RSA Cryptosystem and Prerequisites

Liam Donovan

May 2023

Overview

Overview

Overview

Overview

Overview

Overview

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n
- Modular Multiplicative Inverses

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n
- Modular Multiplicative Inverses
- Semiprimes

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n
- Modular Multiplicative Inverses
- Semiprimes
- Euler's Totient Function

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n
- Modular Multiplicative Inverses
- Semiprimes
- Euler's Totient Function

Background Information: Modular Arithmetic

For most of what we do in cryptography, we will use modular arithmetic, that is: instead of the mathematics of the quotient in a division of two integers, we will look at the remainder. For this we will need a few results.

- Congruence and Equivalence Classes Mod n
- Modular Multiplicative Inverses
- Semiprimes
- Euler's Totient Function

Division and Divisibility

Consider we have 10 objects, and we would like to organize them into groups of 2.

Division and Divisibility

Consider we have 10 objects, and we would like to organize them into groups of 2. That is: $a = 10$, $b = 2$. Well, we can make 5 groups of 2, with nothing left over, which means:

Division and Divisibility

Consider we have 10 objects, and we would like to organize them into groups of 2. That is: $a = 10$, $b = 2$. Well, we can make 5 groups of 2, with nothing left over, which means:

Example 1.0.1

$$10 = 2(5) + 0$$

In this case, we can divide a from b, with no remainder, meaning that we can write a as a *multiple* of b or that b is a *factor* of a. So, in this case 10 is a multiple of 5, and 5 is a factor of 10.

Division and Divisibility

Consider we have 10 objects, and we would like to organize them into groups of 2. That is: $a = 10$, $b = 2$. Well, we can make 5 groups of 2, with nothing left over, which means:

Example 1.0.1

$$10 = 2(5) + 0$$

In this case, we can divide a from b, with no remainder, meaning that we can write a as a *multiple* of b or that b is a *factor* of a. So, in this case 10 is a multiple of 5, and 5 is a factor of 10. If we can write a as a group of b, with no remainder, we say "b divides a", in that b/a is an integer. We denote this $b \mid a$.

Another Example

What if we wanted to make groups of 3? Then, we could make 3 groups of 3, with 1 left:

Another Example

What if we wanted to make groups of 3? Then, we could make 3 groups of 3, with 1 left:

Example 1.0.2

$$10 = 3(3) + 1$$

Another Example

What if we wanted to make groups of 3? Then, we could make 3 groups of 3, with 1 left:

Example 1.0.2

$$10 = 3(3) + 1$$

In this case, we cannot write *a* as a group of *b*, without a remainder, so a/b will not be an integer.
It should be clear that if we make as many groups of *b* as we possible can, that *q* and *r* will be unique.

Modular Congruence

If two integers, a, c have the same remainder, when divided by b, we say they are congruent mod b.[Sma03] We denote this by

$$a \equiv b \pmod{c}$$

Another definition of this relies on the fact that if we subtract the division algorithm of a and b, then the r terms will cancel, meaning that $b \mid (a - b)$.

A Familiar Example

A good example of modular arithmetic is a 12 hour clock. When we tell time, we organize hours into groups of 12. For example, consider it's 12 o'clock, what time will it be in 12 hours?

A Familiar Example

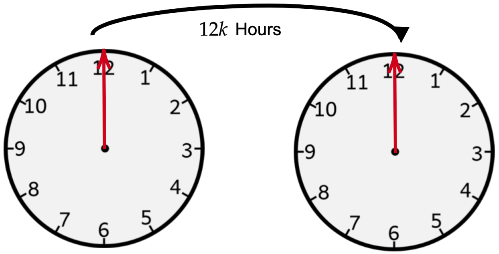
A good example of modular arithmetic is a 12 hour clock. When we tell time, we organize hours into groups of 12. For example, consider it's 12 o'clock, what time will it be in 12 hours? How about 24?

A Familiar Example

A good example of modular arithmetic is a 12 hour clock. When we tell time, we organize hours into groups of 12. For example, consider it's 12 o'clock, what time will it be in 12 hours? How about 24? 120?

A Familiar Example

A good example of modular arithmetic is a 12 hour clock. When we tell time, we organize hours into groups of 12. For example, consider it's 12 o'clock, what time will it be in 12 hours? How about 24? 120? As long as we pick some hour that is a multiple of 12, it will always land on 12. Let k be some integer.



A Familiar Example cont.

Likewise, consider after $3 + 12k$ hours, we would land at 3 and after $4 + 12k$ we would land at 4. Since it takes 12 hours to go back around to 3 after the first pass.

A Familiar Example cont.

Likewise, consider after $3 + 12k$ hours, we would land at 3 and after $4 + 12k$ we would land at 4. Since it takes 12 hours to go back around to 3 after the first pass.

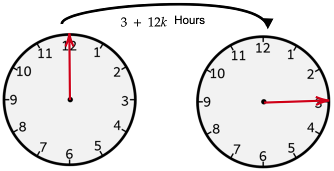


Figure: After $3 + 12k$ hours, the clock is at 3.

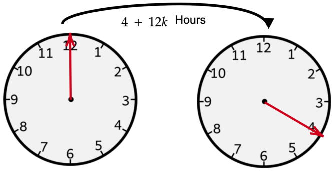


Figure: After $4k + 12k$ hours, the clock is at 4.

We are going to use modular arithmetic to encrypt our data.

The Modular Multiplicative Inverse

Since we encrypted our data using modular arithmetic, we need to find a decryption key, which "undoes" the encryption.

The Modular Multiplicative Inverse

Since we encrypted our data using modular arithmetic, we need to find a decryption key, which "undoes" the encryption. This is called the *modular multiplicative inverse*, $x \in \mathbb{Z}$, which satisfies:

The Modular Multiplicative Inverse

Since we encrypted our data using modular arithmetic, we need to find a decryption key, which "undoes" the encryption. This is called the *modular multiplicative inverse*, $x \in \mathbb{Z}$, which satisfies:

$$ax \equiv 1 \pmod{c}$$

x exists if and only if $gcd(a, c) = 1$. Where *gcd* refers to the greatest common factor, the largest integer which divides both a and c.

The Modular Multiplicative Inverse

Since we encrypted our data using modular arithmetic, we need to find a decryption key, which "undoes" the encryption. This is called the *modular multiplicative inverse*, $x \in \mathbb{Z}$, which satisfies:

$$ax \equiv 1 \pmod{c}$$

x exists if and only if $gcd(a, c) = 1$. Where gcd refers to the greatest common factor, the largest integer which divides both a and c. In this case, we say a, c are *coprime*. That is: they share no common factors, except 1, which is a factor of all integers.
In practice, a computer performs the extended Euclidean algorithm, which can compute x.

Why Modular Arithmetic?

- Forms a finite group

Why Modular Arithmetic?

- Forms a finite group (a set which is predictable).

Why Modular Arithmetic?

- Forms a finite group (a set which is predictable). Computers like finite groups.
- Further strengthens the encryption

Why Modular Arithmetic?

- Forms a finite group (a set which is predictable). Computers like finite groups.
- Further strengthens the encryption If the attacker does not know which group we are working with everything will seem random.

Semiprimes

The product of two primes is *hard to factor*, if it is large enough.

Semiprimes

The product of two primes is *hard to factor*, if it is large enough. We call these numbers *semiprime*.

Semiprimes

The product of two primes is *hard to factor*, if it is large enough. We call these numbers *semiprime*.

Example 3.0.1

21 is semiprime because 7 and 3 are prime and $7 \cdot 3 = 21$

Semiprimes

The product of two primes is *hard to factor*, if it is large enough. We call these numbers *semiprime*.

Example 3.0.1

21 is semiprime because 7 and 3 are prime and $7 \cdot 3 = 21$

Why do we care?

- Computers (at the moment) can only factor a number in exponential time.

Why do we care?

- Computers (at the moment) can only factor a number in exponential time. The best algorithm will still take awhile to factor a large semiprime.
  - So, we use large primes to create a large semiprime to encrypt our data.

Why do we care?

- Computers (at the moment) can only factor a number in exponential time. The best algorithm will still take awhile to factor a large semiprime.
  - So, we use large primes to create a large semiprime to encrypt our data. We can hide the prime factors, while publicizing the semiprime.

Euler's Totient Function

- Euler's totient function, which we denote $\varphi(n)$ is just the amount of integers less than or equal to n which are coprime to n.

Euler's Totient Function

- Euler's totient function, which we denote $\varphi(n)$ is just the amount of integers less than or equal to n which are coprime to n.
- So, if we had a prime number n. Then, everything, except n is coprime to it, since it's only factors are itself and 1.

Euler's Totient Function

- Euler's totient function, which we denote $\varphi(n)$ is just the amount of integers less than or equal to n which are coprime to n.
- So, if we had a prime number n. Then, everything, except n is coprime to it, since it's only factors are itself and 1.
- A property of Euler's totient function is that is is multiplicative

Euler's Totient Function

- Euler's totient function, which we denote $\varphi(n)$ is just the amount of integers less than or equal to n which are coprime to n.
- So, if we had a prime number n. Then, everything, except n is coprime to it, since it's only factors are itself and 1.
- A property of Euler's totient function is that is is multiplicative
  - This means that if a, b are coprime, $\varphi(ab) = \varphi(a)\varphi(b)$.
  - So, if we have two primes, p, q, $\varphi(pq) = \varphi(p)\varphi(q)$

Euler's Totient Function

- Euler's totient function, which we denote $\varphi(n)$ is just the amount of integers less than or equal to n which are coprime to n.
- So, if we had a prime number n. Then, everything, except n is coprime to it, since it's only factors are itself and 1.
- A property of Euler's totient function is that is is multiplicative
  - This means that if a, b are coprime, $\varphi(ab) = \varphi(a)\varphi(b)$.
  - So, if we have two primes, p, q, $\varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$
  - I can use the semiprime pq to create another value that the attacker will not know, $\varphi(pq)$.

Setting It Up

We now will use our knowledge of modular arithmetic to create a system that securely encrypts data.[RSA78]

Setting It Up

We now will use our knowledge of modular arithmetic to create a system that securely encrypts data.[RSA78] Let red variables refer to private keys (that only the recipient should know) and blue variables refer to public keys, which anyone can know.

Setting It Up

red: private

blue: public

1. Make a semiprime, using large primes

Setting It Up

red: private
blue: public

1. Make a semiprime, using large primes
   - let $n = pq$
   - This uses the idea that even if someone knows n, they have a hard time finding pq.
2. Find $\varphi(n) = (p-1)(q-1)$

Setting It Up

red: private
blue: public

1. Make a semiprime, using large primes
   - let $n = pq$
   - This uses the idea that even if someone knows n, they have a hard time finding pq.
2. Find $\varphi(n) = (p-1)(q-1)$
   - This will just make encryption harder to break.

Setting It Up

red: private
blue: public

1. Make a semiprime, using large primes
   - let $n = pq$
   - This uses the idea that even if someone knows n, they have a hard time finding pq.
2. Find $\varphi(n) = (p-1)(q-1)$
   - This will just make encryption harder to break.
3. Choose an integer e, which is less than $\varphi(n)$, but greater than 2.

Setting It Up

red: private
blue: public

1. Make a semiprime, using large primes
   - let $n = pq$
   - This uses the idea that even if someone knows n, they have a hard time finding pq.
2. Find $\varphi(n) = (p-1)(q-1)$
   - This will just make encryption harder to break.
3. Choose an integer e, which is less than $\varphi(n)$, but greater than 2. e for "encryption".

Setting It Up

red: private
blue: public

1. Make a semiprime, using large primes
   - let $n = pq$
   - This uses the idea that even if someone knows n, they have a hard time finding pq.
2. Find $\varphi(n) = (p-1)(q-1)$
   - This will just make encryption harder to break.
3. Choose an integer e, which is less than $\varphi(n)$, but greater than 2. e for "encryption".
4. Find the modular multiplicative inverse of $e \pmod{\varphi(n)}$. Let it be d, for "decryption", $ed \equiv 1 \pmod{\varphi(n)}$

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).
   - Call the integer m
3. Bob encrypts the data, by computing $c \equiv m^e \pmod{n}$ and delivers it to Alice

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).
   - Call the integer m
3. Bob encrypts the data, by computing $c \equiv m^e \pmod{n}$ and delivers it to Alice
   - "c" for ciphertext
   - For convenience, c is usually the smallest number $mod(n)$)

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).
    - Call the integer m
3. Bob encrypts the data, by computing $c \equiv m^e \pmod{n}$ and delivers it to Alice
    - "c" for ciphertext
    - For convenience, c is usually the smallest number $mod(n)$)
4. Alice has a specific public key, so she also has a specific private key.

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).
   - Call the integer m
3. Bob encrypts the data, by computing $c \equiv m^e \pmod{n}$ and delivers it to Alice
   - "c" for ciphertext
   - For convenience, c is usually the smallest number $mod(n)$)
4. Alice has a specific public key, so she also has a specific private key.
   - She computes d: $ed \equiv 1 \pmod{\varphi(n)}$)

The Process

Consider a situation when Bob wants to send a message to Alice.

1. To do this Alice will send her public key (n, e) to Bob.
2. Bob encrypts his message, using some padding scheme (maps a message into a number).
    - Call the integer m
3. Bob encrypts the data, by computing $c \equiv m^e \pmod{n}$ and delivers it to Alice
    - "c" for ciphertext
    - For convenience, c is usually the smallest number $mod(n)$)
4. Alice has a specific public key, so she also has a specific private key.
    - She computes d: $ed \equiv 1 \pmod{\varphi(n)}$)
5. Then, Alice recovers m, since $m^{ed} \equiv m \pmod{n}$

Proof That RSA Works

RSA's functionality relies on the fact that $m^{ed} \equiv m \pmod{n}$, so if this is true, then RSA works.

Proof That RSA Works

RSA's functionality relies on the fact that $m^{ed} \equiv m \pmod{n}$, so if this is true, then RSA works.
To prove this we need a quick theorem.

Euler's Theorem

Theorem

Let a, n be coprime

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Proof that RSA Works

Note that we can assume that m and n are coprime, since n is a product of two primes, it's highly unlikely that they will not be coprime.

Proof that RSA Works

Note that we can assume that m and n are coprime, since n is a product of two primes, it's highly unlikely that they will not be coprime. Even if they are, we can choose a different padding scheme, to change m.

Proof that RSA Works

Note that we can assume that m and n are coprime, since n is a product of two primes, it's highly unlikely that they will not be coprime. Even if they are, we can choose a different padding scheme, to change m.

Claim 5.1

Let $gcd(m, n) = 1 = gcd(e, \varphi(n))$ and $ed \equiv 1 \pmod{\varphi(n)}$

$$m^{ed} \equiv m \pmod{n}$$

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\begin{aligned} \varphi(n) &\mid ed - 1 \\ \implies \exists z \in \mathbb{Z} \quad z\varphi(n) &= ed - 1 \\ 1 + z\varphi(n) &= ed \end{aligned}$$

Now we can plug this into the exponent:

$$\begin{aligned} m^{1+z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot m^{z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot (1)^z &\equiv m \pmod{n} \\ m &\equiv m \pmod{n} \end{aligned}$$

Thus, the proof is complete. [pro] □

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\varphi(n) \mid ed - 1$$

$$1 + z\varphi(n) = ed$$

Now we can plug this into the exponent:

$$\begin{aligned} m^{1+z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot m^{z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot (1)^{z} &\equiv m \pmod{n} \\ m &\equiv m \pmod{n} \end{aligned}$$

Thus, the proof is complete. [pro] $\square$

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\begin{aligned} \varphi(n) &\mid ed - 1 \\ \implies \exists z \in \mathbb{Z} \quad z\varphi(n) &= ed - 1 \\ 1 + z\varphi(n) &= ed \end{aligned}$$

Now we can plug this into the exponent:

$$\begin{aligned} m^{1+z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot m^{z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot (1)^z &\equiv m \pmod{n} \\ m &\equiv m \pmod{n} \end{aligned}$$

Thus, the proof is complete. [pro] $\square$

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\begin{aligned} \varphi(n) &\mid ed - 1 \\ \implies \exists z \in \mathbb{Z} \quad z\varphi(n) &= ed - 1 \\ 1 + z\varphi(n) &= ed \end{aligned}$$

Now we can plug this into the exponent:

$$m^{1+z\varphi(n)} \equiv m \pmod{n}$$

$$\begin{aligned} m \cdot (1)^z &\equiv m \pmod{n} \\ m &\equiv m \pmod{n} \end{aligned}$$

Thus, the proof is complete. [pro] □

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\begin{gathered} \varphi(n) \mid ed - 1 \\ \implies \exists z \in \mathbb{Z} \quad z\varphi(n) = ed - 1 \\ 1 + z\varphi(n) = ed \end{gathered}$$

Now we can plug this into the exponent:

$$m^{1+z\varphi(n)} \equiv m \pmod{n}$$

$$m \equiv m \pmod{n}$$

Thus, the proof is complete. [pro] □

The Proof

Proof.

Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write this by its definition:

$$\begin{aligned} \varphi(n) &\mid ed - 1 \\ \implies \exists z \in \mathbb{Z} \quad z\varphi(n) &= ed - 1 \\ 1 + z\varphi(n) &= ed \end{aligned}$$

Now we can plug this into the exponent:

$$\begin{aligned} m^{1+z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot m^{z\varphi(n)} &\equiv m \pmod{n} \\ m \cdot (1)^z &\equiv m \pmod{n} \\ m &\equiv m \pmod{n} \end{aligned}$$

Thus, the proof is complete. [pro] □

The mathematics behind rsa.

R.L. Rivest, A. Shamir, and L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems, Feb 1978.

Nigel P. Smart.
Cryptography: An introduction.
McGraw-Hill, 2003.