NUSH

CRYPTOGRAPHIC ALGORITHMS
BASED UPON THE BLOCK CIPHER CALLED
“NUSH”

Algorithm “NUSH”

DEFINITIONS

n - 32 (16, 64) - registers length (in bits)
N - 4*n - block size of the algorithm NUSH block
K - initial key of the algorithm (in bits, equals t*n and >= 128)
l - number of round of the NUSH algorithm
L - number of iterations (equals 4*l)
a, b, c, d - four of n-bit long registers of the algorithm NUSH
KR[i] - keys for the iterations (i =0,...,L-1)
KS[i] - initial keys of the algorithm (i=0,...,3)
KF[i] - final keys of the algorithm (i=0,...,3)
C[i] - n-bits registers (i=0,...,L-1)
S[i] - integers from 0 trough n-1 – cyclic rotation (to the right, i.e. to the least bit) lengths for the i-th iteration (i=0,...,L-1)

BASIC OPERATIONS

- binary addition (OR) or binary multiplication(&) of two n-bit vectors,
+ - addition mod2^n or XOR of two n-bit vectors
>>>t - cyclic rotation to the right (to the least bit) of an n-bit register by t bits

TRANSFORMATIONS

R - transformation of the four n-bit registers a, b, c, d;
parameters of the transformation R are: k is an n-bit register,
s is an integer from 0 through n-1
R(a, b, c, d, k, s) = (a1, b1, c1, d1)
c1 = c + k
c1 = c + b
c1 = c >>> s
a1 = a + (c1 # d)
b1 = b
d1 = d
Or
c1 = (c + k + b)>>>s
a1 = a+ c1 # d
b1 = b
d1 = d
We call a transformation R “an iteration”.

Round of the algorithm NUSH

One round of the algorithm consists of the four iterations of the form
R(a, b, c, d, k1, s1)
R(b, c, d, a, k2, s2)
R(c, d, a, b, k3, s3)
R(d, a, b, c, k4, s4)

Main body of the algorithm NUSH

The main body of the algorithm consists of L rounds, the i-th round looks like (i =0, ...,L):
R(a, b, c, d, KR[4*i]+C[4*i],S[4*i])
R(b, c, d, a, KR[4*i+1] +C[4*i+1], S[4*i+1])
R(c, d, a, b, KR[4*i+2] +C[4*i+2], S[4*i+2])
R(d, a, b, c, KR[4*i+3] +C[4*i+3], S[4*i+3])

Initial transformation:

START(a, b, c ,d, KS) – transformation of the registers a, b, c ,d
(a1, b1, c1 ,d1) = START(a, b, c ,d)
a1 = a + KS[0]
b1 = b + KS[1]
c1 = c + KS[2]
d1 = d + KS[3]

Final transformation:

FINAL(a, b, c ,d, KF) – transformation of the registers a, b, c ,d
(a1, b1, c1 ,d1) = FINAL(a, b, c ,d)
a1 = a + KF[0]
b1 = b + KF[1]
c1 = c + KF[2]
d1 = d + KF[3]

ENCRYPTION

The algorithm NUSH transforms four n-bit input registers a, b, c, d to the four output n-bit registers A, B, C, D.
Steps
1. Generation of the keys for the iterations, initial iteration and final iteration keys from the initial key K of the algorithm NUSH.
2. Setting of the algorithm parameters: rotations S[i] and registers C[i]
3. Initial transformation
4. Execution of the L rounds of the algorithm's main body
5. Final transformation

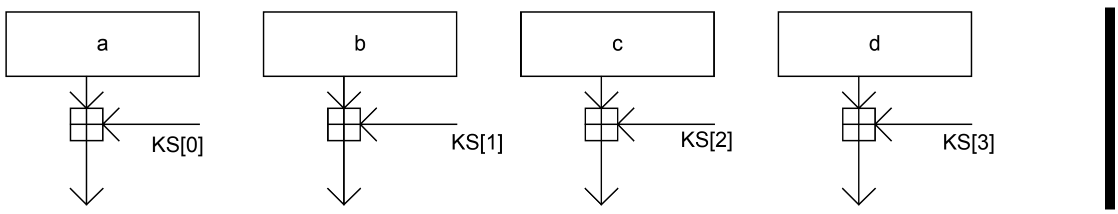
Below we use the following notations:
(A, B, C, D) = NUSH(a, b, c ,d) or
(A, B, C, D) = NUSH(a, b, c ,d, K, S, C), where S and C are notations for the contents of all the registers S[i] and C[i], and the initial key K is used to generate the keys KS[i], KR[i], KF[i].

In pseudo code the main body (without initial settings) of the algorithm NUSH looks like this

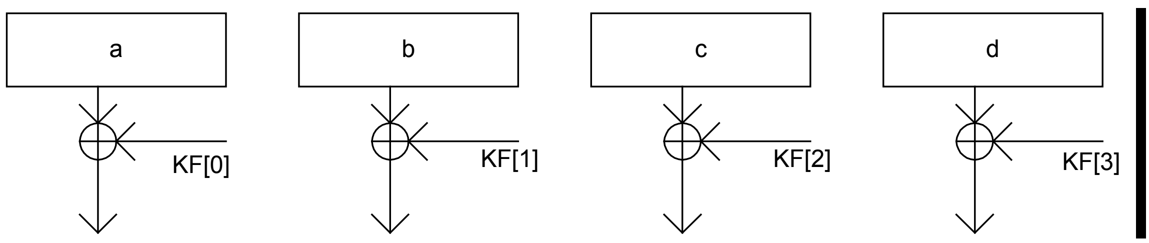
START(a, b, c ,d, KS)
R(a, b, c, d, KR[0]+C[0],S[0])
R(b, c, d, a, KR[1] +C[1], S[1])
R(c, d, a, b, KR[2] +C[2], S[2])
R(d, a, b, c, KR[3] +C[3], S[3])
...
R(a, b, c, d, KR[4*l-4] +C[4*l-4], S[4*l-4])
R(b, c, d, a, KR[4*l-3] +C[4*l-3], S[4*l-3])
R(c, d, a, b, KR[4*l-2] +C[4*l-2], S[4*l-2])
R(d, a, b, c, KR[4*l-1] +C[4*l-1], S[4*l-1])
FINAL(a, b, c ,d, KF)

Visually it may be depicted in the following way:

Initial transformation

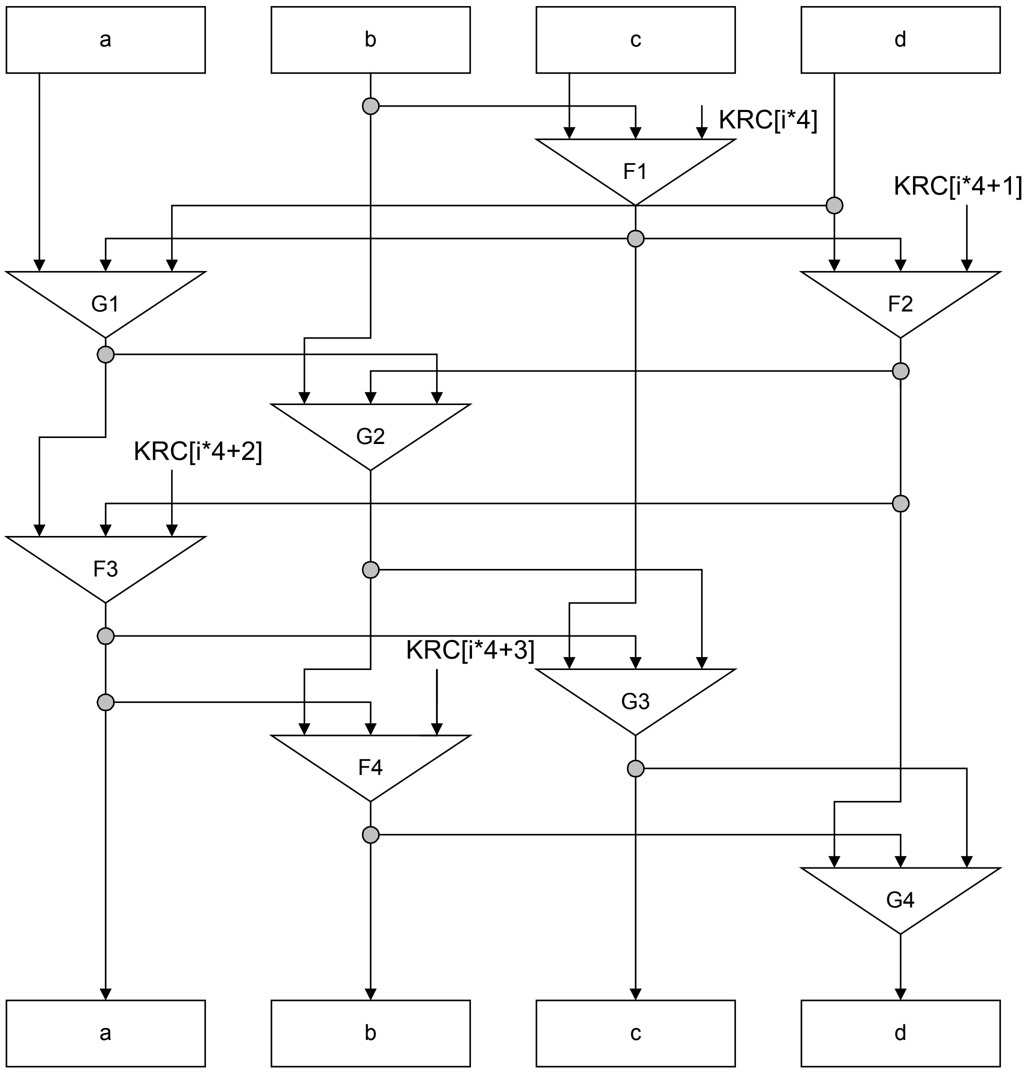


Final transformation



Round function

The i-th round looks like

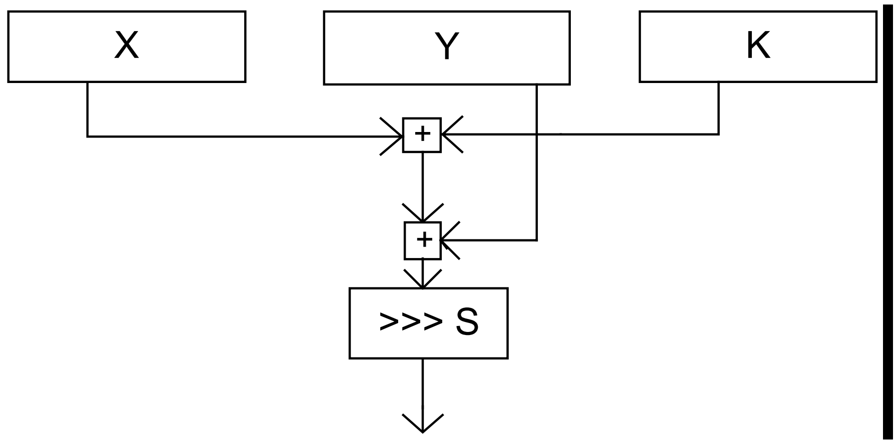


Here KRC[i] is for KR[i] +C[i].

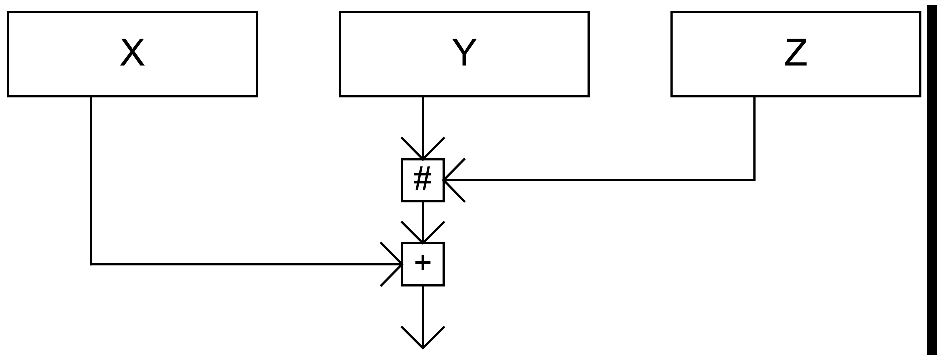
Formally the functions F_i and G_i are different for the different rounds but we do not pay much attention to this fact here.

The pair of functions F_i and G_i combine an iteration and look like:

Functions Fi



Functions Gi



DECRYPTION

The transformations START and FINAL are invertible with the inverses $START^{-1}$ and $FINAL^{-1}$.
The transformation R is also invertible with the inverse to R(a, b, c, d, k, s) of the form

R^{-1}(a1, b1, c1, d1, k, s) = (a, b, c, d)

d = d1
b = b1
a = a1 - (c1 # d)
c = c1 >>> (n-s)
c = c - k
c = c - b

(here by the «-» we denote the inverse operation to +, i.e. subtraction $mod2^n$ or XOR of two binary n-bit vectors)

To decrypt a block of cipher text we perform the following transformations

FINAL $^{-1}$(a, b, c ,d, KF)
R^{-1}(d, a, b, c, KR[4*l-1] +C[4*l-1], S[4*l-1])
R^{-1}(c, d, a, b, KR[4*l-2] +C[4*l-2], S[4*l-2])
R^{-1}(b, c, d, a, KR[4*l-3] +C[4*l-3], S[4*l-3])
R^{-1}(a, b, c, d, KR[4*l-4] +C[4*l-4], S[4*l-4])

. . . .

R^{-1}(d, a, b, c, KR[3] +C[3] S[3])
R^{-1}(c, d, a, b, KR[2] +C[2], S[2])
R^{-1}(b, c, d, a, KR[1] +C[1], S[1])
R^{-1}(a, b, c, d, KR[0]+C[0],S[0])
START $^{-1}$(a, b, c ,d, KS)

If we get the block (A, B, C, D) after decryption of the block (a, b, c, d), we write:
(A, B, C, D) = $NUSH^{-1}$(a, b, c ,d) or
(A, B, C, D) = $NUSH^{-1}$(a, b, c ,d, K, S, C),
where S and C are for all the contents of all the registers S[i] and C[i], with the initial key K used to form the keys KS[i], KR[i], KF[i].

Block cipher “NUSH Block”

GENERAL

The algorithm encrypts data by blocks of N bits each. Current block is read from the input registers a, b ,c ,d (the least bit of the plain text is in the least bit of the register “a”, the most bit of the text is in the most bit of the register “d”).

Cipher text of the plaintext block (a, b, c, d) is the block (A, B, C, D) of the same length N bit
(A, B, C, D) = NUSH(a, b, c ,d, K, S, C)
(the least bit of the cipher text is in the least bit of the register A, and the greatest bit of the cipher text is in the greatest bit of the register D).

Decryption of the block (A, B, C, D) (where the least bit of the cipher text is in the least bit of A and the greatest bit of the cipher text is in the greatest bit of D) is to the block (a, b, c, d) in accordance with the formula
(a, b, c, d) = $NUSH^{-1}$(A, B, C ,D, K, S, C)
(the least bit of plain text is in the least bit of the register “a” and the greatest bit of plain text is in the greatest bit of the register “d”)

PARAMETERS

1. 64-bit block
n=16, N=64
l=9, L=36 (all the constants are hexadecimal)

I	C[i]	i	C[i]	i	C[i]	i	C[i]
0	**ac25**	9	**6a29**	18	**96da**	27	**d25e**
1	**8a93**	10	**6d84**	19	**905f**	28	**a926**
2	**243d**	11	**34bd**	20	**d631**	29	**1c7b**
3	**262e**	12	**a267**	21	**aa62**	30	**5f12**
4	**f887**	13	**cc15**	22	**4d15**	31	**4ecc**
5	**c4f2**	14	**04fe**	23	**70cb**	32	**3c86**
6	**8e36**	15	**b94a**	24	**7533**	33	**28db**
7	**9fa1**	16	**df24**	25	**45fc**	34	**fc01**
8	**7dc0**	17	**40ef**	26	**5337**	35	**7cb1**

I	S[i]	i	S[i]	i	S[i]	i	S[i]
0	4	9	2	18	5	27	13
1	7	10	9	19	1	28	12
2	11	11	4	20	2	29	3
3	8	12	13	21	4	30	6
4	7	13	1	22	12	31	11
5	14	14	14	23	3	32	7
6	5	15	6	24	9	33	15
7	4	16	7	25	2	34	4
8	8	17	12	26	11	35	14

2. 128-bit block

n=32, N=128
l=17, L=68

I	C[i]	i	C[i]	I	C[i]	i	C[i]
0	**9b28a37b**	18	**c443f6cc**	36	**aa7de138**	54	**362f2f4a**
1	**9de5b521**	19	**e84c5bcb**	37	**a674a66c**	55	**6ccb630d**
2	**0b8ee0d7**	20	**f750a732**	38	**b3f54983**	56	**97919d88**
3	**672aa715**	21	**2cde9942**	39	**ae29d0db**	57	**823f95ac**
4	**0e356c9f**	23	**370c437a**	40	**599470cb**	58	**67c99a98**
5	**bf54692a**	23	**da8b5654**	41	**3b2e3fa0**	59	**8e91d0cb**
6	**dc9e15c8**	24	**99a76750**	42	**a354cc6f**	60	**ab796817**
7	**06d736e8**	25	**a1559437**	43	**516af8c4**	61	**356459a7**
8	**9263e8cf**	26	**9ea46718**	44	**ade11d33**	62	**668d9fa8**
9	**1fcd682d**	27	**83e984f8**	45	**860d95f2**	63	**0d4dbf40**
10	**7368b074**	28	**ab5692e4**	46	**bc2731a4**	64	**1acce5d8**
11	**2654f15a**	29	**a6c5c46a**	47	**ccd12baa**	65	**f53b24c1**
12	**00eb3e4d**	30	**25fb110e**	48	**ba518e95**	66	**6db89876**
13	**18d62f6d**	31	**55955b2e**	49	**22f7583a**	67	**5c965da5**
14	**632a557a**	32	**fa639063**	50	**6c0a5fe8**		
15	**1d953d21**	33	**027e4dc6**	51	**8fac2d74**		
16	**cd4b2acd**	34	**919e96b2**	52	**d129e934**		
17	**49a0d3f4**	35	**62e96d0c**	53	**11dce4c9**		

I	Si]	i	S[i]	I	S[i]	i	S[i]
0	**7**	18	**26**	36	**12**	54	**7**
1	**5**	19	**4**	37	**24**	55	**15**
2	**15**	20	**29**	38	**27**	56	**1**
3	**14**	21	**16**	39	**10**	57	**13**
4	**3**	22	**2**	40	**16**	58	**15**
5	**30**	23	**22**	41	**24**	59	**1**
6	**4**	24	**23**	42	**9**	60	**23**
7	**23**	25	**11**	43	**13**	61	**28**
8	**13**	26	**26**	44	**5**	62	**12**
9	**12**	27	**13**	45	**10**	63	**2**
10	**26**	28	**20**	46	**26**	64	**28**
11	**16**	29	**5**	47	**30**	65	**14**
12	**9**	30	**28**	48	**9**	66	**15**
13	**28**	31	**17**	49	**16**	67	**12**
14	**8**	32	**19**	50	**28**		
15	**18**	33	**22**	51	**24**		
16	**23**	34	**6**	52	**27**		
17	**8**	35	**25**	53	**6**		

3. 256-bit block

n=64, N=256
l=33, L=132

I	C[i]	i	C[i]	i	C[i]	i	C[I]
0	**1a028e3b458fe65f**	33	**17c41d9833728fe9**	66	**0289fbe8ce5bd06a**	99	**5036e2ee9c4166b9**
1	**10cb1c5cac3c7a75**	34	**7e692e4db9d09471**	67	**26dbaa50cbc1e8b9**	100	**6d32721cf1269e70**
2	**0aa54c8d55cc6f5e**	35	**c900cde6cb8aa557**	68	**4116b2b8d89aff86**	101	**c51e826355ec445f**
3	**ee4ac8b12e2fc8d5**	36	**eb2b8576c0419fe3**	69	**1d658d6eef814e49**	102	**0e8e66931ef37c41**
4	**f787d15c240344d7**	37	**927c3fe32c9a2365**	70	**a4b511d2427e3f73**	103	**9a94b3039660d3de**
5	**caccaf60f2998693**	38	**427410eb1eacbe4f**	71	**e2a77bd9898e1326**	104	**1ed158ecd9d68529**
6	**4ea93e4df9558e82**	39	**18a6fe2878b4d78d**	72	**65dea88074b941fd**	105	**0ece52dc8f1c3952**
7	**b57cda0316bc1c92**	40	**436eb84357c5342f**	73	**8e55b0dc3cee4398**	106	**86a20a1fffc847e5**
8	**623c7496c0d6fb68**	41	**1b94c23f94c24b3e**	74	**c14e2add6601ebdc**	107	**ff1dadc90c09a612**
9	**bd7b065e84d852a9**	42	**d3d831585e585a9c**	75	**a24f31d25e456e34**	108	**b896156e08c55f6d**
10	**a6cd2e5c6b1a30e7**	43	**f37e22a1587b9670**	76	**ad83615ac0e7aeae**	109	**644dea351c86f456**
11	**788d9efc078281b5**	44	**96a27fa6164197cd**	77	**81fcc39f84a54a8b**	110	**29b4b572556f360d**
12	**d0cf11a8ff9943e4**	45	**c21bc4eaf449ac7e**	78	**d15c7e21fe235136**	111	**875399911a5a79d1**
13	**d04f01c7f3ea8e96**	46	**bcce8974a35a69d4**	79	**5f5ac08e5a961b43**	112	**32ec6f05bc921ba5**
14	**5313f574e5d1d2c8**	47	**7fa98c9b495c2782**	80	**0cec9543f2a66676**	113	**ce0fb52c15c61a97**
15	**dc8ab4437aad50cf**	48	**3b64d65041406ffb**	81	**7c034eba929a8b8e**	114	**7f4e15212953f03d**
16	**66ed63d790921a4d**	49	**af82f6418c48f7dc**	82	**c0f4ce12ec988ebb**	115	**873ca0565bbec3e8**
17	**fa351c5183ebda0b**	50	**13b7d80a170e6ab6**	83	**5d358844ae5699f9**	116	**cdfb94c29b3812f1**
18	**da694b14554d17c9**	51	**09dfc1bbf5a51842**	84	**42e8d74db4919b52**	117	**aaaea6e308e92f68**
19	**0a392fa5de785cd1**	52	**45b2f2934e2becc4**	85	**8250d178f5557f8a**	118	**703cca8345ec51fc**
20	**75b1d5de6561d08c**	53	**f456d827335c90d3**	86	**532394e648e4f3fc**	119	**4618bb1b1b33ef0c**
21	**bc128db2f22c591e**	54	**7a2c6ee4672634d8**	87	**3e2bf92b03691ad8**	120	**f039732aad11fe46**
22	**d19f06a961bc6e36**	55	**3aa0d9523bbbd398**	88	**fa9268e710647d5b**	121	**86d89114ce8de23f**
23	**f3f2d208215dda85**	56	**d578f2aea135f841**	89	**bbd56f8408e2e651**	122	**330aedc7e44b8af0**
24	**d9a5d482f930b1af**	57	**9a6635da5227b8e9**	90	**793c3027eb0c5b8c**	123	**96d7869edd33e500**
25	**fd98b3a189ad9851**	58	**f40f12a5b07bc3d8**	91	**7643d2bb11326b87**	124	**f59cc3b1e9354045**
26	**b671a790fb204ae3**	59	**bbd16b68649b4271**	92	**4b9ff22bb56211e4**	125	**ad3db4f4a1aa8433**
27	**4e3b9db2a290ec98**	60	**042753ce1b63f27b**	93	**aa39e9382f34b664**	126	**724ece1c833975ea**
28	**2ca2afb114df74a2**	61	**a471d892d743f58d**	94	**e212d331bfe06a72**	127	**98516ab5c5303e6e**
29	**705ce63837b3616d**	62	**b6cacf5958204c67**	95	**1755736ea478f948**	128	**acf4fd043b90ccb6**
30	**679d058ea189a2ee**	63	**fb7786e2234aa30a**	96	**59ca19f718a53eaa**	129	**8d8a1da51be5cec1**
31	**8398bab59e3a506c**	64	**97eb25e4c9f33038**	97	**f44b30fa21c0a6ed**	130	**11d0127b77b9427b**
32	**181f8aefd8499ad4**	65	**cd5d27e1802e58f4**	98	**71f47e295da0855c**	131	**67c2de1924caa5ed**

i	S[i]	i	S[i]	i	S[i]	i	S[i]
0	**12**	33	**40**	66	**60**	99	**51**
1	**45**	34	**13**	67	**55**	100	**32**
2	**7**	35	**14**	68	**37**	101	**4**
3	**48**	36	**8**	69	**50**	102	**26**
4	**14**	37	**21**	70	**12**	103	**46**
5	**43**	38	**6**	71	**41**	104	**2**
6	**8**	39	**59**	72	**7**	105	**1**
7	**54**	40	**17**	73	**40**	106	**38**
8	**49**	41	**5**	74	**35**	107	**12**
9	**47**	42	**23**	75	**45**	108	**7**
10	**37**	43	**10**	76	**2**	109	**41**
11	**55**	44	**32**	77	**44**	110	**45**
12	**58**	45	**20**	78	**4**	111	**37**
13	**32**	46	**53**	79	**49**	112	**24**
14	**16**	47	**3**	80	**29**	113	**10**
15	**36**	48	**20**	81	**12**	114	**4**
16	**13**	49	**42**	82	**56**	115	**2**
17	**35**	50	**1**	83	**18**	116	**6**
18	**50**	51	**58**	84	**59**	117	**18**
19	**58**	52	**12**	85	**21**	118	**9**
20	**21**	53	**30**	86	**45**	119	**52**
21	**56**	54	**38**	87	**60**	120	**8**
22	**4**	55	**6**	88	**12**	121	**57**
23	**52**	56	**23**	89	**62**	122	**1**
24	**32**	57	**61**	90	**59**	123	**31**
25	**19**	58	**7**	91	**51**	124	**35**
26	**28**	59	**12**	92	**20**	125	**33**
27	**10**	60	**33**	93	**42**	126	**11**
28	**63**	61	**41**	94	**6**	127	**16**
29	**53**	62	**17**	95	**27**	128	**6**
30	**50**	63	**35**	96	**1**	129	**13**
31	**27**	64	**30**	97	**17**	130	**15**
32	**18**	65	**3**	98	**24**	131	**45**

KEY GENERATION

The initial key is represented by n-bit words of the form (K[0], K[1], ...) and the least bit of the word K[0] is the least bit of the key.

1. Key of the 128 bits

1.1 N=64 (n=16)

KS[0]=K[4]	KF[0]= K[3]
KS[1]=K[5]	KF[1]= K[2]
KS[2]=K[6]	KF[2]= K[1]
KS[3]=K[7]	KF[3]= K[0]

KR[i]=K[i mod 8] , i=0,...35

1.2 N = 128 (n=32)

KS[0]= K[3]	KF[0]= K[1]
KS[1]= K[2]	KF[1]= K[0]
KS[2]= K[1]	KF[2]= K[3]
KS[3]= K[0]	KF[3]= K[2]

KR[i]=K[i mod 4] , i=0,...67

1.3 N = 256 (n=64)

KS[0]= K[1]	KF[0]= K[0]
KS[1]= K[0]	KF[1]= K[1]
KS[2]= K[1]	KF[2]= K[0]
KS[3]= K[0]	KF[3]= K[1]

KR[i]=K[i mod 2] , i=0,...67

2. Key of the 192 bits

2.1 N = 64 (n=16)

KS[0]= K[4]	KF[0]= K[11]
KS[1]= K[5]	KF[1]= K[10]
KS[2]= K[6]	KF[2]= K[9]
KS[3]= K[7]	KF[3]= K[8]

KR[i]=K[i mod 12] , i=0,...35

2.2 N = 128 (n=32)

KS[0]= K[2]	KF[0]= K[5]
KS[1]= K[3]	KF[1]= K[4]
KS[2]= K[4]	KF[2]= K[3]
KS[3]= K[5]	KF[3]= K[2]

KR[i]=K[i mod 6] , i=0,...67

2.3N = 256 (n=64)

KS[0]= K[2]	KF[0]= K[1]
KS[1]= K[1]	KF[1]= K[2]

KS[2]= K[0]   KF[2]= K[2]
KS[3]= K[2]   KF[3]= K[0]
KR[i]=K[i mod 3] , i=0,...67

3. Key of the 256 bits

3.1 N = 64 (n=16)
KS[0]= K[12]   KF[0]= K[13]
KS[1]= K[13]   KF[1]= K[12]
KS[2]= K[14]   KF[2]= K[15]
KS[3]= K[15]   KF[3]= K[14]
KR[i]=K[i mod 16] , i=0,...35

3.2 N = 128 (n=32)
KS[0]= K[4]   KF[0]= K[5]
KS[1]= K[5]   KF[1]= K[4]
KS[2]= K[6]   KF[2]= K[7]
KS[3]= K[7]   KF[3]= K[6]
KR[i]=K[i mod 8] , i=0,...67

3.3N = 256 (n=64)
KS[0]= K[3]   KF[0]= K[2]
KS[1]= K[2]   KF[1]= K[3]
KS[2]= K[1]   KF[2]= K[0]
KS[3]= K[0]   KF[3]= K[1]
KR[i]=K[i mod 4] , i=0,...67

If we do not change the constants C[i] for each iteration, then we can compute KRC[i]=KR[i]+C[i] in advance and use them instead of computing the sums KR[i]+C[i] for each of the iterations..

CHOICE OF OPERATIONS

& - is for Boolean multiplication of two n-bit vectors

| - is for Boolean addition (OR)of two n-bit vectors

^ - is for XOR of two n-bit vectors

+ - is for addition mod2^n of two n-bit integers

Steps of the algorithm now are the following.

1. Initial addition with key
a = a ^ KS[0], b = b ^ KS[1], c = c ^ KS[2], d = d ^ KS[3]

2. Final addition with key
a = a ^ KF[0], b = b ^KF[1], c = c^ KF[2], d = d ^ KF[3]

3. The operations in the main body of the algorithm NUSH.

Let us numerate the operations to be defined.
For the transformation R we have four operations to be defined
R(a, b, c, d, k, s) = (a1, b1, c1, d1)
c1 = c + k (1)
c1 = c + b (2)
c1 = c >>> s (3)
a1 = a + (c1 # d) (4)

b1 = b
d1 = d

We use the additional operation (5), which also has to be defined.
R(a, b, c, d, KR[4*i]+C[4*i],S[4*i]) (5)

Let i be the iteration number
Then, the operation (1) is the operation ^.
The operation (2) is +.
The operation (3) is also +.
The operation (5) is also +.

Choice of the operation (4) is given by the table:

<table>
<tr><th>I</th><th>Operation (4)</th><th>i</th><th>Operation (4)</th><th>i</th><th>Operation (4)</th><th>i</th><th>Operation (4)</th></tr>
<tr><td>0</td><td>&</td><td>16</td><td>|</td><td>32</td><td>|</td><td>48</td><td>&</td></tr>
<tr><td>1</td><td>|</td><td>17</td><td>|</td><td>33</td><td>|</td><td>49</td><td>&</td></tr>
<tr><td>2</td><td>&</td><td>18</td><td>&</td><td>34</td><td>&</td><td>50</td><td>&</td></tr>
<tr><td>3</td><td>|</td><td>19</td><td>&</td><td>35</td><td>|</td><td>51</td><td>&</td></tr>
<tr><td>4</td><td>|</td><td>20</td><td>&</td><td>36</td><td>|</td><td>52</td><td>&</td></tr>
<tr><td>5</td><td>|</td><td>21</td><td>&</td><td>37</td><td>&</td><td>53</td><td>&</td></tr>
<tr><td>6</td><td>|</td><td>22</td><td>&</td><td>38</td><td>|</td><td>54</td><td>|</td></tr>
<tr><td>7</td><td>|</td><td>23</td><td>|</td><td>39</td><td>&</td><td>55</td><td>&</td></tr>
<tr><td>8</td><td>&</td><td>24</td><td>&</td><td>40</td><td>|</td><td>56</td><td>|</td></tr>
<tr><td>9</td><td>|</td><td>25</td><td>|</td><td>41</td><td>&</td><td>57</td><td>|</td></tr>
<tr><td>10</td><td>|</td><td>26</td><td>|</td><td>42</td><td>&</td><td>58</td><td>|</td></tr>
<tr><td>11</td><td>&</td><td>27</td><td>|</td><td>43</td><td>|</td><td>59</td><td>&</td></tr>
<tr><td>12</td><td>|</td><td>28</td><td>&</td><td>44</td><td>|</td><td>60</td><td>&</td></tr>
<tr><td>13</td><td>&</td><td>29</td><td>|</td><td>45</td><td>&</td><td>61</td><td>&</td></tr>
<tr><td>14</td><td>|</td><td>30</td><td>&</td><td>46</td><td>&</td><td>62</td><td>|</td></tr>
<tr><td>15</td><td>|</td><td>31</td><td>&</td><td>47</td><td>&</td><td>63</td><td>|</td></tr>
</table>

For an iteration with number i more than 63 the operations are the same as for the iteration number i(mod 64).

Synchronous stream ciphers "NUSH Stream"

ALGORITHM

Let SYNC be a Boolean vector of a length LENGTH.
We suppose that the vector SYNC is known for encryption and decryption procedures.
The algorithm NUSH Stream can be used as a stream cipher with internal memory of LENGTH bits in the following ways.

Varian 1. Use the algorithm NUSH Block with the block length N =LENGTH (LENGTH = 64, 128, 256)

To generate COUNT number of N-bit blocks of keystream called GAMMA: GAMMA[0], ... , GAMMA[COUNT-1] for encryption or decryption of data we go the same way and compute:

```
SYNC = SYNC ^ NUSH(SYNC)
For i =0 to COUNT -1
{
        GAMMA[i]= NUSH(SYNC)
        SYNC = (SYNC + 65257 ) mod 2^N
}
```

Variant2. Use the algorithm NUSH Block with the block length N=LENGTH / 2 (LENGTH = 128, 256, 512),

Let T be an N-bit register with N = 4*n, and let the register T consists of the following four n-bit words (T[0],T[1],T[2],T[3]), let vector SYNC be the (SYNC[0], SYNC[1]) of N-bit words.
We use register T to modify contents of the registers C[n], C[n+1], C[n+2],C[n+3].

To generate COUNT number of N-bit blocks of keystream called GAMMA: GAMMA[0], . . . , GAMMA[COUNT -1] for encryption and decryption of data we go the same way and compute:

```
SYNC[0] = SYNC[0] ^ NUSH(SYNC[0])
SYNC[1] = SYNC[1] ^ NUSH(SYNC[1])
T=SYNC[1]
For i =0 to COUNT -1
{
        C[n]     = T[0]
        C[n+1] = T[1]
        C[n+2] = T[2]
        C[n+3]= T[3]
        GAMMA[i]= NUSH(SYNC[0])
        SYNC[0] = (SYNC[0] + 65257 ) mod 2^N
        T = (T + 127  ) mod 2^N
}
```

Self-synchronising stream ciphers "NUSH Self-Synchro"

ALGORITHM

Let variable LENGTH means the length of the synchronizing vector SYNC (in bits) and let LG be the number of bits in output vector used to encrypt data, with LG < N. Let GAMMA be the output Boolean vector of LG bits called output encrypting sequence.

By HIGH(A) we denote the high LG bits of the N-bit vector A, by >>t we denote a shift of an N-bit vector to the least bit by t bits with zeroing the most t bits.

Variant 1. Use NUSH Block algorithm with block length N=LENGTH (LENGTH = 64,128, 256).

Encryption of a text block TEXT of the length LTEXT blocks (measured by blocks of LG bits each) is the following.

```
SYNC = SYNC ^ NUSH(SYNC)
For i =0 to LTEXT - 1
{
        GAMMA[i] = HIGH(NUSH(SYNC))
        SYNC = SYNC >> LG
        TEXT[i] = TEXT [i] + GAMMA[i] (this is a bit-wise addition of two
LG-bit vectors)
        HIGH(SYNC) = TEXT[i]
}
```

Decryption is the following.

```
SYNC = SYNC ^ NUSH(SYNC)
For i =0 to LTEXT - 1
{
        TEMP = TEXT[i]
        GAMMA[i]= HIGH(NUSH(SYNC))
        SYNC = SYNC >> LG
        TEXT[i] = TEXT [i] + GAMMA[i] (this is a bit-wise addition of two
LG-bit vectors)
        HIGH(SYNC) = TEMP
}
```

Variant 2. Use NUSH Block algorithm with block length N=LENGTH/2, LENGTH=(128, 256,512),

Let T be a register of length N (in bits, and N = 4*n) that has a form of a four words vector (T[0], T[1], T[2], T[3]) with the N-bit words T[i].
We will modify values of C[i] by the contents of the register T.

Let LOW(A) be the least LG bits of the N-bit vector A, and let SYNC=(SYNC[0], SYNC[1]) be a vector of two n-bit words, and let V be an N-bit vector.

In this case encryption of a text called TEXT of the length LTEXT (in LG-bit blocks) looks like:

```
SYNC[0] = SYNC[0] ^ NUSH(SYNC[0])
SYNC[1] = SYNC[1] ^ NUSH(SYNC[1])
T = SYNC[0]
V= SYNC[1]
For i =0 to LTEXT - 1
{
        C[n] =T[0]
        C[n+1] =T[1]
        C[n+2] =T[2]
        C[n+3] =T[3]
        GAMMA[i]= HIGH(NUSH(V))
        T = T >>LG
        HIGH(T) = LOW (V)
        V = V >>LG
        TEXT[i] = TEXT [i] + GAMMA[i] (bit-wise sum of two LG-bit
vectors)
        HIGH(V) = TEXT[i]
}
```

Decryption cycle looks like

```
SYNC[0] = NUSH(SYNC[0])
SYNC[1] = NUSH(SYNC[1])
T = SYNC[0]
V= SYNC[1]
For i =0 to LTEXT -1
{

        TEMP = TEXT[i]
        C[M =T[0]
        C[M+1] =T[1]
        C[M+2] =T[2]
        C[M+3] =T[3]
        GAMMA[i]= HIGH(NUSH(V)
        T = T << LG
```

HIGH(T) = LOW (V)
SYNC = SYNC << LG
TEXT[i] = TEXT [i] + GAMMA[i] (bit-wise sum of two LG-bit vectors)
HIGH(V) = TEMP
}

4. Message Authentication Codes based upon the algorithm NUSH

MAC COMPUTATION

This algorithm is the same as the hash algorithm “NUSH Hash” described below with the only difference that it uses nonzero keys.

5. - 6. Hash functions based on NUSH Block algorithm.

TEXT EXTENSION

Initial text called TEXT of the length LENGTH bits to be hashed is extended in the following way.

1. Add to the initial text the final bit equal 1.
2. The result text, called TEXT1, extend by zeroes to the text of length multiple N
3. The result text called TEXT2, extend by the additional N-bit vector of the binary representation of the integer LENGTH (mod 2^N).
4. The result TEXT3 extend by the N-bit vector of bit-wise sum (XOR) of all the N-bit vectors of the text TEXT2.

DEFIINITIONS

H is an N-bit register (H[0], . . . , H[3]) represented by the four n-bit words.

M is a binary register of length 4*N contains the array (M[0], . . . , M[15]) represented by the 16 of n-bit vectors M[i]

T is a binary register of length 4*N represented by the n-bit words (T[0], . . . , T[15]).

V is an N-bit vector (V[0], V[1], V[2], V[3]) represented by n-bit words.

The initial value of the register T consists of the constants C[0], . . . , C[15]. The initial value of register M formed by the constants C[16], . . . , C[31], the keys KS, KF, KR equal zero.

HASHING ALGORITHM

Let sequence TEXT be a text of the length LENGTH represented by the N-bit words (TEXT[0], . . . , TEXT[LENGTH-1])
Hashing procedure is the following

```
For i = 0 to LENGTH -1
{
        For j=0 to L/2-1
        {
                C[2*j]      = T[j mod 16]
                C[2*j+1]  =M[j mod16]
        }
        V = TEXT[i]
        H = NUSH(V)
        H[0] = H[0] + V[3]
        H[1] = H[1] + V[2]
        H[2] = H[2] + V[1]
        H[3] = H[3] + V[0]
        For j=15 to 4
                T[j] = T[j-4]
        T[0]=H[0]
        T[1]=H[1]
        T[2]=H[2]
        T[3]=H[3]
        For j=15 to 4
                M[j] = M[j-4]
        M[0]=V[0]
        M[1]=V[1]
        M[2]=V[2]
        M[3]=V[3]

}
For i= 0 to 3
{
        For j=0 to L/2-1
                C[2*j]      = T[j mod 16]
        H = NUSH(M[i])
        For j=15 to 4
                T[j] = T[j-4]
        T[0]=H[0]
        T[1]=H[1]
        T[2]=H[2]
        T[3]=H[3]
}
```

The final result of this hashing is the value of the register T. If we need hash value of the length n*t bits (t <16), then we use the first t of n-bit words of this register.

7. Families of Pseudo-random functions "NUSH PRF"

DEFINITIONS

X is an N-bit vector,
K is the initial key of the algorithm NUSH of the length L bits

BASIC TRANSFORMATIONS

NUSH_MAC(X, K) is a binary vector of the length 4*N, that is equal to the result of MAC computation for the text X with the key K.

FAMILY OF PSEUDO-RANDOM FUNCTIONS

Let F be a function from the N+L dimensional Boolean vector space $V_N(2)$ x $V_L(2)$ to the 4*N dimensional space $V_{4*N}(2)$ defined as

F(X,K) = NUSH_MAC(X,K)

for all vectors X from $V_N(2)$ and keys K from $V_L(2)$.

Let F_K be the corresponding mapping from $V_N(2)$ to $V_{4*N}(2)$ with a fixed key K from $V_L(2)$.

The set of functions { F_K : K from $V_L(2)$ } gives a family of pseudorandom functions with the arguments from $V_N(2)$ and the values in $V_{4*N}(2)$.

To get a function values in the space $V_S(2)$ for S < 4*N the high (4*N-S) bits of the value $F_K(X)$ are ignored.

ASYMMETRIC PRIMITIVES
BASED UPON THE COMPLEXITY OF DISCRETE LOGARITHM
AND THE BLOCK CIPHER CALLED
"NUSH"

DEFINITIONS

Let P be a prime number of n bits. Montgomery multiplication $\otimes$ of two integers A, B is defined by:

$$A \otimes B = \frac{A \bullet B + P \bullet ((A \bullet B) \bullet M \bmod 2^N)}{2^N}, \text{ for } N \geq n,\ M = -\frac{1}{P} \bmod 2^N .$$

Division by 2^N is just an ignoring of the least N bits of this sum (which really are equal zero). For integers $A[i] \in [0,2^n-1]$ Montgomery product of the integers $A[1] \otimes A[2] \otimes \ldots \otimes A[J]$ is from the interval $[0,2^N-1]$, only if $N \geq n+2$.

For our cryptographic reasons we modify this operation and denote the new one by $\Diamond$:

$$\mathbf{A} \Diamond \mathbf{B} = \begin{cases} A \otimes B, if A \otimes B < 2^n \\ A \otimes B - P, else \end{cases},$$

Where the operation $\otimes$ is computed with the parameter $N = n$.

This new binary operation $\Diamond$, as well as the Montgomery multiplication $\otimes$ is commutative, but not necessary associative if defined for the integers from the interval $[1,2^n-1]$.

To form a finite abelian group of the integers with respect to the new operation define the new equivalence relation for the integers from the interval $[1,2^n-1]$:

$$A \equiv B \Leftrightarrow A\text{-}B = \begin{cases} P \\ 0 \\ -P \end{cases}.$$

Then we get a finite abelian group with respect to the operation $\Diamond$. It is isomorphic to group of units F_P^* of a finite prime field F_p, but it has different representation by the integers mod P, then the standard representation of this field.

The isomorphism φ: $F_P^* \to$ G for the standard presentation of F_P^* is given by

$$\varphi: m \to m{*}2^n \bmod P.$$

In particular, there is an integer a from the interval $[1,2^n-1]$, powers of which with respect to the operation ◊ form this group G.
Thus, group exponentiation of an integer a to the power m by ◊ we denote as $a^{<m>}$.

The elements of group G may be represented by the integers of the interval $[1,2^n-1]$ in several different ways: some elements have a unique representation, others have two, but prime P does not have a presentation of this form at all.

To combine two different representations of a group element in one we define a function:

$$|A| = \begin{cases} A, if A < P \\ A - P, else \end{cases},$$

That is, $A \equiv B \Leftrightarrow |A| = |B|$.

By $E_k(M)$ we denote encryption result of a message M by a key k with NUSH algorithm.

The exact algorithm from the class NUSH (block cipher, stream, . . .) and the key length may be taken with respect to required security level and environment.

We regard a key as an integer from the interval $[1,P-1]$, ignoring if necessary the superfluous bits (at the high end of the number).
By $D_k(F)$ we denote a decryption result of cryptogram F with a key k.

In binary representation of an integer the least bit is regarded as the first one.
Concatenation of the texts A, B denoted as $A||B$.

8. Asymmetric encryption schemes "NUSH PKCode".

PUBLIC PARAMETERS OF THE ALGORITHM

Prime number P of n bits such, that the number $\frac{P-1}{2}$ is also prime.

Group generator a of group G.

PRIVATE KEY

Uniformly distributed random integer x from [*1,P-2*].

PUBLIC KEY

Group element $b = a^{<x>}$.

ENCRYPTION

Take at random an integer r from the interval [*1,P-1*].
Compute a group element $c = a^{<r>}$.
By the public key b compute a group element d = $|b^{<r>}|$, the least bits of which will be used as an encryption session key for symmetric NUSH algorithm.

Encrypt message M by the key d with NUSH algorithm:

$$e = E_d(M) = E_{|b^{<r>}|}(M).$$

A cryptogram f equals to concatenation of two vectors: $f = c \| e$.

DECRYPTION

Find in the cryptogram f a header c of n bits and a cipher text e.

With the header c and the private key x find a group element $d = |c^{<x>}|$, the low bits of which form the decryption key for NUSH.

Decrypt this cipher text e by this key d:

$$M = D_d(e) = D_{|c^{<x>}|}(e).$$

CHOICE OF PARAMETERS

Key length: 64, 128, 256 bits.

Prime P has to be taken with respect to complexity evaluation of DLOG problem, which is now as high as $e^{\sqrt[3]{\frac{32}{9}\log P \log^2 \log P}}$ of elementary operations.
Length n in bits of the prime P has to be multiple of 64, that is the most convenient word length for the most recent processors.

PARAMETERS GENERATION

To generate prime number P of n bits and group generator *a* we use a pseudo random sequence of bytes: b_0, b_1, . . . , from which we take the bytes to generate pseudo random numbers.

PRIME NUMBER GENERATION

Generate a decreasing sequence of integers (lengths) $n_0 > n_1 > \ldots n_t$ by the formula:

$n_0 = n-1$,

$$n_1 = [\frac{n_0+1}{2}]+[\frac{3n_0(b_0+b_1 2^8)}{2^{20}}];$$

…

$$n_i = [\frac{n_{i-1}+1}{2}]+[\frac{3n_{i-1}(b_{2i-2}+b_{2i-1}2^8)}{2^{20}}];$$

n_t is the first of integers n[i] =< 32.

To create prime P_t of $n_t \leq 32$ bits we take the next $k = [\frac{n_t+7}{8}]$ bytes of the pseudo random sequence $b_j, b_{j+1}, \ldots b_{j+k-1}$, and combine integer $u = (b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^{n_t}$.

If u < 2^{n_t-1}, then $u = u + 2^{n_t-1}$. If u is even, then increase u by 1.

On the interval $[u, u+16n_t-2]$ we take the least prime =< 2^{n_t} , which is taken as P_t.

Primness of an integer x is proven by a trial division by all the primes up to $\sqrt{x}$.

If we could not find a prime on this particular interval, we take the next k bytes of pseudo random sequence and form the next interval.

Primes $P_{t-1}, \ldots, P_1$ with the lengths $n_{t-1}, \ldots, n_1$ are generated by the following algorithm.

1. To create prime P_i of n_i bits we get the next $k = [\frac{n_i + 7}{8}]$ bytes of pseudo random sequence $b_j, b_{j+1}, \dots b_{j+k-1}$, and construct the integer mod(2**n)

   $u = (b_j + b_{j+1}2^8 + \dots + b_{j+k-1}2^{8(k-1)}) \bmod 2^{n_i}$.

   If u< $2^{n_i - 1}$, then $u = u + 2^{n_i - 1}$.

   If u is even, increase u by 1.

   Compute the integer $h = [\frac{u}{P_{i+1}}]$. If h is odd, then decrease h by 1.

2. Try all the even numbers m from the interval [h, h+16n_i-2] and for each of them

   verify the following conditions:

a). $mP_{i+1}+1 > 2^{n_i - 1}$

If no, then take the next even number m from the interval.

b). $mP_{i+1}+1 < 2^{n_i}$

If no, then go to step 1 and repeat the same computations for the next part of the pseudo random sequence.

c). There exists prime number S from the interval [3,251] such that:

$$(S2^{-3})^m \neq 1 \bmod (mP_{i+1} + 1).$$

$$(S2^{-3})^{mP_{i+1}} = 1 \bmod (mP_{i+1} + 1);$$

If all three of the conditions a) – c) are true, then put

$$P_i = mP_{i+1}+1$$

and go to generation of the next prime P_{i-1}.

If for no even number m from the interval [h, h+16n_i-2] all the conditions a)-c) are true, then go to step 1 and repeat the same computations for the next part of the pseudo random sequence.

Generation of prime number P is the same as above, but for the step with i =0 we have to check the fourth condition of the step 2 above:

d). $2^{2mP_1+2} = 1 \bmod (2mP_1+3)$

If all of the four statements a) – d) are true , then $P = 2mP_1+3$.

CREATION OF GROUP GENERATORS

Take the next k = n/8 bytes of the pseudo random sequence b_j, b_{j+1}, . . . , b_{j+k-1}, and form a integer $a = b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}$.
Then, check the following three conditions:

$$a \neq 0 \ mod\ P$$

$$a^{<\frac{P-1}{2}>} \lozenge 1 \neq 1$$

$$a^{<2>} \lozenge 1 \neq 1.$$

If not all of them are true, then start the process again.

Integer *a,* which satisfies to all three of the conditions above is a group generator we search for.

GENERATION OF PSEUDO RANDOM SEQUENCES

Take the initial random 62 bytes d_0, …, d_{61}.

Form the numbers:

$X_0 = d_0+256*d_1 \bmod 65257$,
$X_1 = d_2 + 256*d_3 \bmod 65257$,
. . . .
$X_{30} = d_{60} + 256*d_{61} \bmod 65257$.

If $X_0 = 0$, then put $X_0 = 1$.

The sequence of the integers X_i for i = 31, … is generated by the formula:

$$X_i = X_{i-31} - X_{i-21} \bmod 65257.$$

By the X_i define recursively the following sequences, the last of which is taken as the pseudo random sequence we need:

$V_0 = X_0$,

$V_i = V_{i-1} + X_i \bmod 2^{16}$;

$W_0 = X_{20}$,
$W_i = 2^{15} * W_{i-1} + [W_{i-1}/2] + X_{i+20} \bmod 2^{16}$;

$H_0 = 0$;
$$H_i = \left[\frac{(V_{i+10} \oplus W_{i+10}) + H_{i-1} \bmod 2^{16}}{256}\right]$$

$b_i = (V_{i+255} \oplus W_{i+255}) + H_{i+244} \bmod 256$

here [Y] is an integer part of the number Y, and the operation ⊕ is XOR.

9. Digital Signature Algorithm "NUSH Sign" based upon the block cipher NUSH.

DEFINITIONS

Let *P* be a prime number of *n* bits. Montgomery multiplication ⊗ of two integers *A, B* is defined by:

$$A \otimes B = \frac{A \bullet B + P \bullet ((A \bullet B) \bullet M \bmod 2^N)}{2^N}, \text{ for } N \geq n, M = -\frac{1}{P} \bmod 2^N.$$

Division by 2^N is just an ignoring of the least N bits of this sum (which really are equal zero). For integers $A[i] \in [0,2^n-1]$ Montgomery product of the integers $A[1] \otimes A[2] \otimes \ldots \otimes A[J]$ is from the interval $[0,2^N-1]$, only if $N \geq n+2$.

For our cryptographic reasons we modify this operation and denote the new one by ◊:

$$\mathbf{A} \lozenge \mathbf{B} = \begin{cases} A \otimes B, if A \otimes B < 2^n \\ A \otimes B - P, else \end{cases},$$

Where the operation ⊗ is computed with the parameter $N = n$.

This new binary operation ◊, as well as the Montgomery multiplication ⊗ is commutative, but not necessary associative if defined for the integers from the interval $[1,2^n-1]$.

To form a finite abelian group of the integers with respect to the new operation define the new equivalence relation for the integers from the interval $[1,2^n-1]$:

$$A \equiv B \Leftrightarrow A\text{-}B = \begin{cases} P \\ 0 \\ -P \end{cases}.$$

Then we get a finite abelian group with respect to the operation ◊. It is isomorphic to group of units F_P^* of a finite prime field F_p, but it has different representation by the integers mod P, then the standard representation of this field.

The isomorphism φ: $F_P^* \rightarrow$ G for the standard presentation of F_P^* is given by

$$\varphi: m \rightarrow m * 2^n \bmod P.$$

In particular, there is an integer a from the interval $[1,2^n-1]$, powers of which with respect to the operation ◊ form this group G.
Thus, group exponentiation of an integer a to the power m by ◊ we denote as $a^{<m>}$.

The elements of group G may be represented by the integers of the interval $[1,2^n-1]$ in several different ways: some elements have a unique representation, others have two, but prime P does not have a presentation of this form at all.

To combine two different representations of a group element in one we define a function:

$$|A| = \begin{cases} A, if A < P \\ A - P, else \end{cases},$$

That is, $A \equiv B \Leftrightarrow |A| = |B|$.
We also use prime number Q of $2m$ bits dividing P-1. For the prime Q we define binary operation ◊ in the same way as we did it for the prime P, the last operation we denote by ° to distinguish it from the first operation.
In group G there is a subgroup H of order Q , generator of the subgroup H we denote by g.

For the numbers $A < Q$, $B \in [0, 2^{2m}$-1] we define $A - B = \begin{cases} A - B, if A - B \geq 0 \\ A - B + Q, else \end{cases}$.

By the $H_k(M)$ we denote the result of hashing of a text M by the NUSH Hash algorithm to a string of k bits.
If k is less than hash value length, then we take the last (high) k bits of the binary representation of the corresponding integer. Remind that the least bit we take as the first one.

PUBLIC PARAMETERS OF THE ALGORITHM

Prime number P of n bits.
Prime number Q of 2m bits, that divides P.
A generator g of a subgroup H of order Q in group G.

PRIVATE KEY TO SIGN

Uniformly distributed random integer x from the interval [1,Q-1].

PUBLIC KEY TO VERIFY SIGNATURES

Group element $b = g^{<x \circ 1>}$.

DIGITAL SIGNATURE COMPUTATION

Get at random integer r from the interval [1,Q-1].
Compute group element $c = |g^{<r>}|$.
Compute the first part of signature

$$d = \mathrm{H_m}(\mathrm{M}||c)$$

from the message M or from its hash value H(M). If d = 0, then we start all the procedure again.
By the private key x compute the second pert of the signature

$$e = r - (x \circ d) \bmod Q.$$

Combine the parts of signature to get the signature as a whole

$$s = d \,\|\, e.$$

DIGITAL SIGNATURE VERIFICATION

Split signature s by prefix d of m bits and suffix e.
If $d = 0$, then the signature rejected.
By the public key b compute group element $h = |\, g^{<e>} \lozenge b^{<d>} \,|$.
By the signed text M (or by its hash value) verify the equality

$$d = \mathrm{H_m}(\mathrm{M} \| h);$$

CHOICE OF PARAMETERS

Prime number P has to be taken with respect to security level required. This choice is defined by the DLOG complexity evaluation from the formula $e^{\sqrt[3]{\frac{32}{9}\log P \log^2 \log P}}$ of elementary operations. The length n of the prime P should be multiple of 64 in regard the of next generation processors.
Prime divisor Q has to have length equal to 2m and multiple 16 with the inequality

$$\sqrt{Q} \geq e^{\sqrt[3]{\frac{32}{9}\log P \log^2 \log P}}.$$

GENERATION OF PARAMETERS

To generate prime P of n bits with a prime divisor Q of $2m$ bits and group generator a we use pseudo random sequence of bytes: $b_0, b_1, \ldots$, which is used to generate all the pseudo random numbers we need.

GENERATION OF PRIME P of n BITS WITH PRIME DIVISOR Q of 2m<n/2 BITS.

Generate two decreasing sequences of integers (lengths of primes) $m_0 > m_1 > \ldots m_t$ and $n_0 > n_1 > \ldots > n_l$ by the formulae:

$m_0 = 2m$,

$$m_1 = [\frac{m_0+1}{2}] + [\frac{3m_0(b_0 + b_1 2^8)}{2^{20}}];$$

…

$$m_i = [\frac{m_{i-1}+1}{2}] + [\frac{3m_{i-1}(b_{2i-2} + b_{2i-1} 2^8)}{2^{20}}];$$

$n_0 = [\frac{n+1}{2}] + [\frac{3(n-2m)(b_{2t}+b_{2t+1}2^8)}{2^{20}}]$;

…

$n_i = [\frac{n_{i-1}+1}{2}] + [\frac{3n_{i-1}(b_{2t+2i}+b_{2t+2i+1}2^8)}{2^{20}}]$;

where [x] is for the integer part of x, and m_t, n_l are the first of numbers =< 32.

Generate primes $Q_t, Q_{t-1}, \ldots, Q_0 = Q$ of the $m_t, m_{t-1}, \ldots, m_0$, bits each and then primes $P_l, P_{l-1}, \ldots, P_0$ of the $n_l, n_{l-1}, \ldots, n_0$ bits.

To create prime Q_t of $n_t \le 32$ bits we take the next $k = [\frac{m_t+7}{8}]$ bytes of the pseudo random sequence $b_j, b_{j+1}, \ldots b_{j+k-1}$, and the combine an integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^{m_t}$. If u< 2^{m_t-1}, then we put $u = u + 2^{m_t-1}$. If u is even, then increase it by 1.

Then from the interval $[u, u+16m_t-2]$ we take the least prime =< 2^{m_t}, and take it as Q_t. Primness testing of x is made by the trial division by the primes up to $\sqrt{x}$. If we do not find the primes from the interval, then we take the next k bytes of the initial pseudo random sequence and repeat the same procedure.

Primes $Q_{t-1}, \ldots, Q_0$ of $m_{t-1}, \ldots, m_0$ bits are generated by the following algorithm.

1.To create prime Q_i of m_i bits we take the next $k = [\frac{m_i+7}{8}]$ bytes of the pseudo random sequence of bytes $b_j, b_{j+1}, \ldots b_{j+k-1}$, and combine the integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^{m_i}$.

If u< 2^{m_i-1}, then put $u = u + 2^{m_i-1}$. If u is even, then increase it by 1.

Compute $R = [\frac{u}{Q_{i+1}}]$. If R is even, the decrease it by 1.

2. Try the even numbers h from $[R, R+16m_i-2]$ and check the conditions:

a). $hQ_{i+1}+1 > 2^{m_i-1}$

If no, then take the next number h.

b). $hQ_{i+1}+1 < 2^{m_i}$

If no, then go to step 1 and do the same computations for the new sequence.

c). There exists prime number S from [3,251] with the following properties:

$$(S2^{-3})^h \neq 1 \mod(hQ_{i+1}+1).$$

$$(S2^{-3})^{hQ_{i+1}} = 1 \bmod (hQ_{i+1}+1);$$

If a) – c) are true, then put $Q_i = hQ_{i+1}+1$ and go to Q_{i-1}.

If the conditions a)- c) are not true for the h's from [R, R+16m_i-2], then go to step 1 and do the same computations with the next part of pseudo random sequence.

The sequence of primes $P_l, P_{l-1}, \ldots, P_0$ is generated the same way by the next part of the pseudo random sequence $b_0, b_1, \ldots$.

The prime P we need may be calculated from the numbers P_0 and $Q = Q_0$ in the following way.

1. Take the next $k = [\frac{n+7}{8}]$ bytes of $b_j, b_{j+1}, \ldots b_{j+k-1}$, and integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^n$. If $u< 2^{n-1}$, then put $u = u+2^{n-1}$. If u is even, then increase u by 1. Calculate $R = [\frac{u}{QP_0}]$. If R is even, decrease it by 1.

2. Try all the even integers h from [R, R+16n-2] and for each of them check:

a). $hQP_0 +1 > 2^{n-1}$

If no, then take the next h.

b). $hQP_0 +1 < 2^n$

If no, then go to step 1 and do the same for the next part of the pseudo random sequence.

c). There exists prime number S from [3,251] with the following conditions:

$$(S2^{-3})^{hQ} \neq 1 \bmod (hQP_0+1),$$

$$(S2^{-3})^{hQP_0} = 1 \bmod (hQP_0+1);$$

If all the conditions a) – c) are true, then put $P = hQP_0+1$.

If for all numbers h from [R, R+16n-2] these three conditions are not true together, then go to step 1 with the next part of the pseudo random sequence.

SUBGROUP GENERATOR

Take the next k = n/8 bytes of $b_j, b_{j+1}, \ldots, b_{j+k-1}$, and integer $u = b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}$, compute group element $g = u^{<\frac{P-1}{Q}>}$ and check the conditions:

- $g \neq 0$
- $g \neq P$
- $g \Diamond 1 \neq 1$

If any of them is not true, then repeat the procedure again.

Group element g satisfying all three of them is the generator we search for.

PSEUDO RANDOM SEQUENCES

The initial vector consists of the 62 bytes $d_0, \ldots, d_{61}$.

We form the following numbers
$X_0 = d_0+256*d_1 \mod 65257$,
$X_1 = d_2 + 256*d_3 \mod 65257, \ldots, X_{30} = d_{60} + 256*d_{61} \mod 65257$.
If $X_0 = 0$, put $X_0 = 1$.
The sequence X_i , $i = 31, \ldots$ is computed by the rule: $X_i = X_{i-31} - X_{i-21} \mod 65257$.

We define from the sequence X_i the following sequences, last of which we take as the pseudo random sequence we need.

$V_0 = X_0$,
$V_i = V_{i-1} + X_i \mod 2^{16}$;

$W_0 = X_{20}$,
$W_i = 2^{15}*W_{i-1} + [W_{i-1} /2] + X_{i+20} \mod 2^{16}$;

$H_0 = 0$;
$$H_i = [\frac{(V_{i+10} \oplus W_{i+10}) + H_{i-1} \mod 2^{16}}{256}]$$

$b_i = (V_{i+255} \oplus W_{i+255}) + H_{i+244} \mod 256$.

10. Asymmetric identification schemes "NUSH IDS" based on the block cipher NUSH

DEFINITIONS

Let P be a prime number of n bits. Montgomery multiplication $\otimes$ of two integers A, B is defined by:

$$A \otimes B = \frac{A \bullet B + P \bullet ((A \bullet B) \bullet M \mod 2^N)}{2^N}, \text{ for } N \geq n,\ M = -\frac{1}{P} \mod 2^N .$$

Division by 2^N is just an ignoring of the least N bits of this sum (which really are equal zero). For integers $A[i] \in [0,2^n-1]$ Montgomery product of the integers $A[1] \otimes A[2] \otimes \ldots \otimes A[J]$ is from the interval $[0,2^N-1]$, only if $N \geq n+2$.

For our cryptographic reasons we modify this operation and denote the new one by ◊:

$$\mathbf{A \lozenge B} = \begin{cases} A \otimes B, if A \otimes B < 2^n \\ A \otimes B - P, else \end{cases},$$

Where the operation ⊗ is computed with the parameter $N = n$.

This new binary operation ◊, as well as the Montgomery multiplication ⊗ is commutative, but not necessary associative if defined for the integers from the interval $[1,2^n-1]$.

To form a finite abelian group of the integers with respect to the new operation define the new equivalence relation for the integers from the interval $[1,2^n-1]$:

$$A \equiv B \Leftrightarrow A\text{-}B = \begin{cases} P \\ 0 \\ -P \end{cases}.$$

Then we get a finite abelian group with respect to the operation ◊. It is isomorphic to group of units F_P^* of a finite prime field F_p, but it has different representation by the integers mod P, then the standard representation of this field.

The isomorphism φ: $F_P^* \rightarrow$ G for the standard presentation of F_P^* is given by

$$\varphi\text{: } m \rightarrow m*2^n \ mod \ P.$$

In particular, there is an integer a from the interval $[1,2^n-1]$, powers of which with respect to the operation ◊ form this group G.
Thus, group exponentiation of an integer a to the power m by ◊ we denote as $a^{<m>}$.

The elements of group G may be represented by the integers of the interval $[1,2^n-1]$ in several different ways: some elements have a unique representation, others have two, but prime P does not have a presentation of this form at all.

To combine two different representations of a group element in one we define a function:

$$|A| = \begin{cases} A, if A < P \\ A - P, else \end{cases},$$

That is, $A \equiv B \Leftrightarrow |A| = |B|$.

We also use prime number Q of $2m$ bits dividing P-1.
For the prime Q we define binary operation like ◊ in the same way as we did it for the prime P, the last operation we denote by ° to distinguish it from the operation defined for the prime P.
In the group G there is a subgroup H of order Q , generator of the subgroup H we denote by g.

For the numbers $A < Q$, $B \in [0, 2^{2m}\text{-}1]$ we define $A - B = \begin{cases} A - B, if A - B \geq 0 \\ A - B + Q, else \end{cases}$.

By the $H_k(M)$ we denote the result of hashing of a text M by the NUSH Hash algorithm to a string of k bits.
If k is less than hash value length, then we take the last (high) k bits of the binary representation of the corresponding integer. Remind that the least bit we take as the first one.

PUBLIC PARAMETERS OF THE ALGORITHM

Prime number P of n bits.
Prime number Q of 2m bits, that divides P.
A generator g of a subgroup H of order Q in group G.

PRIVATE IDENTIFICATION KEY

Uniformly distributed random integer x from [1,Q-1].

PUBLIC IDENTIFICATION KEY

Group element $b = g^{<x \circ 1>}$.

IDENTIFICATION

1. First round

Prover generates random integer r from [1,Q-1] and computes

$$c = H_m(|\, g^{<r>} \,|),$$

which he sends to verifier.

2. Second round

Verifier generates random integer k from $[0, 2^t\text{-}1]$ and sends it to prover.

Here t < 2m is reliability level.

3. Third round

Prover computes

$$d = r - x \circ H_m(c \| k) \text{ mod Q}$$

and sends it to verifier.

Verifier check that

$$H_m(| g^{<d>} \Diamond b^{H_m(c\|k)} |) = c.$$

CHOICE OF PARAMETERS

Prime number P has to be taken with respect to security level required. This choice is defined by the DLOG complexity evaluation from the formula $e^{\sqrt[3]{\frac{32}{9}\log P \log^2 \log P}}$ of elementary operations. The length *n* of the prime P should be multiple of 64 in regard the of next generation processors.
Prime divisor Q has to have length equal to 2m and multiple 16 with the inequality

$$\sqrt{Q} \geq e^{\sqrt[3]{\frac{32}{9}\log P \log^2 \log P}}.$$

GENERATION OF PARAMETERS

To generate prime *P* of *n* bits with a prime divisor *Q* of *2m* bits and group generator *a* we use pseudo random sequence of bytes: b_0, b_1, . . . , which is used to generate all the pseudo random numbers we need.

GENERATION OF PRIME P of n BITS WITH PRIME DIVISOR Q of 2m<n/2 BITS.

Generate two decreasing sequences of integers (lengths of primes) $m_0 > m_1 > \ldots m_t$ and $n_0 > n_1 > \ldots > n_l$ by the formulae:

$m_0 = 2m$,

$$m_1 = [\frac{m_0 + 1}{2}] + [\frac{3m_0(b_0 + b_1 2^8)}{2^{20}}];$$

…

$$m_i = [\frac{m_{i-1} + 1}{2}] + [\frac{3m_{i-1}(b_{2i-2} + b_{2i-1} 2^8)}{2^{20}}];$$

$$n_0 = [\frac{n + 1}{2}] + [\frac{3(n - 2m)(b_{2t} + b_{2t+1} 2^8)}{2^{20}}];$$

…

$n_i = [\frac{n_{i-1}+1}{2}] + [\frac{3n_{i-1}(b_{2t+2i} + b_{2t+2i+1}2^8)}{2^{20}}]$;

where [x] is for the integer part of x, and m_t, n_l are the first of numbers =< 32.

Generate primes $Q_t, Q_{t-1}, \ldots, Q_0 = Q$ of the $m_t, m_{t-1}, \ldots, m_0$, bits each and then primes $P_l, P_{l-1}, \ldots, P_0$ of the $n_l, n_{l-1}, \ldots, n_0$ bits.

To create prime Q_t of $n_t \le 32$ bits we take the next $k = [\frac{m_t+7}{8}]$ bytes of the pseudo random sequence $b_j, b_{j+1}, \ldots b_{j+k-1}$, and the combine an integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^{m_t}$. If u< 2^{m_t-1}, then we put $u = u + 2^{m_t-1}$. If u is even, then increase it by 1.

Then from the interval $[u, u+16m_t-2]$ we take the least prime =< 2^{m_t}, and take it as Q_t. Primness testing of x is made by the trial division by the primes up to $\sqrt{x}$. If we do not find the primes from the interval, then we take the next k bytes of the initial pseudo random sequence and repeat the same procedure.

Primes $Q_{t-1}, \ldots, Q_0$ of $m_{t-1}, \ldots, m_0$ bits are generated by the following algorithm.

1.To create prime Q_i of m_i bits we take the next $k = [\frac{m_i+7}{8}]$ bytes of the pseudo random sequence of bytes $b_j, b_{j+1}, \ldots b_{j+k-1}$, and combine the integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^{m_i}$.

If u< 2^{m_i-1}, then put $u = u + 2^{m_i-1}$. If u is even, then increase it by 1.

Compute $R = [\frac{u}{Q_{i+1}}]$. If R is even, the decrease it by 1.

2. Try the even numbers h from $[R, R+16m_i-2]$ and check the conditions:

a). $hQ_{i+1}+1 > 2^{m_i-1}$

If no, then take the next number h.

b). $hQ_{i+1}+1 < 2^{m_i}$

If no, then go to step 1 and do the same computations for the new sequence.

c). There exists prime number S from [3,251] with the following properties:

$$(S2^{-3})^h \neq 1 \mod(hQ_{i+1}+1).$$

$$(S2^{-3})^{hQ_{i+1}} = 1 \mod(hQ_{i+1}+1);$$

If a) – c) are true, then put $Q_i = hQ_{i+1}+1$ and go to Q_{i-1}.

If the conditions a)- c) are not true for the h's from [R, R+16m_i-2], then go to step 1 and do the same computations with the next part of pseudo random sequence.

The sequence of primes $P_l, P_{l-1}, \ldots, P_0$ is generated the same way by the next part of the pseudo random sequence $b_0, b_1, \ldots$.

The prime P we need may be calculated from the numbers P_0 and $Q = Q_0$ in the following way.

1. Take the next $k = [\frac{n+7}{8}]$ bytes of $b_j, b_{j+1}, \ldots b_{j+k-1}$, and integer u = $(b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}) \bmod 2^n$. If $u< 2^{n-1}$, then put $u = u+2^{n-1}$. If u is even, then increase u by 1. Calculate $R = [\frac{u}{QP_0}]$. If R is even, decrease it by 1.

2. Try all the even integers h from [R, R+16n-2] and for each of them check:

a). $hQP_0 +1 > 2^{n-1}$

If no, then take the next h.

b). $hQP_0 +1 < 2^n$

If no, then go to step 1 and do the same for the next part of the pseudo random sequence.

c). There exists prime number S from [3,251] with the following conditions:

$$(S2^{-3})^{hQ} \neq 1 \bmod(hQP_0 +1),$$

$$(S2^{-3})^{hQP_0} = 1 \bmod(hQP_0 + 1);$$

If all the conditions a) – c) are true, then put $P = hQP_0+1$.

If for all numbers h from [R, R+16n-2] these three conditions are not true together, then go to step 1 with the next part of the pseudo random sequence.

SUBGROUP GENERATOR

Take the next k = n/8 bytes of $b_j, b_{j+1}, \ldots, b_{j+k-1}$, and integer $u = b_j+b_{j+1}2^8+\ldots+b_{j+k-1}2^{8(k-1)}$, compute group element $g = u^{<\frac{P-1}{Q}>}$ and check the conditions:

- $g \neq 0$
- $g \neq P$
- $g \Diamond 1 \neq 1$

If any of them is not true, then repeat the procedure again.

Group element *g* satisfying all three of them is the generator we search for.

PSEUDO RANDOM SEQUENCES

The initial vector consists of the 62 bytes $d_0, \ldots, d_{61}$.

We form the following numbers
$X_0 = d_0+256*d_1 \bmod 65257$,
$X_1 = d_2 + 256*d_3 \bmod 65257, \ldots, X_{30} = d_{60} + 256*d_{61} \bmod 65257$.
If $X_0 = 0$, put $X_0 = 1$.
The sequence X_i , $i = 31, \ldots$ is computed by the rule: $X_i = X_{i-31} - X_{i-21} \bmod 65257$.

We define from the sequence X_i the following sequences, last of which we take as the pseudo random sequence we need.

$V_0 = X_0$,
$V_i = V_{i-1} + X_i \bmod 2^{16}$;

$W_0 = X_{20}$,
$W_i = 2^{15}*W_{i-1} + [W_{i-1} /2] + X_{i+20} \bmod 2^{16}$;

$H_0 = 0$;
$$H_i = [\frac{(V_{i+10} \oplus W_{i+10}) + H_{i-1} \bmod 2^{16}}{256}]$$

$b_i = (V_{i+255} \oplus W_{i+255}) + H_{i+244} \bmod 256$.