RSA - The keys to security

Ilija Afanasyev

10. Juni 2021

Table of contents

Motivation

- dealt with encryption (f.i. https protocol, ssh)
- while discussing encryption with Theel: "...something with RSA..."
- RSA was mentioned in university algebra

Motivation

- dealt with encryption (f.i. https protocol, ssh)
- while discussing encryption with Theel: "...something with RSA..."
- RSA was mentioned in university algebra

$\rightarrow$ How does RSA work again?

What to expect?

- hopefully some better understanding of RSA
- how RSA works with example and implementation
- insight in mathematical detail with idea for proof
- evaluation of security level

Definition

Wikipedia: "RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission."

Definition

Wikipedia: "RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission."

- *public key* encryption technique/algorithm
- named after 3 founders: Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
- helps with data transmission

Definition

Wikipedia: "RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission."

- *public key* encryption technique/algorithm
- named after 3 founders: Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
- helps with data transmission

→ What is a public key encryption technique?
→ How is data transmitted?

Symmetric and asymmetric cryptography

Imagine you want to encrypt and decrypt some secret data.

Symmetric and asymmetric cryptography

Imagine you want to encrypt and decrypt some secret data.

1. Symmetric cryptography: there is only one key. Use same key for encryption and decryption.

Symmetric and asymmetric cryptography

Imagine you want to encrypt and decrypt some secret data.

1. Symmetric cryptography: there is only one key. Use same key for encryption and decryption.
2. Asymmetric cryptography (= public key cryptography): There are two keys - public key and private key. If locked with public key $\rightarrow$ unlock with private key. If locked with private key $\rightarrow$ unlock with public key.

Symmetric and asymmetric cryptography

Imagine you want to encrypt and decrypt some secret data.

1. Symmetric cryptography: there is only one key. Use same key for encryption and decryption.
2. Asymmetric cryptography (= public key cryptography): There are two keys - public key and private key. If locked with public key $\rightarrow$ unlock with private key. If locked with private key $\rightarrow$ unlock with public key.

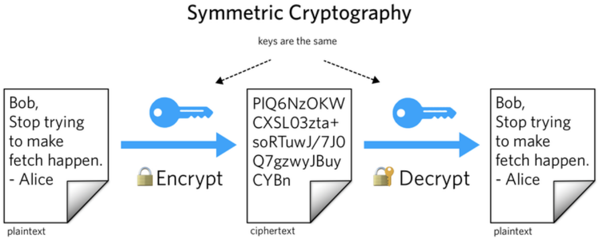
$\rightarrow$ Why public and private? And still: how is data transmitted?

Symmetric cryptography

Alice wants to send Bob a message.

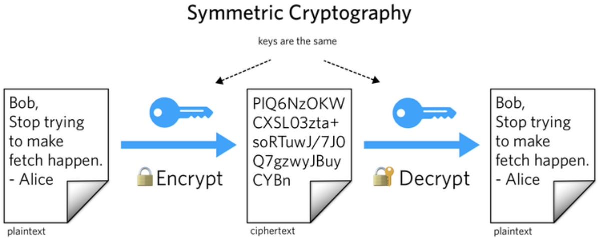
Symmetric cryptography

Alice wants to send Bob a message.



Symmetric cryptography

Alice wants to send Bob a message.

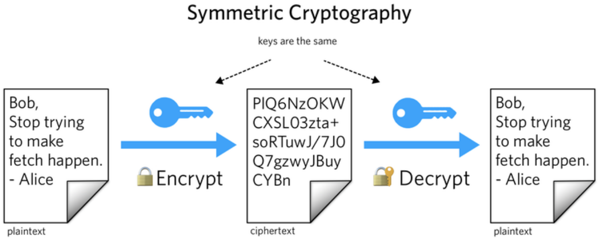


→ How does Alice get the key?

→ Why doesn't she simply get the message instead?

Symmetric cryptography

Alice wants to send Bob a message.



$\rightarrow$ How does Alice get the key?

$\rightarrow$ Why doesn't she simply get the message instead?

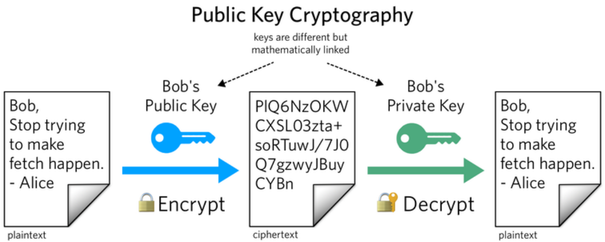
Good Questions!

Assymetric cryptography

Alice wants to send Bob a message.

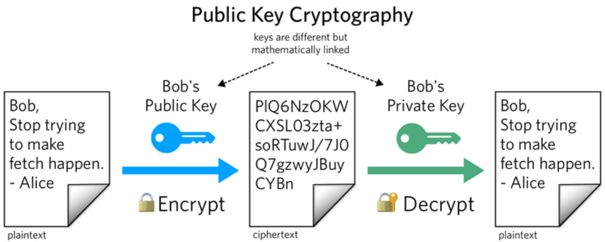
Assymetric cryptography

Alice wants to send Bob a message.



Assymetric cryptography

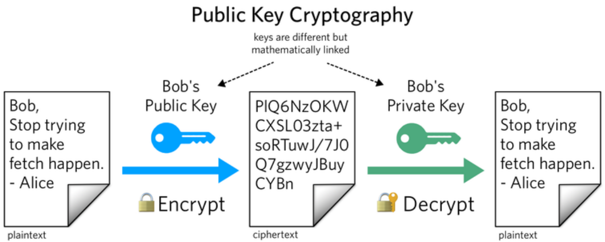
Alice wants to send Bob a message.



1. Alice has Bob's public key. (Public key can be shared with anyone, who wants to send a message to Bob.)
2. Only Bob has his private key.

Assymetric cryptography

Alice wants to send Bob a message.



1. Alice has Bob's public key. (Public key can be shared with anyone, who wants to send a message to Bob.)
2. Only Bob has his private key.

$\rightarrow$ That's nice! How are the keys generated?

$\rightarrow$ How are they „connected“?

Key generation

Short summary, we need:

1. two keys
2. minimum requirement: one key decrypts messages sent by the other
3. there is no other key with feature (2)
4. the keys are very hard to guess

Key generation

Short summary, we need:

1. two keys
2. minimum requirement: one key decrypts messages sent by the other
3. there is no other key with feature (2)
4. the keys are very hard to guess

Prerequisites:

- **prime number** (Primzahl): number with two (natural) factors: 1 and itself
- **modulo**: system of arithmetic, where numbers „wrap around“ after certain value, f.e. $18 \equiv 6 \mod 12$
- **coprime** (teilerfremd): two integers are called coprime, if they have no common factor/divisor (except for the obvious 1)

Algorithm

1. Choose (big) prime numbers p and q, $p \neq q$
2. $n \rightarrow p \cdot q$
3. Compute $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$
4. Choose e with $1 < e < \phi(n)$, such that $\phi(n), e$ coprime
5. Compute d, such that $d \cdot e \equiv 1 \mod \phi(n)$

Algorithm

1. Choose (big) prime numbers p and q, $p \neq q$
2. $n \rightarrow p \cdot q$
3. Compute $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$
4. Choose e with $1 < e < \phi(n)$, such that $\phi(n), e$ coprime
5. Compute d, such that $d \cdot e \equiv 1 \mod \phi(n)$

$\rightarrow$ How do we encrypt?
$\rightarrow$ How do we find p, q?
$\rightarrow$ How do we find e?
$\rightarrow$ How do we find d?
$\rightarrow$ Most important: Why does this work?

Algorithm Example

1. (Choose prime numbers p and q)
   $p \to 11,\ q \to 13$

Algorithm Example

1. (Choose prime numbers p and q)
   $p \rightarrow 11,\ q \rightarrow 13$
2. $n \rightarrow p \cdot q$
   $11 \cdot 13 = 143 \leftarrow n$

Algorithm Example

1. (Choose prime numbers p and q)
   $p \to 11,\ q \to 13$
2. $n \to p \cdot q$
   $11 \cdot 13 = 143 \leftarrow n$
3. (Compute $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$)
   $10 \cdot 12 = 120 \leftarrow \phi(n)$

Algorithm Example

1. (Choose prime numbers p and q)
   $p \rightarrow 11,\ q \rightarrow 13$
2. $n \rightarrow p \cdot q$
   $11 \cdot 13 = 143 \leftarrow n$
3. (Compute $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$)
   $10 \cdot 12 = 120 \leftarrow \phi(n)$
4. (Choose e with $1 < e < \phi(n)$, such that $\phi(n), e$ coprime)
   $e \rightarrow 23$, (e is prime number, e is not divisor of 120)

Algorithm Example

1. (Choose prime numbers p and q)
   $p \to 11,\ q \to 13$
2. $n \to p \cdot q$
   $11 \cdot 13 = 143 \leftarrow n$
3. (Compute $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$)
   $10 \cdot 12 = 120 \leftarrow \phi(n)$
4. (Choose e with $1 < e < \phi(n)$, such that $\phi(n), e$ coprime)
   $e \to 23$, (e is prime number, e is not divisor of 120)
5. (Compute d, such that $d \cdot e \equiv 1 \mod \phi(n)$)
   Extended Euclidean Algorithm: $d \to 47$
   $(23 \cdot 47 = 1081 = 9 \cdot 120 + 1)$

Encryption

1. We have prime numbers p, q, their product n, coprime numbers e, $\phi(n)$ and d
2. **Pairs** (e, n) **and** (d, n) **are our keys! (In our case** $(23, 143), (47, 143)$**)**

Encryption

1. We have prime numbers p, q, their product n, coprime numbers e, $\phi(n)$ and d
2. **Pairs** (e, n) **and** (d, n) **are our keys! (In our case** $(23, 143)$, $(47, 143)$**)**

public key $= (23, 143)$
private key $= (47, 143)$

1. Alice takes a message and encodes it, f.e.
   Hi: 89 (8th and 9th letters of the alphabet)

public key $= (23, 143)$
private key $= (47, 143)$

1. Alice takes a message and encodes it, f.e.
   Hi: 89 (8th and 9th letters of the alphabet)
2. Alice takes the number m, computes
   $m^e \mod n$
   $89^{23} \mod 143 \equiv 45$.
   This is the message encrypted with Bob's public key (e, n)

public key $= (23, 143)$
private key $= (47, 143)$

1. Alice takes a message and encodes it, f.e.
   Hi: 89 (8th and 9th letters of the alphabet)
2. Alice takes the number m, computes
   $m^e \mod n$
   $89^{23} \mod 143 \equiv 45$.
   This is the message encrypted with Bob's public key (e, n)
3. Bob takes the encrypted message c and computes $c^d \mod n$ with his private key (d, n).
   $45^{47} \mod 143 \equiv 89$.

public key $= (23, 143)$
private key $= (47, 143)$

1. Alice takes a message and encodes it, f.e.
   Hi: 89 (8th and 9th letters of the alphabet)
2. Alice takes the number m, computes
   $m^e \mod n$
   $89^{23} \mod 143 \equiv 45$.
   This is the message encrypted with Bob's public key (e, n)
3. Bob takes the encrypted message c and computes $c^d \mod n$ with his private key (d, n).
   $45^{47} \mod 143 \equiv 89$.

   That's the original message!

public key $= (23, 143)$
private key $= (47, 143)$

1. Alice takes a message and encodes it, f.e.
   Hi: 89 (8th and 9th letters of the alphabet)
2. Alice takes the number m, computes
   $m^e \mod n$
   $89^{23} \mod 143 \equiv 45$.
   This is the message encrypted with Bob's public key (e, n)
3. Bob takes the encrypted message c and computes $c^d \mod n$ with his private key (d, n).
   $45^{47} \mod 143 \equiv 89$.

That's the original message! $\rightarrow$ Again, why does this work?

Mathematical Details

Main question: given keys (e, n) and (d, n), message m: Why is

$$((m^e \mod n)^d \mod n) \equiv m\,?$$

Mathematical Details

Main question: given keys (e, n) and (d, n), message m: Why is

$$((m^e \mod n)^d \mod n) \equiv m\,?$$

Using modular multiplication property one can simplify to

$$m \equiv m^{e^d} \mod n$$

Mathematical Details

Main question: given keys (e, n) and (d, n), message m: Why is

$$((m^e \mod n)^d \mod n) \equiv m\,?$$

Using modular multiplication property one can simplify to

$$m \equiv m^{e^d} \mod n$$

Idea of proof (complete proof in the sources):

- $e \cdot d \equiv 1 \mod \phi(n)$ means there is a number k with $e \cdot d = 1 + k \cdot \phi(n)$
- Use *Fermat-Euler theorem* to show $m = m^{1+k\cdot\phi(n)} \mod p$ and $m = m^{1+k\cdot\phi(n)} \mod q$
- Use *Chinese Remainder theorem* to show $m \cdot m^{k\cdot\phi(n)} \equiv m \mod p \cdot q$

Security

- BSI: at least 2000 bit length until March 2023

Security

- BSI: at least 2000 bit length until March 2023
- „It would take a classical computer around 300 trillion years to break a RSA-2048 bit encryption key“

Security

- BSI: at least 2000 bit length until March 2023
- „It would take a classical computer around 300 trillion years to break a RSA-2048 bit encryption key“
- The complexity lies either in factorizing n to get p and q or in computing the message through encrypted message and public key
- For both problems no polynomial-time algorithm is known

Security

- BSI: at least 2000 bit length until March 2023
- „It would take a classical computer around 300 trillion years to break a RSA-2048 bit encryption key"
- The complexity lies either in factorizing n to get p and q or in computing the message through encrypted message and public key
- For both problems no polynomial-time algorithm is known
- Quantum computers **could** crack the encryption in less than a day

Implementation

This section is about a specific implementation (`https://github.com/ingoaf/rsa-example`). Yet, there are basic concepts which can be applied to other implementations.

Implementation

This section is about a specific implementation (`https://github.com/ingoaf/rsa-example`). Yet, there are basic concepts which can be applied to other implementations.

- Encoding, Decoding: string $\leftrightarrow$ numbers (f.e. string $\leftrightarrow$ []rune)
- Encryption: two main packages: **math/big** and **crypto/rand**

Implementation

This section is about a specific implementation (`https://github.com/ingoaf/rsa-example`). Yet, there are basic concepts which can be applied to other implementations.

- Encoding, Decoding: string $\leftrightarrow$ numbers (f.e. string $\leftrightarrow$ []rune)
- Encryption: two main packages: **math/big** and **crypto/rand**
- math/big is for **all** necessary arithmetic operations on big integers

Implementation

This section is about a specific implementation (`https://github.com/ingoaf/rsa-example`). Yet, there are basic concepts which can be applied to other implementations.

- Encoding, Decoding: string $\leftrightarrow$ numbers (f.e. string $\leftrightarrow$ []rune)
- Encryption: two main packages: **math/big** and **crypto/rand**
- math/big is for **all** necessary arithmetic operations on big integers
- crypto/rand is for generating big primes, returns big integers!

Implementation

This section is about a specific implementation (`https://github.com/ingoaf/rsa-example`). Yet, there are basic concepts which can be applied to other implementations.

- Encoding, Decoding: string $\leftrightarrow$ numbers (f.e. string $\leftrightarrow$ []rune)
- Encryption: two main packages: **math/big** and **crypto/rand**
- math/big is for **all** necessary arithmetic operations on big integers
- crypto/rand is for generating big primes, returns big integers!

Problem: slow, workaround: combine with symmetric cryptography

Summary

- generates two keys: public and private
- uses modulo arithmetic with big prime numbers
- key prime numbers are modular inverse to each other
- hard to crack, because no **practical** algorithm for factorization is known
- implementation in Go relies on big integers (math/big) and crypto packages (crypto/rand)
- can be combined with symmetric cryptography algorithms

Sources

- RSA proof
- RSA general information
- Computing modular inverse: example
- Breaking RSA Encryption
- What is a rune?
- math/big docs
- crypto/rand docs