RSA in practice

$ whoami

- Michał Janiszewski
- janisozaur
- Mainly C++
- Software developer @ Mobica (Cognizant)

Agenda

1. What is RSA?
2. Maths refresher
3. RSA example
4. Key handling
5. Implementations
6. Use cases
7. Security considerations

RSA - introduction

RSA is an encryption algorithm that allows for secure communication over the internet. Invented by Ron **Rivest**, Adi **Shamir**, and Leonard **Adleman** in 1977, RSA has become one of the most widely used encryption systems in the world, partly thanks to many readily available open source implementations.

Asymmetric cryptography

- Asymmetric cryptography is also known as public-key cryptography.
- It uses two different keys for encryption and decryption.
- A public key is used for encryption, while a private key is used for decryption.
- The public key is made widely available, while the private key is kept secret.
- This system allows for secure communication between two parties who have never met before and do not share a secret key.

Symmetric cryptography

- Uses the same key for both encryption and decryption.
- The key must be kept secret by both parties.
- Requires a secure method of key exchange between the parties.
- Generally faster and more efficient than asymmetric cryptography.
- Suitable for encrypting large amounts of data.

Maths refresher

Counting 101

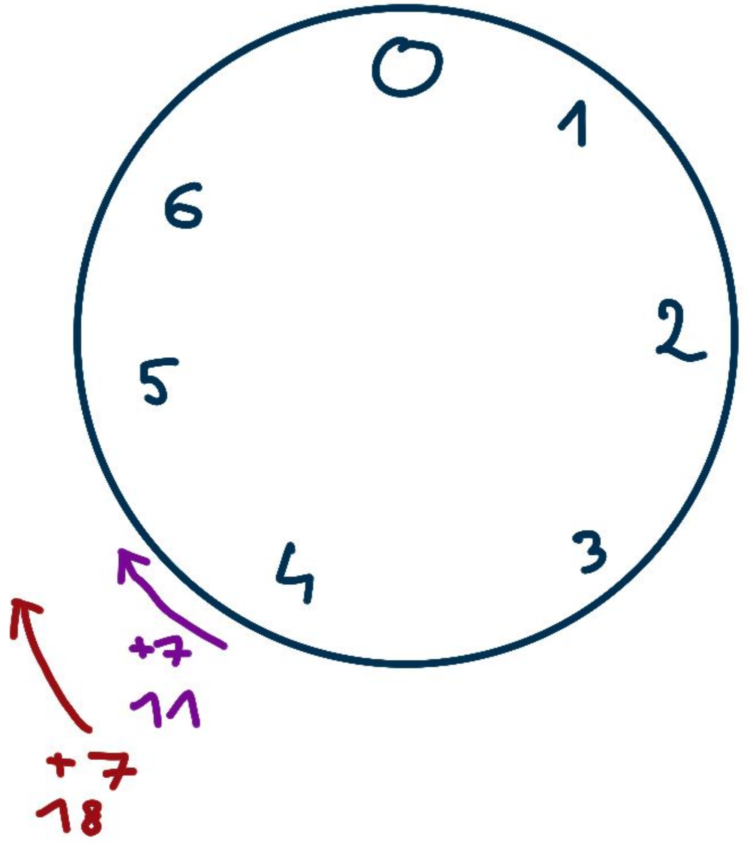
Modular arithmetic

Modular arithmetic is a system of arithmetic for
integers that involves taking remainders after division.

4 mod 7 = 4, because remainder is 4

11 mod 7 = 4, because 11 = 1 * 7 + 4

18 mod 7 = 4, because 18 = n * 7 + 4



Congruence

Congruence is a mathematical concept that deals with the equivalence of two numbers with respect to a modulus. In modular arithmetic, two numbers are considered congruent if they have the same remainder when divided by a fixed modulus. Modular arithmetic is used in cryptography to perform calculations on large numbers that are too big to handle using regular arithmetic.

Congruence is denoted with a triple-lined equal sign:

$4 \equiv 11 \pmod{7} \equiv 18 \pmod{7}$

Exponentiation

Exponentiation is a generic way to express multiplication of a number.

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n}$$

Commutative property

$(a^n)^m = a^{n*m} = a^{m*n} = (a^m)^n$

Prime numbers

A **factor** is a whole number that can be divided evenly into another number. A **prime number** is a whole number greater than 1 whose only factors are 1 and itself. Numbers that have more than two factors are called **composite numbers**. Two numbers are **coprime** if the only positive integer that is a divisor of both of them (GCD) is 1.

Prime numbers

Prime numbers:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

Composite numbers:

4, 6, 8, 9, 10, 12, 14, 15, 16, 18

Factorization:

```
5 = 1 * 5 // prime, 1 is usually omitted
6 = 2 * 3
48 = 16 * 3 = 2^4 * 3
```

Coprimes:

```
GCD(6, 7) = 1 // composite with prime
GCD(14, 15) = 1 // composite with composite
```

RSA

The Algorithm

RSA principle

The RSA algorithm relies on the principle that it is easy to find a congruent value, but hard to determine the original value that was used initially.

$$(m^e)^d \equiv m \bmod n$$

In this equation, *e* and *d* are integers, *n* is a product of two large prime numbers, and *m* is the message being encrypted.

RSA principle

$$(m^e)^d \equiv m \bmod n$$

Given only the values of *e*, *n* and *m*, it is infeasible to compute the value of *d*. This is because there is no known algorithm that can efficiently compute the modular inverse of a large number. Instead, it would require trying out every possible value of *d*, which is impractical for large numbers.

RSA principle

To make use of prime factorization, RSA operates on large prime numbers. The larger the prime factors, the more secure the encryption.

RSA example

Demo:

- Jupyter notebook
- SageMath

Working example

RSA in action

Key generation

To generate an RSA key pair, we first choose two random prime numbers p and q that are similar in size.

    p=13, q=17

Compute n=p*q

    n=13*17=221

Compute least common multiple of (q-1,p-1):

    l=lcm(q-1,p-1)=lcm(13-1,17-1)=lcm(12,16)=48

Key generation

Choose a number e in range 1<e<48 that is coprime (meaning their only common divisor is 1) with 48.

Let e=5

Find d such that

1 ≡ (e*d) mod l ≡ (5*d) mod 48 ≡ (5*29) mod 48 ≡ 145 mod 48

In our example the **public key is (n=221,e=5)** and the **private key is (n=221,d=29)**.

Encryption and decryption

Using equation $(m^e)^d \equiv m \bmod n$ lets encrypt a sample message m=22

$m^e \bmod n \equiv 22^5 \bmod 221 \equiv 5153632 \bmod 221 \equiv 133$

And decrypt it

$(m^e)^d \bmod n \equiv 133^{29} \bmod 221 \equiv$
$39056883657367139614227739052562278609342949109376229274301653 \bmod 221 \equiv 22$

Signing and verification

Asymmetric cryptography can also be used to create digital signatures of messages, which uses private key for signing and public key for verification. Encryption's

$$(m^{e})^{d} \equiv m \bmod n$$

becomes

$$(m^{d})^{e} \equiv m \bmod n$$

Signing and verification

Let’s create a signature s of the same message m=22 using our private key:

$s=m^d \bmod n \equiv 22^{29} \bmod 221 \equiv$
$851643319086537701956194499721106030592 \bmod 221 \equiv 3$

And verify this signature:

$s^e \bmod n \equiv 3^5 \bmod 221 \equiv 243 \bmod 221=22=m$

Key distribution

How to distribute RSA keys

PEM key format

Real-world use cases demand keys much larger than our paltry 3-digit-long example. They also require standardised key distribution format.

Enter PEM, Privacy Enhanced Mail. Standardised in RFC 1422 describes format for key management.

PEM key format

```
$ openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt
rsa_keygen_bits:1024

-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAPHVJRs6TpFIyXYY
3r9PvTq3LBZ1pcW2qmy2z4O8zSESyTJ3F4RhGHkEmBwObzJTRvpKESz7OAgct9dL
31/OkpOpZhRtnxiMBhTJZMpYfMoyQsR778xcnVZ6z8ma45fpXDSg60p/BTK+lKVh
                                   …
SPgE4PZP9o9CD/QxK6hnASrsJUlZtQJBAPCIGJSkFViZnuQ3sRnfWy3dEUgWsROZ
PyYXqwyfSg3uus+HhjpUf7vq5vdsOjB//3E6GZvzKUWTOMqb3+8EdaUCQCYdQAJ5
/danZoc+wBBtkqlYNYlKbxl0oE/HFLYA0LAL+COsi9NtZAItAQ1zEeNzpLsVzaFy
72/ZHjKL6hozhKM=
-----END PRIVATE KEY-----
```

PEM key format

Demo again

Implementations

Where to get RSA

Implementations

RSA software implementations are widely available including:

- mbedTLS (open source)
- OpenSSL/WolfSSL/LibreSSL (open source)
- Windows CNG
- Bouncy Castle (open source)
- And more

Implementations

RSA is also available with dedicated hardware blocks:

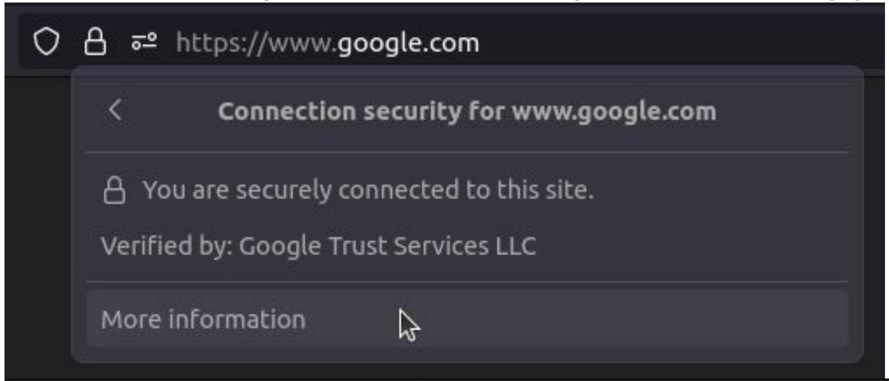
- IBM z/TPF
- AMD Zynq (FPGA)
- Intel QAT
- more…

Use cases

Where to use RSA

Use cases

- TLS (Transport Layer Security)



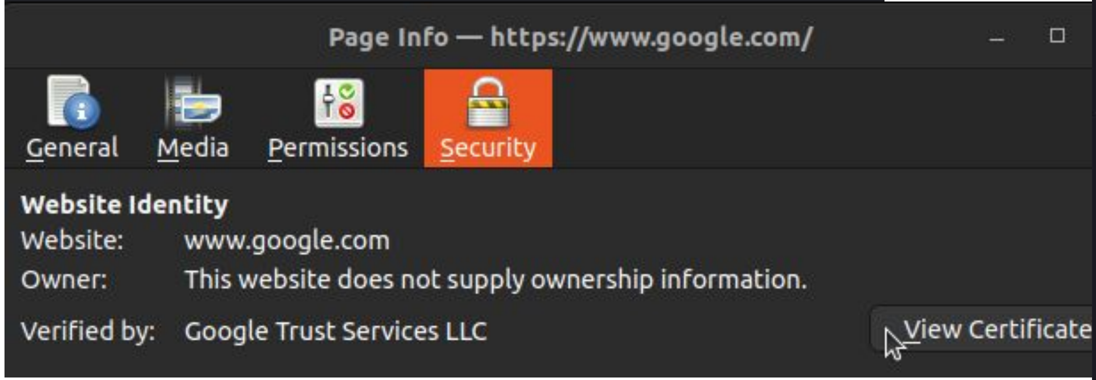
https://www.google.com

Connection security for www.google.com

You are securely connected to this site.

Verified by: Google Trust Services LLC

More information

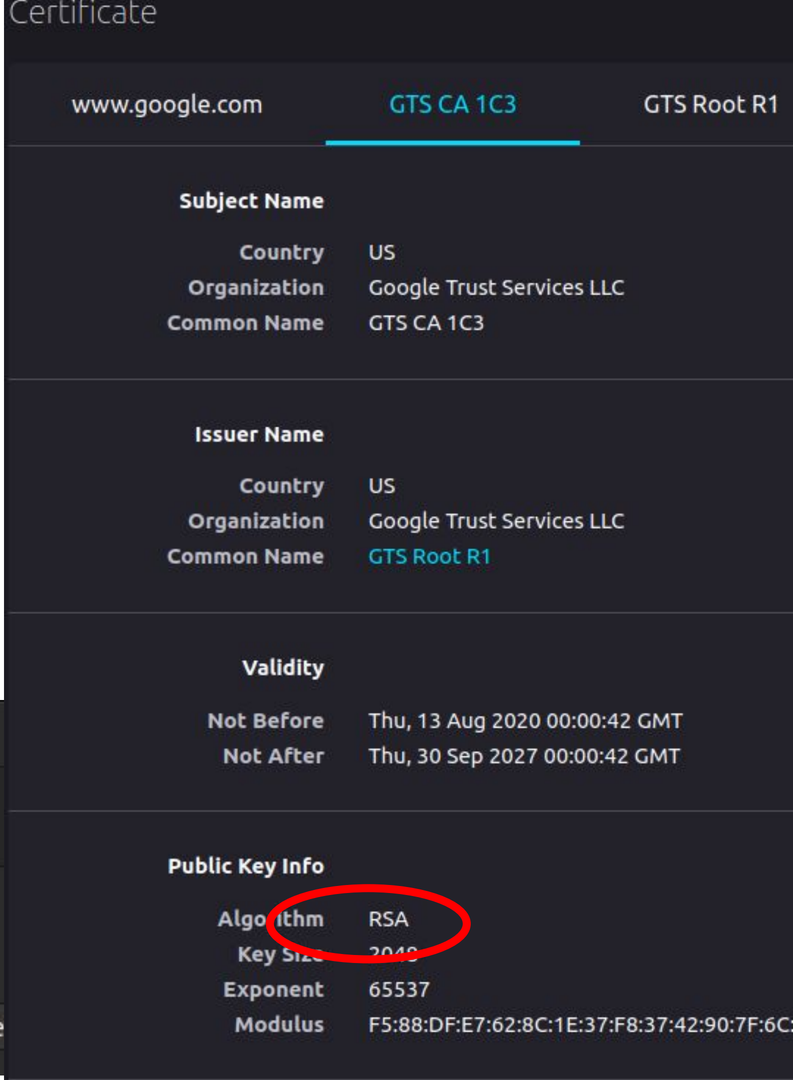


Page Info — https://www.google.com/

Website Identity

Website:	www.google.com
Owner:	This website does not supply ownership information.
Verified by:	Google Trust Services LLC

View Certificate



Certificate

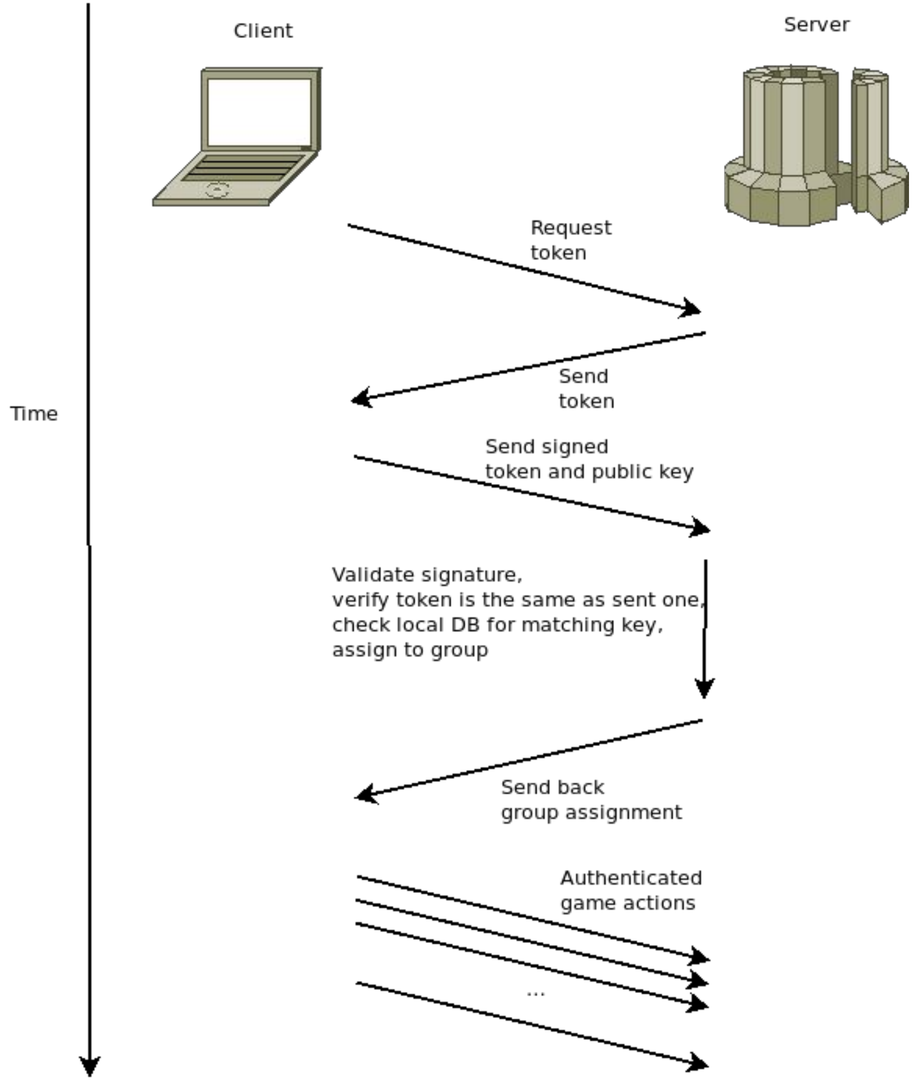
Subject Name	
Country	US
Organization	Google Trust Services LLC
Common Name	GTS CA 1C3
Issuer Name	
Country	US
Organization	Google Trust Services LLC
Common Name	GTS Root R1
Validity	
Not Before	Thu, 13 Aug 2020 00:00:42 GMT
Not After	Thu, 30 Sep 2027 00:00:42 GMT
Public Key Info	
Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	F5:88:DF:E7:62:8C:1E:37:F8:37:42:90:7F:6C:

Use cases

- TLS (Transport Layer Security)
- Message signatures
- SSH (Secure SHell)
- VPN (Virtual Private Network)
- Code signing

Use cases

- Multiplayer authentication


Client
Server
Time
Request token
Send token
Send signed token and public key
Validate signature, verify token is the same as sent one, check local DB for matching key, assign to group
Send back group assignment
Authenticated game actions
...

Use cases

- Intel QAT KPT (Key Protection Technology)
  - https://www.coursera.org/lecture/network-transformation-102/intel-quickassist-technology-key-protection-technology-kpt-Aavdk
  -

Use cases

- Live demo of hardware accelerator

Attacks

How (not) to use RSA

Attacks

RSA can be tricky to implement correctly. If done incorrectly, it can be subject to many attacks. It is important to only use well-tested crypto libraries.

Attacks

Modulus factorisation

- Attack
  - Factoring modulus n to primes q and p
  - RSA-512 was factored in 1999 using 8400 machine-years
  - Quantum computers with enough qubits using Shor's algorithm
- Mitigations
  - Using larger keys
  - Current recommendation (NIST 800-57) is at least 2048 bits

Attacks

Faulty key generation

- Attack
    - If p or q are not truly random, some information might be recovered
    - If p and q are close to each other, n can be factored in feasible time
    - If p or q is reused, it is possible to calculate relations between them and factor n
- Mitigations
    - Use good quality randomness source
    - Validate relationship between q and p

Attacks

Side-channel leaks

- Attack
  - It is possible to analyse performance characteristic of a computer executing RSA algorithm due to its relatively low performance
  - Power, timing, branch predictor and emitted sounds can be analysed
  - Timing analysis can be performed over network
- Mitigations
  - Adapting implementation to behave consistently regardless of key/message
  - Blinding

Attacks

Chosen ciphertext attack

- Attack
  - Abuses message padding scheme PKCS#1 v1
  - Repeatedly submits chosen ciphertext and infers data from responses
- Mitigations
  - Use PKCS#1 v2 (RSAES-OAEP; Encryption/decryption scheme using Optimal Asymmetric Encryption Padding)

Implementation detail

Message padding is a scheme of extending a short message (such as m=22) for purposes of:

- Adding structure to the message
- Ensuring proper block length for chosen algorithm
- Thwarting cryptanalysis
- Prevent replay attacks

Other algorithms

What to replace RSA with

Other algorithms

RSA is not the only asymmetric cryptography algorithm. Other algorithms include:

- ECC (Elliptic Curve Cryptography)
- ElGamal
- DSS
- YAK

Q&A

Your turn

Thank you