Lagrange's Theorem, Cryptography

Daniel H. Luecking
MASC

November 4, 2022

Application of groups: RSA cryptography

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

This is impractical today so a public-key method is used: If I want to *receive* secure messages, I publish an *encryption key* which others can use to encrypt messages, but I keep secret the *decryption key*.

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

This is impractical today so a public-key method is used: If I want to *receive* secure messages, I publish an *encryption key* which others can use to encrypt messages, but I keep secret the *decryption key*.

The Rivest-Shamir-Adleman (RSA) system amounts to the following (I am simplifying). A message m, being just 0s and 1s, is interpreted as a number in $\mathbb{Z}_n$ for some large n (or sequence of numbers if $m \geq n$).

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

This is impractical today so a public-key method is used: If I want to *receive* secure messages, I publish an *encryption key* which others can use to encrypt messages, but I keep secret the *decryption key*.

The Rivest-Shamir-Adleman (RSA) system amounts to the following (I am simplifying). A message m, being just 0s and 1s, is interpreted as a number in $\mathbb{Z}_n$ for some large n (or sequence of numbers if $m \geq n$).

I make the number n public as well as a certain number $e < n$. Then m is encrypted by computing $c = m^e \bmod n$ and c is what is sent to me.

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

This is impractical today so a public-key method is used: If I want to *receive* secure messages, I publish an *encryption key* which others can use to encrypt messages, but I keep secret the *decryption key*.

The Rivest-Shamir-Adleman (RSA) system amounts to the following (I am simplifying). A message m, being just 0s and 1s, is interpreted as a number in $\mathbb{Z}_n$ for some large n (or sequence of numbers if $m \geq n$).

I make the number n public as well as a certain number $e < n$. Then m is encrypted by computing $c = m^e \bmod n$ and c is what is sent to me.

I keep another number $d < n$ secret. It has the property that $c^d \bmod n = m^{ed} \bmod n = m$ and I then have the message m.

Application of groups: RSA cryptography

In the distant past (say the 1970's) encrypted communication required the two parties to meet and determine the encryption codes.

This is impractical today so a public-key method is used: If I want to *receive* secure messages, I publish an *encryption key* which others can use to encrypt messages, but I keep secret the *decryption key*.

The Rivest-Shamir-Adleman (RSA) system amounts to the following (I am simplifying). A message m, being just 0s and 1s, is interpreted as a number in $\mathbb{Z}_n$ for some large n (or sequence of numbers if $m \geq n$).

I make the number n public as well as a certain number $e < n$. Then m is encrypted by computing $c = m^e \bmod n$ and c is what is sent to me.

I keep another number $d < n$ secret. It has the property that $c^d \bmod n = m^{ed} \bmod n = m$ and I then have the message m.

The mathematics behind this is the subject of this lecture. It is possible to deduce d from n and e, but only if n can be factored. This is a well-known difficult problem.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

There is a method that relies on the observation that raising to the 2^k power requires only k multiplications: m^{2^k} is obtained by squaring m to get m^2 then squaring that to get m^{2^2} then squaring that to get m^{2^3}, etc.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

There is a method that relies on the observation that raising to the 2^k power requires only k multiplications: m^{2^k} is obtained by squaring m to get m^2 then squaring that to get m^{2^2} then squaring that to get m^{2^3}, etc.

Since e is a sum of powers of 2 (at most as many as the number of bits in e), an algorithm can be use that produces m^e in about twice as many steps as there are bits in e.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

There is a method that relies on the observation that raising to the 2^k power requires only k multiplications: m^{2^k} is obtained by squaring m to get m^2 then squaring that to get m^{2^2} then squaring that to get m^{2^3}, etc.

Since e is a sum of powers of 2 (at most as many as the number of bits in e), an algorithm can be use that produces m^e in about twice as many steps as there are bits in e. Thus one needs only dozens of multiplications. Also, since we work in $\mathbb{Z}_n$ the numbers never get larger than n.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

There is a method that relies on the observation that raising to the 2^k power requires only k multiplications: m^{2^k} is obtained by squaring m to get m^2 then squaring that to get m^{2^2} then squaring that to get m^{2^3}, etc.

Since e is a sum of powers of 2 (at most as many as the number of bits in e), an algorithm can be use that produces m^e in about twice as many steps as there are bits in e. Thus one needs only dozens of multiplications. Also, since we work in $\mathbb{Z}_n$ the numbers never get larger than n.

The existence of the e and d that satisfy $m^{ed} \bmod n = m$ requires n to have a special form: it must be a product of distinct primes.

Raising to powers

While modern computers can do millions of multiplications per second, security requires the chosen n to have thousands of bits and e and d are required to have dozens of bits. We're talking numbers larger than a million million million or so. So one needs an efficient method to take powers.

There is a method that relies on the observation that raising to the 2^k power requires only k multiplications: m^{2^k} is obtained by squaring m to get m^2 then squaring that to get m^{2^2} then squaring that to get m^{2^3}, etc.

Since e is a sum of powers of 2 (at most as many as the number of bits in e), an algorithm can be use that produces m^e in about twice as many steps as there are bits in e. Thus one needs only dozens of multiplications. Also, since we work in $\mathbb{Z}_n$ the numbers never get larger than n.

The existence of the e and d that satisfy $m^{ed} \bmod n = m$ requires n to have a special form: it must be a product of distinct primes. These primes must be large, for security, so the system specifies $n = pq$ for two large primes p and q.

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity.

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity. If we consider this in $u(\mathbb{Z}_n)$ which has size $\phi(n)$ we conclude that if a is a unit in $\mathbb{Z}_n$ we have $a^{\phi(n)} \bmod n = 1$.

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity. If we consider this in $u(\mathbb{Z}_n)$ which has size $\phi(n)$ we conclude that if a is a unit in $\mathbb{Z}_n$ we have $a^{\phi(n)} \bmod n = 1$.

If p is a prime, then every element of $\mathbb{Z}_p$ is a unit except 0 and $\phi(p) = p - 1$ so $a^{p-1} \bmod p = 1$ for all $a \neq 0$ in $\mathbb{Z}_p$.

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity. If we consider this in $u(\mathbb{Z}_n)$ which has size $\phi(n)$ we conclude that if a is a unit in $\mathbb{Z}_n$ we have $a^{\phi(n)} \bmod n = 1$.

If p is a prime, then every element of $\mathbb{Z}_p$ is a unit except 0 and $\phi(p) = p - 1$ so $a^{p-1} \bmod p = 1$ for all $a \neq 0$ in $\mathbb{Z}_p$. If we raise that to a power we also get $a^{k(p-1)} = 1$. If we multiply that by a we get

$$a^{k\phi(p)+1} \bmod p = a \text{ for any integer } k.$$

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity. If we consider this in $u(\mathbb{Z}_n)$ which has size $\phi(n)$ we conclude that if a is a unit in $\mathbb{Z}_n$ we have $a^{\phi(n)} \bmod n = 1$.

If p is a prime, then every element of $\mathbb{Z}_p$ is a unit except 0 and $\phi(p) = p - 1$ so $a^{p-1} \bmod p = 1$ for all $a \neq 0$ in $\mathbb{Z}_p$. If we raise that to a power we also get $a^{k(p-1)} = 1$. If we multiply that by a we get

$$a^{k\phi(p)+1} \bmod p = a \text{ for any integer } k.$$

This continues to hold when $a = 0$ so it is true for all a in $\mathbb{Z}_p$.

If we replace p by any n, the above may not be true if a is not a unit and not 0, but if n is a product of distinct primes, for example if $n = pq$, where p and q are different primes, then it is true.

The group of units

Recall that Lagrange's theorem implies that if G is a group and a is in G then $a^{|G|}$ equals the group's identity. If we consider this in $u(\mathbb{Z}_n)$ which has size $\phi(n)$ we conclude that if a is a unit in $\mathbb{Z}_n$ we have $a^{\phi(n)} \bmod n = 1$.

If p is a prime, then every element of $\mathbb{Z}_p$ is a unit except 0 and $\phi(p) = p - 1$ so $a^{p-1} \bmod p = 1$ for all $a \neq 0$ in $\mathbb{Z}_p$. If we raise that to a power we also get $a^{k(p-1)} = 1$. If we multiply that by a we get

$$a^{k\phi(p)+1} \bmod p = a \text{ for any integer } k.$$

This continues to hold when $a = 0$ so it is true for all a in $\mathbb{Z}_p$.

If we replace p by any n, the above may not be true if a is not a unit and not 0, but if n is a product of distinct primes, for example if $n = pq$, where p and q are different primes, then it is true. That is, for any a in $\mathbb{Z}_n$ with $n = pq$

$$a^{k\phi(n)+1} \bmod n = a, \text{ for any integer } k.$$

The Chinese Remainder Theorem

The reason for the last equation in $\mathbb{Z}_{pq}$ is the Chinese Remainder Theorem:

Theorem

If $n = pq$ where $\gcd(p, q) = 1$ then there is a one-to-one homomorphism between $\mathbb{Z}_n$ and $\mathbb{Z}_p \times \mathbb{Z}_q$.

The Chinese Remainder Theorem

The reason for the last equation in $\mathbb{Z}_{pq}$ is the Chinese Remainder Theorem:

Theorem

If $n = pq$ where $\gcd(p, q) = 1$ then there is a one-to-one homomorphism between $\mathbb{Z}_n$ and $\mathbb{Z}_p \times \mathbb{Z}_q$.

A homomorphism of rings is a function that preserves both addition and scalar multiplication.

The Chinese Remainder Theorem

The reason for the last equation in $\mathbb{Z}_{pq}$ is the Chinese Remainder Theorem:

Theorem

If $n = pq$ where $\gcd(p, q) = 1$ then there is a one-to-one homomorphism between $\mathbb{Z}_n$ and $\mathbb{Z}_p \times \mathbb{Z}_q$.

A homomorphism of rings is a function that preserves both addition and scalar multiplication. This means that one can do computations in $\mathbb{Z}_n$ by transferring elements m in $\mathbb{Z}_n$ to elements (r, s) in $\mathbb{Z}_p \times \mathbb{Z}_q$, doing the computations there, then returning to $\mathbb{Z}_n$.

The Chinese Remainder Theorem

The reason for the last equation in $\mathbb{Z}_{pq}$ is the Chinese Remainder Theorem:

Theorem

If $n = pq$ where $\gcd(p, q) = 1$ then there is a one-to-one homomorphism between $\mathbb{Z}_n$ and $\mathbb{Z}_p \times \mathbb{Z}_q$.

A homomorphism of rings is a function that preserves both addition and scalar multiplication. This means that one can do computations in $\mathbb{Z}_n$ by transferring elements m in $\mathbb{Z}_n$ to elements (r, s) in $\mathbb{Z}_p \times \mathbb{Z}_q$, doing the computations there, then returning to $\mathbb{Z}_n$. The function that turns m into (r, s) is easy: $r = m \bmod p$ and $s = m \bmod q$.

The Chinese Remainder Theorem

The reason for the last equation in $\mathbb{Z}_{pq}$ is the Chinese Remainder Theorem:

Theorem

If $n = pq$ where $\gcd(p,q) = 1$ then there is a one-to-one homomorphism between $\mathbb{Z}_n$ and $\mathbb{Z}_p \times \mathbb{Z}_q$.

A homomorphism of rings is a function that preserves both addition and scalar multiplication. This means that one can do computations in $\mathbb{Z}_n$ by transferring elements m in $\mathbb{Z}_n$ to elements (r,s) in $\mathbb{Z}_p \times \mathbb{Z}_q$, doing the computations there, then returning to $\mathbb{Z}_n$. The function that turns m into (r,s) is easy: $r = m \bmod p$ and $s = m \bmod q$.

The return function is almost as simple if we use the fact that $1 = ap + bq$ for integers a and b. Then we can return to $\mathbb{Z}_n$ by
$(r,s) \mapsto m = (bqr + aps) \bmod n$

Now for any positive integers j and l we have $r^{j\phi(p)+1} = r$ in $\mathbb{Z}_p$ and $s^{l\phi(q)+1} = s$ for all s in $\mathbb{Z}_q$. It follows that $(r, s)^{k\phi(q)\phi(p)+1} = (r, s)$ in $\mathbb{Z}_p \times \mathbb{Z}_q$ for any positive integer l.

Now for any positive integers j and l we have $r^{j\phi(p)+1} = r$ in $\mathbb{Z}_p$ and $s^{l\phi(q)+1} = s$ for all s in $\mathbb{Z}_q$. It follows that $(r, s)^{k\phi(q)\phi(p)+1} = (r, s)$ in $\mathbb{Z}_p \times \mathbb{Z}_q$ for any positive integer l. Transferring these computations back to $\mathbb{Z}_n$, and using the fact that $\phi(n) = \phi(p)\phi(q)$ we get

$$m^{k\phi(n)+1} = m \quad \text{for any positive integer } \; k.$$

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$.

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$. This is equivalent to $ed \bmod \phi(n) = 1$ or $e \cdot d = 1$ in the ring $\mathbb{Z}_{\phi(n)}$. As a consequence all we need to do is pick $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and set d equal to its inverse in $\mathbb{Z}_{\phi(n)}$.

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$. This is equivalent to $ed \bmod \phi(n) = 1$ or $e \cdot d = 1$ in the ring $\mathbb{Z}_{\phi(n)}$. As a consequence all we need to do is pick $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and set d equal to its inverse in $\mathbb{Z}_{\phi(n)}$.

Since p and q are odd, $\phi(n) = (p-1)(q-1)$ is even so e must be odd.

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$. This is equivalent to $ed \bmod \phi(n) = 1$ or $e \cdot d = 1$ in the ring $\mathbb{Z}_{\phi(n)}$. As a consequence all we need to do is pick $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and set d equal to its inverse in $\mathbb{Z}_{\phi(n)}$.

Since p and q are odd, $\phi(n) = (p-1)(q-1)$ is even so e must be odd. In practice a good fraction of the odd numbers less that $\phi(n)$ have $\gcd(e, \phi(n)) = 1$, so e can be found eventually just by choosing odd numbers $k < \phi(n)$ at random and testing the value of $\gcd(k, \phi(n))$.

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$. This is equivalent to $ed \bmod \phi(n) = 1$ or $e \cdot d = 1$ in the ring $\mathbb{Z}_{\phi(n)}$. As a consequence all we need to do is pick $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and set d equal to its inverse in $\mathbb{Z}_{\phi(n)}$.

Since p and q are odd, $\phi(n) = (p-1)(q-1)$ is even so e must be odd. In practice a good fraction of the odd numbers less that $\phi(n)$ have $\gcd(e, \phi(n)) = 1$, so e can be found eventually just by choosing odd numbers $k < \phi(n)$ at random and testing the value of $\gcd(k, \phi(n))$.

[The textbook incorrectly computes the probability of getting a suitable e in one try. The argument there correctly produces the odds of getting a unit in $\mathbb{Z}_n$, but e has to be a unit in $\mathbb{Z}_{\phi(n)}$.]

Where do e and d come from?

From the previous slide we see that we get $m^{k\phi(n)+1} = m$ for any message $m \in \mathbb{Z}_n$. In order to translate this into $m^{ed} = m$ we only need e and d to satisfy $ed = k\phi(n) + 1$. This is equivalent to $ed \bmod \phi(n) = 1$ or $e \cdot d = 1$ in the ring $\mathbb{Z}_{\phi(n)}$. As a consequence all we need to do is pick $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and set d equal to its inverse in $\mathbb{Z}_{\phi(n)}$.

Since p and q are odd, $\phi(n) = (p-1)(q-1)$ is even so e must be odd. In practice a good fraction of the odd numbers less that $\phi(n)$ have $\gcd(e, \phi(n)) = 1$, so e can be found eventually just by choosing odd numbers $k < \phi(n)$ at random and testing the value of $\gcd(k, \phi(n))$.

[The textbook incorrectly computes the probability of getting a suitable e in one try. The argument there correctly produces the odds of getting a unit in $\mathbb{Z}_n$, but e has to be a unit in $\mathbb{Z}_{\phi(n)}$.]

There are ways to get e without random choosing. For example, picking e to be the larger of p or q always works, but that would be an insecure choice.

Where do p and q come from

It turns out there are ways to test whether a number is prime without trying to eliminate all possible factorizations.

Where do p and q come from

It turns out there are ways to test whether a number is prime without trying to eliminate all possible factorizations. Since obtaining p and q has to be done only once (e.g., when you install your browser) it doesn't matter too much if it takes a little time.

Where do p and q come from

It turns out there are ways to test whether a number is prime without trying to eliminate all possible factorizations. Since obtaining p and q has to be done only once (e.g., when you install your browser) it doesn't matter too much if it takes a little time. So typically they are found by randomly selecting numbers in an appropriate range of sizes and testing them for primality until two primes are found.

Where do p and q come from

It turns out there are ways to test whether a number is prime without trying to eliminate all possible factorizations. Since obtaining p and q has to be done only once (e.g., when you install your browser) it doesn't matter too much if it takes a little time. So typically they are found by randomly selecting numbers in an appropriate range of sizes and testing them for primality until two primes are found.

There are some other concerns besides size and primality. The two primes cannot be too close to each other. They also shouldn't match what others have chosen.

A more thorough coverage of the RSA system can be found on Wikipedia:
`https://en.wikipedia.org/wiki/RSA_(cryptosystem)`
Coverage of primality testing can be found at
`https://en.wikipedia.org/wiki/Primality_test`

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e.

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e. This can be done easily if the factorization of n can be found. It is thought (though not proven) that determining d is computationally as hard as factoring n. How hard could that be?

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e. This can be done easily if the factorization of n can be found. It is thought (though not proven) that determining d is computationally as hard as factoring n. How hard could that be?

Using state-of-the-art factoring algorithms, the record for finding p and q is a 795 bit number n. It was done in 2019 and took 900 years of CPU time (distributed over thousands of computers that donated CPU time).

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e. This can be done easily if the factorization of n can be found. It is thought (though not proven) that determining d is computationally as hard as factoring n. How hard could that be?

Using state-of-the-art factoring algorithms, the record for finding p and q is a 795 bit number n. It was done in 2019 and took 900 years of CPU time (distributed over thousands of computers that donated CPU time). Numbers n up to 512 bits can be routinely factored in a few weeks on common hardware. Most numbers n used these days are longer than 1024 bits and recommendations for the future range from 2048 to 4096 bits.

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e. This can be done easily if the factorization of n can be found. It is thought (though not proven) that determining d is computationally as hard as factoring n. How hard could that be?

Using state-of-the-art factoring algorithms, the record for finding p and q is a 795 bit number n. It was done in 2019 and took 900 years of CPU time (distributed over thousands of computers that donated CPU time). Numbers n up to 512 bits can be routinely factored in a few weeks on common hardware. Most numbers n used these days are longer than 1024 bits and recommendations for the future range from 2048 to 4096 bits.

A *quantum computer*, should one ever be developed for practical use, could theoretically quickly factor almost any size number.

Is this cryptography system unbreakable?

No one knows how secure the RSA system is. Its security lies in being unable to determine d from n and e. This can be done easily if the factorization of n can be found. It is thought (though not proven) that determining d is computationally as hard as factoring n. How hard could that be?

Using state-of-the-art factoring algorithms, the record for finding p and q is a 795 bit number n. It was done in 2019 and took 900 years of CPU time (distributed over thousands of computers that donated CPU time). Numbers n up to 512 bits can be routinely factored in a few weeks on common hardware. Most numbers n used these days are longer than 1024 bits and recommendations for the future range from 2048 to 4096 bits.

A *quantum computer*, should one ever be developed for practical use, could theoretically quickly factor almost any size number.

There are attacks on RSA that involve the special nature of some messages. Other parts of the RSA system (e.g. scrambling m) are designed to avoid such attacks.