

# Intro To Asymmetric Cryptography RPISEC

Avi Weinstock (aweinstock)

October 29, 2019

```
; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init          ; DATA XREF: _start+1A70
                          ; .text:init70

PUSH.W                  {R3-R9,LR}
MOV                     R7, R8
LDR                     R6, =(__do_global_ctors_aux_fini_array_entry - 0x104E0)
MOV                     R8, R1
LDR                     R5, =(__frame_dummy_init_array_entry - 0x104E8)
MOV                     R9, R2
ADD                     R6, PC ; __do_global_ctors_aux_fini_array_entry
BLX                     .init_proc
MOV                     R6, PC ; __frame_dummy_init_array_entry
SUBS                    R8, R6, R5
ASRS                    R6, R6, #2
BEQ                     locret_104F8
MOV                     R4, #0

loc_104E6                ; CODE XREF: __libc_csu_init+2E4j
                          ADDS                    R4, #1
                          LDR.W                   R3, [R5],#4
                          MOV                     R2, R9
                          MOV                     R1, R8
                          MOV                     R0, R7
                          BLX                     R3 ; frame_dummy
                          CMP                     R4, R6
                          BNE                     loc_104E6

locret_104F8             ; CODE XREF: __libc_csu_init+1A7j
                          POP.W                   {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC                DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
                          ; DATA XREF: __libc_csu_init+67r
off_10500                DCD __frame_dummy_init_array_entry - 0x104E8
                          ; DATA XREF: __libc_csu_init+A7r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini          ; DATA XREF: _start+1070
                          ; .text:off_1039870

BX                     LR
; End of function __libc_csu_fini

; -----
```

# Symmetric vs Asymmetric Cryptosystems

Examples of symmetric crypto:      Examples of asymmetric crypto:

- ▶ classical ciphers (caesar, vigenere)
- ▶ block ciphers (DES, AES)
- ▶ stream ciphers (OTP, RC4, Salsa20)
- ▶ hash functions (MD5, SHA256)
- ▶ MACs (HMAC, Poly1305)

- ▶ key exchange (Diffie-Hellman)
- ▶ encryption (RSA, ElGamal)
- ▶ signatures (RSA, DSA, Schnorr)
- ▶ homomorphic encryption (Paillier, RSA, Gentry)

```
; ===== SUBROUTINE =====
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1A70
; .text:01070

PUSH.W      {R3-R9,LR}
MOV         R7, R8
LDR         R6, =(_do_global_ctors_aux_fini_array_entry - 0x104E0)
MOV         R8, R1
MOV         R9, R2
MOV         R6, PC ; __do_global_ctors_aux_fini_array_entry
ADD         R5, PC ; __frame_dummy_init_array_entry
SUB         R6, R6, R5
PUSH        R6, R6, #2
BEQ         locret_104F8

loc_104E6:
; CODE XREF: __libc_csu_init+2E4j
MOV         R3, R6
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3 ; frame_dummy
BNE         loc_104E8

loc_104E8:
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC: DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
; DATA XREF: __libc_csu_init+64r
off_10500: DCD __frame_dummy_init_array_entry - 0x104E8
; DATA XREF: __libc_csu_init+47r

; ===== SUBROUTINE =====
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+1070
; .text:off_103980

BX          LR
; End of function __libc_csu_fini

; -----
```

# What is the significance of RSA?

- ▶ Historically the first asymmetric cryptosystem (published in 1977)
- ▶ Named for its inventors: Rivest, Shamir, Adleman
- ▶ Very flexible: can be used for both encryption and signing, is multiplicatively homomorphic
- ▶ Easy to implement
- ▶ Easy to mess up implementing, on account of its flexibility

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init ; DATA XREF: __start+1A70  
; .text:01A70  
  
PUSH.W {R3-R9,LR}  
MOV R7, R8  
LDR R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)  
MOV R8, R1  
LDR R5, =(__frame_dummy_init_array_entry - 0x104E8)  
MOV R9, R2  
ADD R6, PC ; __do_global_ctors_aux_fini_array_entry  
BLX .init_proc  
ADD R5, PC ; __frame_dummy_init_array_entry  
SUBS R0, R0, R0  
ADRS R0, R0, R0  
BEQ locret_104F8  
MOVVS R4, #0  
  
loc_104E6 ; CODE XREF: __libc_csu_init+2E4j  
R4, #1  
LDR.W R3, [R5], #4  
MOV R2, R9  
LDR R1, R0  
MOV R8, R7  
BLX R3 ; frame_dummy  
CMP R4, R6  
BNE loc_104E6  
  
locret_104F8 ; CODE XREF: __libc_csu_init+1A7j  
POP.W {R3-R9,PC}  
; End of function __libc_csu_init  
; *****  
off_104FC DCD __do_global_ctors_aux_fini_array_entry - 0x104DA  
; DATA XREF: __libc_csu_init+67r  
off_10500 DCD __frame_dummy_init_array_entry - 0x104E8  
; DATA XREF: __libc_csu_init+47r  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini ; DATA XREF: __start+1070  
; .text:0103980  
  
BX LR  
; End of function __libc_csu_fini  
; *****
```

# What is RSA?

- ▶  $p$  and  $q$  are distinct large primes
- ▶  $n = p * q$
- ▶  $\varphi(n) = (p - 1) * (q - 1)$
- ▶  $e * d \equiv 1 \pmod{\varphi(n)}$
- ▶  $(n, e)$  is the "public key"
- ▶  $(p, q, d)$  is the "private key"
- ▶  $\text{enc}(x) = \text{rem}(x^e, n)$
- ▶  $\text{dec}(x) = \text{rem}(x^d, n)$

```
; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init          ; DATA XREF: __start+1A70
                          ; .text:init70

        PUSH.W           {R3-R9,LR}
        MOV              R7, R8
        LDR              R6, =(__do_global_ctors_aux_fini_array_entry - 0x104E0)
        MOV              R8, R1
        LDR              R5, =(__frame_dummy_init_array_entry - 0x104E0)
        MOV              R9, R2
        ADD              R6, PC ; __do_global_ctors_aux_fini_array_entry
        BLX              .init_proc
        ADD              R5, PC ; __frame_dummy_init_array_entry
        SUBS              R6, R6, R5
        ASRS              R6, R6, #2
        BEQ              locret_104F8
        MOVS              R4, #0

loc_104E6                ; CODE XREF: __libc_csu_init+2E4j
        ADDS              R4, #1
        LDR.W            R3, [R5], #4
        MOV              R2, R9
        MOV              R1, R8
        MOV              R0, R7
        BLX              R3 ; frame_dummy
        CMP              R4, R6
        BNE              loc_104E6

locret_104F8             ; CODE XREF: __libc_csu_init+1A7j
        POP.W            {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC               DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
                        ; DATA XREF: __libc_csu_init+64r
off_10500               DCD __frame_dummy_init_array_entry - 0x104E0
                        ; DATA XREF: __libc_csu_init+A7r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini          ; DATA XREF: __start+1070
                          ; .text:off_1039870

        BX              LR
; End of function __libc_csu_fini

; -----
```

►  $\forall x, y \exists q, r (y * q \overset{MWS}{+} r \overset{R4, R8}{=} x)$

```
loc_104E6          ; CODE XREF: __libc_csu_init+2E4j
```

```

        ADDS      R4, #1
        LDR, W    R3, [R5], #4
        MOV       R2, R9
        MOV       R1, R8
        MOV       R0, R7
        BLX       R3          ; frame_dummy
        CNP       R4, R6
        BNE       loc_104E6

```

```

3 * 27 + 1 = 82  locret 10458 ; CODE XREF: __libc_csu_init+1A7j
                  POP.W    {R3-R9,PC}
                  ; End of function  libc csu init

```

# Modular congruence and primes

Divides cleanly/Is divisor of:

- ▶  $\text{divides}(x, y) \leftrightarrow \text{rem}(y, x) = 0$
- ▶ e.g.  $\text{divides}(5, 10)$ , since  $10 \% 5 == 0$

Modular congruence:

- ▶  $x \equiv y \pmod{n} \leftrightarrow \text{divides}(x - y, n)$
- ▶  $x \equiv y \pmod{n} \leftrightarrow \text{rem}(x, n) = \text{rem}(y, n)$

Primeness:

- ▶  $\text{prime}(p) \leftrightarrow \forall k \in [2, p - 1](\neg \text{divides}(k, p))$

- ▶ `def prime(p):`  
    `return all([p % k != 0 for k in range(2,p)])`

`assert filter(prime, range(2, 20)) == [2, 3, 5, 7, 11, 13, 17, 19]`

Greatest Common Divisor:

- ▶  $\text{gcd}(x, y) = \max\{k | \text{divides}(k, x) \wedge \text{divides}(k, y)\}$

- ▶ `def gcd(x, y):`  
    `return max([1]+[k for k in range(1, x*y) if x % k == 0 and y % k == 0])`

```
; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init      ; DATA XREF: __start+1A70
                    ; .text:00170

PUSH.W              {R3-R9,LR}
MOV                 R7, R8
LDR                 R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)
MOV                 R8, R1
LDR                 R5, =(_frame_dummy_init_array_entry - 0x184E8)
MOV                 R9, R2
ADD                 R6, PC ; __do_global_ctors_aux_fini_array_entry
BLX                 .init_proc
ADD                 R5, PC ; _frame_dummy_init_array_entry
SUBS                 R6, R6, R5
ASRS                 R6, R6, #2
BEQ                 locret_104F8
MOVS                 R4, #0

loc_104E6            ; CODE XREF: __libc_csu_init+2E4j
ADDS                 R4, #1
LDR.W               R3, [R5], #4
MOV                 R2, R9
MOV                 R1, R8
MOV                 R0, R7
BLX                 R3 ; frame_dummy
CNR                 R4, R6
B.E                 loc_104E6

locret_104F8         ; CODE XREF: __libc_csu_init+1A7j
POP.W               {R3-R9,PC}
; End of function __libc_csu_init

off_104F8            DCD __do_global_ctors_aux_fini_array_entry - 0x184DA
                    ; DATA XREF: __libc_csu_init+67r
off_10500            DCD _frame_dummy_init_array_entry - 0x184E8
                    ; DATA XREF: __libc_csu_init+A7r

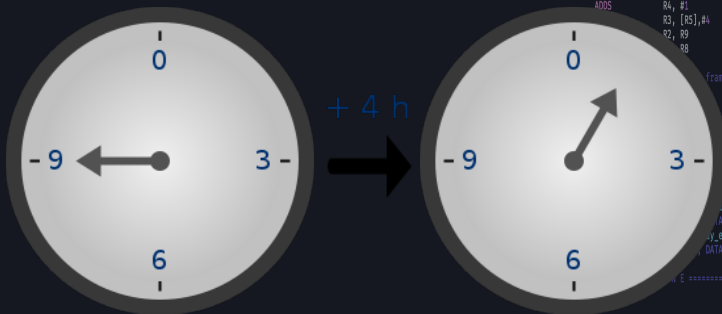
; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini      ; DATA XREF: __start+1070
                    ; .text:off_1039800

BX                 LR
; End of function __libc_csu_fini
```

# Modular arithmetic

- ▶ Let  $\mathbb{Z}_n$  denote  $\{\text{rem}(x, n) \mid x \in \mathbb{Z}\}$ , or equivalently,  $[0, n)$ .
- ▶ We can truncate addition and multiplication to work within  $\mathbb{Z}_n$  by calculating remainders after each operation.
- ▶ "Clock arithmetic": in  $\mathbb{Z}_{12}$ ,  $9 + 4 = 1$ , since  $\text{rem}(9 + 4, 12) = \text{rem}(13, 12) = 1$ .



```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1A70
; .text:00000000

PUSH.W      {R3-R9,LR}
MOV         R7, R8
LDR         R6, =(&_do_global_ctors_aux_fini_array_entry - 0x104E0)
MOV         R8, R7
MOV         R5, =(&_frame_dummy_init_array_entry - 0x104E0)
MOV         R9, R2
MOV         R6, PC
ADD         R5, PC
ADD         R6, R5
PUSH        R6, R5
ADDS        R6, R6, #2
BEQ         locret_104F8
; loc_104E6
; CODE XREF: __libc_csu_init+2E4j
ADDS        R4, #1
R3, [R5], #4
R2, R9
R8
; frame_dummy
; XREF: __libc_csu_init+1A7j
; _array_entry - 0x104DA
; XREF: __libc_csu_init+6tr
; _y_entry - 0x104E0
; DATA XREF: __libc_csu_init+4tr
; E *****

EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+1070
; .text:00000000

BX         LR
; End of function __libc_csu_fini

```

# Addition and Multiplication mod 7

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init                ; DATA XREF: __start+1A70
                                ; .text:init70

                                PUSH.W    {R3-R9,LR}
                                MOV     R7, R8
                                LDR     R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)
                                MOV     R8, R1
                                LDR     R5, =(__frame_dummy_init_array_entry - 0x104E8)
                                MOV     R9, R2
                                ADD     R6, PC ; __do_global_ctors_aux_fini_array_entry
                                BLX     .init_proc
                                ADD     R5, PC ; __frame_dummy_init_array_entry
                                SUBS    R0, R6, R8
                                BEQ     loc_104E6
                                MOVS    R4, #0
                                loc_104E6
                                ADDS    R4, #1
                                LDR.W   R5, [R5], #4
                                MOV     R2, R9
                                MOV     R4, R8
                                MOV     R7, R7
                                BLX     R3 ; frame_dummy
                                CMP     R6, R6
                                BNE     loc_104E8
                                loc_104E8
                                POP.W   {R3-R9,PC}
                                ; End of function __libc_csu_init

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini                ; DATA XREF: __start+1070
                                ; .text:off_10398070

                                BX      LR
; End of function __libc_csu_fini

```



# Addition and Multiplication mod 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init                ; DATA XREF: _start+1A70
                                ; .text:0x1A70

                                PUSH.W    {R3-R9,LR}
                                MOV        R7, R8
                                LDR        R6, =(__do_global_ctors_aux_fini_array_entry - 0x104E8)
                                MOV        R8, R1
                                LDR        R5, =(__frame_dummy_init_array_entry - 0x104E8)
                                MOV        R9, R2
                                ADD        R6, PC, ; __do_global_ctors_aux_fini_array_entry
                                BLX        __init_pro
                                ADD        R3, PC, ; __frame_dummy_init_array_entry
                                SUBS       R6, R6, R5
                                ASRS       R6, R6, #1
                                BEQ        locret_104F8
                                MOVS      R1, #0
                                loc_104E6    ; CODE XREF: __libc_csu_init+2E4j
                                ADDS       R3, #1
                                LDR.W      R3, [R5], #4
                                MOV        R9, R9
                                MOV        R8, R8
                                MOV        R0, R7
                                BLX        __frame_dummy
                                CMP        R4, R6
                                BNE        loc_104E6
                                locret_104F8    ; CODE XREF: __libc_csu_init+1A7j
                                POP.W      {R3-R9,PC}
; End of function __libc_csu_init

off_104FC    DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
                                ; DATA XREF: __libc_csu_init+67r
off_10500    DCD __frame_dummy_init_array_entry - 0x104E8
                                ; DATA XREF: __libc_csu_init+47r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini                ; DATA XREF: _start+1B70
                                ; .text:off_103980

                                BX         LR
; End of function __libc_csu_fini

```

# Multiplicative inverses

- ▶ In  $\mathbb{Q}$  and  $\mathbb{R}$ , inverses exist everywhere except zero:  $x * \frac{1}{x} = 1$
- ▶ In  $\mathbb{Z}$ , inverses only exist at 1
- ▶ In  $\mathbb{Z}_p$ , inverses exist everywhere except zero, and can be found via fermat's little theorem (e.g.  $3 * 4 \equiv 12 \equiv 1 \pmod{11}$ )
- ▶ In  $\mathbb{Z}_n$ , multiples of factors of  $n$  act as additional zeros for the purposes of not having an inverse; when they exist, they can be found via an extension of the algorithm for euclidean division

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1A70
; .text:00000000

PUSH.W      {R3-R9,LR}
MOV         R7, R8
LDR         R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry - 0x104E8)
MOV         R9, R2
ADD         R6, PC, ; __do_global_ctors_aux_fini_array_entry
BLX         __init_ctors
; DATA XREF: __frame_dummy_init_array_entry
SUBS        R6, R6, R5
ASRS        R6, R6, #2
BEQ         locret_104F8
MOV         R4, #0

locret_104F8:
; DATA XREF: __frame_dummy_init_array_entry
ADD         R4, #1
LDR.W       R3, [R5], #4
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3, ; frame_dummy
CMP         R4, R6
BEQ         locret_104F8
; DATA XREF: __frame_dummy_init_array_entry
PUSH.W      {R3-R9,LR}
; DATA XREF: __start+1A70
; .text:00000000
; End of function __libc_csu_init

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+1070
; .text:00000000

BX         LR
; End of function __libc_csu_fini

```



# Multiplication mod 13

$$x * x^{p-2} \equiv 1 \pmod{p}$$

*	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12
2	0	2	4	6	8	10	12	1	3	5	7	9	11
3	0	3	6	9	12	2	5	8	11	1	4	7	10
4	0	4	8	12	3	7	11	2	6	10	1	5	9
5	0	5	10	2	7	12	4	9	1	6	11	3	8
6	0	6	12	5	11	4	10	3	9	2	8	1	7
7	0	7	1	8	2	9	3	10	4	11	5	12	6
8	0	8	3	11	6	1	9	4	12	7	2	10	5
9	0	9	5	1	10	6	2	11	7	3	12	8	4
10	0	10	7	4	1	11	8	5	2	12	9	6	3
11	0	11	9	7	5	3	1	12	10	8	6	4	2
12	0	12	11	10	9	8	7	6	5	4	3	2	1

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init      ; DATA XREF: _start+1A70
                    ; .text:1A70

PUSH.W              {R3-R9,LR}
MOV                 R7, R8
LDR                 R6, =(&__do_global_ctors_aux_fini_array_entry - 0x18)
MOV                 R8, R1
LDR                 R5, =(&_frame_dummy_init_array_entry - 0x104E8)
MOV                 R9, R2
ADD                 R6, PC, ; __do_global_ctors_aux_fini_array_entry
BLX                 .init_proc
ADD                 R5, PC, ; _frame_dummy_init_array_entry
SUBS                 R6, R6, R5
ASRS                 R6, R6, #2
BEQ                 locret_104F8
MOV                 R4, #8

                    ; CODE XREF: __libc_csu_init+2E4j
ADDS                 R4, #1
LDR.W               R3, [R5], #4
MOV                 R2, R9
MOV                 R1, R8
MOV                 R0, R7
BLX                 R3, ; frame_dummy
CMP                 R4, R6
BNE                 loc_104E6

loc_104E6:
                    ; CODE XREF: __libc_csu_init+1A7j
POP.W               {R3-R9,PC}
; End of function __libc_csu_init
;-----
f_104FC: DCD &__do_global_ctors_aux_fini_array_entry - 0x104DA
                    ; DATA XREF: __libc_csu_init+67r
ff_10500: DCD &_frame_dummy_init_array_entry - 0x104E8
                    ; DATA XREF: __libc_csu_init+A7r
;-----
***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini      ; DATA XREF: _start+1070
                    ; .text:off_103980

BX                 LR
; End of function __libc_csu_fini

```

# Exponentiation mod 13

$$x * x^{p-2} \equiv 1 \pmod{p}$$

$x^y$	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	3	6	12	11	9	5	10	7	1
3	1	3	9	1	3	9	1	3	9	1	3	9	1
4	1	4	3	12	9	10	1	4	3	12	9	10	1
5	1	5	12	8	1	5	12	8	1	5	12	8	1
6	1	6	10	8	9	2	12	7	3	5	4	11	1
7	1	7	10	5	9	11	12	6	3	8	4	2	1
8	1	8	12	5	1	8	12	5	1	8	12	5	1
9	1	9	3	1	9	3	1	9	3	1	9	3	1
10	1	10	9	12	3	4	1	10	9	12	3	4	1
11	1	11	4	5	3	7	12	2	9	8	10	6	1
12	1	12	1	12	1	12	1	12	1	12	1	12	1

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init      ; DATA XREF: _start+1A70
                    ; .text:1A70

PUSH.W              {R3-R9,LR}
MOV R7, R8
LDR R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)
MOV R8, R1
LDR R5, =(_frame_dummy_init_array_entry - 0x184E8)
MOV R9, R2
ADD R6, PC, ; __do_global_ctors_aux_fini_array_entry
BLX .init_proc
ADD R5, PC, ; _frame_dummy_init_array_entry
SUBS R6, R6, R5
ASRS R6, R6, #2
BEQ locret_104F8
MOVS R4, #8

                    ; CODE XREF: __libc_csu_init+2E4j
ADDS R4, #1
LDR.W R3, [R5], #4
MOV R2, R9
MOV R1, R8
MOV R0, R7
BLX R3, ; frame_dummy
CMP R4, R6
BNE loc_104E6

                    ; CODE XREF: __libc_csu_init+1A7j
POP.W              {R3-R9,PC}
; End of function __libc_csu_init
;
;-----
; f_104FC          DCD __do_global_ctors_aux_fini_array_entry - 0x184DA
;                  ; DATA XREF: __libc_csu_init+67r
; ff_10500         DCD _frame_dummy_init_array_entry - 0x184E8
;                  ; DATA XREF: __libc_csu_init+A7r
;-----
; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini      ; DATA XREF: _start+1070
                    ; .text:off_103980

BX LR
; End of function __libc_csu_fini

```

# Euler's totient theorem and phi

- ▶  $\gcd(x, n) = 1 \rightarrow x^{\varphi(n)} \equiv 1 \pmod{n}$
- ▶  $\varphi(n) = |\{k | 1 \leq k < n \wedge \gcd(k, n) = 1\}|$
- ▶ def phi(n):  
    return len([k for k in range(1, n) if gcd(k, n) == 1])
- ▶ This definition is inefficient to calculate (linear in the value of  $n$ , so exponential in the bitlength of  $n$ )

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init
    ; DATA XREF: __start+1A70
    ; .text:01A70

    PUSH.W    {R3-R9,LR}
    MOV       R7, R8
    LDR       R6, =(__do_global_ctors_aux_fini_array_entry - 0x104E0)
    MOV       R8, R1
    LDR       R5, =(__frame_dummy_init_array_entry - 0x104E0)
    MOV       R9, R2
    ADD       R6, PC ; __do_global_ctors_aux_fini_array_entry
    BLX       .init_proc
    ADD       R5, PC ; __frame_dummy_init_array_entry
    SUBS      R6, R6, R5
    ASRS      R6, R6, #2
    BEQ       locret_104F8
    MOVS      R4, #0

    ; CODE XREF: __libc_csu_init+2E4j
    ADDS      R4, #1
    LDR.W     R3, [R5], #4
    MOV       R2, R9
    MOV       R1, R7
    MOV       R0, R7
    BLX       R3 ; frame_dummy
    CND       R4, R6
    BNE       loc_104F8

locret_104F8
    ; CODE XREF: __libc_csu_init+1A7j
    POP.W     {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC    DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
            ; DATA XREF: __libc_csu_init+64r
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
            ; DATA XREF: __libc_csu_init+47r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini
    ; DATA XREF: __start+1070
    ; .text:off_103980

    BX        LR
; End of function __libc_csu_fini

```

# Fundamental Theorem of Arithmetic

- Any integer can be decomposed (uniquely) into a product of prime powers
- $\forall n \exists \vec{v} (n = \prod_{i=1}^{len(\vec{v})} p_i^{v_i})$
- e.g. for  $n = 84 = 2 * 42 = 2^2 * 21 = 2^2 * 3 * 7$ ,  $\vec{v} = [2, 1, 0, 1]$
- Computing  $\vec{v}$  from  $n$  is expensive (algorithms like GNFS and ECM find  $\vec{v}$  slower than polytime in the bitlength of  $n$ )
- Computing  $n$  from  $\vec{v}$  is fast ( $p_i$  can be found via sieving, and multiplication is cheap)
- When coding, a sparse representation of the prime vector is more convenient:

```

histogram = lambda s: (lambda d: [[d.__setitem__(c, d.get(c, 0)+1) for c in s]](dict()))
product = lambda xs: reduce(__import__('operator').mul, xs, 1)
assert product([2,2,3,7]) == 84
assert histogram([2,2,3,7]) == {2: 2, 3: 1, 7: 1}

```

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: _start+1A70
; .text:01070

PUSH.W      {R3-R9,LR}
MOV         R7, R8
LDR         R6, =(_do_global_dtors_aux_fini_array_entry - 0x104E8)
MOV         R8, R1
MOV         R9, R2
ADD         R6, PC, ; _do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC, ; _frame_dummy_init_array_entry
SUBS       R6, R6, R5
ASRS       R6, R6, #2
BEQ         locret_104F8
MOV         R4, #0
loc_104E6:
ADDS       R4, #1
RSC.W      R3, R3, #1
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3, ; frame_dummy
POP         R6
BNE        loc_104E0
locret_104F8
; CODE XREF: __libc_csu_init+2E4j
POP.W      {R3-R9,PC}
; End of function __libc_csu_init
;
;-----
off_104FC: DCD _do_global_dtors_aux_fini_array_entry - 0x104DA
; DATA XREF: __libc_csu_init+67r
off_10500: DCD _frame_dummy_init_array_entry - 0x104F4
; DATA XREF: __libc_csu_init+61r
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+1070
; .text:off_1039800
BX         LR
; End of function __libc_csu_fini
;
;-----

```

```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1A70
; .text:initt0

PUSH.W    {R3-R9,LR}
MOV       R7, R0
LDR       R6, =( __do_global_ctors_aux_fini_array_entry - 0x104D0)
MOV       R8, R1
LDR       R5, =( __frame_dummy_init_array_entry - 0x104E0)
MOV       R0, R2
MOV       R1, R3
BLK       .init_proc
ADD       R5, PC, __frame_dummy_init_array_entry
SUBS     R6, R6, R5
ASRS     R6, R6, #2
BEQ      locret_104F8
MOVW     R4, #0
```

- ```

=1.  $\prod_{i=1}^{\text{len}(v)} p_i^{v_i-1}$ 
MOV R4, #1
CALL 0x104AFC ; [R6,R4]
MOV R4, #9
MOV R1, R8
MOV R0, R7
BZL R3, #0 ; frame_dummy
CMO R4, R6
104AE6

__toret_104AF6
get(c,(c)+1) for c in s], d[-1])(dict))
; xs if function __libc_csu_init
; {R3-R9,PC}

program(factors_of_n).items())
off_104AFC DCD __do_global_ctors_aux._ini_array_entry - 0x104DA
; DATA XREF: __libc_csu_init+6tr
24ff_10508 DCD _frame_dummy_init_array_entry - 0x104E0
; DATA XREF: libc.csu.init+Atr

```



- ```
EXPORT __libc_csu_init
; DATA XREF: __start+1A70
; .text:initfo

PUSH.W      {R3-R9,LR}
MOV         R7,R0
LDR         R6,=(__do_global_dtors_aux_fini_array_entry - 0x104D0)
MOV         R8,R1
LDR         R5,=(__frame_dummy_init_array_entry - 0x104E0)
MOV         R9,R2
ADD         R6,PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5,PC ; __frame_dummy_init_array_entry
SUBS       R6,R6,R5
ASRS       R6,R6,#2
BEQ        locret_104F8
MOVS       R4,R0

loc_104E6                                ; CODE XREF: __libc_csu_init+2E6j
ADD        R4,#1
LDR.W      R3,[R5],#4
MOV        R2,R9
MOV        R1,R8
MOV        R0,R7
BLX        R3 ; frame_dummy
BNE        loc_104E6

loc_104F8                                ; CODE XREF: __libc_csu_init+3A7j
FOR.W      {R3-R5,PC}

; End of function __libc_csu_init

;-----
off_104FC   DCD __do_global_dtors_aux_fini_array_entry - 0x104D0
; DATA XREF: __libc_csu_init+67r
off_10500   DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init+A7r

] == [1, 4, 5, 2, 3, 6]
;----- SUBROUTINE -----
] == [1, 4, 5, 2, 3, 6]

EXPORT __libc_csu_fini
__libc_csu_fini                        ; DATA XREF: __start+1070
; .text:off_103980

BX        LR

; End of function __libc_csu_fini
;-----
```

```
LDR.W    R3, [R5],#4
```

```
BLX      R3      ; f
```

## ideal algorithm

the bezout coeff

SECRET 10150

from bezout coef

```
off 10500      DCD    frame dummy init arra
```

$$] == [1 \ 4 \ 5 \ 2$$

```

] ===== SUBROUTINE =====

```

```
] == [1, 4, 5, 2,
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶

# Correctness of RSA<sup>1</sup>

- ▶  $e * d \equiv 1 \pmod{\varphi(n)}$
- ▶  $\text{enc}(x) \equiv x^e \pmod{n}$
- ▶  $\text{dec}(x) \equiv x^d \pmod{n}$
- ▶  $\text{dec}(\text{enc}(x)) \equiv (x^e)^d \equiv x^{e*d} \equiv x^{\text{Euler}} \equiv x^1 \equiv x \pmod{n}$

```

; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init          ; DATA XREF: __start+1A70
                        ; .text:init70

PUSH.W                  {R3-R9,LR}
MOV                     R7, R8
LDR                     R6, =(__do_global_ctors_aux_fini_array_entry - 0x10)
MOV                     R8, R1
LDR                     R5, =(__frame_dummy_init_array_entry - 0x104E8)
MOV                     R9, R2
ADD                     R6, PC ; __do_global_ctors_aux_fini_array_entry
BLX                     .init_proc
ADD                     R5, PC ; __frame_dummy_init_array_entry
SUBS                     R6, R6, R5
ASRS                     R6, R6, #2
BEQ                     locret_104F8
MOV                     R4, #0

loc_104E6                ; CODE XREF: __libc_csu_init+2E4j
                        ADDS                     R4, #1
                        LDR.W                     R3, [R5], #4
                        MOV                     R2, R9
                        MOV                     R1, R8
                        MOV                     R0, R7
                        BLX                     R3 ; frame_dummy
                        DW                     0
                        BNE                     loc_104E6

locret_104F8             ; CODE XREF: __libc_csu_init+1A7j
                        POP.W                     {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC                DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
                        ; DATA XREF: __libc_csu_init+64r
off_10500                DCD __frame_dummy_init_array_entry - 0x104E8
                        ; DATA XREF: __libc_csu_init+A7r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini          ; DATA XREF: __start+1070
                        ; .text:off_1039870

BX                     LR
; End of function __libc_csu_fini

```

<sup>1</sup>Assumes  $\gcd(x, n) = 1$ , full proof is a little bit more involved

# Exploiting RSA: Cube root of small message

```
▶ from codecs import encode
from gmpy import invert, next_prime
import os
d = 0
while d == 0:
    p = next_prime(int(encode(os.urandom(1024/8), 'hex'), 16))
    q = next_prime(int(encode(os.urandom(1024/8), 'hex'), 16))
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 3
    d = invert(e, phi)
```

```
message = int(encode('hello', 'hex'), 16)
ciphertext = pow(message, e, n)
assert pow(ciphertext, d, n) == message
assert round(ciphertext**(1.0/3)) == message
```

```
; ===== SUBROUTINE =====
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: _start+1A70
; .text:01A70
PUSH.W {R3-R9,LR}
MOV R7, R8
LDR R6, =(__do_global_ctors_aux_fini_array_entry - 0x18)
MOV R8, R1
LDR R5, =(__frame_dummy_init_array_entry - 0x184E8)
MOV R9, R2
ADD R6, PC, __do_global_ctors_aux_fini_array_entry
BLX .init_proc
ADD R5, PC, __frame_dummy_init_array_entry
SUBS R6, R6, R5
ASRS R6, R6, #2
BEQ locret_104F8
MOVVS R4, #8

locret_104F8
; CODE XREF: __libc_csu_init+2E4j
POP.W {R3-R9,PC}
; End of function __libc_csu_init
; -----
off_104FC DCD __do_global_ctors_aux_fini_array_entry - 0x184DA
; DATA XREF: __libc_csu_init+64r
off_10508 DCD __frame_dummy_init_array_entry - 0x184E8
; DATA XREF: __libc_csu_init+47r
; ===== SUBROUTINE =====
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+1070
; .text:off_103980
BX LR
; End of function __libc_csu_fini
; -----
```

# Resources

- ▶ [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
- ▶ [https://en.wikipedia.org/wiki/Extended\\_Euclidean\\_Algorithm](https://en.wikipedia.org/wiki/Extended_Euclidean_Algorithm)
- ▶ [https://en.wikipedia.org/wiki/Modular\\_arithmetic](https://en.wikipedia.org/wiki/Modular_arithmetic)
- ▶ <https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf>
- ▶ <https://cryptopals.com/>, Sets 5 and 6

```
; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init          ; DATA XREF: __start+1A70
                          ; .text:init70

PUSH.W                  {R3-R9,LR}
MOV                     R7, R8
LDR                     R6, =(__do_global_ctors_aux_fini_array_entry - 0x104E0)
MOV                     R8, R1
LDR                     R5, =(__frame_dummy_init_array_entry - 0x104E0)
MOV                     R9, R2
ADD                     R6, PC ; __do_global_ctors_aux_fini_array_entry
BLX                     .init_proc
ADD                     R5, PC ; __frame_dummy_init_array_entry
SUBS                     R6, R6, R5
ASRS                     R6, R6, #2
PUSH.W                  {R4, #0}
MOV                     R4, #0
; loc_104E6
; CODE XREF: __libc_csu_init+2E4j
ADD                     R4, #1
LDR.W                   R3, [R5], #4
MOV                     R7, R9
MOV                     R1, R8
MOV                     R8, R7
PUSH.W                  {R3}
; frame_dummy
; loc_104E8
BNE                     loc_104E8
; CODE XREF: __libc_csu_init+1A7j
POP.W                   {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC DCD __do_global_ctors_aux_fini_array_entry - 0x104DA
          ; DATA XREF: __libc_csu_init+64r
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0
          ; DATA XREF: __libc_csu_init+47r

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini          ; DATA XREF: __start+1070
                          ; .text:off_103980

BX                     LR
; End of function __libc_csu_fini

; -----
; -----
```