Tampering Protection

Link

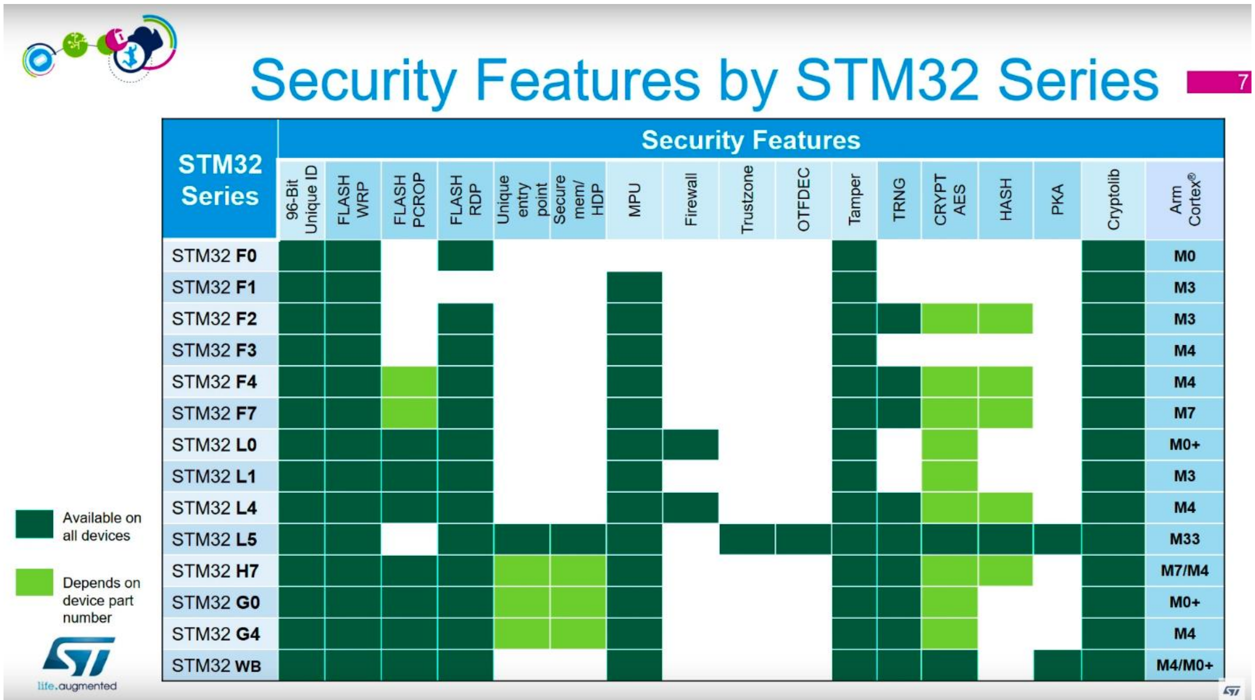
https://www.youtube.com/watch?v=qCuVBD2dmTA&list=PLnMKNibPkDnFzux3PHKUEi14ftDn9Cbm7&index=27

Description

In this paperwork, we will learn how to use asymmetric encryption on our board.

Contents

Prerequisites



STM32 Board

ST-Link cable

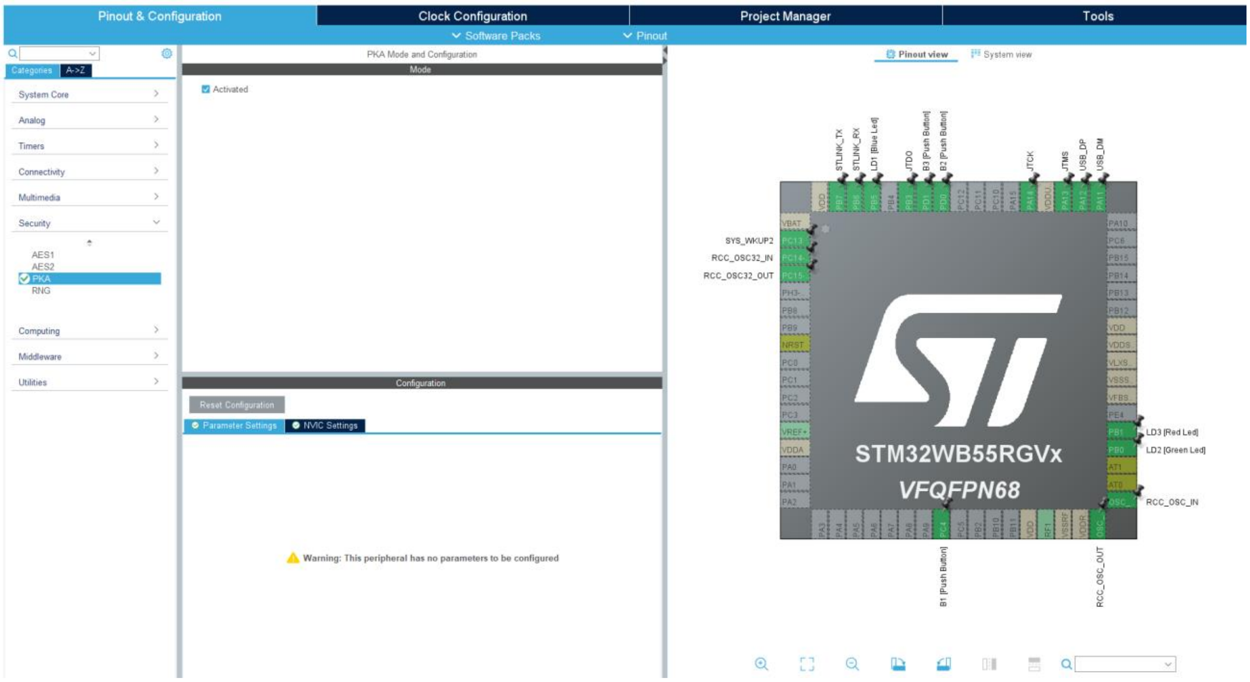
STM32CubeMX

STM32CubeIDE

Walkthrough

Step 1 : Launch STM32CubeMX and generate the code

Launch STM32CubeMX and select the right board depending on the one you are using. In my case I use the WB55 Nucleo board. Then you can generate the code of your project.
Then you have to activate PKA.



Step 2 : Generate the private and public key.

To generate your keys, you have to install openssl first. Once it is installed, you can generate the following commands using the application to generate them.

Generate private key :

```
┌──(user㉿kali)-[~]
└─$ openssl rsa -in MyPrivKey.pem
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC1tZiF2UJFwA3ds4mspvqmxjBIyf5Terq7Xzwkh2QOjUCzW1aD
3r0OQqK/JhiY4NcluzkSnXSasRrY5qxbSsjGElC8VgsW61BsdB3Nh+UCfibO9czb
Y7DIX7pdaIQMiSyUadhasKzTe02c620VeABAjgXizWqbGpjwAeJa6ziHmQIDAQAB
AoGABS3j5wzGQs6ylnsjlXQ8+Lv1bF21jhOVdvnD8Raa139XNMWJtcCHivyDPweK
8/CUsVKg0dMDG9Woej4483EyP8atCdPT9hJQOQX1JNqx8gfQBUrwDr+1ffwFxUyd
hEoomWqWn+bmUcJE9Dpt4JLHfnlV6BS7tN+mOZunQ4ZQHcECQQDyPvxRwdWSRxpQ
sjBee+M2fzcynDpZNI6hLrZ9ouRR88TVCW3WVG4sTF9Hou8rN8tG0XgUJnHpPMmZ
obJdK/z9AkEAwAa3rOFjkkfnfh8IFDI0awFXxKZBsIz8sFOOeHjg/y+M0I3mENve
YCzAvUP7c7HPnTeDB69iSrMjaK5oM6YFzQJAWFtg7PEmVRRaJNTZj5zgYyBDodIZ
9i+VVnUTWv/vB3VCdfHafjKNfNreZeKoGbtgCZSdl7vuEIR7g+3WgOVqgQJAawjT
hMe1KqptvH0rkaZSVXrQI0rQvso3Z0mL1lb6gwNMKEuP+8GyeEU5wcWM+XYZVXbF
0JjP3vdvO0BL0M4v2QJBAI8kcn+nlpVXEOSFdY7njnnzh4LZd9wCDfLWyWNsTWuq
2x5Hf/ZxEzPJKj6OA90Q9CDfvePtVKek9T1VVUQEnRg=
-----END RSA PRIVATE KEY-----
```

Read private key components :

```
┌──(user㉿kali)-[~]
└─$ openssl rsa -in MyPrivKey.pem -text
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:b5:b5:98:85:d9:42:45:c0:0d:dd:b3:89:ac:a6:
    fa:a6:c6:30:48:c9:fe:53:7a:ba:bb:5f:3c:24:87:
    64:0e:8d:40:b3:5b:56:83:de:bd:0e:42:a2:bf:26:
    18:98:e0:d7:25:bb:39:12:9d:74:9a:b1:1a:d8:e6:
    ac:5b:4a:c8:c6:12:50:bc:56:0b:16:eb:50:6c:74:
    1d:cd:87:e5:02:7e:26:ce:f5:cc:db:63:b0:c8:5f:
    ba:5d:68:84:0c:89:2c:94:69:d8:5a:b0:ac:d3:7b:
    4d:9c:eb:6d:15:78:00:40:8e:05:e2:cd:6a:9b:1a:
    98:f0:01:e2:5a:eb:38:87:99
publicExponent: 65537 (0x10001)
privateExponent:
    05:2d:e3:e7:0c:c6:42:ce:b2:96:7b:23:95:74:3c:
    f8:bb:f5:6c:5d:b5:8e:13:95:76:f9:c3:f1:16:9a:
    d7:7f:57:34:c5:89:b5:c0:87:8a:fc:83:3f:07:8a:
    f3:f0:94:b1:52:a0:d1:d3:03:1b:d5:a8:7a:3e:38:
    f3:71:32:3f:c6:ad:09:d3:d3:f6:12:50:39:05:f5:
    24:da:b1:f2:07:d0:05:4a:f0:0e:bf:b5:7d:fc:05:
    c5:4c:9d:84:4a:28:99:6a:96:9f:e6:e6:51:c2:44:
    f4:3a:6d:e0:92:c7:7e:79:55:e8:14:bb:b4:df:a6:
    39:9b:a7:43:86:50:1d:c1
```

Generate public key :

```
┌──(user㉿kali)-[~]
└─$ openssl rsa -in MyPrivKey.pem -pubout -out MyPubKey.pem
writing RSA key
```

Step 3 : Write the main.c

When you have the informations about your private key, just put them in the same format than in the screenshot. You have to enter the modulus, the public exponent, the private one, and a plaintext to encrypt.

You can use apps like notepad to format the values in the correct format.

We will also set a variable for the encryted and decrypted data.

```
/* USER CODE BEGIN PV */
uint8_t modulus[]= {0xb5, 0xb5, 0x98, 0x85, 0xd9, 0x42, 0x45, 0xc0, 0x0d, 0xdd, 0xb3, 0x89, 0xac, 0xa6, 0xfa, 0xa6, 0xc6, 0x30, 0x48, 0xc9,
uint8_t publicExponent[]={0x00,0x01,0x00,0x01};
uint8_t privateExponent[]={0x05, 0x2d, 0xe3, 0xe7, 0x0c, 0xc6, 0x42, 0xce, 0xb2, 0x96, 0x7b, 0x23, 0x95, 0x74, 0x3c, 0xf8, 0xbb, 0xf5, 0x6c,
uint8_t plaintext[128]={
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
};

uint8_t encrypted_data[128];
uint8_t decrypted_data[128];
/* USER CODE END PV */
```

Then add a variable in the main function to set your PKA parameters.

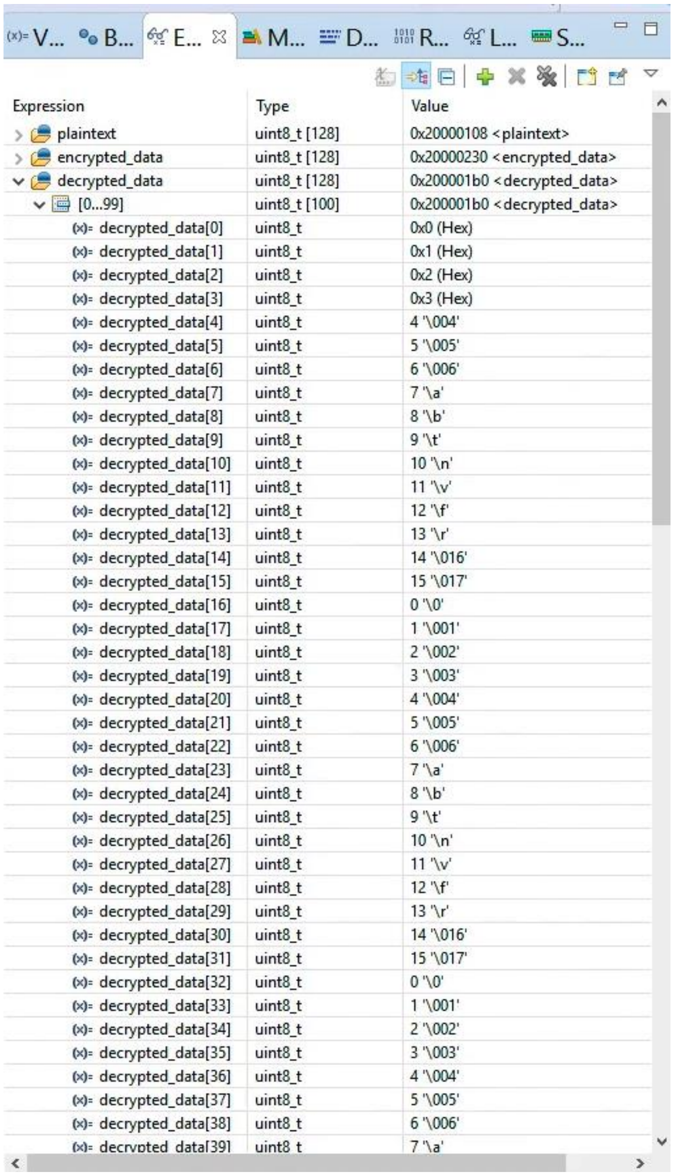
```
/* USER CODE BEGIN 1 */
  PKA_ModExpInTypeDef in;
/* USER CODE END 1 */
```

Then we will set the code for the encryption and the decryption. We basically set the values and just run the encryption. We set an error handler if something unexpected happens.

```
  //encryption
  in.expSize=sizeof(publicExponent);
  in.OpSize=sizeof(modulus);
  in.pExp=publicExponent;
  in.pMod=modulus;
  in.pOp1=plaintext;
if (HAL_PKA_ModExp(&hpka, &in, 1000) != HAL_OK)
{
    Error_Handler();
}

HAL_PKA_ModExp_GetResult(&hpka, encrypted_data);

MX_GPIO_Init();
  MX_PKA_Init();
  MX_USART1_UART_Init();
  MX_USB_PCD_Init();

  //decryption
  in.expSize=sizeof(privateExponent);
  in.OpSize=sizeof(modulus);
  in.pExp=privateExponent;
  in.pMod=modulus;
  in.pOp1=encrypted_data;
if (HAL_PKA_ModExp(&hpka, &in, 5000) != HAL_OK)
{
    Error_Handler();
}

HAL_PKA_ModExp_GetResult(&hpka, decrypted_data);
```

Step 4 : Check the encryption

Finally we just have to put our variables in the debug expressions and check them after the program has been launched.



You could normally see that the plaintext is equal to the decrypted data.