Implementation memo rev.1

Toshiba Corporation
Hideo Shimizu

Domain parameters*

parameter	symbol	description
dX	DX	Total degree of public key polynomial
dr	DR	Total degree of encryption random (1)
q	Q	Prime number for base field
ℓ	L	Encode parameter
n	N	Degree of residue class polynomial Prime number*
mlen	MLEN	Message byte size

Above parameter is defined in parameter.h

*rev.1

variables

variable	type	description
M	OS	message to encrypt
C	OS	ciphertext
PK	OS	public key
SK	OS	secret key
m	R_ℓ	decoded message
c	Pq(dX+dr)	decoded ciphertext
ux	R_ℓ	decoded secret key (1)
uy	R_ℓ	decoded secret key (2)
X	Pq(dX)	decoded public key
r	Pq(dr)	encryption random (1)
e	P_ℓ(dX+dr)	encryption random (2)

※OS : Octet String

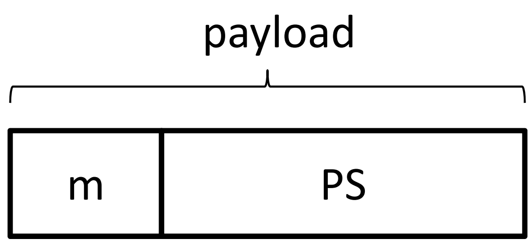
size*

|ℓ|		bit size of ℓ (= 2)
|q|		byte size of q (= 4)
#terms(d)	(d + 1)(d + 2) / 2	Number of terms of bivariate polynomial with total degree d
ux	|ℓ| * n	secret key 1 (bit size), ceil(n / 4) bytes*
uy	|ℓ| * n	secret key 2 (bit size), ceil(n / 4) bytes*
(ux, uy)	2*|ℓ| * n	secret key (bit size), 2*ceil(n / 4) bytes*
X	|q|* #terms(dX) * n	public key (byte size)
payload	|ℓ| * n	The number of bits that IEC can encrypt, ceil(n / 4) bytes*
C	|q|*#terms(dX+dr) * n	ciphertext (byte size)

*rev.1

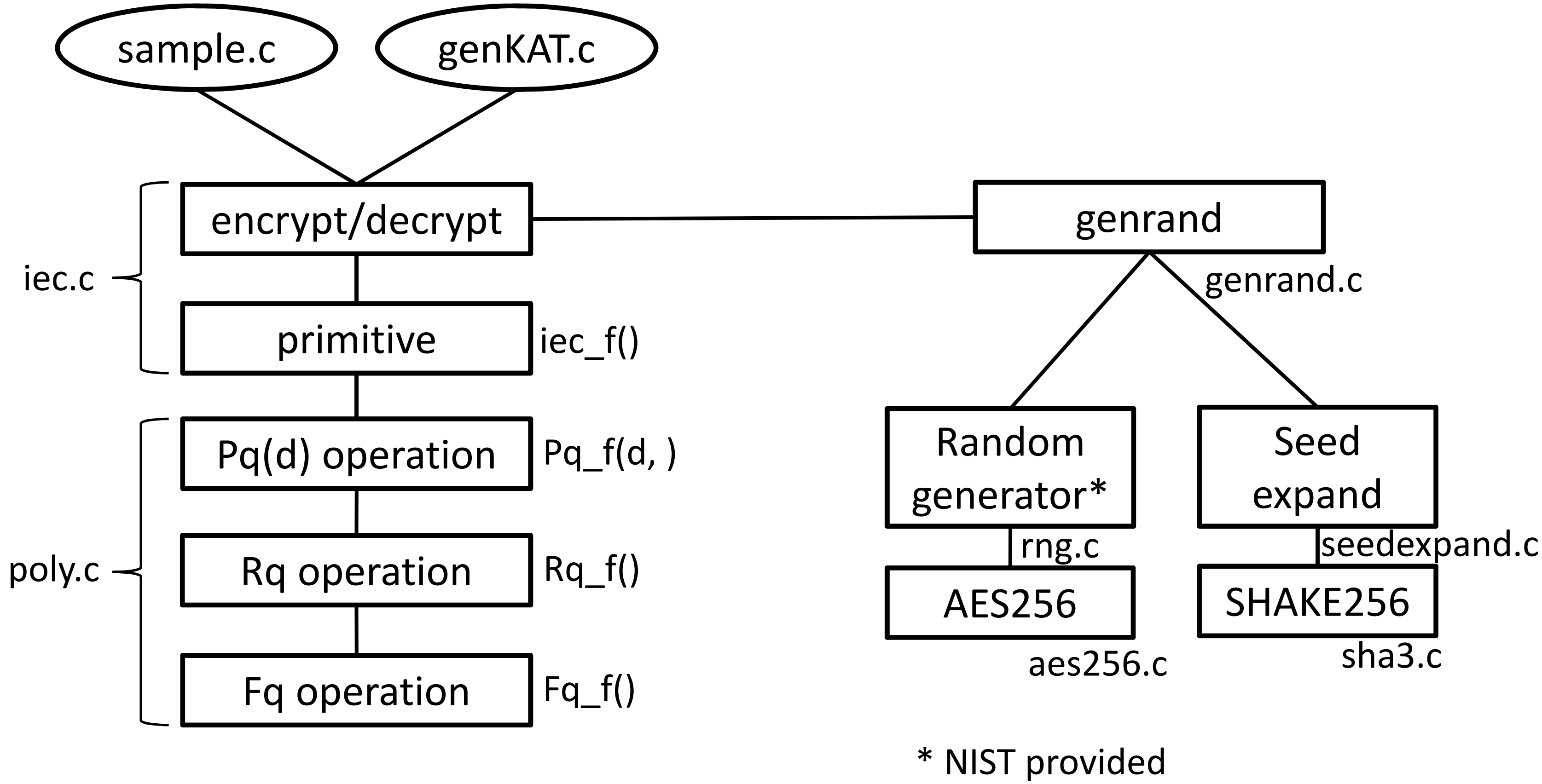
Parameters*

Security	n	m	sk	pk	c	payload	PS
128bit	1201	16byte	602byte	14412byte	28824byte	301byte	285byte
192bit	1733	24byte	868byte	20796byte	41592byte	434byte	410byte
256bit	2267	32byte	1134byte	27204byte	54408byte	567byte	535byte
			2*ceil(n/4)	12*n	24*n	ceil(n/4)	payload-m



*rev.1

Software layer



Benchmark*

Reference implementation

	keygen	encrypt	decrypt
IEC384	92909566	178456036	335353573
IEC448	160497017	378860493	716243384
IEC512	239510004	626677271	1186128486

(cycles)

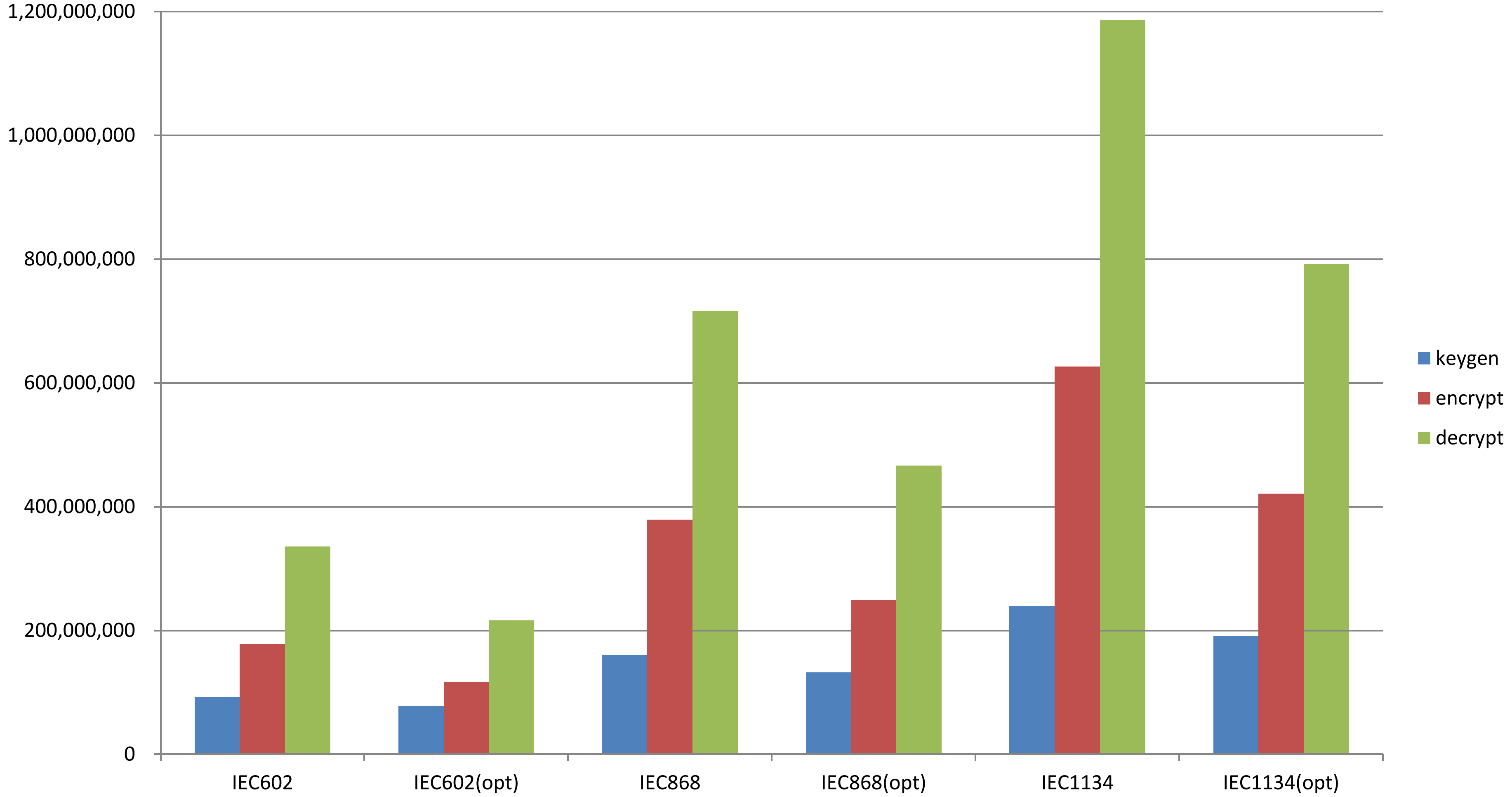
Optimized implementation

	keygen	encrypt	decrypt
IEC384	78272627	116773401	216049724
IEC448	131971731	248815749	466577361
IEC512	191246205	420543208	792576864

(cycles)

† Xeon E5-1620 3.6GHz Windows 7 64bit 32GB memory *rev.1

Benchmark*



*rev.1

Optimization technique

- Fast prime (Mersenne prime 2^{31}-1)
- Lazy reduction

```
S = A * B % q
T = C * D % q
R = (S + T) % q
```

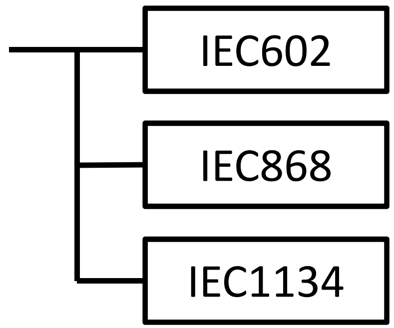
```
S = A * B
T = C * D
R = S + T
R = R % q
```

- Parallel implementation (not done)
  AVX2(Intel), NEON(ARM), GPU…

Side channel countermeasure

- Execution time independent to input data (done)
- Random masking (not done)

Source directory*



All difference is parameter.h only.

*rev.1

Difference from NIST API

- IND-CCA2 decryption with FO conversion needs public key as input for the purpose of verification.
- NIST provides seed expander $\{0,1\}^{320} \rightarrow \{0,1\}^*$. But we need seed expander $\{0,1\}^* \rightarrow \{0,1\}^*$. So we use SHAKE256 as seed expander.
- Original AES256. We use original AES256 instead of libssl. In windows environment, it is hard to use libssl.

All algorithms

- IECKG
- IECENC
- IECDEC
- keygen
- encrypt
- decrypt

IECKG

- Key generation primitive
- (ux, uy, X) = IECKG()
  return ux: secret key 1 in R_ℓ, uy: secret key 2 in R_ℓ, X: public key in Pq(dX)

  (1) ux $\in_R R_\ell$
  (2) uy $\in_R R_\ell$
  (3) X $\in_R$ Pq(dX)
  (4) a_{00} = X(ux, uy)
  (5) X = X – a00
  (6) return (ux, uy, X)

- X(ux, uy) means substitute x = ux(t), y = uy(t) for X(x, y). The result is in Rq.
- $\in_R$ uses random generator.

IECENC

- Encryption primitive
- c = IECENC(m, X)
  input m: message in R_ℓ, X:public key in Pq(dX)
  return c: ciphertext in Pq(dX+dr)

  (1) r $\in_R$ Pq(dr)
  (2) e $\in_R$ P_ℓ(dX+dr)
  (3) c = m + X*r + ℓ*e
  (4) return c

- $\in_R$ uses seed expander.

IECDEC

- Decryption primitive
- m = IECDEC(c, ux, uy)
  input c: ciphertext in Pq(dX+dr), ux: secret key 1 in R_ℓ, uy: secret key 2 in R_ℓ
  return m: message in R_ℓ

  (1) m = c(ux, uy)
  (2) m = reduce(m, ℓ)
  (3) return m

- c(ux, uy) means substitute x = ux(t), y = uy(t) for c(x, y). The result is in Pq.
- reduce(m, ℓ) means make all the coefficients of m(t) remainder divided by ℓ.

keygen

- Key generation
- SK, PK = keygen()
  return SK: secret key in OS, PK: public key in OS

  (1) (ux, uy, X) = IECKG()
  (2) SK = R_ℓ2OS(ux) | R_ℓ2OS(uy)
  (3) PK = Pq2OS(X)
  (4) return (SK, PK)

encrypt*

- IND-CCA2 encryption with FO conversion
- C = encrypt(msg, PK)
  input msg: message in OS with length mlen, PK: public key in OS
  return C: ciphertext in OS

  (1) PS $\in_R$ OS with byte length of payload – mlen
  Let the last 8 - 2* (n % 4) bits of PS set to 0.*
  (2) M = msg | PS
  (3) Initialize seed expander with coin = H(M)
  (4) m = $OS2R_\ell$(M)
  (5) X = OS2Pq(dX, PK)
  (6) c = IECENC(m, X)
  (7) C = Pq2OS(dX+dr, c)
  (8) return C

- H(･) is seed expander. In this implementation, we use SHAKE3-256 as seed expander.
- In this implementation, |PS| = ceil(n / 4) – mlen bytes.

*rev.1

decrypt

- IND-CCA2 decryption with FO conversion
- M = decrypt(C, SK, PK)
  input C: ciphertext in OS, SK: secret key in OS, PK: public key in OS
  return M: message in OS or IEC_NG

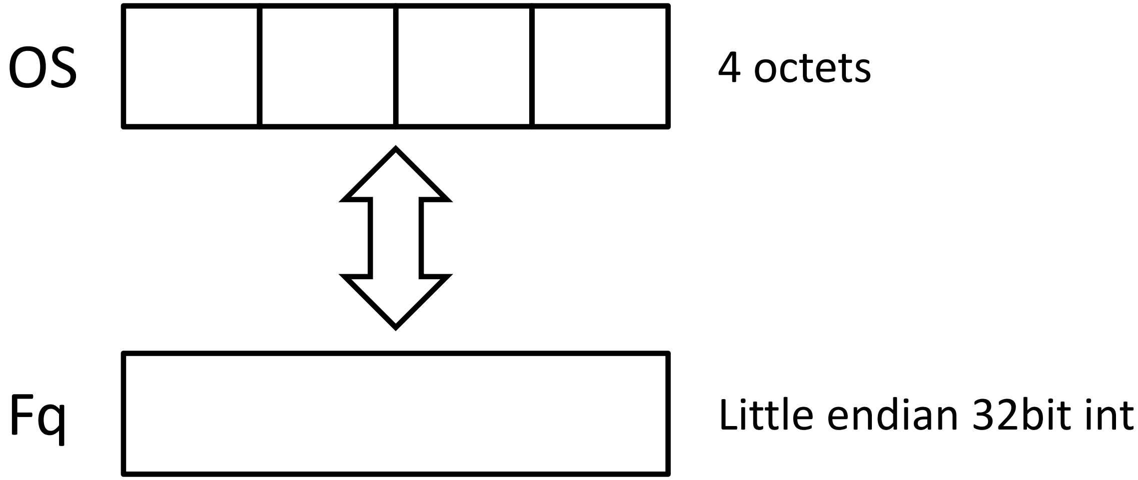
  (1) c = OS2Pq(dX+dr, C)
  (2) decompose SK to SK1, SK2
  (3) ux = OS2R$_\ell$(SK1)
  (4) uy = OS2R$_\ell$(SK2)
  (5) m = IECDEC(c, ux, uy)
  # verify whether C is correctly generated ciphertext
  (6) M = R$_\ell$2OS(m)
  (7) Initialize seed expander with coin = H(M)
  (8) X = OS2Pq(dX, PK)
  (9) c2 = IECENC(m, X)
  (10) if c == c2: return first mlen bytes of M
  (11) return IEC_NG

- |SK| = n / 2 bytes, |SK1| = |SK2| = n / 4 bytes.

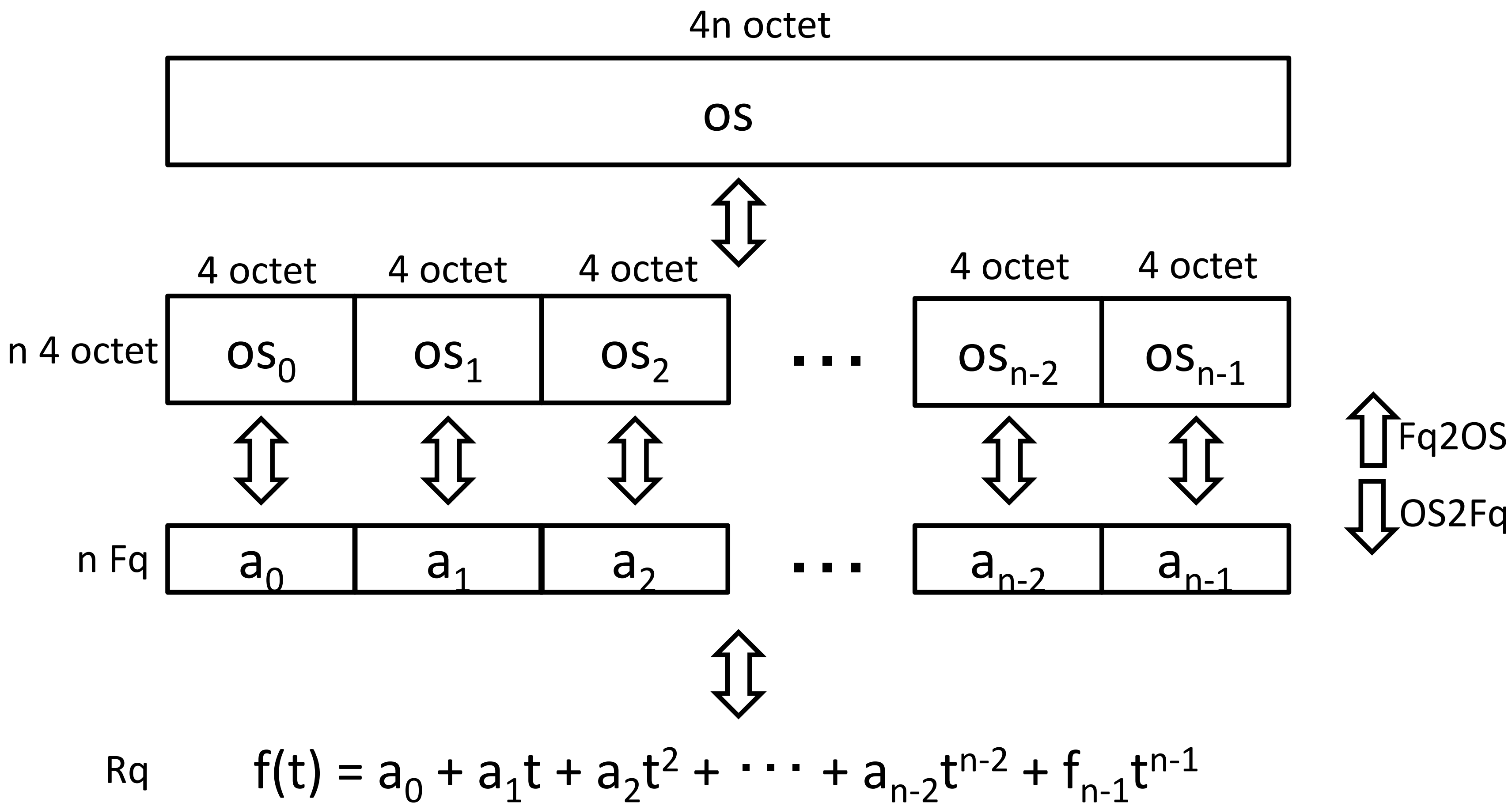
Data conversion

- OS2Fq / Fq2OS
- OS2Rq / Rq2OS
- OS2R_ℓ / R_ℓ2OS
- OS2Pq / Pq2OS

OS2Fq / Fq2OS

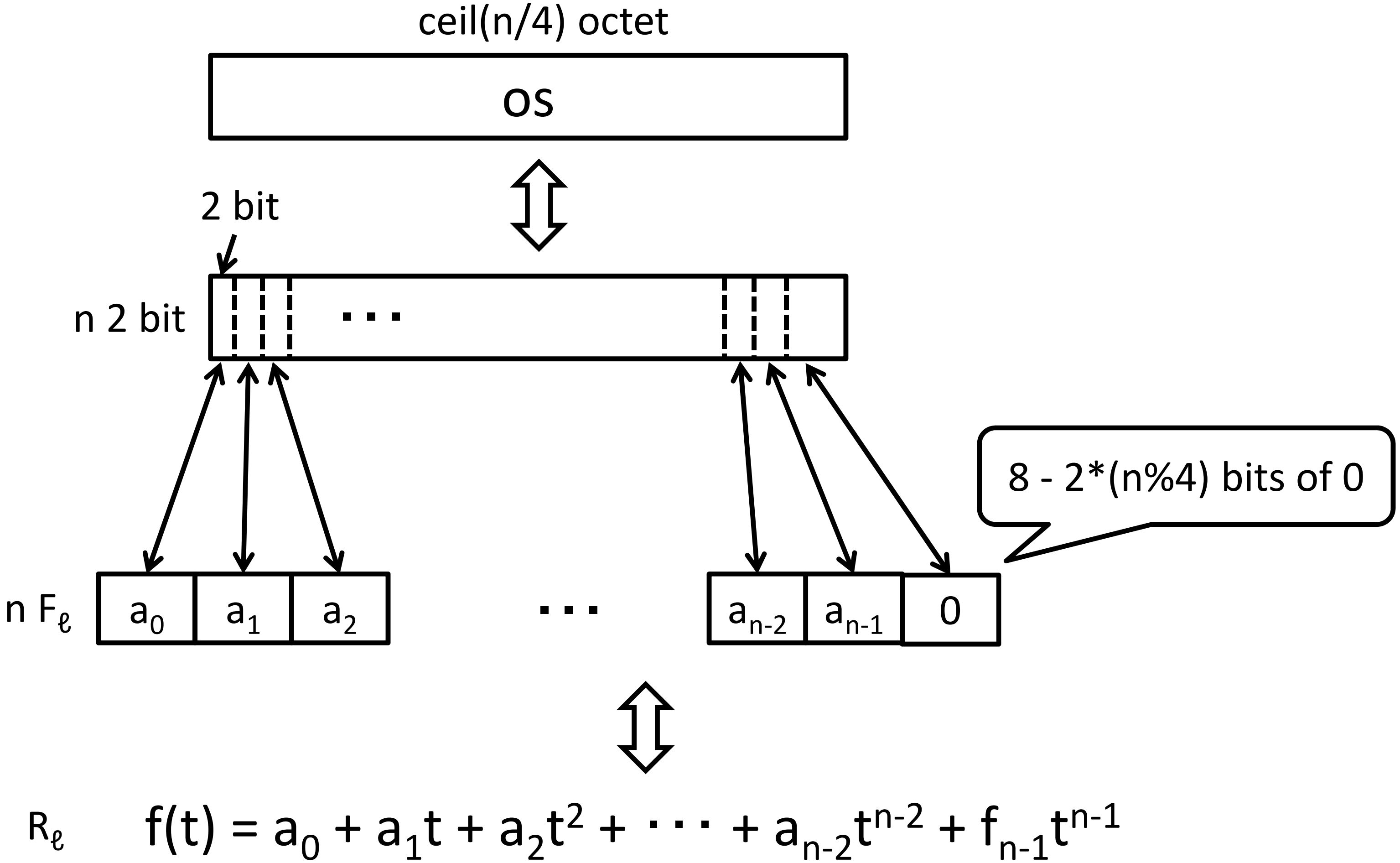


OS2Rq / Rq2OS



4n octet
os
4 octet
4 octet
4 octet
4 octet
4 octet
n 4 octet
os_0
os_1
os_2
. . .
os_{n-2}
os_{n-1}
Fq2OS
OS2Fq
n Fq
a_0
a_1
a_2
. . .
a_{n-2}
a_{n-1}
Rq
$f(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_{n-2} t^{n-2} + f_{n-1} t^{n-1}$

$OS2R_\ell$ / $R_\ell 2OS$*

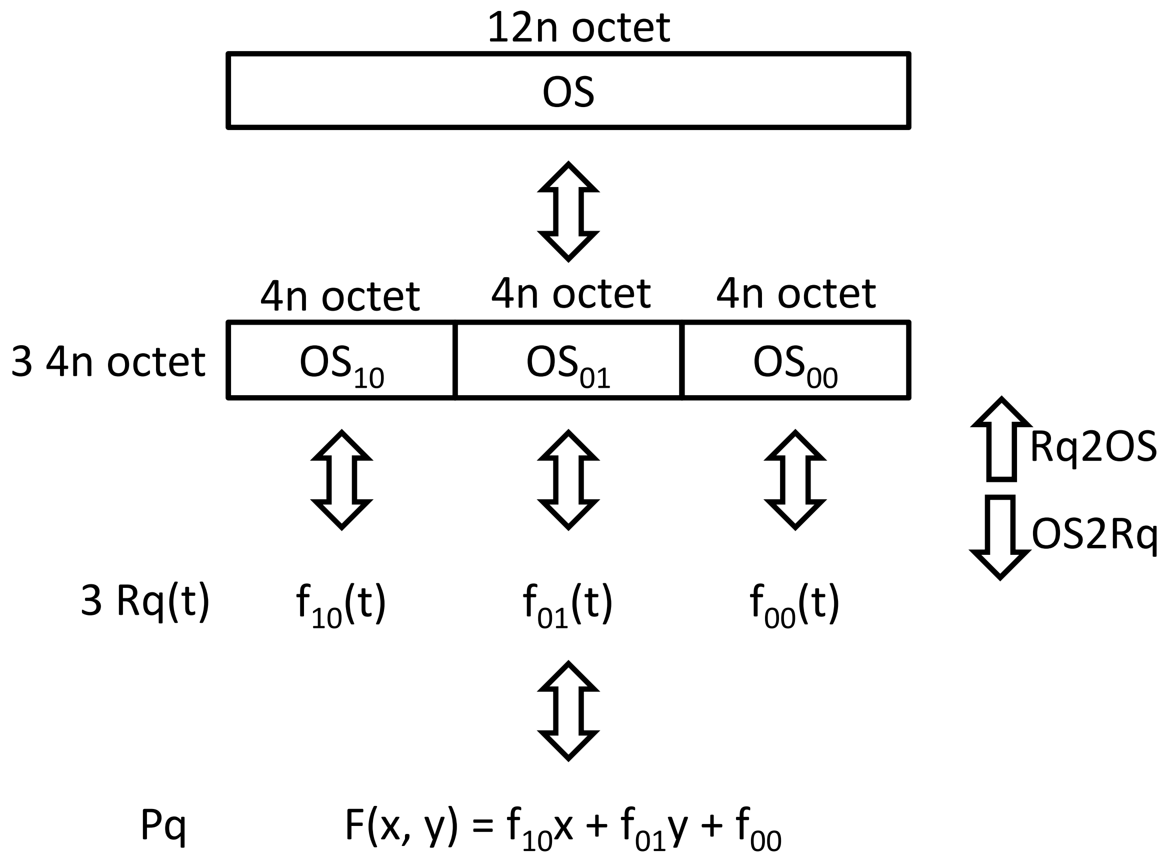


$$f(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_{n-2} t^{n-2} + f_{n-1} t^{n-1}$$

*rev.1

OS2Pq / Pq2OS

degree 1 case



OS2Pq / Pq2OS

degree 2 case

24n octet

OS

	4n octet	4n octet	4n octet	4n octet	4n octet	4n octet
6 4n octet	OS_{20}	OS_{11}	OS_{02}	OS_{10}	OS_{01}	OS_{00}
6 Rq(t)	$f_{20}(t)$	$f_{11}(t)$	$f_{02}(t)$	$f_{10}(t)$	$f_{01}(t)$	$f_{00}(t)$

$\Uparrow$ Rq2OS

$\Downarrow$ OS2Rq

Pq $F(x, y) = f_{20}x^2 + f_{11}xy + f_{02}y^2 + f_{10}x + f_{01}y + f_{00}$

Difference of previous submit*

- The algorithm name is now Giophantus.
- n must be prime number. (p.2)
  This affects followings:
  - Various sizes are changed ceil(n / 4). (p.3)
  - encode/decode rule of OS2Rℓ / RℓOS are changed. (p.23) Now we need zero padding.
- For security reason, parameters are changed. (p.5)

*rev.1