xRSA: Construct Larger Bits RSA on Low-Cost Devices

Fan DANG, Lingkun LI, Jiajie CHEN

Background

OPC UA

MQTT

Background

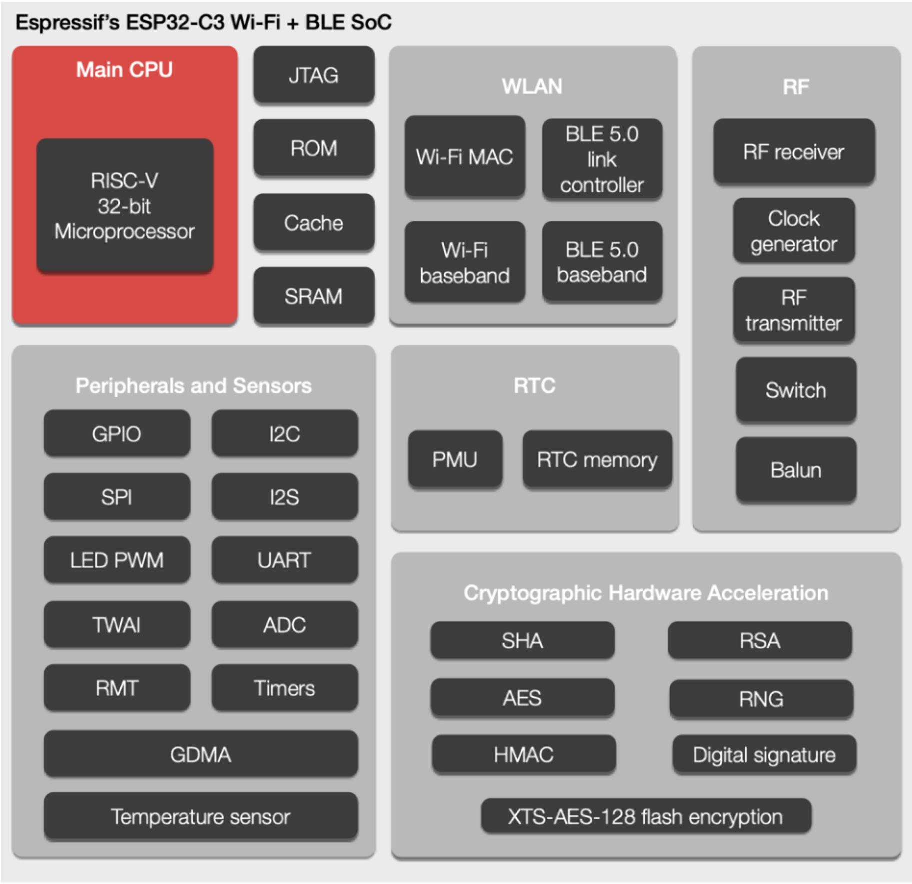
STM32L562E Cortex-M33 at 110 MHz

Symmetric Algorithm	Software (MB/s)	Accelerated (MB/s)
AES-CBC-128	0.121	4.468
AES-GCM-128	0.008	3.662
SHA-256	0.136	1.855

Asymmetric Algorithm	Software (ops/sec)	Accelerated (ops/sec) SP Math Cortex-M	Accelerated (ops/sec) ST PKA ECC
RSA 2048 public	9.208	18.083	18.083
RSA 2048 private	0.155	0.526	0.526
DH 2048 key gen	0.833	1.129	1.129
DH 2048 agree	0.411	1.128	1.128
ECC 256 key gen	0.661	35.608	10.309
ECDHE 256 agree	0.661	16.575	10.619
ECDSA 256 sign	0.652	21.912	20.542
ECDSA 256 verify	1.014	10.591	10.667

RSA is too heavy for low-cost devices (e.g., MCUs)

Background



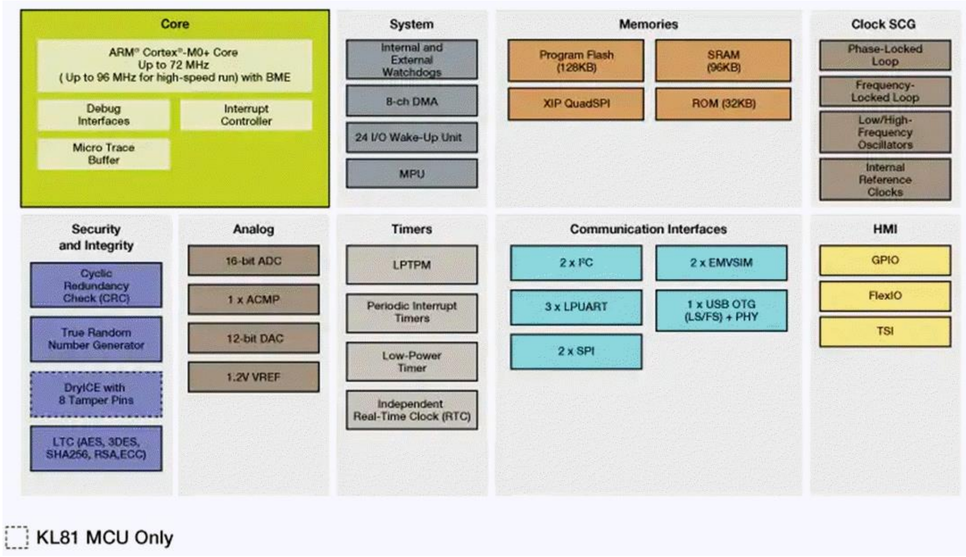
Block Diagram of ESP32-C3

life.augmented

STM32L562xx

Ultra-low-power Arm® Cortex®-M33 32-bit MCU+TrustZone®+FPU, 165DMIPS, up to 512KB Flash, 256KB SRAM, SMPS, AES+PKA

Datasheet - production data



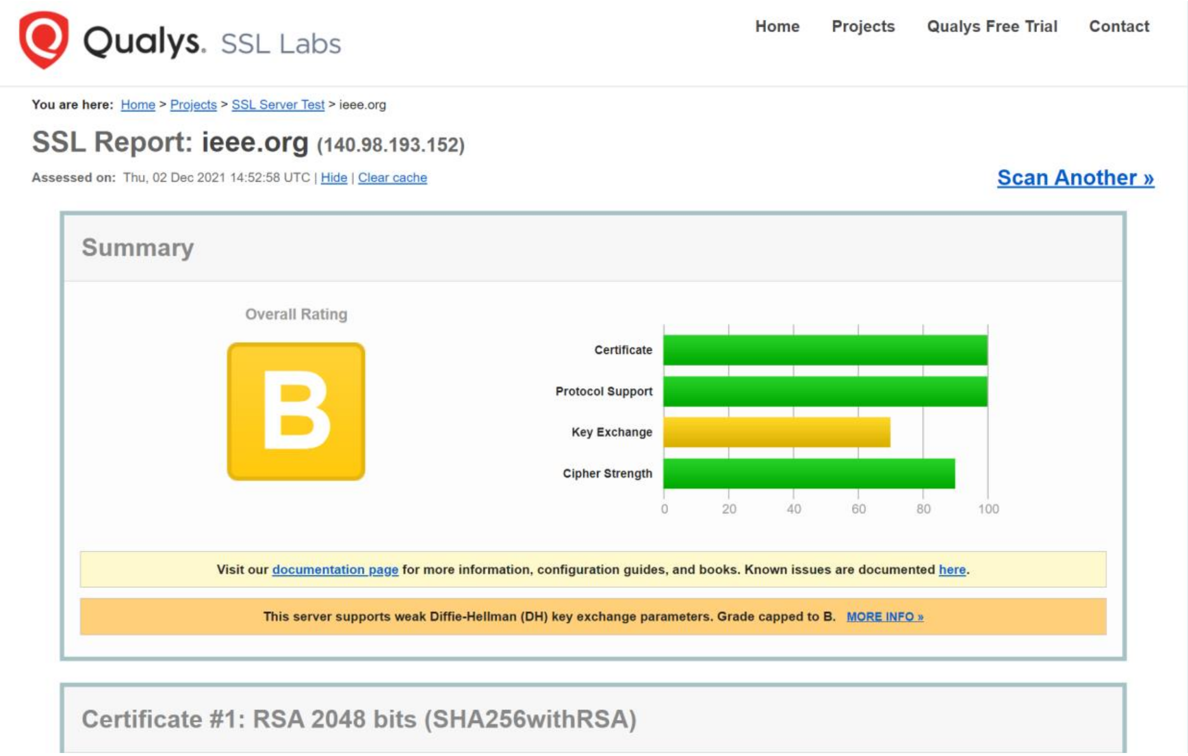
Background

Security

ESP32-C3 ensures that the availability of features, such as the RSA-3072-based secure boot and the AES-128-XTS-based flash encryption, can be used to build connected devices securely. The innovative digital signature

PKA main features:

- Acceleration of RSA, DH and ECC over GF(p) operations, based on the Montgomery method for fast modular multiplications. More specifically:
  - RSA modular exponentiation, RSA Chinese remainder theorem (CRT) exponentiation
  - ECC scalar multiplication, point on curve check
  - ECDSA signature generation and verification
- Capability to handle operands up to 3136 bits for RSA/DH and 640 bits for ECC.
- Arithmetic and modular operations such as addition, subtraction, multiplication, modular reduction, modular inversion, comparison, and Montgomery multiplication.



Requires RSA-4096 to get A+

Preliminaries

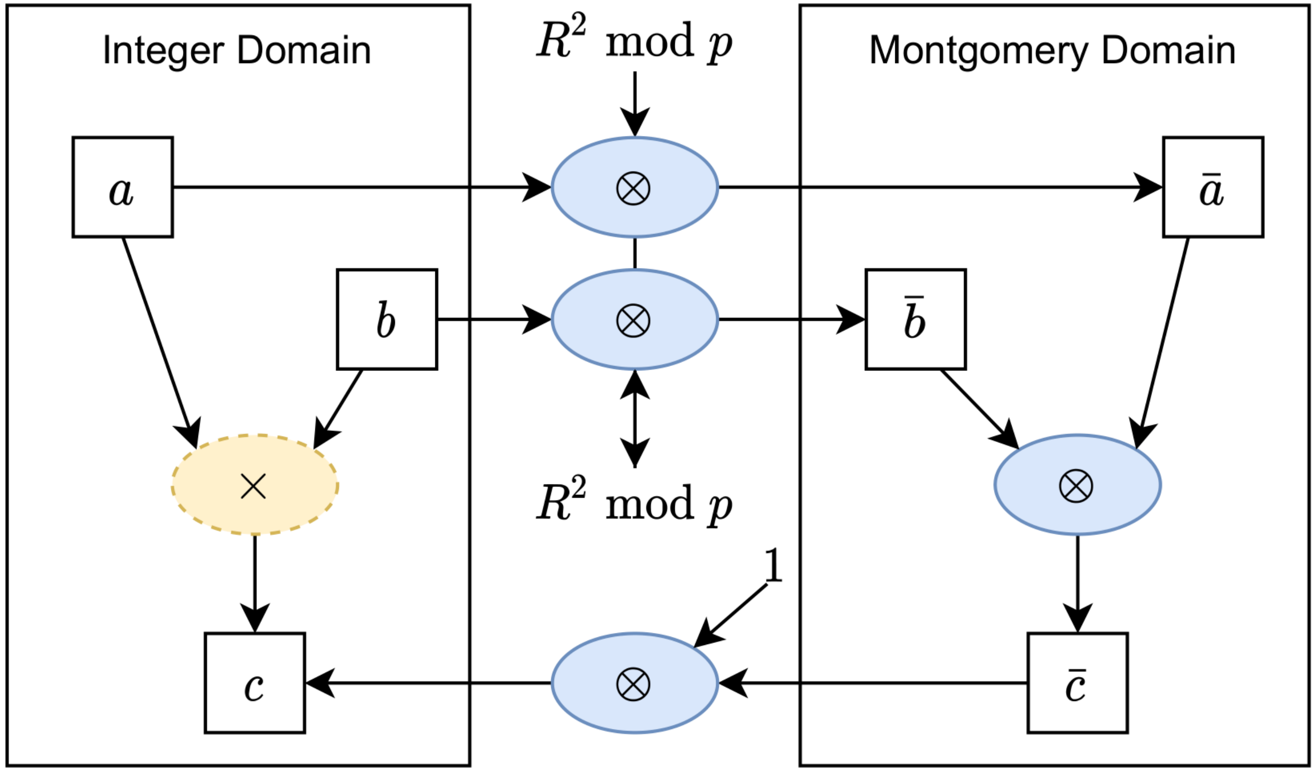
- How does an MCU accelerate RSA?

  Montgomery Modular Multiplication

- How do we compute RSA fast?

  Chinese Remainder Theorem

Preliminaries: Montgomery Modular Multiplication



Preliminaries: Montgomery Modular Multiplication

- The modulus is a ***k***-bit prime number ***p***.
- Let R=2^k.
- A number a in its Montgomery form is
$$\bar{a} = a \cdot R \bmod p$$
- The Montgomery Modular Multiplication is defined as
$$a \otimes b = a \cdot b \cdot R^{-1} \bmod p$$

Preliminaries: Montgomery Modular Multiplication

- With Montgomery modular multiplications
- Turn a number into Montgomery domain

$$\overline{a} = a \otimes R^2 = a \cdot R \bmod p$$

- Turn a number back

$$a = \overline{a} \otimes 1$$

Preliminaries: Chinese Remainder Theorem

RSA-4096

- Raw RSA
  - Public key: (p, q, e)
  - Private key: (p, q, d). Plaintext $m = M^d \bmod \boxed{N}$. ⟸ **4096-bit**
- RSA-CRT
  - Public key: (p, q, e)
  - Private key: $(p, q, d_p, d_q, q_{inv})$, where

    $d_p = d \bmod \boxed{(p-1)}$, $d_q = d \bmod \boxed{(q-1)}$, $q_{inv} = q^{-1} \bmod \boxed{p}$ ⟸ **2048-bit**

Preliminaries: Chinese Remainder Theorem

Algorithm 1 Private-key operation of RSA-CRT.

Require: message m, private key $(p, q, d_p, d_q, q_{inv})$

Ensure: $m^d \bmod N$

1: $S_p = m^{d_p} \bmod p$

2: $S_q = m^{d_q} \bmod q$ ⟸ **2048-bit**

3: $h = q_{inv} \cdot (S_p - S_q) \bmod p$

4: $S = S_q + h \cdot q \bmod N$ ⟸ **4096-bit**

5: **return** S

Algorithm

- Challenge 1: compute R^2, where $R = 2^{2048}$

$r = (R - 1) \oplus 1$

$r_1 = r \oplus r = 2 \cdot R \bmod p$

$r_2 = r_1 \otimes r_1 = 2^2 \cdot R \bmod p$

$r_3 = r_2 \otimes r_2 = 2^3 \cdot R \bmod p$

$\ldots$

$r_{2048} = r_{2047} \otimes r_{2047} = 2^{2048} \cdot R \bmod p$

Algorithm

- Challenge 2: compute $m^{d_p} \bmod p$

  Divide m into two parts: m_1(highest 2048 bits) & m_2(lowest 2048 bits) , i.e.,

  $$m = m_1 \cdot R + m_2$$

  $$m \bmod p = (m_1 \cdot R + m_2) \bmod p$$

  $$= \left(m_1 \otimes R^2\right) \oplus m_2$$

Algorithm

- Challenge 2: compute $m^{d_p} \bmod p$

Fast exponentiation with a constant time

Algorithm 3 A variant of the fast exponentiation algorithm.

Require: $\overline{m} = m \bmod p$, and d_p

Ensure: $m^{d_p} \bmod p$

$y = 1 \otimes R^2$

$t = \overline{m} \otimes R^2$

for $i = 1;\ i \leq 2048;\ i \leftarrow i + 1$ **do**

  if the rightmost bit of d_p is 1 **then**

    $y \leftarrow y \otimes t$

  else

    $dummy \leftarrow y \otimes t$

  end if

  $t \leftarrow t \otimes t$

  $d_p \leftarrow d_p >> 1$

end for

return $y \otimes 1$

Algorithm

$$S = S_q + h \cdot q \bmod N$$

- Challenge 3: compute $x \cdot y$, where x, y are 2048-bit numbers

Divide x, y into two parts, respectively:

x_1, y_1 (highest 1024 bits) &
x_2, y_2(lowest 1024 bits)

Let HI(x) denote highest 1024 bits of x,

LO(x) denote lowest 1024 bits of x.

THE COMPOSITION OF $x \cdot y$

4096~3073	3072~2049	2048~1023	1024~1
$\mathrm{HI}(x_1 y_1)$	$\mathrm{LO}(x_1 y_1)$ $\mathrm{HI}(x_1 y_2)$ $\mathrm{HI}(x_2 y_1)$	$\mathrm{LO}(x_1 y_2)$ $\mathrm{LO}(x_2 y_1)$ $\mathrm{HI}(x_2 y_2)$	$\mathrm{LO}(x_2 y_2)$

- Why can we use the MM to compute a normal multiplication?

Algorithm

- Why can we use the MM to compute a normal multiplication?
  - $R^{-1} \equiv 1 \bmod (R-1)$
  - $a \otimes b = a \cdot b \bmod (R-1)$
  - Since $a, b < 2^{1024}$, we have $a \cdot b < R - 1$

Complexity

Algorithm 1 Private-key operation of RSA-CRT.

Require: message m, private key $(p, q, d_p, d_q, q_{inv})$

Ensure: $m^d \bmod N$

1: $S_p = m^{d_p} \bmod p$ ⟵ **6148** $\otimes$ ops

2: $S_q = m^{d_q} \bmod q$ ⟵ **6148** $\otimes$ ops

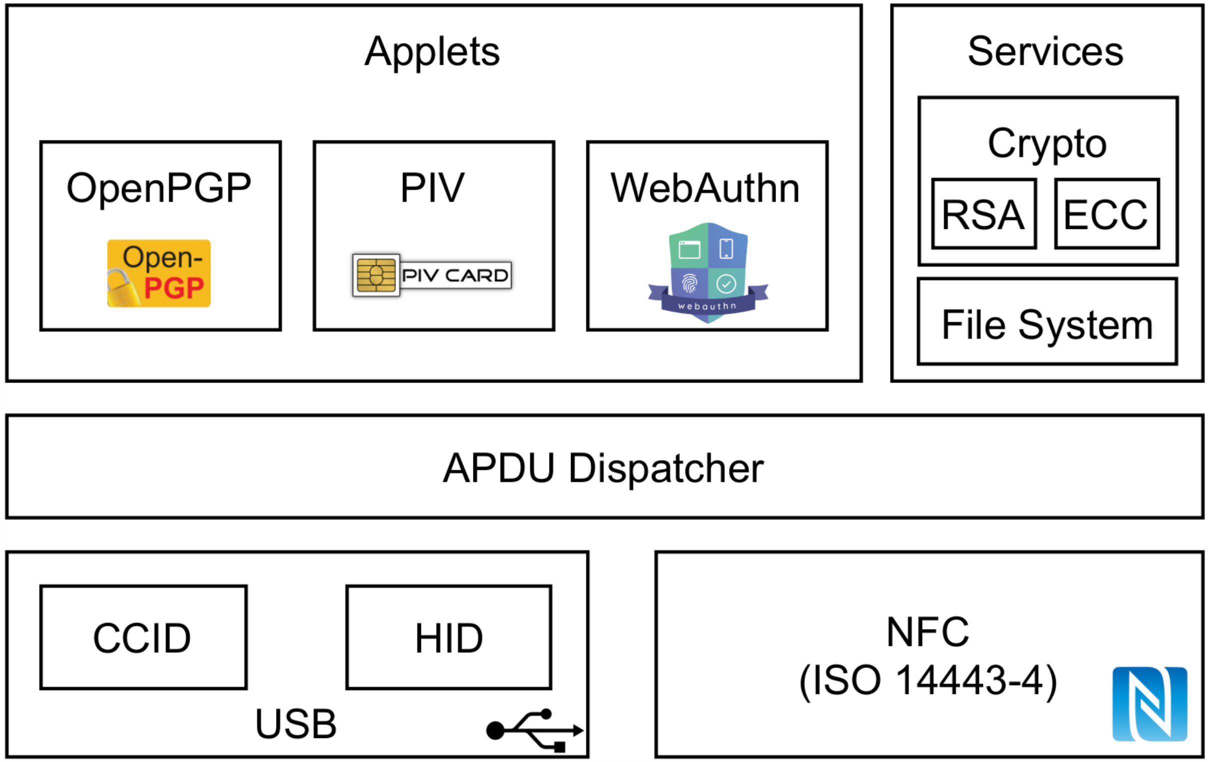
3: $h = q_{inv} \cdot (S_p - S_q) \bmod p$ ⟵ **4** $\otimes$ ops

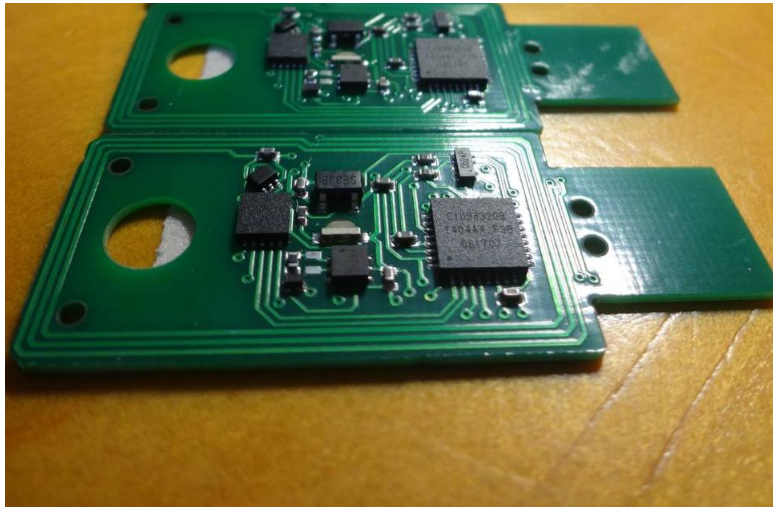
4: $S = S_q + h \cdot q \bmod N$ ⟵ **4** $\otimes$ ops

5: **return** S

12,304 $\otimes$ ops

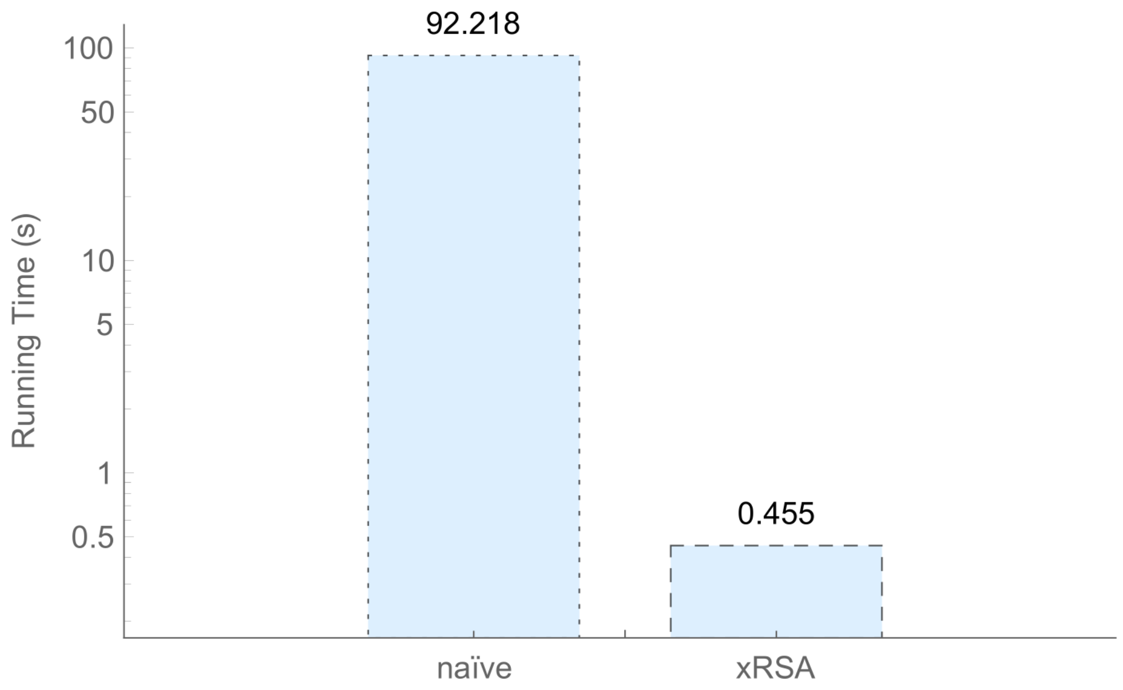
Implementation





https://github.com/canokeys

Evaluation



RSA-4096 performance on a 48 MHz MCU: 203x faster

RUNNING TIME OF SIGNING USING GNUPG

	CanoKey	YubiKey 5 NFC
Average running time	869 ms	670 ms

29.7% slower than the native RSA-4096 acceleration

```
## RSA4096 key import
Addkey 4 4096 # [10] gen RSA4096 key
Key2card 10 3 # key[10] to Authentication key
Addkey 6 4096 # [11] gen RSA4096 key
Key2card 11 2 # key[11] to Encryption key
GPGAuth
GPGEnc
Addkey 4 4096 # [12] gen RSA4096 key
Key2card 12 1 # key[12] to Signature key
GPGSign
```

Automated correctness test

Conclusion

- We design an algorithm that uses the most existing 2048-bit Montgomery modular multiplier to achieve a 4096-bit RSA cryptography mechanism without replacing any circuit component.
- We implement the 4096-bit RSA cryptography on an existing device, which is equipped with a 2048-bit Montgomery modular multiplier.
- Experiment results show that our method achieves the correct behavior of 4096-bit RSA cryptography, and makes it over 200x faster than the software-based solution.

Thanks!