct brute-force attack is not feasible for Eve to perform because 2^{56} is a fairly large computation even by modern standards. (Remember that it is 1983 in this example.)
  - Eve instead gambles that the key Alice encoded was a composite number with prime factors that were not unreasonably large, say $m = ab$ for some integers a, b with $a, b \leq 2^{30}$. This is reasonably likely to occur in practice.
  - Eve then computes a list of the values of $x^e \pmod N$ for all $1 \leq x \leq 2^{30}$ and all values $cy^{-e} \pmod N$ for all $1 \leq y \leq 2^{30}$.
  - If Eve finds an element common to both lists with then she knows $x^e \equiv c\,y^{-e} \pmod N$ so that $(xy)^e \equiv c \pmod N$. Raising to the dth power gives $xy \equiv c^d \equiv m \pmod N$, so Eve can compute m since she knows x and y.
  - This attack is much more efficient because Eve only needs to store two lists of 2^{30} elements each (only a few terabytes) and compare them to each other.
  - This attack is easy to defeat using a padding procedure: if Alice instead tells Bob ahead of time that she will be including 100 random digits before and after her 56-bit key, Bob can simply delete them once he decodes the message, but Eve's attack will no longer work since the message m is not likely to have a factorization into small terms.

- Attack 5 (Low decryption exponent): If the decryption exponent d is sufficiently small relative to $N = pq$ and the primes p and q are reasonably close together, it is possible to compute d very rapidly using continued fractions.

  - Specifically, if $q < p < 2q$ and if $d < \frac{1}{3}N^{1/4}$, then d can be computed rapidly.
  - First, observe that $N - \varphi(N) = p + q - 1 < 3\sqrt{N}$ by the assumptions on p and q.
  - Now if $de = 1 + k\varphi(N)$, since $d < \frac{1}{3}N^{1/4}$ and $e < \varphi(N)$ we see $\varphi(N)k < de < \frac{1}{3}\varphi(N) \cdot N^{1/4}$ so $k < \frac{1}{3}N^{1/4}$.
  - Then $0 < \dfrac{k}{d} - \dfrac{e}{N} = \dfrac{kn - ed}{dN} = \dfrac{k(n - \varphi(N))}{dN} < \dfrac{1/3 \cdot N^{1/4} \cdot 3\sqrt{N}}{dN} = \dfrac{n^{3/4}}{dN} < \dfrac{1}{3d^2}$.
  - So what this means is that $\left|\dfrac{e}{N} - \dfrac{k}{d}\right| < \dfrac{1}{3d^2}$. Recall that we are trying to compute d: what this says is that the rational number k/d is very close to e/N.
  - From the theory of continued fractions, it is known that for any real number α, if $\left|\alpha - \dfrac{p}{q}\right| < \dfrac{1}{2q^2}$ then p/q is one of the convergents to the continued fraction expansion of α. (We will not actually discuss the details of computing continued fractions since it would take us too far afield, but suffice it to say that they are easy to compute.)
  - Since k/d lies within the required bound, it must be one of the convergents to the continued fraction of e/N. Eve can compute the convergents very rapidly, and then test whether each pair (k, d) gives an integer solution to $de = 1 + k\varphi(N)$ for $\varphi(N)$.

- When she obtains an integer solution for $\varphi(N)$, she can then compute the factorization of N since she has both N and $\varphi(N)$.

- Attack 6 (Partially-known plaintext, low exponent): If a portion of a plaintext is known and the encryption exponent is low, it can be possible to decode the remaining piece of the ciphertext.
  - An example where this kind of situation can occur is when interacting with automated scripts: when resetting a password over a secure channel, the response message is likely to be something of the form "Thank you for changing your password. As confirmation, your new password is ———" (where the dashes indicate the user's password).
  - We will illustrate the basic ideas in the case where $e = 3$. If the beginning of the plaintext is known, then the message m has the form $m = A + x$ where A is a known large number and x is an unknown small number.
  - The ciphertext is then $c \equiv (A + x)^3 = x^3 + 3Ax^2 + 3Ax + A^3 \pmod{N}$.
  - Eve therefore wants to find a small solution x to the polynomial congruence $x^3+3Ax^2+3A^2x+(A^3-c) \equiv 0 \pmod{N}$.
  - Eve can apply a lattice reduction algorithm such as LLL to some well-chosen vectors to find a small solution to the modular congruence, and then solve the resulting polynomial over the real numbers to determine the value of x.
  - When N is sufficiently large relative to x, this procedure will be much faster than any kind of brute-force attack using the known plaintext information.

- Attack 7 (Timing attack): Depending on the algorithms that are used, it is possible to determine information about the decryption exponent by measuring how long it takes to perform each step of the computation.
  - Such attacks have nothing to do with the RSA cryptosystem itself, but rather the potentially insecure ways in which the algorithm is implemented by a computer.
  - Here is the rough idea behind a timing attack: most implementations on binary computers use "power chain" squaring to do modular exponentiation, since this requires much less memory than the direct successive squaring procedure we described.
  - When doing power-chain squaring, when there is a bit 1 in the exponent the computer must do a multiplication before squaring, and when there is a bit 0 the computer only needs to square.
  - If, for example, each multiplication and each squaring took 1 nanosecond, and the computer required 20 nanoseconds to decrypt a message whose decryption exponent has 16 bits, then there were 16 squarings (because of the size of the decryption exponent) and thus 4 multiplications.
  - This would tell us that the decryption exponent had exactly 4 ones in its binary representation.
  - Of course in practice, there is substantial variation in hardware and software computing speeds, but taking an average over a sufficiently large sample would give a fairly good estimate of the number of multiplications.
  - To extract additional information about the positioning of the ones in the binary representation, we can study the variation in the computing speed.
  - Explicitly: if a particular bit in the decryption exponent is equal to 0, then the amount of time it takes to do a single multiplication at that stage will be independent of the amount of time the remaining computation takes (since the computation does not require a multiplication there). But if the bit is equal to 1, the time it takes to do a single multiplication will not be independent (since that multiplication is part of the computation).
  - It is possible to determine whether two random variables are correlated or uncorrelated using basic statistical analysis (e.g., by plotting the values for many observations and computing a regression coefficient): by doing this, Eve can determine whether each successive bit in the decryption exponent was 0 or 1.
  - A variation on this attack is instead to measure the power consumption of the computer: it will use more power when it is doing a computation than when it is not, so one can glean information about what is occurring during the different steps of the computation by recording the processor's power usage.

- This kind of timing / power measurement attack is hard to guard against for reasons that are deeply ingrained in modern computer architecture.
    - One way to try to prevent timing attacks is to intentionally mask the speed of the calculations by making the computer evaluate a multiplication at every stage (even though the result is not actually used half the time). Then each step will take essentially the same amount of time no matter what the bits of the decryption exponent are, so there is no information leaked by measuring how long the computations take.
    - However, most software compilers are specifically designed to streamline program code when converting it to machine language, and they will do things like removing computations that are not used at any subsequent time. From a programming perspective this is an extremely helpful thing for a compiler to do, since any unused branches of code are simply a drag on the computer: removing them from the machine code will speed up the computation.
    - But of course, in the case where we intentionally add an unused calculation to prevent a timing attack, the compiler's attempts to be helpful will make the implementation vulnerable again!
    - Even when we can arrange matters so that the compiler does not add a vulnerability (which is for obvious reasons a difficult computational problem), the computer processor itself might create one.
    - Most (perhaps all) computer processors use a "branch predictor" to try to guess whether a particular branch will be taken in a program will be taken before it actually occurs. The goal is to improve the speed of the program by partially executing the branch that the computer predicts will be taken: if the prediction is correct, the processor has effectively saved time by evaluating a later part of the code already, but if the prediction is wrong, the processor will have to back up and use the correct branch.
    - By analyzing the performance of a branch predictor in an appropriately devious way, it is essentially possible to reproduce a timing attack even when the power-chain squaring algorithm is modified to have essentially the same computations regardless of the bits in the decryption exponent.
    - There are ways to code the successive squaring algorithm in such a way to avoid this kind of attack, but (at the least) it is not clear whether other implementations might not be vulnerable to other kinds of attacks.

2.5 Primality and Compositeness Testing

- In order to implement the public-key cryptosystems we have discussed, we need a way to generate large prime numbers.
- It might seem that finding large prime numbers would be very difficult, but it is actually relatively simple.
    - The Prime Number Theorem says that the approximate number of primes less than X is $\dfrac{X}{\ln X}$.
    - Therefore (roughly) the probability that a randomly-chosen large integer N is prime is about $\dfrac{1}{\ln N}$.
    - So for example, if we choose a random integer with 100 digits (in base 10), it has an approximately $\dfrac{1}{\ln(10^{100})} \approx 0.4\%$ chance of being prime.
    - However, this probability includes the possibility that we chose N to be even, or divisible by 3, or 5, or 7, and so forth. If we throw away integers divisible by primes less than 20, the probability of picking a prime randomly increases to about 2.5%.
    - We would like to develop efficient methods for testing whether a given large integer is prime: if we can, then it should be relatively straightforward to find large primes by choosing essentially random numbers until we get one that passes all the tests.
    - For example, if we take 200 randomly chosen 100-digit integers with no divisors less than 20 (it is easy to screen out integers with small divisors), the probability that at least one of them is actually prime is about $1 - 0.975^{200} \approx 99.4\%$, which is extremely high!
- Thus, the only remaining ingredient for generating big primes is a method for determining whether a given large integer n is prime or composite, without needing to factor it in the event that it is composite.

- There are various naive methods for doing this (such as attempting to divide n by each prime smaller than $\sqrt{n}$ to see if it divides evenly), but these are extremely impractical if n has hundreds of digits. Our goal is to describe several effective primality testing methods that are motivated by the results we have already proven.
- We will note that there are a wide variety of primality/compositeness tests and factorization algorithms of varying complexity, many of which we do not possess the background to discuss. Therefore, we will cover only a few of the most approachable techniques.

2.5.1 The Fermat Compositeness Test

- Fermat's Little Theorem says that if p is prime, then $a^p \equiv a \pmod{p}$ for every a. By taking the contrapositive, we obtain a sufficient condition for an integer to be composite.
- Test (Fermat Test): If a is an integer such that $a^n \not\equiv a \pmod{n}$, then n is composite.
  - Warning: The Fermat test is not a primality test: it is a compositeness test. There are only two possible outcomes of the test: either it shows that n is composite, or it yields no result. In particular, it can never be used to say that an integer is actually prime.
- Example: Apply the Fermat test to determine whether $n = 56\,011\,607$ is composite.
  - Using successive squaring, we can compute $2^{56\,011\,607} \equiv 48\,437\,830 \pmod{n}$: therefore, n is $\boxed{\text{composite}}$.
  - Note that the test does not tell us anything about the factorization of n: we know is that n is composite, but we don't have any information about the factorization.
  - In fact, n is the product of the two primes 6653 and 8419.
- It would be quite pleasant if the Fermat test were successful for every composite number. Unfortunately, this is not the case, as it is possible to make a bad choice for a.
- Example: Apply the Fermat test to decide whether $n = 341$ is composite.
  - Using successive squaring, we can compute that $2^{341} \equiv 2 \pmod{341}$, so the test provides no information with $a = 2$.
  - We instead try $a = 3$: successive squaring yields $3^{341} \equiv 168 \pmod{341}$, whence we see that 341 is composite.
- We might still hope that there will always be some a for which the Fermat test succeeds. Unfortunately, this is not the case either: there exist integers with the property that the Fermat test fails for *every* residue class a.
- Proposition: The Fermat test fails, for every a, to recognize $561 = 3 \cdot 11 \cdot 17$ as composite.
  - Proof: By the Chinese Remainder Theorem, it is enough to see that $a^{561} \equiv a$ modulo 3, 11, and 17 for every a.
  - Fermat's Little Theorem implies that $a^3 \equiv a \pmod{3}$. Multiplying both sides by a^2 gives $a^5 \equiv a^3 \equiv a$. Iterating, we see more generally that $a^{2k+1} \equiv a \pmod{3}$ for any k. In particular, taking $k = 280$ yields $a^{561} \equiv a \pmod{3}$.
  - In the same way we see that $a^{10k+1} \equiv a \pmod{11}$, so in particular taking $k = 56$ gives $a^{561} \equiv a \pmod{11}$. Similarly, we have $a^{16k+1} \equiv a \pmod{17}$, so by taking $k = 35$ we see $a^{561} \equiv a \pmod{17}$.
- Definition: An integer m for which the Fermat test fails modulo m for every a is called a Carmichael number (or pseudoprime).
  - It has been shown that there are infinitely many Carmichael numbers, but that they are significantly less common than primes (in an appropriate sense).
- In practice, Fermat's test is fairly effective when performed for enough values of a. Nonetheless, because of the existence of Carmichael numbers, it has a positive probability of failing to identify a number as composite.

2.5.2 The Miller-Rabin Compositeness Test

- We would like to improve on the Fermat test, since it has a positive probability of failing to yield any results. To begin, suppose p is prime and consider the solutions to $r^2 \equiv 1 \pmod{p}$.
  - This congruence is equivalent to $p|(r^2-1) = (r-1)(r+1)$, so since p is prime, the solutions are $r \equiv \pm 1 \pmod{p}$.
  - Now, if m is an odd integer that is prime, and a is any nonzero residue class, Fermat's Little Theorem implies that for $r = a^{(m-1)/2}$, we have $r^2 = a^{m-1} \equiv 1 \pmod{m}$.
  - By the above, we can conclude that $a^{(m-1)/2} \equiv \pm 1 \pmod{m}$.
  - Furthermore, in the event that $r \equiv 1 \pmod{m}$ and $m-1$ is divisible by 4, we see that for $s = a^{(m-1)/4}$, we have $s^2 = a^{(m-1)/2} = r \equiv 1 \pmod{m}$.
  - By the above logic applied again, we necessarily have $s = \pm 1 \pmod{m}$.
  - We can clearly repeat the above argument if $s \equiv 1 \pmod{m}$ and $m-1$ is divisible by 8, and so on and so forth.
- Test (Miller-Rabin Test): Let m be an odd integer and write $m - 1 = 2^k d$ for d odd. For a residue class a modulo m, calculate each of the values a^d, a^{2d}, a^{4d}, ... , $a^{2^k d}$ modulo m. If the last entry is $\not\equiv 1 \pmod{m}$ then m is composite. Furthermore, if any entry in the list is $\equiv 1 \pmod{m}$ and the previous entry is not $\equiv \pm 1 \pmod{m}$, then m is composite.
  - Proof: The first statement is simply the Fermat test. The second statement is an application of the contrapositive of the statement "$r^2 \equiv 1 \pmod{p}$ implies $r \equiv \pm 1 \pmod{p}$", proven above.
  - Warning: Like with the Fermat test, a single application of the Miller-Rabin test cannot prove affirmatively that a given number is prime: it can only show that m is composite.
- Example: Use the Miller-Rabin test to determine whether 561 is prime.
  - We will try $a = 2$ with $m = 561$. Observe $m - 1 = 2^4 \cdot 35$, so $k = 4$ and $d = 35$.
  - We need to compute a^{35}, a^{70}, a^{140}, a^{280}, a^{560} modulo 561.
  - We can do this rapidly by successive squaring: this yields the list 263, 166, 67, 1, 1.
  - Since the fourth term is 1 and the previous term is not $\equiv \pm 1 \pmod{561}$, we conclude that 561 is composite.
- Example: Use the Miller-Rabin test to determine whether 2047 is prime.
  - We try $a = 2$ with $m = 2047$. Observe $m - 1 = 2 \cdot 1023$, so $k = 1$ and $d = 1023$.
  - We need to compute a^{1023}, a^{2046} modulo 2047.
  - Successive squaring yields the values 1, 1: thus, the test is inconclusive for $a = 2$.
  - Next we try $a = 3$: successive squaring yields 1565, 1013. The last entry is not $\equiv 1$, so m is composite.
- The Miller-Rabin test is much stronger than the Fermat test, as can be seen from the example above: we showed earlier that 561 is a Carmichael number, meaning that the Fermat test will never show it is composite. On the other hand, the Miller-Rabin test succeeds in showing 561 is composite using only the residue $a = 2$.
- Definition: If m is odd and composite, and the Miller-Rabin test fails for a modulo m, we say that m is a strong pseudoprime to the base a.
  - It turns out that strong pseudoprimes are fairly uncommon. For example, it has been proven that, for any odd composite m, the Miller-Rabin test succeeds for at least 75% of the residue classes modulo m.
  - In particular, there are no "Carmichael numbers" for the Miller-Rabin test, where the test fails for every residue class.
  - Furthermore, if an integer m passes the Miller-Rabin test for more than $m/4$ residue classes modulo m, then m is prime. This is not a computationally effective way to show that an integer is prime, since it requires $m/4$ calculations (far more than trial division).

- However, it is believed that the bound can be substantially lowered from $m/4$. If we assume the Generalized Riemann Hypothesis (which is typically believed to be true), then it has been proven that testing the first $2(\log m)^2$ residues modulo m is sufficient.

- In practice, the Miller-Rabin test is used "probabilistically": we apply the test many times to the integer m, and if it passes sufficiently many times, we say m is probably prime.
  - Any given residue has at least a $3/4$ probability of showing that m is composite, so the probability that a composite integer m can pass the test k times with randomly-chosen residues a is at most $1/4^k$.
  - Taking $k = 100$ gives a probability negligable enough to use for all practical purposes (since the probability of having a hardware or programming error is certainly higher than $1/4^{100}$).
  - The Miller-Rabin test is very fast: a single application of the test to an integer m requires approximately $(\log m)^2$ calculations (to perform the required modular exponentiations), so even for integers with hundreds or thousands of digits, the method will quickly return a result that is correct with extremely high probability.
  - As noted above, if we assume the Generalized Riemann Hypothesis then the Miller-Rabin test would give a proof of primality in roughly $2(\log m)^4$ steps.

2.5.3 The Lucas Primality Criterion

- The tests we have examined so far test only for compositeness: they cannot actually prove a given integer is a prime. We will now give an example of a method that can prove a given integer is prime (though it suffers from some drawbacks).

- The basic idea is as follows: if $n = ab$ is composite, then $\varphi(n) < n - 1$ because there will be integers less than n sharing a common prime divisor with n. Conversely, if p is prime, $\varphi(p) = p - 1$.

- Therefore, if we can show the existence of an element modulo n whose order is $n - 1$, then n is necessarily prime.
  - We will postpone discussion of why such elements will exist modulo p when p is a prime, but they do exist and they are fairly common.

- This is the idea behind the Lucas primality criterion:

- Criterion (Lucas Criterion): Suppose $n > 1$ and that there exists a modulo n such that $a^{n-1} \equiv 1 \pmod{n}$ but such that $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ for any prime divisor q of n. Then n is prime.
  - Proof: The first statement says that a is necessarily a unit modulo n, so suppose the order of a is k.
  - By properties of order, since $a^{n-1} \equiv 1 \pmod{n}$ we see that k divides $n - 1$.
  - Now suppose that k is not equal to $n-1$. Then at least one prime q in the factorization of k must appear to a lower power than in the factorization of n: hence k divides $(n-1)/q$. So by properties of order, we would then have $a^{(n-1)/q} \equiv 1 \pmod{n}$, contrary to assumption.
  - Thus, $k = n - 1$, so a has order $n - 1$ modulo n. By Euler's theorem, the order of any element modulo n divides $\varphi(n)$, so we see $\varphi(n) \geq n - 1$. But this requires every positive integer less than n to be relatively prime to n, so since $n > 1$ we conclude that n is prime.

- Example: Use the Lucas criterion to show that $n = 20563$ is prime, using $a = 3$.
  - First, we compute $3^{20562} \equiv 1 \pmod{20563}$, so the first part of the criterion holds.
  - Now we factor $n - 1 = 20562 = 2 \cdot 3 \cdot 23 \cdot 149$.
  - Next, we find

$$\begin{array}{rcrcr} 3^{20562/2} & = & 3^{10281} & \equiv & 20562 \pmod{n} \\ 3^{20562/3} & = & 3^{6854} & \equiv & 3065 \pmod{n} \\ 3^{20562/23} & = & 3^{894} & \equiv & 15551 \pmod{n} \\ 3^{20562/149} & = & 3^{148} & \equiv & 19307 \pmod{n} \end{array}$$

    and since all of the results are not equal to 1 modulo n, we conclude that n is prime.

- The Lucas criterion can certainly work well, but there are serious computational issues with it.
- First, if n does happen to be prime, it does not give any indication on how to construct an appropriate value of a. (This is why we called the result a "criterion" rather than a test.)
    - As it turns out, there will usually be many such a that work, but their distribution will often be rather random.
    - This is not usually such an issue in practice because we can simply test a number of values of a (such as $a = 2, 3, 5, 6, 7, 10, ...$) until we find one that works.
- Second, and much more critically, we need to find the prime factorization of $p-1$ in order to prove that p is prime.
    - For some primes p, it is certainly the case that $p-1$ is easy to factor: for example, if $p = 2q+1$ or $4q+1$ or $6q+1$, ... where q is another prime. (Of course, we would then need to verify that q is itself actually prime in order to know we had the correct factorization....)
    - However, it can certainly happen that $p = 2q_1q_2 + 1$ where q_1 and q_2 are themselves large primes of roughly equal size. In this case to prove that p is prime would require factoring an integer whose size is roughly $p/2$, and (as we will discuss) factorization is usually quite difficult.
- Ultimately, there is really no way to avoid having to find a factorization in order to apply the Lucas test.
    - If, however, someone does expend the effort to find the factorization of $p-1$, it is straightforward to use the results to verify that the computations are correct after the fact.
- The "Pratt certificate" of a prime number consists of a valid a satisfying the Lucas criterion, together with the factorization of $p-1$.
    - Verifying the Pratt certificate is much faster than generating it, since the verification requires only computing the appropriate modular exponentiations.
    - One minor issue with the Pratt certificate is that it requires the prime factorization of $p-1$ to be correct: one could attempt to circumvent the procedure by giving a "factorization" of $p-1$ into terms that are not actually prime, and choosing a value of a which still makes the indicated modular exponentiations come out correctly.
    - To give a fully verifiable primality certificate, one would need also to include certificates showing that all of the prime divisors q_1, q_2, ... of $p-1$ are also themselves prime. This in turn would require certificates showing that each of the prime divisors of $q_1 - 1$, $q_2 - 1$ is prime, and so on and so forth, until all the divisors are sufficiently small that they can simply be compared to a list of known primes.
    - It can be shown that the total length of a certificate would be roughly on the order of $(\log p)^2$ digits (note that p itself has about $\log p$ digits), and that all of the computations can be checked in time roughly equal to $(\log p)^4$. Generating the certificate itself, of course, is likely to take substantially longer due to the factorizations that are required.

2.5.4 The AKS Primality Test

- What we still lack is a provably fast algorithm that determines whether a given integer m is prime. A starting point is the following observation:
- Proposition: If a is relatively prime to m and x is variable, then $(x+a)^m \equiv x^m + a$ (modulo m) holds (as a polynomial in x with coefficients modulo m) if and only if m is prime.
    - For example, if $a = 2$ and $m = 5$, the result says $(x+2)^5 = x^5 + 10x^4 + 40x^3 + 80x^2 + 80x + 32$ is equivalent (modulo 5) to the polynomial $x^5 + 2$, which is indeed the case.
    - Conversely, if $a = 1$ and $m = 4$, the result says $(x+1)^4 = x^4+4x^3+6x^2+4x+1$ should not be equivalent (modulo 4) to the polynomial $x^4 + 1$,which it is not.
    - Proof: Expanding out the power with the binomial theorem shows that $(x+a)^m \equiv x^m + a$ (modulo m) is equivalent to saying that $\binom{m}{k}$ is divisible by m for all $0 < k < m$, and $a^m \equiv a \pmod{m}$.

- If p is prime, then we can write $\binom{p}{k} = \dfrac{p!}{k! \cdot (p-k)!}$ and observe that the numerator is divisible by p but the denominator is not. Furthermore, Fermat's Little Theorem says $a^p \equiv a \pmod{p}$.
- Now suppose m is not prime. We claim that one of the binomial coefficients $\binom{m}{k}$ with $0 < k < m$ is not divisible by m. Explicitly, if $m = p^r d$ where p is prime and d is not divisible by p, then $\binom{m}{p} = \dfrac{m(m-1)\cdots(m-p+1)}{p!}$ is not divisible by p^r, since the only term in the numerator divisible by p is $n = p^r$, but there is a factor of p in the denominator.

- Although this result is a primality test, it is not especially useful, since computing the necessary binomial coefficients takes quite a long time. One way to speed up the computation is to take both sides modulo the polynomial $x^r - 1$ for some small r: in other words, to check whether the relation $(x+a)^m \equiv x^m + a \pmod{x^r - 1}$ holds, where coefficients are also considered modulo m.
  - The difficulty is that we may lose information by doing this. It turns out that if we choose r carefully, and verify the congruence for enough different values of a, we can prove that it necessarily holds in general.
- Test (Agrawal-Kayal-Saxena Test): Let $m > 1$ be an odd integer that is not of the form a^b for any $b > 1$.
  - Let r be the smallest value such that the order of r modulo m is greater than $(\log m)^2$.
    - This value can be computed simply by finding the orders of 2, 3, ... , until one of them has an order exceeding this bound.
    - If any of these integers divides m, then m is clearly composite.
    - If there is no such residue less than m, m is prime. (This can only happen for $m < 10^7$.)
    - There will always be such an r satisfying $r \leq 1 + (\log m)^5$.
  - For each a with $1 \leq a \leq \sqrt{\varphi(r)} \log m$, check whether $(x+a)^m \equiv x^m + a \pmod{x^r - 1, m}$.
    - If any of these congruences fails, m is composite.
    - Otherwise, m is prime.
- We will not prove the correctness of the AKS algorithm here.
  - However, we will note that the algorithm gives an affirmative declaration of whether m is prime or composite (unlike the previous tests we have discussed).
  - Furthermore, the runtime of this algorithm is a polynomial in $\log m$: it is much more efficient than (say) trial division, which has a runtime of roughly $\sqrt{m}$.
  - The version above runs in time roughly equal to $(\log m)^{12}$, and there have been subsequent modifications that lowered the time to approximately $(\log m)^6$. However, this is much slower than the "probabilistic" tests like the Miller-Rabin test, which is believed to run in time approximately $(\log m)^4$.

2.6 Factorization Algorithms

- All of the compositeness tests we have discussed so far are nonconstructive: they show that an integer m is composite without explicitly finding a divisor.
- We now turn to the question of actually factoring large integers. In general, factorization seems to be very much more computationally difficult than primality testing. We will describe some of the most common techniques.

2.6.1 The Fermat Factorization

- Suppose we wish to factor $n = pq$, where $p < q$ are odd numbers. (Usually they will be primes in the examples we consider, but this is not necessary.)
  - From the difference-of-squares identity, we write $n = \left(\dfrac{q+p}{2}\right)^2 - \left(\dfrac{q-p}{2}\right)^2$.

- If $q-p$ is small, then the term $\left(\frac{q-p}{2}\right)^2$ will be much smaller than $\left(\frac{q+p}{2}\right)^2$. This means that $\left(\frac{q+p}{2}\right)^2$ will be a perfect square that is only a small amount larger than n.
- We can then simply compute the first integer $k \geq \sqrt{n}$ and then check whether any of the integers k^2-n, $(k+1)^2-n$, $(k+2)^2-n$, ... ends up being a perfect square. If it is, the difference-of-squares identity yields a factorization.
- Note that we can compute square roots to very high accuracy, extremely rapidly, using logarithms, since $\sqrt{n} = e^{\ln(n)/2}$.

- The method above is called the Fermat factorization: we search for a and b such that $n = a^2-b^2 = (a+b)(a-b)$.
- Example: Factor $n = 1298639$.
  - We try looking for a Fermat factorization. We can compute numerically that $\sqrt{n} \approx 1139.58$.
  - We then compute $1140^2 - n = 961$, which is 31^2.
  - Hence we get the factorization $n = (1140-31)\cdot(1140+31) = 1109 \cdot 1171$. (Both of these integers turn out to be prime.)
- Example: Factor $n = 2789959$.
  - We try looking for a Fermat factorization. We compute $\sqrt{n} \approx 1670.32$.
  - We then compute $1671^2 - n = 2282$, which is not a square.
  - Next we try $1672^2 - n = 5625$, which is 75^2.
  - We obtain the factorization $n = (1672-75)\cdot(1672+75) = 1597 \cdot 1747$. (Both of these integers turn out to be prime.)
- Of course, the effectiveness of the Fermat factorization technique depends on how small $q-p$ is.
  - The number of iterations is more or less equal to $\sqrt{(q+p)/2} - \sqrt{n}$, which, if $q-p < n^{1/3}$, is bounded above by $2n^{1/6}$. Trial division takes roughly $\sqrt{n} = n^{1/2}$ iterations in the worst case, so the Fermat factorization is significantly faster than trial division if $q-p$ is small.
  - On the other hand, if q is roughly $2p$, then we would need to examine about p squares before we would find the factorization. In this case, $n \approx 2p^2$, so in terms of n we end up doing about $\sqrt{n}$ checks, which is the same as trial division.
- There are ways to modify the Fermat factorization that can overcome the issue of having q be larger than p.
  - For example, if it is known or suspected that $n = pq$ where q/p is close to 2, then applying the Fermat factorization technique to $8N$ will quickly return $8N = 4p \cdot 2q$, because $4p$ and $2q$ are very close together. (Multiplying by 8 is necessary because $2N = (p+q/2)^2 - (p-q/2)^2$ is not a difference of squares of integers.)
- Ultimately, however, the Fermat factorization is totally ineffective if p and q each have hundreds of digits: even if their first few digits are the same, searching for perfect squares will only be marginally faster than trial division.

2.6.2 Pollard's $p-1$ Algorithm

- One way to look for divisors of an integer n is as follows:
  - If a is a random residue modulo $n = pq$, then it is likely that the order of a modulo p is different from the order of a modulo q.
  - Suppose that the order of a modulo p is k, and is less than the order of a modulo q.
  - Then $a^k \equiv 1 \pmod{p}$ while $a^k \not\equiv 1 \pmod{q}$, meaning that $\gcd(a^k-1, n) = p$.

- Of course, if $n = pq$ is a product of two primes, then it is likely that the order of a modulo p and modulo q is quite large, and so a direct search for the order would be very inefficient.
- One way to speed the computation is to notice that we do not need to find the exact order of a: any multiple of it will suffice, as long as that multiple is not also divisible by the order of a modulo q.
- A reasonably effective option that is also easy to implement is to evaluate the values $a^{1!}$, $a^{2!}$, $a^{3!}$, $a^{4!}$, ... , $a^{B!}$ modulo n (for some bound B), since the jth term is simply the jth power of the previous term. This procedure is guaranteed to return a result congruent to 1 modulo p provided that the order of a divides $B!$.

- Algorithm (Pollard's $(p-1)$-Algorithm): Suppose n is composite. Choose a bound B and a residue a modulo n. Set $x_1 = a$, and for $2 \leq j \leq B$, define $x_j = x_{j-1}^j \pmod{n}$. Compute $\gcd(x_B - 1, n)$: if the gcd is between 1 and n then we have found a divisor of n. If the gcd is 1 or n, start over with a new residue a.
  - If the gcd is 1, it may be possible to extend the computation to obtain a divisor by increasing the bound B. It is easy to resume such an aborted computation: we can simply compute additional terms x_j past x_B using the same recursion.
  - If the gcd is n, it may have been the case that B was chosen too large (i.e., we carried the computation sufficiently far that $B!$ was a multiple of the order modulo p and modulo q). In such a case, we could repeat the computation with a smaller bound (e.g., $B/2$) to see if stopping the calculation earlier would catch one of the orders modulo p or modulo q but not the other.
  - The idea behind the algorithm is if p is a prime divisor of n such that $p-1$ has only small prime factors, then the order of any element modulo p will divide $B!$ where B is comparatively small. On the other hand, if the other prime factors q_i of n are such that $q_i - 1$ has a large prime factor, it is unlikely that a randomly chosen residue will have small order modulo q.
  - Thus, when we apply Pollard's $(p-1)$-algorithm to a composite integer $n = pq$ where $p-1$ has only small prime divisors, it is likely that the procedure will quickly find the factorization. (This is the reason for the algorithm's name.)
  - It is a nontrivial problem in analytic number theory to determine the optimal choice for the bound B. In practice, for integers with 20 digits or fewer, one usually chooses B on the order of 10^5 or so: such a computation can be done essentially instantaneously by a computer.

- Example: Use Pollard's $(p-1)$-algorithm with $a = 2$ to find a divisor of $n = 4913429$.
  - We start with $a = 2$, so that $x_1 = 2$. For emphasis we will compute $\gcd(x_j - 1, n)$ for each value of j until we find a gcd larger than 1:

Value	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
x_j	2	4	64	2036929	251970	3059995	1426887
$\gcd(x_j - 1, n)$	1	1	1	1	1	1	2521

  - After the 7th step, we obtain a nontrivial divisor 2521, giving the factorization $n = \boxed{2521 \cdot 1949}$.
  - Observe that $2521 - 1 = 2520 = 2^3 3^2 5^1 7^1$ has only small divisors, and indeed 2520 divides 7! (so we were guaranteed to obtain it by the 7th iteration of the procedure).
  - On the other hand, $1949 - 1 = 2^2 \cdot 481$ has a large prime divisor, so it would usually take $B = 481$ to find 1949 as a divisor.

- We will remark that the speed of Pollard's $(p-1)$-algorithm depends on the size of the largest prime divisor of $p-1$, which can vary quite substantially even for primes p of approximately the same size.
  - If p is an odd prime, then $p-1$ is clearly even, so the "worst case scenario" for Pollard's $(p-1)$-algorithm is to have $n = pq$ where $p = 2p_0 + 1$ and $q = 2q_0 + 1$ with p, q, p_0, q_0 all prime and where p and q are roughly equal. In such a case, we would require $B \approx p_0 \approx \frac{1}{2}\sqrt{n}$ in order to find the factorization (unless we are lucky with our choice of a).
  - As an aside, a prime p_0 such that $2p_0 + 1$ is also prime is called a "Sophie Germain prime". It is not known whether there are infinitely many such primes, although heuristically it is expected there should be infinitely many.

- It is also a rather involved analytic number theory problem to estimate the "expected" running time for the algorithm. In general, if we use a bound $B = n^{\alpha/2}$, then we would expect to have a probability roughly $\alpha^{-\alpha}$ of finding a factorization. When $\alpha = 1/2$ this says we would have about a 25% chance of obtaining a factorization if we take $B = n^{1/4}$.
- When generating an RSA modulus, it is very unlikely to choose a prime p such that $p-1$ only has small divisors by accident (at least, as long as one is choosing them randomly). Nevertheless, it is often a good idea to generate the modulus in such a way that each prime p has another large prime divisor of $p-1$.
- If, for example, one wants to generate a 250-digit prime p such that $p-1$ has a large prime divisor, one could first generate a 200-digit prime p_0 and a random 50-digit number k, and then test the numbers $p = (k+r)p_0 + 1$ for integers $r \geq 0$ until a prime is found. By construction, $p-1$ will then have the 200-digit prime p_0 as a divisor.

2.6.3 Pollard's ρ-Algorithm

- Another way we can try to generate divisors is in the following way: if we choose k integers modulo $n = pq$ at random, where $k > 2\sqrt{p}$, then it is likely that two of the k integers will be congruent modulo p, but different modulo n.
  - The reason for this is as follows: if we choose k integers modulo p, then the probability that they all lie in different residue classes is $\binom{p}{k}/p^k = \left(1 - \frac{1}{p}\right) \cdot \left(1 - \frac{2}{p}\right) \cdot \cdots \cdot \left(1 - \frac{k-1}{p}\right)$.
  - Taking the natural logarithm of this expression yields $\sum_{j=1}^{k-1} \log\left(1 - \frac{j-1}{p}\right) < -\sum_{j=1}^{k-1} \frac{j-1}{p} < -\frac{k^2}{2p}$, where we invoked the inequality $\log(1-x) < -x$, which is true for small positive x.
  - Thus, the probability that two of the k integers lie in the same residue class modulo p is at least $1 - e^{-k^2/(2p)}$: so in particular, if $k = 2\sqrt{p}$, the probability will be roughly $1 - e^{-2} \approx 0.86$.
  - A typical application of this result in basic probability is to set $p = 365$ and $k = 23$, which yields the often-surprising result that if 23 people are chosen at random, the probability that two or more of them share a common birthday exceeds 1/2. There are a number of applications of this idea in cryptography, which we will discuss later.
- Choosing k random residue classes and trying to see whether any of them are congruent modulo p is not much faster than trial division, since $\binom{k}{2} \approx \frac{1}{2}k^2$ comparisons would be needed in order to find two that are equal mod p, and if $k \approx 2\sqrt{p}$ then we do not obtain an improvement over trial division (which would also take p attempts).
- The observation, initially made by Pollard, is that we can speed up this process by generating the residue classes by iterating a polynomial map in the following way:
  - Let $p(x)$ be a polynomial with integer coefficients, and choose an arbitrary a, and consider the values a, $a_1 = p(a)$, $a_2 = p(p(a))$, $a_3 = p(p(p(a)))$, ..., taken modulo n, where in general we set $a_i = p(a_{i-1})$.
  - Absent any reason to expect otherwise, we would guess that the values $p(a_i)$ mod n will be essentially random, and so if we compare roughly $\sqrt{p}$ of them to each other, we are likely to find two that are congruent modulo p, if p is the smallest prime divisor of n.
  - The advantage lies in the fact that if $a_i \equiv a_j \pmod{p}$ for some $i < j$, then $a_{i+1} \equiv a_{j+1} \pmod{p}$, in turn implying $a_{i+2} \equiv a_{j+2} \pmod{p}$ and so forth. So the sequence becomes periodic with period $j - i$.
  - In particular, if $t \geq i$ is any multiple of the period $j - i$, then $a_t \equiv a_{2t} \pmod{p}$. This means we can detect the periodicity of this sequence by looking only at pairs of the form (a_t, a_{2t}), which is a vast improvement over having to search all pairs (a_i, a_j).
  - To obtain a divisor of n, we therefore want to compute $\gcd(a_{2t} - a_t, n)$.
- Collecting the above observations yields the following algorithm:

- Algorithm (Pollard's ρ-Algorithm): Suppose n is composite and set $p(x) = x^2 + 1$. Choose an arbitrary a, set $x_0 = y_0 = a$, and then define $x_i = p(x_{i-1}) \bmod n$ and $y_i = p(p(y_{i-1})) \bmod n$. Compute $\gcd(y_i - x_i, n)$ for each i until the gcd exceeds 1. If the gcd is n, repeat the procedure with a different u_0 or a different polynomial $p(x)$.
    - Note that $y_i = x_{2i}$, so we could just have computed the sequence x_i and $\gcd(x_{2i} - x_i, n)$ for each i.
    - However, we organize the algorithm in this way because it only requires a fixed amount of storage space: we only need to keep track of the most recent pair (x_i, y_i) to compute the next one.
    - If we kept track of the x_i only, we would need to use more memory: at the step where we compute $\gcd(x_{2i} - x_i, n)$, we would need to keep the values x_{i+1}, x_{i+2}, ... , x_{2i-1} for use later on. As i increases, so does the number of terms we need to keep track of.
    - There is also no particularly compelling reason to choose $p(x) = x^2 + 1$ aside from the fact that it seems to work well in practice. (Linear functions do not work, and more complicated polynomials would take longer to compute.)
- Example: Use Pollard's ρ-algorithm to find a divisor of $n = 1242301$.
    - We start with $u = 1$, so that $x_1 = 2$ and $y_1 = 5$, and successively keep track of the terms x_i and y_i modulo n.

Value	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
x_i	2	5	26	677	458330	743607	710748	894671	544825	121987
y_i	5	677	743607	894671	121987	636498	581703	1109702	1195126	635655
$\gcd(y_i - x_i, n)$	1	1	1	1	1	1	1	1	1	281

    - We see that at the 10th step, we obtain a nontrivial divisor 281, yielding the factorization $n = 281 \cdot 4421$.
- We will remark that Pollard's ρ-algorithm is not guaranteed to find a divisor on any given attempt: rather, it is only expected to do so according to some heuristics.
    - However, we would expect that it should find the smallest prime divisor of n after roughly $\sqrt{p}$ steps, and since $p \leq \sqrt{n}$, the expected runtime is on the order of $n^{1/4}$.
    - This is significantly faster than the $n^{1/2}$ we obtain from trial division, but it is still enormously large if n has hundreds of digits. On the other hand, it is at least more of a guarantee than from Pollard's $(p-1)$-algorithm, which

2.6.4 Sieving Methods

- We can improve on the Fermat factorization method by using the following fact from modular arithmetic, which already essentially came up when we discussed Rabin encryption:
- Proposition: If n is composite and $a^2 \equiv b^2 \pmod{n}$ with $a \neq \pm b \pmod{n}$, then $1 < \gcd(a + b, n) < n$.
    - Note that $a^2 \equiv b^2$ with $a \neq \pm b$ can only happen if n is composite, since there are only at most two solutions to any quadratic equation modulo a prime.
    - Proof: The given hypotheses imply $n | (a + b)(a - b)$ and that n does not divide either factor.
    - Hence the gcd of $a + b$ and n cannot be 1 (since then necessarily n would divide $a - b$), and it cannot be n (since then necessarily n would divide $a + b$). Hence the gcd must be strictly between 1 and n.
- The point of this proposition is that, since we can compute the gcd rapidly using the Euclidean algorithm, having such an a and b immediately yields a divisor of n.
    - Example: Since $10^2 \equiv 3^2 \pmod{91}$, we can find a divisor of 91 by computing $\gcd(10 + 3, 91) = 13$. Indeed, 13 is a divisor of 91, with quotient 7.
- The task is then to find a more efficient way to construct such an a and b than brute-force searching.

- Note that the goal of the Fermat factorization method is to construct a and b such that $n = a^2 - b^2$, which is a special case of what we are looking for.
- It is possible to find such a and b when using the Miller-Rabin test: if the test successfully shows m is composite by finding $c^{2j} \equiv 1 \pmod{m}$ with $c^j \not\equiv \pm 1 \pmod{m}$, then $(c^j)^2 \equiv 1 \pmod{m}$ while $c^j \neq \pm 1 \pmod{m}$: then $a = c^j$ and $b = 1$ satisfy the desired conditions.

- The method known as the quadratic sieve is one of the fastest procedures for factoring numbers less than 90 digits. Here is an outline of the procedure; the procedure we describe is properly called Dixon's factorization method, of which the quadratic sieve is an optimization:
  - Instead of trying to find a single value of a for which a^2 modulo n is a square, we instead compute a number of different values of a such that a^2 modulo n has all of its prime divisors in a small fixed set. Then, by taking products of some of these values, one can obtain a congruence of the form $a^2 \equiv b^2 \pmod{n}$ with $a \neq \pm b \pmod{n}$.
  - For example, modulo 2077, if we search for powers that have small prime divisors we will find $46^2 \equiv 3^1 13^1$ and $59^2 \equiv 2^2 3^3 13^1$. Multiplying them yields the equality $(46 \cdot 59)^2 \equiv (2^1 3^2 13^1)^2$, which is the same as $637^2 \equiv 234^2$. Then $\gcd(637 - 234, 2077) = 31$, which gives a divisor of 2077.
  - In general, this kind of search requires (i) finding many squares whose factorizations only involve small primes, and then (ii) finding a product of such factorizations that has a square value.
  - Goal (i) is in general rather difficult, and we will not describe in detail the methods used to do it.
  - Goal (ii), however, can be done efficiently with linear algebra: the idea is to find a nonzero linear dependence between the vectors of prime-factorization exponents, considered modulo 2.
  - For example, if we want to find a set of elements among 6, 10, 30, 150 whose product is a perfect square, we can find the prime factorizations $6 = 2^1 3^1 5^0$, $10 = 2^1 3^0 5^1$, $30 = 2^1 3^1 5^1$, $150 = 2^1 3^1 5^2$, we would take the four vectors of exponents $\langle 1, 1, 0 \rangle$, $\langle 1, 0, 1 \rangle$, $\langle 1, 1, 1 \rangle$, $\langle 1, 1, 2 \rangle$ and search for a linear combination of these vectors whose entries are all even.
  - In this case, we can see that $\langle 1, 1, 0 \rangle + \langle 1, 1, 2 \rangle = \langle 2, 2, 2 \rangle$, corresponding to the product $6 \cdot 150 = 900 = 30^2$.
  - There are simple linear-algebra procedures for doing this by row-reducing an appropriate matrix (which is quite computationally efficient).

- We will also remark that there is an improvement on the quadratic sieve called the general number field sieve that operates on the same kind of principle, except instead of doing its computations with the set of rational numbers $\mathbb{Q}$, it works using more general number fields such as $\mathbb{Q}(\sqrt{2}) = \{a + b\sqrt{2} : a, b \in \mathbb{Q}\}$. (Again, the details are too technical for us to treat properly here.)

- We will mention that the sieving algorithms run in speed that is asymptotically far faster than other methods for sufficiently large integers.
  - Specifically, the computational complexity for the general number field sieve to factor n is approximately $e^{1.95(\ln n)^{1/3}(\ln \ln n)^{2/3}}$, while the complexity for the quadratic sieve is approximately $e^{(\ln n)^{1/2}(\ln \ln n)^{1/2}}$.
  - As a comparison, Pollard's ρ-algorithm is expected to run in roughly $n^{1/4} = e^{0.25(\ln n)^1}$ steps.
  - For small n, the exponent $0.25(\ln n)^1$ will be less than $(\ln n)^{1/2}(\ln \ln n)^{1/2}$ because of the constant 0.25, but for large n, the expression with the smaller power of $\ln n$ will be smaller.
  - Similarly, for medium-sized n, $1.95(\ln n)^{1/3}(\ln \ln n)^{2/3}$ will be bigger than $(\ln n)^{1/2}(\ln \ln n)^{1/2}$ because of the constant 1.95, but for large n, $1.95(\ln n)^{1/3}(\ln \ln n)^{2/3}$ is smaller.
  - Thus, for small integers (roughly 40 base-10 digits or fewer), Pollard's ρ-algorithm will be fastest, while for integers up to about 90 digits the quadratic sieve is best. For larger integers, the general number field sieve is the best.
  - There is an algorithm known as Shor's algorithm that could run on a quantum computer that can factor an integer in approximately $(\ln n)^2(\ln \ln n)(\ln \ln \ln n)$ steps, vastly faster than the other algorithms we have mentioned.

Well, you're at the end of my handout. Hope it was helpful.