Workgroup: Internet Engineering Task Force
Published: 11 May 2023
Intended Status: Informational
Expires: 12 November 2023
Authors: S. Josefsson

Streamlined NTRU Prime: sntrup761

Abstract

We provide an algorithm introduction, reference code and test vectors for the Streamlined NTRU Prime key-encapsulation mechanism "sntrup761".

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 November 2023.

Table of Contents

1. Introduction

Streamlined NTRU Prime [NTRUPrime] [NTRUPrimePQCS] provides post-quantum small lattice-based key-encapsulation mechanisms. The variant sntrup761 has been implemented widely.

The focus of this document is to aid implementers that wish to implement Streamlined NTRU Prime sntrup761, by providing an introduction to the algorithm, Sage and C reference code with test vectors.

The original NTRU Prime paper [NTRUPrime] and the primary submission document of NTRU Prime to NIST's Post-Quantum Cryptography Standardization Project" [NTRUPrimePQCS] provide background, and is the source for the majority of content in this document.

Instead of attempting to describe the algorithm in human language, we refer to the exact algorithm description offered in SageMath [SageMath]. For traditional implementation, we also offer a reference C implementation and a test driver that is able to reproduce the test vectors.

2. Streamlined NTRU Prime: key generation

The following algorithm outputs the Streamlined NTRU Prime public/ private key-pair, and the session key.

```
def KEM_KeyGen():
  pk,sk = ZKeyGen()
  sk += pk
  rho = Inputs_randomenc()
  sk += rho
  sk += Hash4(pk)
  return pk,sk
```

Figure 1

3. Streamlined NTRU Prime: encapsulation

The following algorithm outputs the Streamlined NTRU Prime encapsulated session key.

```
def Encap(pk):
  r = Inputs_random()
  C,r_enc = Hide(r,pk)
  return C,HashSession(1,r_enc,C)
```

Figure 2

4. Streamlined NTRU Prime: decapsulation

The following algorithm outputs the Streamlined NTRU Prime decapsulated session key.

```
def Decap(C,sk):
  Corig = C
  c_inner,C = C[:Ciphertexts_bytes],C[Ciphertexts_bytes:]
  gamma,C = C[:Confirm_bytes],C[Confirm_bytes:]
  assert len(C) == 0

  sk_inner,sk = sk[:SecretKeys_bytes],sk[SecretKeys_bytes:]
  pk,sk = sk[:PublicKeys_bytes],sk[PublicKeys_bytes:]
  rho,sk = sk[:Inputs_bytes],sk[Inputs_bytes:]
  cache = None
  cache,sk = sk[:Hash_bytes],sk[Hash_bytes:]
  assert len(sk) == 0

  r = ZDecrypt(c_inner,sk_inner)
  Cnew,r_enc = Hide(r,pk,cache)

  assert len(r_enc) == Inputs_bytes
  assert len(rho) == Inputs_bytes

  if Cnew == Corig: return HashSession(1,r_enc,Corig)
  return HashSession(0,rho,Corig)
```

Figure 3

5. Streamlined NTRU Prime: Complete Sage implementation

The functions above call several other functions, which are specified below.

```
sntrup = { # p,q,w
  'sntrup761':    (761,4591,286),
  'sntrup653':    (653,4621,288),
  'sntrup857':    (857,5167,322),
  'sntrup953':    (953,6343,396),
  'sntrup1013':   (1013,7177,448),
  'sntrup1277':   (1277,7879,492),
}

def setparameters(system,random8):
  global p
  global q
  global w

  global PublicKeys_bytes
  global SecretKeys_bytes
  global Ciphertexts_bytes
  global Inputs_bytes
  global Confirm_bytes
  global Hash_bytes
  global Rq_encode
  global Inputs_random
  global ZKeyGen
  global ZEncrypt
  global ZDecrypt

  if system in sntrup:
    p,q,w = sntrup[system]
  else:
    raise Exception('%s is not one of the selected parameter sets' % sys

  # ----- parameter requirements
  # tested later: irreducibility of x^p-x-1 mod q

  assert p.is_prime()
  assert q.is_prime()
  assert w > 0
  assert 2*p >= 3*w
  assert q >= 16*w+1
  assert q%6 == 1 # spec allows 5 but these tests do not
  assert p%4 == 1 # spec allows 3 but ref C code does not

  # ----- arithmetic mod 3

  F3 = GF(3)
  def ZZ_fromF3(c):
    assert c in F3
    return ZZ(c+1)-1

  # ----- arithmetic mod q
```

```
Fq = GF(q)
q12 = ZZ((q-1)/2)
def ZZ_fromFq(c):
  assert c in Fq
  return ZZ(c+q12)-q12

# ----- polynomials over integers

global R

Zx.<x> = ZZ[]
R.<xp> = Zx.quotient(x^p-x-1)

def Weightw_is(r):
  assert r in R
  return w == len([i for i in range(p) if r[i] != 0])

def Small_is(r):
  assert r in R
  return all(abs(r[i]) <= 1 for i in range(p))

def Short_is(r):
  return Small_is(r) and Weightw_is(r)

# ----- polynomials mod 3

F3x.<x3> = F3[]
R3.<x3p> = F3x.quotient(x^p-x-1)

def R_fromR3(r):
  assert r in R3
  return R([ZZ_fromF3(r[i]) for i in range(p)])

def R3_fromR(r):
  assert r in R
  return R3([r[i] for i in range(p)])

# ----- polynomials mod q

Fqx.<xq> = Fq[]
assert (xq^p-xq-1).is_irreducible()

global Rq
Rq.<xqp> = Fqx.quotient(x^p-x-1)

global R_fromRq
def R_fromRq(r):
  assert r in Rq
  return R([ZZ_fromFq(r[i]) for i in range(p)])
```

```
global Rq_fromR
def Rq_fromR(r):
  assert r in R
  return Rq([r[i] for i in range(p)])

# ----- rounded polynomials mod q

def Rounded_is(r):
  assert r in R
  return (all(r[i]%3 == 0 for i in range(p))
    and all(r[i] >= -q12 for i in range(p))
    and all(r[i] <= q12 for i in range(p)))

def Round(a):
  assert a in Rq
  c = R_fromRq(a)
  r = [3*round(c[i]/3) for i in range(p)]
  assert all(abs(r[i]-c[i]) <= 1 for i in range(p))
  r = R(r)
  assert Rounded_is(r)
  return r

# ----- sorting to generate short polynomial

global Short_fromlist
def Short_fromlist(L): # L is list of p uint32
  L = [L[i]&-2 for i in range(w)] + [(L[i]&-3)|1 for i in range(w,p)]
  assert all(L[i]%2 == 0 for i in range(w))
  assert all(L[i]%4 == 1 for i in range(w,p))
  L.sort()
  L = [(L[i]%4)-1 for i in range(p)]
  assert all(abs(L[i]) <= 1 for i in range(p))
  assert sum(abs(L[i]) for i in range(p)) == w
  r = R(L)
  assert Short_is(r)
  return r

# ----- underlying hash function

import hashlib

global sha512
def sha512(s):
  h = hashlib.sha512()
  h.update(s)
  return h.digest()

Hash_bytes = 32
def Hash(s): return sha512(s)[:Hash_bytes]
```

```
def Hash0(s): return Hash(chr(0)+s)
def Hash1(s): return Hash(chr(1)+s)
def Hash2(s): return Hash(chr(2)+s)
def Hash3(s): return Hash(chr(3)+s)
def Hash4(s): return Hash(chr(4)+s)

# ----- higher-level randomness

def urandom32():
  c0 = random8()
  c1 = random8()
  c2 = random8()
  c3 = random8()
  return c0 + 256*c1 + 65536*c2 + 16777216*c3

global Short_random
def Short_random(): # R element with w coeffs +-1
  L = [urandom32() for i in range(p)]
  return Short_fromlist(L)

def randomrange3():
  return ((urandom32() & 0x3fffffff) * 3) >> 30

def Small_random():
  r = R([randomrange3()-1 for i in range(p)])
  assert Small_is(r)
  return r

# ----- Streamlined NTRU Prime Core

global KeyGen
global Encrypt
global Decrypt

def KeyGen():
  while True:
    g = Small_random()
    if R3_fromR(g).is_unit(): break
  f = Short_random()
  h = Rq_fromR(g)/Rq_fromR(3*f)
  return h,(f,1/R3_fromR(g))

def Encrypt(r,h):
  assert Short_is(r)
  assert h in Rq
  return Round(h*Rq_fromR(r))

def Decrypt(c,k):
  f,v = k
```

```
  assert Rounded_is(c)
  assert Short_is(f)
  assert v in R3
  e = R3_fromR(R_fromRq(3*Rq_fromR(f)*Rq_fromR(c)))
  r = R_fromR3(e*v)
  if Weightw_is(r): return r
  return R([1]*w+[0]*(p-w))

# ----- strings

global tostring
def tostring(s):
  return ''.join(chr(si) for si in s)

def fromstring(s):
  return [ord(si) for si in s]

# ----- encoding small polynomials (including short polynomials)

Small_bytes = ceil(p/4)

def Small_encode(r):
  assert Small_is(r)
  R = [r[i]+1 for i in range(p)]
  while len(R) < 4*Small_bytes: R += [0]
  assert all(R[i] >= 0 for i in range(4*Small_bytes))
  assert all(R[i] <= 2 for i in range(4*Small_bytes))
  assert len(R) >= p
  assert len(R)%4 == 0
  S = [R[i]+4*R[i+1]+16*R[i+2]+64*R[i+3] for i in range(0,len(R),4)]
  return tostring(S)

def Small_decode(s):
  S = fromstring(s)
  r = [(S[i//4]//4^(i%4))%4 for i in range(p)]
  assert all(r[i] >= 0 for i in range(p))
  assert all(r[i] <= 2 for i in range(p))
  r = [r[i]-1 for i in range(p)]
  return R(r)

# ----- infrastructure for more general encoding

limit = 16384

def Encode(R,M):
  if len(M) == 0: return []
  S = []
  if len(M) == 1:
    r,m = R[0],M[0]
    while m > 1:
```

```
      S += [r%256]
      r,m = r//256,(m+255)//256
    return S
  R2,M2 = [],[]
  for i in range(0,len(M)-1,2):
    m,r = M[i]*M[i+1],R[i]+M[i]*R[i+1]
    while m >= limit:
      S += [r%256]
      r,m = r//256,(m+255)//256
    R2 += [r]
    M2 += [m]
  if len(M)&1:
    R2 += [R[-1]]
    M2 += [M[-1]]
  return S+Encode(R2,M2)

def Decode(S,M):
  if len(M) == 0: return []
  if len(M) == 1: return [sum(S[i]*256**i for i in range(len(S)))%M[0]
  k = 0
  bottom,M2 = [],[]
  for i in range(0,len(M)-1,2):
    m,r,t = M[i]*M[i+1],0,1
    while m >= limit:
      r,t,k,m = r+S[k]*t,t*256,k+1,(m+255)//256
    bottom += [(r,t)]
    M2 += [m]
  if len(M)&1:
    M2 += [M[-1]]
  R2 = Decode(S[k:],M2)
  R = []
  for i in range(0,len(M)-1,2):
    r,t = bottom[i//2]
    r += t*R2[i//2]
    R += [r%M[i]]
    R += [(r//M[i])%M[i+1]]
  if len(M)&1:
    R += [R2[-1]]
  return R

# ----- encoding general polynomials

def Rq_encode(r):
  assert r in Rq
  R = [ZZ_fromFq(r[i])+q12 for i in range(p)]
  M = [q]*p
  assert all(0 <= R[i] for i in range(p))
  assert all(R[i] < M[i] for i in range(p))
  return tostring(Encode(R,M))
```

```
def Rq_decode(s):
  assert len(s) == Rq_bytes
  M = [q]*p
  R = Decode(fromstring(s),M)
  assert all(0 <= R[i] for i in range(p))
  assert all(R[i] < M[i] for i in range(p))
  r = [R[i]-q12 for i in range(p)]
  return Rq(r)

global Rq_bytes
Rq_bytes = len(Rq_encode(Rq(0)))

# ----- encoding rounded polynomials

def Rounded_encode(r):
  assert Rounded_is(r)
  R = [ZZ((ZZ_fromFq(r[i])+q12)/3) for i in range(p)]
  M = [ZZ((q-1)/3+1)]*p
  assert all(0 <= R[i] for i in range(p))
  assert all(R[i] < M[i] for i in range(p))
  return tostring(Encode(R,M))

def Rounded_decode(s):
  assert len(s) == Rounded_bytes
  M = [ZZ((q-1)/3+1)]*p
  r = Decode(fromstring(s),M)
  assert all(0 <= r[i] for i in range(p))
  assert all(r[i] < M[i] for i in range(p))
  r = [3*r[i]-q12 for i in range(p)]
  return R(r)

global Rounded_bytes
Rounded_bytes = len(Rounded_encode(R(0)))

# ----- Streamlined NTRU Prime Core plus encoding

Inputs_random = Short_random
Inputs_encode = Small_encode
Inputs_bytes = Small_bytes

Ciphertexts_bytes = Rounded_bytes
SecretKeys_bytes = 2*Small_bytes
PublicKeys_bytes = Rq_bytes

def Inputs_randomenc():
  rho = [random8() for i in range(Small_bytes)]
  return tostring(rho)

def ZKeyGen():
```

```
  h,(f,v) = KeyGen()
  return Rq_encode(h),Small_encode(f)+Small_encode(R_fromR3(v))

def ZEncrypt(r,pk):
  assert len(pk) == PublicKeys_bytes
  h = Rq_decode(pk)
  return Rounded_encode(Encrypt(r,h))

def ZDecrypt(c,sk):
  assert len(sk) == SecretKeys_bytes
  assert len(c) == Ciphertexts_bytes
  f = Small_decode(sk[:Small_bytes])
  v = R3_fromR(Small_decode(sk[Small_bytes:]))
  c = Rounded_decode(c)
  return Decrypt(c,(f,v))

# ----- confirmation hash

Confirm_bytes = 32

def HashConfirm(r,K,cache=None):
  assert len(r) == Inputs_bytes
  assert len(K) == PublicKeys_bytes
  if not cache: cache = Hash4(K)
  assert cache == Hash4(K)
  r = Hash3(r)
  return Hash2(r+cache)

# ----- session-key hash

global HashSession
def HashSession(b,y,z):
  assert len(y) == Inputs_bytes
  assert len(z) == Ciphertexts_bytes+Confirm_bytes
  assert b in [0,1]
  y = Hash3(y)
  if b == 1: return Hash1(y+z)
  return Hash0(y+z)

# ----- Streamlined NTRU Prime

# KeyGen' in Streamlined NTRU Prime spec
global KEM_KeyGen
def KEM_KeyGen():
  pk,sk = ZKeyGen()
  sk += pk
  rho = Inputs_randomenc()
  sk += rho
  sk += Hash4(pk)
  return pk,sk
```

```
global Hide
def Hide(r,pk,cache=None):
  r_enc = Inputs_encode(r)
  c = ZEncrypt(r,pk)
  gamma = HashConfirm(r_enc,pk,cache)
  c = c+gamma
  return c,r_enc

global Encap
def Encap(pk):
  r = Inputs_random()
  C,r_enc = Hide(r,pk)
  return C,HashSession(1,r_enc,C)

global Decap
def Decap(C,sk):
  Corig = C
  c_inner,C = C[:Ciphertexts_bytes],C[Ciphertexts_bytes:]
  gamma,C = C[:Confirm_bytes],C[Confirm_bytes:]
  assert len(C) == 0

  sk_inner,sk = sk[:SecretKeys_bytes],sk[SecretKeys_bytes:]
  pk,sk = sk[:PublicKeys_bytes],sk[PublicKeys_bytes:]
  rho,sk = sk[:Inputs_bytes],sk[Inputs_bytes:]
  cache = None
  cache,sk = sk[:Hash_bytes],sk[Hash_bytes:]
  assert len(sk) == 0

  r = ZDecrypt(c_inner,sk_inner)
  Cnew,r_enc = Hide(r,pk,cache)

  assert len(r_enc) == Inputs_bytes
  assert len(rho) == Inputs_bytes

  if Cnew == Corig: return HashSession(1,r_enc,Corig)
  return HashSession(0,rho,Corig)
```

Figure 4

6. Streamlined NTRU Prime: Complete C implementation

6.1. sntrup761.h

```
/*
 * Derived from public domain source, written by (in alphabetical order)
 * - Daniel J. Bernstein
 * - Chitchanok Chuengsatiansup
 * - Tanja Lange
 * - Christine van Vredendaal
 */

#ifndef SNTRUP761_H
#define SNTRUP761_H

#include <string.h>
#include <stdint.h>

#define SNTRUP761_SECRETKEY_SIZE 1763
#define SNTRUP761_PUBLICKEY_SIZE 1158
#define SNTRUP761_CIPHERTEXT_SIZE 1039
#define SNTRUP761_SIZE 32

typedef void sntrup761_random_func (void *ctx, size_t length, uint8_t *d

void
sntrup761_keypair (uint8_t *pk, uint8_t *sk,
                   void *random_ctx, sntrup761_random_func *random);

void
sntrup761_enc (uint8_t *c, uint8_t *k, const uint8_t *pk,
               void *random_ctx, sntrup761_random_func *random);

void
sntrup761_dec (uint8_t *k, const uint8_t *c, const uint8_t *sk);

#endif /* SNTRUP761_H */
```

Figure 5

6.2. sntrup761.c

```
/*
 * Derived from public domain source, written by (in alphabetical order)
 * - Daniel J. Bernstein
 * - Chitchanok Chuengsatiansup
 * - Tanja Lange
 * - Christine van Vredendaal
 */

#include "sntrup761.h"

#include "sha512.h"

/* from supercop-20201130/crypto_sort/int32/portable4/int32_minmax.inc *
#define int32_MINMAX(a,b) \
do { \
  int64_t ab = (int64_t)b ^ (int64_t)a; \
  int64_t c = (int64_t)b - (int64_t)a; \
  c ^= ab & (c ^ b); \
  c >>= 31; \
  c &= ab; \
  a ^= c; \
  b ^= c; \
} while(0)

/* from supercop-20201130/crypto_sort/int32/portable4/sort.c */
static void
crypto_sort_int32 (void *array, long long n)
{
  long long top, p, q, r, i, j;
  int32_t *x = array;

  if (n < 2)
    return;
  top = 1;
  while (top < n - top)
    top += top;

  for (p = top; p >= 1; p >>= 1)
    {
      i = 0;
      while (i + 2 * p <= n)
        {
          for (j = i; j < i + p; ++j)
            int32_MINMAX (x[j], x[j + p]);
          i += 2 * p;
        }
      for (j = i; j < n - p; ++j)
        int32_MINMAX (x[j], x[j + p]);
```

```
      i = 0;
      j = 0;
      for (q = top; q > p; q >>= 1)
        {
          if (j != i)
            for (;;)
              {
                if (j == n - q)
                  goto done;
                int32_t a = x[j + p];
                for (r = q; r > p; r >>= 1)
                  int32_MINMAX (a, x[j + r]);
                x[j + p] = a;
                ++j;
                if (j == i + p)
                  {
                    i += 2 * p;
                    break;
                  }
              }
          while (i + p <= n - q)
            {
              for (j = i; j < i + p; ++j)
                {
                  int32_t a = x[j + p];
                  for (r = q; r > p; r >>= 1)
                    int32_MINMAX (a, x[j + r]);
                  x[j + p] = a;
                }
              i += 2 * p;
            }
          /* now i + p > n - q */
          j = i;
          while (j < n - q)
            {
              int32_t a = x[j + p];
              for (r = q; r > p; r >>= 1)
                int32_MINMAX (a, x[j + r]);
              x[j + p] = a;
              ++j;
            }

        done:;
        }
    }
}

/* from supercop-20201130/crypto_sort/uint32/useint32/sort.c */
```

```
/* can save time by vectorizing xor loops */
/* can save time by integrating xor loops with int32_sort */

static void
crypto_sort_uint32 (void *array, long long n)
{
  uint32_t *x = array;
  long long j;
  for (j = 0; j < n; ++j)
    x[j] ^= 0x80000000;
  crypto_sort_int32 (array, n);
  for (j = 0; j < n; ++j)
    x[j] ^= 0x80000000;
}

/* from supercop-20201130/crypto_kem/sntrup761/ref/uint32.c */

/*
CPU division instruction typically takes time depending on x.
This software is designed to take time independent of x.
Time still varies depending on m; user must ensure that m is constant.
Time also varies on CPUs where multiplication is variable-time.
There could be more CPU issues.
There could also be compiler issues.
*/

static void
uint32_divmod_uint14 (uint32_t * q, uint16_t * r, uint32_t x, uint16_t m
{
  uint32_t v = 0x80000000;
  uint32_t qpart;
  uint32_t mask;

  v /= m;

  /* caller guarantees m > 0 */
  /* caller guarantees m < 16384 */
  /* vm <= 2^31 <= vm+m-1 */
  /* xvm <= 2^31 x <= xvm+x(m-1) */

  *q = 0;

  qpart = (x * (uint64_t) v) >> 31;
  /* 2^31 qpart <= xv <= 2^31 qpart + 2^31-1 */
  /* 2^31 qpart m <= xvm <= 2^31 qpart m + (2^31-1)m */
  /* 2^31 qpart m <= 2^31 x <= 2^31 qpart m + (2^31-1)m + x(m-1) */
  /* 0 <= 2^31 newx <= (2^31-1)m + x(m-1) */
  /* 0 <= newx <= (1-1/2^31)m + x(m-1)/2^31 */
  /* 0 <= newx <= (1-1/2^31)(2^14-1) + (2^32-1)((2^14-1)-1)/2^31 */
```

```
  x -= qpart * m;
  *q += qpart;
  /* x <= 49146 */

  qpart = (x * (uint64_t) v) >> 31;
  /* 0 <= newx <= (1-1/2^31)m + x(m-1)/2^31 */
  /* 0 <= newx <= m + 49146(2^14-1)/2^31 */
  /* 0 <= newx <= m + 0.4 */
  /* 0 <= newx <= m */

  x -= qpart * m;
  *q += qpart;
  /* x <= m */

  x -= m;
  *q += 1;
  mask = -(x >> 31);
  x += mask & (uint32_t) m;
  *q += mask;
  /* x < m */

  *r = x;
}


static uint16_t
uint32_mod_uint14 (uint32_t x, uint16_t m)
{
  uint32_t q;
  uint16_t r;
  uint32_divmod_uint14 (&q, &r, x, m);
  return r;
}

/* from supercop-20201130/crypto_kem/sntrup761/ref/int32.c */

static void
int32_divmod_uint14 (int32_t * q, uint16_t * r, int32_t x, uint16_t m)
{
  uint32_t uq, uq2;
  uint16_t ur, ur2;
  uint32_t mask;

  uint32_divmod_uint14 (&uq, &ur, 0x80000000 + (uint32_t) x, m);
  uint32_divmod_uint14 (&uq2, &ur2, 0x80000000, m);
  ur -= ur2;
  uq -= uq2;
  mask = -(uint32_t) (ur >> 15);
  ur += mask & m;
  uq += mask;
```

```
  *r = ur;
  *q = uq;
}


static uint16_t
int32_mod_uint14 (int32_t x, uint16_t m)
{
  int32_t q;
  uint16_t r;
  int32_divmod_uint14 (&q, &r, x, m);
  return r;
}

/* from supercop-20201130/crypto_kem/sntrup761/ref/paramsmenu.h */
#define p 761
#define q 4591
#define Rounded_bytes 1007
#define Rq_bytes 1158
#define w 286

/* from supercop-20201130/crypto_kem/sntrup761/ref/Decode.h */

/* Decode(R,s,M,len) */
/* assumes 0 < M[i] < 16384 */
/* produces 0 <= R[i] < M[i] */

/* from supercop-20201130/crypto_kem/sntrup761/ref/Decode.c */

static void
Decode (uint16_t * out, const unsigned char *S, const uint16_t * M,
        long long len)
{
  if (len == 1)
    {
      if (M[0] == 1)
        *out = 0;
      else if (M[0] <= 256)
        *out = uint32_mod_uint14 (S[0], M[0]);
      else
        *out = uint32_mod_uint14 (S[0] + (((uint16_t) S[1]) << 8), M[0])
    }
  if (len > 1)
    {
      uint16_t R2[(len + 1) / 2];
      uint16_t M2[(len + 1) / 2];
      uint16_t bottomr[len / 2];
      uint32_t bottomt[len / 2];
      long long i;
```

```
      for (i = 0; i < len - 1; i += 2)
        {
          uint32_t m = M[i] * (uint32_t) M[i + 1];
          if (m > 256 * 16383)
            {
              bottomt[i / 2] = 256 * 256;
              bottomr[i / 2] = S[0] + 256 * S[1];
              S += 2;
              M2[i / 2] = (((m + 255) >> 8) + 255) >> 8;
            }
          else if (m >= 16384)
            {
              bottomt[i / 2] = 256;
              bottomr[i / 2] = S[0];
              S += 1;
              M2[i / 2] = (m + 255) >> 8;
            }
          else
            {
              bottomt[i / 2] = 1;
              bottomr[i / 2] = 0;
              M2[i / 2] = m;
            }
        }
      if (i < len)
        M2[i / 2] = M[i];
      Decode (R2, S, M2, (len + 1) / 2);
      for (i = 0; i < len - 1; i += 2)
        {
          uint32_t r = bottomr[i / 2];
          uint32_t r1;
          uint16_t r0;
          r += bottomt[i / 2] * R2[i / 2];
          uint32_divmod_uint14 (&r1, &r0, r, M[i]);
          r1 = uint32_mod_uint14 (r1, M[i + 1]);        /* only needed f
          *out++ = r0;
          *out++ = r1;
        }
      if (i < len)
        *out++ = R2[i / 2];
    }
}

/* from supercop-20201130/crypto_kem/sntrup761/ref/Encode.h */

/* Encode(s,R,M,len) */
/* assumes 0 <= R[i] < M[i] < 16384 */

/* from supercop-20201130/crypto_kem/sntrup761/ref/Encode.c */
```

```
/* 0 <= R[i] < M[i] < 16384 */
static void
Encode (unsigned char *out, const uint16_t * R, const uint16_t * M,
        long long len)
{
  if (len == 1)
    {
      uint16_t r = R[0];
      uint16_t m = M[0];
      while (m > 1)
        {
          *out++ = r;
          r >>= 8;
          m = (m + 255) >> 8;
        }
    }
  if (len > 1)
    {
      uint16_t R2[(len + 1) / 2];
      uint16_t M2[(len + 1) / 2];
      long long i;
      for (i = 0; i < len - 1; i += 2)
        {
          uint32_t m0 = M[i];
          uint32_t r = R[i] + R[i + 1] * m0;
          uint32_t m = M[i + 1] * m0;
          while (m >= 16384)
            {
              *out++ = r;
              r >>= 8;
              m = (m + 255) >> 8;
            }
          R2[i / 2] = r;
          M2[i / 2] = m;
        }
      if (i < len)
        {
          R2[i / 2] = R[i];
          M2[i / 2] = M[i];
        }
      Encode (out, R2, M2, (len + 1) / 2);
    }
}

/* from supercop-20201130/crypto_kem/sntrup761/ref/kem.c */

/* ----- masks */
```

```
/* return -1 if x!=0; else return 0 */
static int
int16_t_nonzero_mask (int16_t x)
{
  uint16_t u = x;              /* 0, else 1...65535 */
  uint32_t v = u;              /* 0, else 1...65535 */
  v = -v;                      /* 0, else 2^32-65535...2^32-1 */
  v >>= 31;                    /* 0, else 1 */
  return -v;                   /* 0, else -1 */
}

/* return -1 if x<0; otherwise return 0 */
static int
int16_t_negative_mask (int16_t x)
{
  uint16_t u = x;
  u >>= 15;
  return -(int) u;
  /* alternative with gcc -fwrapv: */
  /* x>>15 compiles to CPU's arithmetic right shift */
}

/* ----- arithmetic mod 3 */

typedef int8_t small;

/* F3 is always represented as -1,0,1 */
/* so ZZ_fromF3 is a no-op */

/* x must not be close to top int16_t */
static small
F3_freeze (int16_t x)
{
  return int32_mod_uint14 (x + 1, 3) - 1;
}

/* ----- arithmetic mod q */

#define q12 ((q-1)/2)
typedef int16_t Fq;
/* always represented as -q12...q12 */
/* so ZZ_fromFq is a no-op */

/* x must not be close to top int32 */
static Fq
Fq_freeze (int32_t x)
{
  return int32_mod_uint14 (x + q12, q) - q12;
}
```

```
static Fq
Fq_recip (Fq a1)
{
  int i = 1;
  Fq ai = a1;

  while (i < q - 2)
    {
      ai = Fq_freeze (a1 * (int32_t) ai);
      i += 1;
    }
  return ai;
}

/* ----- small polynomials */

/* 0 if Weightw_is(r), else -1 */
static int
Weightw_mask (small * r)
{
  int weight = 0;
  int i;

  for (i = 0; i < p; ++i)
    weight += r[i] & 1;
  return int16_t_nonzero_mask (weight - w);
}

/* R3_fromR(R_fromRq(r)) */
static void
R3_fromRq (small * out, const Fq * r)
{
  int i;
  for (i = 0; i < p; ++i)
    out[i] = F3_freeze (r[i]);
}

/* h = f*g in the ring R3 */
static void
R3_mult (small * h, const small * f, const small * g)
{
  small fg[p + p - 1];
  small result;
  int i, j;

  for (i = 0; i < p; ++i)
    {
      result = 0;
      for (j = 0; j <= i; ++j)
```

```
        result = F3_freeze (result + f[j] * g[i - j]);
      fg[i] = result;
    }
  for (i = p; i < p + p - 1; ++i)
    {
      result = 0;
      for (j = i - p + 1; j < p; ++j)
        result = F3_freeze (result + f[j] * g[i - j]);
      fg[i] = result;
    }

  for (i = p + p - 2; i >= p; --i)
    {
      fg[i - p] = F3_freeze (fg[i - p] + fg[i]);
      fg[i - p + 1] = F3_freeze (fg[i - p + 1] + fg[i]);
    }

  for (i = 0; i < p; ++i)
    h[i] = fg[i];
}

/* returns 0 if recip succeeded; else -1 */
static int
R3_recip (small * out, const small * in)
{
  small f[p + 1], g[p + 1], v[p + 1], r[p + 1];
  int i, loop, delta;
  int sign, swap, t;

  for (i = 0; i < p + 1; ++i)
    v[i] = 0;
  for (i = 0; i < p + 1; ++i)
    r[i] = 0;
  r[0] = 1;
  for (i = 0; i < p; ++i)
    f[i] = 0;
  f[0] = 1;
  f[p - 1] = f[p] = -1;
  for (i = 0; i < p; ++i)
    g[p - 1 - i] = in[i];
  g[p] = 0;

  delta = 1;

  for (loop = 0; loop < 2 * p - 1; ++loop)
    {
      for (i = p; i > 0; --i)
        v[i] = v[i - 1];
      v[0] = 0;
```

```
      sign = -g[0] * f[0];
      swap = int16_t_negative_mask (-delta) & int16_t_nonzero_mask (g[0]
      delta ^= swap & (delta ^ -delta);
      delta += 1;

      for (i = 0; i < p + 1; ++i)
        {
          t = swap & (f[i] ^ g[i]);
          f[i] ^= t;
          g[i] ^= t;
          t = swap & (v[i] ^ r[i]);
          v[i] ^= t;
          r[i] ^= t;
        }

      for (i = 0; i < p + 1; ++i)
        g[i] = F3_freeze (g[i] + sign * f[i]);
      for (i = 0; i < p + 1; ++i)
        r[i] = F3_freeze (r[i] + sign * v[i]);

      for (i = 0; i < p; ++i)
        g[i] = g[i + 1];
      g[p] = 0;
    }

  sign = f[0];
  for (i = 0; i < p; ++i)
    out[i] = sign * v[p - 1 - i];

  return int16_t_nonzero_mask (delta);
}

/* ----- polynomials mod q */

/* h = f*g in the ring Rq */
static void
Rq_mult_small (Fq * h, const Fq * f, const small * g)
{
  Fq fg[p + p - 1];
  Fq result;
  int i, j;

  for (i = 0; i < p; ++i)
    {
      result = 0;
      for (j = 0; j <= i; ++j)
        result = Fq_freeze (result + f[j] * (int32_t) g[i - j]);
      fg[i] = result;
    }
```

```
  for (i = p; i < p + p - 1; ++i)
    {
      result = 0;
      for (j = i - p + 1; j < p; ++j)
        result = Fq_freeze (result + f[j] * (int32_t) g[i - j]);
      fg[i] = result;
    }

  for (i = p + p - 2; i >= p; --i)
    {
      fg[i - p] = Fq_freeze (fg[i - p] + fg[i]);
      fg[i - p + 1] = Fq_freeze (fg[i - p + 1] + fg[i]);
    }

  for (i = 0; i < p; ++i)
    h[i] = fg[i];
}

/* h = 3f in Rq */
static void
Rq_mult3 (Fq * h, const Fq * f)
{
  int i;

  for (i = 0; i < p; ++i)
    h[i] = Fq_freeze (3 * f[i]);
}

/* out = 1/(3*in) in Rq */
/* returns 0 if recip succeeded; else -1 */
static int
Rq_recip3 (Fq * out, const small * in)
{
  Fq f[p + 1], g[p + 1], v[p + 1], r[p + 1];
  int i, loop, delta;
  int swap, t;
  int32_t f0, g0;
  Fq scale;

  for (i = 0; i < p + 1; ++i)
    v[i] = 0;
  for (i = 0; i < p + 1; ++i)
    r[i] = 0;
  r[0] = Fq_recip (3);
  for (i = 0; i < p; ++i)
    f[i] = 0;
  f[0] = 1;
  f[p - 1] = f[p] = -1;
  for (i = 0; i < p; ++i)
```

```
    g[p - 1 - i] = in[i];
  g[p] = 0;

  delta = 1;

  for (loop = 0; loop < 2 * p - 1; ++loop)
    {
      for (i = p; i > 0; --i)
        v[i] = v[i - 1];
      v[0] = 0;

      swap = int16_t_negative_mask (-delta) & int16_t_nonzero_mask (g[0]
      delta ^= swap & (delta ^ -delta);
      delta += 1;

      for (i = 0; i < p + 1; ++i)
        {
          t = swap & (f[i] ^ g[i]);
          f[i] ^= t;
          g[i] ^= t;
          t = swap & (v[i] ^ r[i]);
          v[i] ^= t;
          r[i] ^= t;
        }

      f0 = f[0];
      g0 = g[0];
      for (i = 0; i < p + 1; ++i)
        g[i] = Fq_freeze (f0 * g[i] - g0 * f[i]);
      for (i = 0; i < p + 1; ++i)
        r[i] = Fq_freeze (f0 * r[i] - g0 * v[i]);

      for (i = 0; i < p; ++i)
        g[i] = g[i + 1];
      g[p] = 0;
    }

  scale = Fq_recip (f[0]);
  for (i = 0; i < p; ++i)
    out[i] = Fq_freeze (scale * (int32_t) v[p - 1 - i]);

  return int16_t_nonzero_mask (delta);
}

/* ----- rounded polynomials mod q */

static void
Round (Fq * out, const Fq * a)
{
  int i;
```

```
  for (i = 0; i < p; ++i)
    out[i] = a[i] - F3_freeze (a[i]);
}

/* ----- sorting to generate short polynomial */

static void
Short_fromlist (small * out, const uint32_t * in)
{
  uint32_t L[p];
  int i;

  for (i = 0; i < w; ++i)
    L[i] = in[i] & (uint32_t) - 2;
  for (i = w; i < p; ++i)
    L[i] = (in[i] & (uint32_t) - 3) | 1;
  crypto_sort_uint32 (L, p);
  for (i = 0; i < p; ++i)
    out[i] = (L[i] & 3) - 1;
}

/* ----- underlying hash function */

#define Hash_bytes 32

/* e.g., b = 0 means out = Hash0(in) */
static void
Hash_prefix (unsigned char *out, int b, const unsigned char *in, int inl
{
  unsigned char x[inlen + 1];
  unsigned char h[64];
  int i;

  x[0] = b;
  for (i = 0; i < inlen; ++i)
    x[i + 1] = in[i];
  crypto_hash_sha512 (h, x, inlen + 1);
  for (i = 0; i < 32; ++i)
    out[i] = h[i];
}

/* ----- higher-level randomness */

static uint32_t
urandom32 (void *random_ctx, sntrup761_random_func * random)
{
  unsigned char c[4];
  uint32_t out[4];

  random (random_ctx, 4, c);
```

```
  out[0] = (uint32_t) c[0];
  out[1] = ((uint32_t) c[1]) << 8;
  out[2] = ((uint32_t) c[2]) << 16;
  out[3] = ((uint32_t) c[3]) << 24;
  return out[0] + out[1] + out[2] + out[3];
}

static void
Short_random (small * out, void *random_ctx, sntrup761_random_func * ran
{
  uint32_t L[p];
  int i;

  for (i = 0; i < p; ++i)
    L[i] = urandom32 (random_ctx, random);
  Short_fromlist (out, L);
}

static void
Small_random (small * out, void *random_ctx, sntrup761_random_func * ran
{
  int i;

  for (i = 0; i < p; ++i)
    out[i] = (((urandom32 (random_ctx, random) & 0x3fffffff) * 3) >> 30)
}

/* ----- Streamlined NTRU Prime Core */

/* h,(f,ginv) = KeyGen() */
static void
KeyGen (Fq * h, small * f, small * ginv, void *random_ctx,
        sntrup761_random_func * random)
{
  small g[p];
  Fq finv[p];

  for (;;)
    {
      Small_random (g, random_ctx, random);
      if (R3_recip (ginv, g) == 0)
        break;
    }
  Short_random (f, random_ctx, random);
  Rq_recip3 (finv, f);          /* always works */
  Rq_mult_small (h, finv, g);
}

/* c = Encrypt(r,h) */
static void
```

```
Encrypt (Fq * c, const small * r, const Fq * h)
{
  Fq hr[p];

  Rq_mult_small (hr, h, r);
  Round (c, hr);
}

/* r = Decrypt(c,(f,ginv)) */
static void
Decrypt (small * r, const Fq * c, const small * f, const small * ginv)
{
  Fq cf[p];
  Fq cf3[p];
  small e[p];
  small ev[p];
  int mask;
  int i;

  Rq_mult_small (cf, c, f);
  Rq_mult3 (cf3, cf);
  R3_fromRq (e, cf3);
  R3_mult (ev, e, ginv);

  mask = Weightw_mask (ev);     /* 0 if weight w, else -1 */
  for (i = 0; i < w; ++i)
    r[i] = ((ev[i] ^ 1) & ~mask) ^ 1;
  for (i = w; i < p; ++i)
    r[i] = ev[i] & ~mask;
}

/* ----- encoding small polynomials (including short polynomials) */

#define Small_bytes ((p+3)/4)

/* these are the only functions that rely on p mod 4 = 1 */

static void
Small_encode (unsigned char *s, const small * f)
{
  small x;
  int i;

  for (i = 0; i < p / 4; ++i)
    {
      x = *f++ + 1;
      x += (*f++ + 1) << 2;
      x += (*f++ + 1) << 4;
      x += (*f++ + 1) << 6;
      *s++ = x;
```

```
    }
  x = *f++ + 1;
  *s++ = x;
}

static void
Small_decode (small * f, const unsigned char *s)
{
  unsigned char x;
  int i;

  for (i = 0; i < p / 4; ++i)
    {
      x = *s++;
      *f++ = ((small) (x & 3)) - 1;
      x >>= 2;
      *f++ = ((small) (x & 3)) - 1;
      x >>= 2;
      *f++ = ((small) (x & 3)) - 1;
      x >>= 2;
      *f++ = ((small) (x & 3)) - 1;
    }
  x = *s++;
  *f++ = ((small) (x & 3)) - 1;
}

/* ----- encoding general polynomials */

static void
Rq_encode (unsigned char *s, const Fq * r)
{
  uint16_t R[p], M[p];
  int i;

  for (i = 0; i < p; ++i)
    R[i] = r[i] + q12;
  for (i = 0; i < p; ++i)
    M[i] = q;
  Encode (s, R, M, p);
}

static void
Rq_decode (Fq * r, const unsigned char *s)
{
  uint16_t R[p], M[p];
  int i;

  for (i = 0; i < p; ++i)
    M[i] = q;
```

```
  Decode (R, s, M, p);
  for (i = 0; i < p; ++i)
    r[i] = ((Fq) R[i]) - q12;
}

/* ----- encoding rounded polynomials */

static void
Rounded_encode (unsigned char *s, const Fq * r)
{
  uint16_t R[p], M[p];
  int i;

  for (i = 0; i < p; ++i)
    R[i] = ((r[i] + q12) * 10923) >> 15;
  for (i = 0; i < p; ++i)
    M[i] = (q + 2) / 3;
  Encode (s, R, M, p);
}

static void
Rounded_decode (Fq * r, const unsigned char *s)
{
  uint16_t R[p], M[p];
  int i;

  for (i = 0; i < p; ++i)
    M[i] = (q + 2) / 3;
  Decode (R, s, M, p);
  for (i = 0; i < p; ++i)
    r[i] = R[i] * 3 - q12;
}

/* ----- Streamlined NTRU Prime Core plus encoding */

typedef small Inputs[p];        /* passed by reference */
#define Inputs_random Short_random
#define Inputs_encode Small_encode
#define Inputs_bytes Small_bytes

#define Ciphertexts_bytes Rounded_bytes
#define SecretKeys_bytes (2*Small_bytes)
#define PublicKeys_bytes Rq_bytes

/* pk,sk = ZKeyGen() */
static void
ZKeyGen (unsigned char *pk, unsigned char *sk, void *random_ctx,
         sntrup761_random_func * random)
{
  Fq h[p];
```

```
  small f[p], v[p];

  KeyGen (h, f, v, random_ctx, random);
  Rq_encode (pk, h);
  Small_encode (sk, f);
  sk += Small_bytes;
  Small_encode (sk, v);
}

/* C = ZEncrypt(r,pk) */
static void
ZEncrypt (unsigned char *C, const Inputs r, const unsigned char *pk)
{
  Fq h[p];
  Fq c[p];
  Rq_decode (h, pk);
  Encrypt (c, r, h);
  Rounded_encode (C, c);
}

/* r = ZDecrypt(C,sk) */
static void
ZDecrypt (Inputs r, const unsigned char *C, const unsigned char *sk)
{
  small f[p], v[p];
  Fq c[p];

  Small_decode (f, sk);
  sk += Small_bytes;
  Small_decode (v, sk);
  Rounded_decode (c, C);
  Decrypt (r, c, f, v);
}

/* ----- confirmation hash */

#define Confirm_bytes 32

/* h = HashConfirm(r,pk,cache); cache is Hash4(pk) */
static void
HashConfirm (unsigned char *h, const unsigned char *r,
             /* const unsigned char *pk, */ const unsigned char *cache)
{
  unsigned char x[Hash_bytes * 2];
  int i;

  Hash_prefix (x, 3, r, Inputs_bytes);
  for (i = 0; i < Hash_bytes; ++i)
    x[Hash_bytes + i] = cache[i];
  Hash_prefix (h, 2, x, sizeof x);
```

```
}

/* ----- session-key hash */

/* k = HashSession(b,y,z) */
static void
HashSession (unsigned char *k, int b, const unsigned char *y,
             const unsigned char *z)
{
  unsigned char x[Hash_bytes + Ciphertexts_bytes + Confirm_bytes];
  int i;

  Hash_prefix (x, 3, y, Inputs_bytes);
  for (i = 0; i < Ciphertexts_bytes + Confirm_bytes; ++i)
    x[Hash_bytes + i] = z[i];
  Hash_prefix (k, b, x, sizeof x);
}

/* ----- Streamlined NTRU Prime */

/* pk,sk = KEM_KeyGen() */
void
sntrup761_keypair (unsigned char *pk, unsigned char *sk, void *random_ct
                   sntrup761_random_func * random)
{
  int i;

  ZKeyGen (pk, sk, random_ctx, random);
  sk += SecretKeys_bytes;
  for (i = 0; i < PublicKeys_bytes; ++i)
    *sk++ = pk[i];
  random (random_ctx, Inputs_bytes, sk);
  sk += Inputs_bytes;
  Hash_prefix (sk, 4, pk, PublicKeys_bytes);
}

/* c,r_enc = Hide(r,pk,cache); cache is Hash4(pk) */
static void
Hide (unsigned char *c, unsigned char *r_enc, const Inputs r,
      const unsigned char *pk, const unsigned char *cache)
{
  Inputs_encode (r_enc, r);
  ZEncrypt (c, r, pk);
  c += Ciphertexts_bytes;
  HashConfirm (c, r_enc, cache);
}

/* c,k = Encap(pk) */
void
sntrup761_enc (unsigned char *c, unsigned char *k, const unsigned char *
```

```
               void *random_ctx, sntrup761_random_func * random)
{
  Inputs r;
  unsigned char r_enc[Inputs_bytes];
  unsigned char cache[Hash_bytes];

  Hash_prefix (cache, 4, pk, PublicKeys_bytes);
  Inputs_random (r, random_ctx, random);
  Hide (c, r_enc, r, pk, cache);
  HashSession (k, 1, r_enc, c);
}

/* 0 if matching ciphertext+confirm, else -1 */
static int
Ciphertexts_diff_mask (const unsigned char *c, const unsigned char *c2)
{
  uint16_t differentbits = 0;
  int len = Ciphertexts_bytes + Confirm_bytes;

  while (len-- > 0)
    differentbits |= (*c++) ^ (*c2++);
  return (1 & ((differentbits - 1) >> 8)) - 1;
}

/* k = Decap(c,sk) */
void
sntrup761_dec (unsigned char *k, const unsigned char *c, const unsigned
{
  const unsigned char *pk = sk + SecretKeys_bytes;
  const unsigned char *rho = pk + PublicKeys_bytes;
  const unsigned char *cache = rho + Inputs_bytes;
  Inputs r;
  unsigned char r_enc[Inputs_bytes];
  unsigned char cnew[Ciphertexts_bytes + Confirm_bytes];
  int mask;
  int i;

  ZDecrypt (r, c, sk);
  Hide (cnew, r_enc, r, pk, cache);
  mask = Ciphertexts_diff_mask (c, cnew);
  for (i = 0; i < Inputs_bytes; ++i)
    r_enc[i] ^= mask & (r_enc[i] ^ rho[i]);
  HashSession (k, 1 + mask, r_enc, c);
}
```

Figure 6

6.3. sntrup761-test.c

This is a test driver to compare test vectors of sntrup761. It can be used to verify correctness of the sntrup761.c implementation.

```
#include <stdio.h>
#include <stdlib.h>

#include "sntrup761.h"

#include "drbg-ctr.h"

static void
print_hex (size_t n, const uint8_t *d)
{
  size_t i;
  for (i = 0; i < n; i++)
    printf ("%02x", d[i]);
  printf ("\n");
}

static void
test_sntrup (struct drbg_ctr_aes256_ctx *rngctx,
             sntrup761_random_func * rngfun,
             const uint8_t * xpk, const uint8_t * xsk,
             const uint8_t * xct, const uint8_t * xk)
{
  uint8_t pk[SNTRUP761_PUBLICKEY_SIZE];
  uint8_t sk[SNTRUP761_SECRETKEY_SIZE];
  uint8_t ct[SNTRUP761_CIPHERTEXT_SIZE];
  uint8_t k1[SNTRUP761_SIZE];
  uint8_t k2[SNTRUP761_SIZE];

  sntrup761_keypair (pk, sk, rngctx, rngfun);

  if (memcmp (pk, xpk, SNTRUP761_PUBLICKEY_SIZE)
      || memcmp (sk, xsk, SNTRUP761_SECRETKEY_SIZE))
    {
      printf ("\nnstrup761_keypair:\npk = ");
      print_hex (sizeof pk, pk);
      printf ("\nsk = ");
      print_hex (sizeof sk, sk);
      abort ();
    }

  sntrup761_enc (ct, k1, pk, rngctx, rngfun);

  if (memcmp (ct, xct, SNTRUP761_CIPHERTEXT_SIZE)
      || memcmp (k1, xk, SNTRUP761_SIZE))
    {
      printf ("\nnstrup761_enc:\nct = ");
      print_hex (sizeof ct, ct);
      printf ("\nk1 = ");
      print_hex (sizeof k1, k1);
      abort ();
```

```
    }

  sntrup761_dec (k2, ct, sk);

  if (memcmp (k2, xk, SNTRUP761_SIZE))
    {
      printf ("\nnstrup761_dec:\nk2 = ");
      print_hex (sizeof k2, k2);
      abort ();
    }

  if (memcmp (k1, k2, SNTRUP761_SIZE))
    {
      printf ("\nsntrup761 k1 != k2\n");
      abort ();
    }
}

static char *H(const char* in)
{
  static const unsigned char TBL[] = {
    0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  58,  59,  60,  61,
    62,  63,  64,  10,  11,  12,  13,  14,  15,  71,  72,  73,  74,  75,
    76,  77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,
    90,  91,  92,  93,  94,  95,  96,  10,  11,  12,  13,  14,  15
  };
  static const unsigned char *LOOKUP = TBL - 48;
  const char* end = in + strlen (in);
  char *out, *tmp;

  out = tmp = malloc (strlen (in));
  if (!out)
    abort();

  while(in < end)
    *(tmp++) = LOOKUP[*(in++)] << 4 | LOOKUP[*(in++)];

  return out;
}

void
main (void)
{
  struct drbg_ctr_aes256_ctx rng;
  uint8_t seed_material[DRBG_CTR_AES256_SEED_SIZE];
  uint8_t tmp[DRBG_CTR_AES256_SEED_SIZE];
  size_t i;

  for (i = 0; i < DRBG_CTR_AES256_SEED_SIZE; i++)
    seed_material[i] = i;
```

```
drbg_ctr_aes256_init (&rng, seed_material);

drbg_ctr_aes256_random (&rng, DRBG_CTR_AES256_SEED_SIZE, tmp);
if (memcmp (tmp,
            H ("061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479"
               "D09D86DC9ABCFDE7056A8C266F9EF97ED08541DBD2E1FFA1"),
            DRBG_CTR_AES256_SEED_SIZE))
  {
    printf ("drbg_ctr_aes256 = ");
    print_hex (DRBG_CTR_AES256_SEED_SIZE, tmp);
    abort ();
  }

drbg_ctr_aes256_init (&rng,
                      H ("061550234D158C5EC95595FE04EF7A25767F2E24CC2B
                         "D09D86DC9ABCFDE7056A8C266F9EF97ED08541DBD2E1
test_sntrup (&rng, (sntrup761_random_func *) drbg_ctr_aes256_random,
             H ("36C969CF1008A6AA9551A784941C65A9BF68C2DC33FA36B5"
                "D266B25171B346679F2D22BF3123A79C790D6DEC68E1BC44"
                "420A6824F5357C78E3C336FEE0551E620DCB975F563682A3"
                "12A3353B521C727F57CABED0C3228F09317CAE8B58158EBF"
                "5B26BDC6E6365AA601ACAD2ABD37F5830D0BBFE355705C0A"
                "62B76A5C910AD04E55E5DAD749C7393D2E2E8AB643E62E47"
                "57AD2201CAA33203F53B4A9D4757E7274D72BDB036A31D7D"
                "E11E5D1C66CF3059F33B6A2972C1E1D9C9FB2AEBC78B2C05"
                "5D48D79C3A7C996C08B7DEF0791CDE895053885D8DA1D254"
                "EE19C090AF34F4720B0B77139108D8498982FEAB0B54934C"
                "EE3DCD24B049F981A84C928028A64A26CDF87052313C3E50"
                "B2E1F539394502433C0962996C3599189B15174281B6595B"
                "567B8C4CD80902860E613AE1906A55607CBF0E11AD6C0C0D"
                "F38C006A5F535FA6E6FB49D10B78B9CB6473BF0630518DB6"
                "FDECFD70DED201C7E35FFB3BB78D8AEC181F6DE960C316FE"
                "354B7CC69E8048201965AAFEF4EA3F808737FF45255A1779"
                "DE57A4B68DB587263D7B7F6CB07D8B01224DD291237EFD9C"
                "01676E30154A2A7D60174536580FE64EFEDDA5FB42C13ED8"
                "C5768A3CBCAE7A2343B3128CD5C1C663A07C7DC0E1E2642C"
                "0D9A02F349C964154A6C4308D8ECF30F47E9A81EED2A32A2"
                "BB44DFC36A28A66AE77BB139FA416B6327EEFE33632469CC"
                "C21229587573C4F7752CE1CC79CA0C6BDA08CD79E25720D2"
                "D9092EA2AB13F31E9A3AF1C69FC6379D8E2AD2B87B514C81"
                "7A087338B7B0736B8954DE9223225B40079C92601248C14B"
                "2104901E74F849D9EEE9F5636C1DAD6031AB477C573197E7"
                "EB6CE535D9F0F69183DD9DF5521595E5C9E98D846CE08AA6"
                "55B70CC0D8041401929690298F645D9112DBB03B189CDB1F"
                "CF4D512F6874B409BF55EE1CC284A05B698B8818F043C259"
                "1C9F4ABA29CF4259915D6A0BE71B6B93963C618CBB567838"
                "E5EADE6500DE9AD250083A01EB3EB44C00EECB2B0F874CDA"
                "165FAA6524CA13D435C938DB9469292FE97283C69222107E"
```

```
    "A2F9EFCA1B6D41DCA5B149B2A8CC2244A1BC54261CC11742"
    "ACD27B7F2352C33FB83FE143386479D78D6D5C3E51684B60"
    "A56714182449C94327139B0289B9BDE0AE3FEF319FE2C605"
    "AB5507C894D2C2761A3BD3EA30F4BA928F9F23303061617E"
    "2F042DCF229AFF2C9345A6B3CDCF90D3E3BCC3A1110C8561"
    "6BD585C31CDE3E69ADC12A18BFA5797DF0543435A7C874A8"
    "ACF3786FCCBA4A6CEAA65E5666230AA7206AE78EB1B98CB5"
    "236508E6357E18CCCFAEC5693532BE4EE38022C48FE94F62"
    "B0293A088BF4D737F48748F23BCB338B58B4D666AD3C64E9"
    "ECD07265F971B07AF716D4A5B719C3F5FE35744734CBA543"
    "0381661E372B6F6510D61B11E4697A1A961589949DBA53B5"
    "BC5BDB76FA09324387D799536506182EFF7078034B34E1DD"
    "D2612B0A40E7F1A294FBA869DA2E46C4C36BB08C7E9BEF09"
    "A94459E8FC2D3CC579D15284DF1F19EF77E03041511ACC39"
    "BF9CA8AC8DE6A0E5EEA0BBD4289B6DAE38E9E82EB50A0397"
    "B3EE6C52FBDC7DEEA2C376825E89016C859B09488548BF76"
    "CA62D696DF94A61E8F0E13C69EE816CE5AA2768987B0A782"
    "D74C673EC005"),
H ("55565515155545556569159955A255545545616145059455"
    "655691505589561554885295545655009854499555 5AA546"
    "5400A5555558956510A6515596559665996A449154154144"
    "655655555511056241569549446269405555551656659555"
    "55A5159155595659156565154240408919545456955 8590"
    "64258656285054569505A6A95694944666141558011556 50"
    "55485955259049A5545525455504526A195455554185525A"
    "899625525556582484644514556589656258515459150185"
    "08154A51A0448959112A286212608A52911A949119A28956"
    "64165900A861541A200A52104029544600425A5A1198101A"
    "1A50A094A551425A484959A1506486A618555A8990561502"
    "9622A2A46696A2A6595A466826225250504 89A8040562808"
    "168156854185026540A1059A94AAA291811948A259816991"
    "110951802409AA5A2A062900924514466556084040968 98A"
    "A562205622851508919140661 10A0905A841044865042A26"
    "88AA12441AAA46064669155555A2109659299922820036C9"
    "69CF1008A6AA9551A784941C65A9BF68C2DC33FA36B5D266"
    "B25171B346679F2D22BF3123A79C790D6DEC68E1BC44420A"
    "6824F5357C78E3C336FEE0551E620DCB975F563682A312A3"
    "353B521C727F57CABED0C3228F09317CAE8B58158EBF5B26"
    "BDC6E6365AA601ACAD2ABD37F5830D0BBFE355705C0A62B7"
    "6A5C910AD04E55E5DAD749C7393D2E2E8AB643E62E4757AD"
    "2201CAA33203F53B4A9D4757E7274D72BDB036A31D7DE11E"
    "5D1C66CF3059F33B6A2972C1E1D9C9FB2AEBC78B2C055D48"
    "D79C3A7C996C08B7DEF0791CDE895053885D8DA1D254EE19"
    "C090AF34F4720B0B77139108D8498982FEAB0B54934CEE3D"
    "CD24B049F981A84C928028A64A26CDF87052313C3E50B2E1"
    "F539394502433C0962996C3599189B15174281B6595B567B"
    "8C4CD80902860E613AE1906A55607CBF0E11AD6C0C0DF38C"
    "006A5F535FA6E6FB49D10B78B9CB6473BF0630518DB6FDEC"
    "FD70DED201C7E35FFB3BB78D8AEC181F6DE960C316FE354B"
```

```
    "7CC69E8048201965AAFEF4EA3F808737FF45255A1779DE57"
    "A4B68DB587263D7B7F6CB07D8B01224DD291237EFD9C0167"
    "6E30154A2A7D60174536580FE64EFEDDA5FB42C13ED8C576"
    "8A3CBCAE7A2343B3128CD5C1C663A07C7DC0E1E2642C0D9A"
    "02F349C964154A6C4308D8ECF30F47E9A81EED2A32A2BB44"
    "DFC36A28A66AE77BB139FA416B6327EEFE33632469CCC212"
    "29587573C4F7752CE1CC79CA0C6BDA08CD79E25720D2D909"
    "2EA2AB13F31E9A3AF1C69FC6379D8E2AD2B87B514C817A08"
    "7338B7B0736B8954DE9223225B40079C92601248C14B2104"
    "901E74F849D9EEE9F5636C1DAD6031AB477C573197E7EB6C"
    "E535D9F0F69183DD9DF5521595E5C9E98D846CE08AA655B7"
    "0CC0D8041401929690298F645D9112DBB03B189CDB1FCF4D"
    "512F6874B409BF55EE1CC284A05B698B8818F043C2591C9F"
    "4ABA29CF4259915D6A0BE71B6B93963C618CBB567838E5EA"
    "DE6500DE9AD250083A01EB3EB44C00EECB2B0F874CDA165F"
    "AA6524CA13D435C938DB9469292FE97283C69222107EA2F9"
    "EFCA1B6D41DCA5B149B2A8CC2244A1BC54261CC11742ACD2"
    "7B7F2352C33FB83FE143386479D78D6D5C3E51684B60A567"
    "14182449C94327139B0289B9BDE0AE3FEF319FE2C605AB55"
    "07C894D2C2761A3BD3EA30F4BA928F9F23303061617E2F04"
    "2DCF229AFF2C9345A6B3CDCF90D3E3BCC3A1110C85616BD5"
    "85C31CDE3E69ADC12A18BFA5797DF0543435A7C874A8ACF3"
    "786FCCBA4A6CEAA65E5666230AA7206AE78EB1B98CB52365"
    "08E6357E18CCCFAEC5693532BE4EE38022C48FE94F62B029"
    "3A088BF4D737F48748F23BCB338B58B4D666AD3C64E9ECD0"
    "7265F971B07AF716D4A5B719C3F5FE35744734CBA5430381"
    "661E372B6F6510D61B11E4697A1A961589949DBA53B5BC5B"
    "DB76FA09324387D799536506182EFF7078034B34E1DDD261"
    "2B0A40E7F1A294FBA869DA2E46C4C36BB08C7E9BEF09A944"
    "59E8FC2D3CC579D15284DF1F19EF77E03041511ACC39BF9C"
    "A8AC8DE6A0E5EEA0BBD4289B6DAE38E9E82EB50A0397B3EE"
    "6C52FBDC7DEEA2C376825E89016C859B09488548BF76CA62"
    "D696DF94A61E8F0E13C69EE816CE5AA2768987B0A782D74C"
    "673EC0059FA532AE97F8BDF90F90130926654FB8B469D772"
    "049D72A8375CD8459D06CC1B90633EB3A899685D21491B50"
    "62A9FC73FB6E878D7A73198EFA0B569D9CE665CE126FFDC9"
    "862EB00D11F457A7995555F7C92011C24E1CEC45C270FB5F"
    "121F08177F97FC3F631C0EF86A92E99D557FB69A0F6FCD8B"
    "1C0EF94AA7429B3A8A11614D847202D3D04A98313FC0D63A"
    "BF9FD75CA321C879458BF8837B7E74CCF5179C7714FA9800"
    "D1821EE3F9639D28136B7910872631F85AE7B6DD289E0103"
    "4217210859D4E53C65487CE38AFF621DA76BA1C4E2E77ED5"
    "380B47B8D983CCB5BB793E"),
H ("84F327D38929039EF84366AB796A96E6CC268454D5AD8D05"
    "410D3E969B6D228B6CED2CBAD7BAB3B4B7EE453747D3331D"
    "43B21CD2ECC3BF557D696E81773EE69F687FE1F61E63742A"
    "F913A35418575467118409D4FFDBBAC5CA062DFF3F04D761"
    "54D17EC34212DEBFCA967A91E461F73786D949D6A9765140"
    "B9DC7849639A487058F45DDA919576090D35E783804BA7B8"
```

```
                "B33B5C6A8C1EBB7C39F042696CF9F2B624D26EA5D6D10148"
                "9E242CC35126B573928B6B8BF9945269AD22E7DA70CCB7DC"
                "374F75641978D5F6D5F540A1169EB098570A7CFD65C9257C"
                "4C196F42C6DA4F4466D8F11A17C6A75A9BDD45B5AC77A836"
                "7B005ED22141A81B995A19E4DC9E6743B4B45E0329E7A00D"
                "3FF90C7A917CDC9CA75A34FFFDD9A321751B5C8B2AB9E1EA"
                "037CF45C04B412A97F52E7C6D66508456D9117961257B4F5"
                "0BB9F500DE99B6F061C377D44F495D711AB2ED3E88CCF14D"
                "F10F8C60BD00E29CD83AD160FAEF2984FAC7E897CBB8CDA1"
                "44D50E4C1AD20238A2617598DA6EFE786AA82E0931B0B5D1"
                "A7FA024B856353AFAEDB0A1150A42DA182353CE5A00403BB"
                "6B7A13EFFC14B0FD97E1117302ED8A66FAAED88B43B34CFC"
                "73CB33E49A4C34D0BFDEE2FEADC2B4C8BB1EC41215586FC6"
                "6550D496373305B99CEE254EAC71D3CDAC689A777062ACAE"
                "65B031B5A1A973835BE6A4BB061F178531C45DE3B33E7763"
                "B483621821E037FE5EF48FD971D8D2BEFDA63250E9B1181D"
                "8B8086BED9CDA8CDE6644CDE8EBF7C1F1038AFE798A2A4C4"
                "92B603B7F4F8805D0FDF19D8274888CEBEC3043CC9ACF024"
                "BD30D8B0D28D3119859301103865987 75CF84E0848B11E09"
                "0905A65F56142FC17D2573536FC51E7D5B2E0FF7B0947115"
                "5008561156FBAEC215031ED0D844EB14D6C609173355FFF8"
                "AE22C6E5BCCD19B2832E5FBE625C2C0E516E1369D1FCF49A"
                "5FDBAF8CF572F04317DB8ED70EE1605B96D077B4F054C72B"
                "139F5BCAA5BB627E39BE5BCC97AF922C5E70835BD918281B"
                "95E557FC86A43C75980A0D9F0F9162F230014CD867DD07DB"
                "23260934546C04499D6B8627B8D5DEC73FA45327F8498352"
                "9BAF3AA10E885724FE76055099D136CCAACC159B4CE38784"
                "404CC9B04723910F990AF77C64D42A29D784FC897955758F"
                "1D2A12BB7C6F6A8E5B94EFA740F662CF0DA2D099C61EE1CC"
                "6B661A64F0887B805AF53C7B64C4456DAF228582C1B2212A"
                "8F718E56DF6B72D7EFA0C5D26582CE96AEFCE13CB2830C3F"
                "B32E5115AACC20D3A335F7A4C12E0CD1DE532D8D9D42D69B"
                "C2D0BE0AC7DF3B8F416B9A53DB261C47272725A8A8829867"
                "8351B42103219092076B6ECE744D993388A8F2C475079CB8"
                "BC4DC492A4797147C01EA3508225826F7A3D32BF502B0F45"
                "52F36FC9F6FB91E82DB1949338B45436EC0FF63B53F900F2"
                "7F8147D420F93D33C83794980E94143FD361F944DDB56D31"
                "14EE974C96A155"),
             H ("344CA5E25F6DA5EA95E4A695B1C5446ECA9859334532E4A9"
                "537669F012C743A2"));

drbg_ctr_aes256_init (&rng,
                      H ("D81C4D8D734FCBFBEADE3D3F8A039FAA2A2C9957E835
                         "B22E75BF57BB556AC81ADDE6AEEB4A5A875C3BFCADFA
test_sntrup (&rng, (sntrup761_random_func *) drbg_ctr_aes256_random,
             H ("D2530F125EE5F208B1976A66BCBC917161F6929E636BA8C7"
                "3470DE18065F6057528D718744E9248DFFF6BB55C188CEAC"
                "B9419863C3C456B46A21354834ADA6B2132C67747C9EE70D"
                "02560268EE650E56C84BCE6A6700C5E612999110E53A866A"
```

```
    "E6B4F778230367B8B886C4FEC089C6267F91C7F24D6CE53C"
    "754D9CCCDD25756D76CD211D5954F0E8A11343679C1F692C"
    "E5C0D0E42A02381144E1E0201B6F49F00628A86E09918488"
    "BC3E1E1071561700544C2CE50F4B7BAE4757CA7A0DD0A221"
    "260E0574D1F8F81AF072FCBA8061B5B8BB450FDCC6732D35"
    "CDBBBC4EF1798EA7E263BCF369059EED3A86E2C61C72CBFC"
    "D69521C3A1FD4868AF5638148043162B8B6E39F82B56D6C9"
    "C7724807F623C06CC08FF619F7961EC972B1BD2856F87D5E"
    "6940DC15B9575264A4AB0CA229C5B02C9AA5BB8DA2BD8EB0"
    "9A7A5E824C2D666A17C4845CA9530F3A5E45BE5A380B83C8"
    "1B3965180C706E2CC9ABF4DC8A1559CCF176EFD0FAAD1B8E"
    "184AAA08E24CA6F5D070E040D64CD65A1628E99F3BACAB5A"
    "7206B5C7D1F7D282651448E259A839E128774530C931C4E4"
    "E6F60AB9CDEB645AA24012C3A0983750C04914C59E34AF67"
    "B9128CE906AD162D70E582C42E21898EF8023A9D1BC3B5F5"
    "BDE6415CD168F1DE726A2ECB27C48A73356FBBEE8F405A6F"
    "8C34F8F5A864C109045618B03FBFDCBAFD8BC93465863407"
    "0D40E5C62A03F9B544C325BCC56D8C70D67DA39879356E98"
    "71B016A0CCF3FE1C6AB941B69417FF514B616C9CFE9EA066"
    "FBB4B081B25D584E40430B0950A7EBFBFA9B4E4EE003F297"
    "BA668F7139F26F218026DC9BFDBBBE6FD1CDE03CAC981429"
    "9126E8CBDAFFD54101E07C09B65B88C9743B85C464E44C72"
    "56F506F07BC7D3877E2578D589CD2001E5124C9E647FAC21"
    "34B34E74DC188478E538BBFB750B47600191C51B71F18460"
    "CECD5FBDABAA48D6DD6DBE7E26CEF9498220379F1FF25AFA"
    "CA60CEFFA408CEB677E4CAFB3A43B3E5FAA5B731EA608A94"
    "5EDF645136ADC77B07DB34E191D9E786ED7CFD97E2330548"
    "FFE021997E2E8B774E4A5F1666A91F05D471874E765DF42C"
    "7C3D9EF37CD946E0D69C9EA83CEC9B1AA41A25F1A3A458C9"
    "935CBC7D294B64D628DA7F1B0FA4F24E6375DF31AE82815F"
    "5DFF4ACD0DD9BD8A8740CB92633D1E000191B837021F143F"
    "64B08388A78B9A0F55FE7B824FA9D4E85709EAC46EAB5B24"
    "C46BBC2D1D8D39FEC8BA130EB68A7F55769606F07CB7B8CA"
    "0AD99739C68A365E415983B964F1F2D261145057B7A76D72"
    "300AD49D9FFFFFFE9C41BD48B82F8450274C6C25DCECE61D"
    "4C443FEE0D3C359FDD4E4409AD607AE7A707B83B8629134D"
    "ACCBD54C618EEF0B2F04E848F7B62C494DEC89F2830BACED"
    "B3B876670309B36F7D70BE0219F5D05A3C1E5BB58B1CF053"
    "FE92D2E3F934AA2047F963172E04B7457FED3956C08A705C"
    "9C441F1FEE07E05344C63817F5DA7F298D5323BF88FB490E"
    "1C628015ECD09C5D89978EB42A2DFD32E0820B005BDB60FF"
    "FB15EBE1A123C6107530FE48C72C925FB85465749EF1A46C"
    "6BA7F0F31D911D9E78608DAA64EC87CB3C82ABEC988FA365"
    "6734A038FB5A3F072B7E9E9F383227EA3D5ACE37A6A5536B"
    "9E3AA926E900"),
H ("64515999655565914856965695410550601945189559549"
    "84866951658555588521521601525454456 9A65216995559"
    "85415514151155925466465965555565555614616559508"
    "955596965965645699555A1596555A966566614655A409"
```

"5155955466959061565 9A5515551255565955519555699 15"
"A5614089A15501582951945A1555865955555556541104911"
"69A011565664915624555924415 5A584145650516445A555"
"55595548565545568116646951 1A59642155555501990285"
"891A5140624165801502984000 62840021A908660880A85A"
"28008958212528A0A604218026444A2AA5452A9522051201"
"99055041A99129298A989A895206621800648A98A8806986"
"4862A4651468A14114268209A55249440A89519065056895"
"8622984881626685219 8A2A2096459194A55040291048662"
"2949A09448210A94429A444126A4A66A0AA18921A4A54514"
"595A81248446A86601905118A2850492910185041822A524"
"22565641A19596906885024058864840 8A14990A6801D253"
"0F125EE5F208B1976A66BCBC917161F6929E636BA8C73470"
"DE18065F6057528D718744E9248DFFF6BB55C188CEACB941"
"9863C3C456B46A21354834ADA6B2132C67747C9EE70D0256"
"0268EE650E56C84BCE6A6700C5E612999110E53A866AE6B4"
"F778230367B8B886C4FEC089C6267F91C7F24D6CE53C754D"
"9CCCDD25756D76CD211D5954F0E8A11343679C1F692CE5C0"
"D0E42A02381144E1E0201B6F49F00628A86E09918488BC3E"
"1E1071561700544C2CE50F4B7BAE4757CA7A0DD0A221260E"
"0574D1F8F81AF072FCBA8061B5B8BB450FDCC6732D35CDBB"
"BC4EF1798EA7E263BCF369059EED3A86E2C61C72CBFCD695"
"21C3A1FD4868AF5638148043162B8B6E39F82B56D6C9C772"
"4807F623C06CC08FF619F7961EC972B1BD2856F87D5E6940"
"DC15B9575264A4AB0CA229C5B02C9AA5BB8DA2BD8EB09A7A"
"5E824C2D666A17C4845CA9530F3A5E45BE5A380B83C81B39"
"65180C706E2CC9ABF4DC8A1559CCF176EFD0FAAD1B8E184A"
"AA08E24CA6F5D070E040D64CD65A1628E99F3BACAB5A7206"
"B5C7D1F7D282651448E259A839E128774530C931C4E4E6F6"
"0AB9CDEB645AA24012C3A0983750C04914C59E34AF67B912"
"8CE906AD162D70E582C42E21898EF8023A9D1BC3B5F5BDE6"
"415CD168F1DE726A2ECB27C48A73356FBBEE8F405A6F8C34"
"F8F5A864C109045618B03FBFDCBAFD8BC934658634070D40"
"E5C62A03F9B544C325BCC56D8C70D67DA39879356E9871B0"
"16A0CCF3FE1C6AB941B69417FF514B616C9CFE9EA066FBB4"
"B081B25D584E40430B0950A7EBFBFA9B4E4EE003F297BA66"
"8F7139F26F218026DC9BFDBBBE6FD1CDE03CAC9814299126"
"E8CBDAFFD54101E07C09B65B88C9743B85C464E44C7256F5"
"06F07BC7D3877E2578D589CD2001E5124C9E647FAC2134B3"
"4E74DC188478E538BBFB750B47600191C51B71F18460CECD"
"5FBDABAA48D6DD6DBE7E26CEF9498220379F1FF25AFACA60"
"CEFFA408CEB677E4CAFB3A43B3E5FAA5B731EA608A945EDF"
"645136ADC77B07DB34E191D9E786ED7CFD97E2330548FFE0"
"21997E2E8B774E4A5F1666A91F05D471874E765DF42C7C3D"
"9EF37CD946E0D69C9EA83CEC9B1AA41A25F1A3A458C9935C"
"BC7D294B64D628DA7F1B0FA4F24E6375DF31AE82815F5DFF"
"4ACD0DD9BD8A8740CB92633D1E000191B837021F143F64B0"
"8388A78B9A0F55FE7B824FA9D4E85709EAC46EAB5B24C46B"
"BC2D1D8D39FEC8BA130EB68A7F55769606F07CB7B8CA0AD9"

```
    "9739C68A365E415983B964F1F2D261145057B7A76D72300A"
    "D49D9FFFFFFE9C41BD48B82F8450274C6C25DCECE61D4C44"
    "3FEE0D3C359FDD4E4409AD607AE7A707B83B8629134DACCB"
    "D54C618EEF0B2F04E848F7B62C494DEC89F2830BACEDB3B8"
    "76670309B36F7D70BE0219F5D05A3C1E5BB58B1CF053FE92"
    "D2E3F934AA2047F963172E04B7457FED3956C08A705C9C44"
    "1F1FEE07E05344C63817F5DA7F298D5323BF88FB490E1C62"
    "8015ECD09C5D89978EB42A2DFD32E0820B005BDB60FFFB15"
    "EBE1A123C6107530FE48C72C925FB85465749EF1A46C6BA7"
    "F0F31D911D9E78608DAA64EC87CB3C82ABEC988FA3656734"
    "A038FB5A3F072B7E9E9F383227EA3D5ACE37A6A5536B9E3A"
    "A926E900B736895EF729F2159BA82A0D090B0D4A26FFC467"
    "511911F39D3EC248467A576EE1F8DCA10DD0C3BE961080D9"
    "25B823AD9538477F258DD445AF4872DFDEB8E4A819DDE314"
    "766683246379B03AA738907ED3671359999FF298C4A44133"
    "417821912013E792C90D939815BC3AEBB565E1D6B42BB356"
    "CC6A6C79EB6D640001C9D0ED847B4D39B2C38FC2123609EF"
    "94608B766EBEB91DD12D228123D29D14C1B4169D8417D260"
    "54304B5C900E5CF78159735D0FFA15B691369BC66811A9F1"
    "1ADB3ED280E3151830BAC71B6E5EE77238F632911067ED8C"
    "525887CF983CFE4A2572D3"),
H ("8CE9955F6DC23A0E49D9B263C43026609612696FD84D37DF"
    "A1192BA0D9412D2125E25A7C64209A92E5F1C6F170D5C4C8"
    "91D5050E336F628544620D6B0A9C5058DBBB51C4F540AE3C"
    "19BD941E7CD3105A7BCD1774FF05B0B65E310A1D6F88253C"
    "FA36C2E7F34130B14613B188D8B9D6C9BE040CE2446F33F7"
    "5E2A61AF7D7C11FFF54505F1E1EEE49172234915D722599C"
    "6BD77D0BCEBEC69435BAC571A194939861CE3ED3960C0FD0"
    "FDB8D7CBD272659BBB881C29A70ECB62388062C4ABB2A565"
    "A0D9CE73E23E9CA7D589DE524E8289762499F867697003DB"
    "B87E1657313D05E5E8E7C7FF407DF7931EB6842C82521936"
    "9E92D2E65E29CFCF31DCD0F40706AF8B4F70D9E8AF6147AC"
    "7D8B1900773D9EE2CE6C4EEC38C4FF44F2E397D95684CE5E"
    "E6FD5FE4975216819BAE496D2CE888A6F630FE448A4872BB"
    "C5164A70A33A9988A3C4AEA17A8688346191CD4BE2F8D021"
    "CC97D5A20BFFFFC628ED6F267145C41BCFB83161C2326294"
    "3B530DF4BD4B6BDD85D78A2DAF0499B8DE97AFE6191AFD3A"
    "35E8268A4D08EC55DBA8BEC15BDE4F823D4B3C62F7990EF7"
    "A655D03DD4897F640ABDD60D58A3244F745CEA09ACCDD56D"
    "E4D8F08F43025E084E91634E25FC7B39BCAFFC1803E8D480"
    "6786548263D6BF8D8ACC84FA9A01D786F438FEDB811DF9FC"
    "0EFFCF9C6A787B17228B260D6ACCE43075E7F80CC57C6821"
    "01CD9CDF12B2D3F931AAAE1A2434AAE772C11CCEE49E2503"
    "DAE5CBC98E41F6AD6E885B1F011E9883CE078B33D99FC414"
    "71CCA14FC0DC12789464CE92B93A08E8F3D25822827A0827"
    "3FA2E5114909F0D4A378A482F0EBDF781991E1E28B3F72CA"
    "D3886575FF7E59F2B8607CE314D8292E27C1D6187EC5C76C"
    "B1347185D760D69F91B7F455DDE52257B03CBB9845B78867"
    "EABAA28535B6DEA69D14693666B1E978503F65412C5AFF4A"
```

```
                "FEC1F537C416EAC390D5EEA0C3B4694DB4FAB95CAC381D12"
                "F42F176F27A4BEE0C0A0A458B51E5C71D82724593014C154"
                "30BECDBB472659D91B093192590A43A6E56BB6AD3C6C5BAA"
                "7AA49D888E1CFA656200680A31ECDB99A4B0109133523653"
                "D9769A945ADEC3890734841B94ACC62A8AB0805F0E9E9E3B"
                "4471552DA0B5E78935C149E3D4DFCB20F75BDA04A86611B4"
                "192BC6AF2B5C3C1C4424E40E8AF9E1A7A3E57B9AACA0B730"
                "585931386329C19846D1BF091084C12380D7B2FB3348E7BC"
                "25EF9BC39FEBA174B8FC9A191A11C345E0E3CB3170593928"
                "429F0729BB96D771BAFCB84633B80377658EBB5866631AE2"
                "EBAE05940AFDBA13F34A0341E923EA2C31F11E2A1A421C45"
                "22AB50153A77B7950834CBF23A16D01462A0FCD270BCF66B"
                "9F6E0A5A39F5989D75643DFC8EC4D5B051AE94C6DB74D777"
                "9C54016EAD109D10AC190B07816CDCD2B8EC9BC596090DF0"
                "0FE23B4E862FE7E0D7EBCBF6B2A1A565435DA662CB34489B"
                "B570286BA114F8"),
            H ("16C15126F734E51268BA916CE3B39A72E171AE79B8C2B6A6"
                "8B34AB0DC5621B7E"));
}
```

Figure 7

6.4. drbg-ctr.h

This is the C header file for the DRBG-CTR-AES256 implementation, used by the sntrup761-test.c test driver to reproduce test vectors.

```
#ifndef DRBG_CTR_H
#define DRBG_CTR_H

#include <string.h>
#include <stdint.h>

#include "aes256_ecb.h"

#define DRBG_CTR_AES256_SEED_SIZE (AES_BLOCK_SIZE + AES256_KEY_SIZE)

struct drbg_ctr_aes256_ctx
{
  uint8_t K[AES256_KEY_SIZE];
  uint8_t V[AES_BLOCK_SIZE];
};

/* Initialize using DRBG_CTR_AES256_SEED_SIZE bytes of
   SEED_MATERIAL.  */
void
drbg_ctr_aes256_init (struct drbg_ctr_aes256_ctx *ctx,
                      uint8_t *seed_material);

/* Output N bytes of random data into DST.  */
void
drbg_ctr_aes256_random (struct drbg_ctr_aes256_ctx *ctx,
                        size_t n, uint8_t *dst);

#endif /* DRBG_CTR_H */
```

Figure 8

6.5. drbg-ctr.c

This is the C implementation file for the DRBG-CTR-AES256 implementation, used by the sntrup761-test.c test driver to reproduce test vectors.

```
#include "drbg-ctr.h"

#include <string.h>

static void memxor(uint8_t *dst, const uint8_t *src, size_t n)
{
  size_t i;
  for (i = 0; i < n; i++)
    dst[i] ^= src[i];
}

static void increment (uint8_t *V)
{
  size_t i;

  for (i = AES_BLOCK_SIZE - 1; i > 0; i--)
    {
      if (V[i] == 0xFF)
        V[i] = 0;
      else
        {
          V[i]++;
          break;
        }
    }
}

static void
drbg_ctr_aes256_update (uint8_t *Key, uint8_t *V, uint8_t *provided_data
{
  uint8_t tmp[DRBG_CTR_AES256_SEED_SIZE];

  increment (V);
  aes256_encrypt (Key, tmp, V);

  increment (V);
  aes256_encrypt (Key, tmp + AES_BLOCK_SIZE, V);

  increment (V);
  aes256_encrypt (Key, tmp + 2 * AES_BLOCK_SIZE, V);

  if (provided_data)
    memxor (tmp, provided_data, 48);

  memcpy (Key, tmp, AES256_KEY_SIZE);
  memcpy (V, tmp + AES256_KEY_SIZE, AES_BLOCK_SIZE);
}

void
drbg_ctr_aes256_init (struct drbg_ctr_aes256_ctx *ctx, uint8_t *seed_mat
```

```
{
  memset (ctx->K, 0, AES256_KEY_SIZE);
  memset (ctx->V, 0, AES_BLOCK_SIZE);

  drbg_ctr_aes256_update (ctx->K, ctx->V, seed_material);
}

void
drbg_ctr_aes256_random (struct drbg_ctr_aes256_ctx *ctx,
                        size_t n, uint8_t *dst)
{
  while (n >= AES_BLOCK_SIZE)
    {
      increment (ctx->V);
      aes256_encrypt (ctx->K, dst, ctx->V);
      dst += AES_BLOCK_SIZE;
      n -= AES_BLOCK_SIZE;
    }

  if (n > 0)
    {
      uint8_t block[AES_BLOCK_SIZE];

      increment (ctx->V);
      aes256_encrypt (ctx->K, block, ctx->V);
      memcpy (dst, block, n);
    }

  drbg_ctr_aes256_update (ctx->K, ctx->V, NULL);
}
```

Figure 9

6.6. sha512.h, sha512.c, aes256_ecb.h, aes256_ecb.c

These are helper code used by the sntrup761-test.c test driver to reproduce test vectors, and require OpenSSL but you may substitute your own implementation of these cryptographic primitives.

```
extern void crypto_hash_sha512 (unsigned char *out,
                                const unsigned char *in,
                                unsigned long long inlen);
```

Figure 10

```
#include <openssl/sha.h>
#include "sha512.h"

void crypto_hash_sha512 (unsigned char *out,
                         const unsigned char *in,
                         unsigned long long inlen)
{
  SHA512(in, inlen, out);
}
```

Figure 11

```
#define AES_BLOCK_SIZE 16
#define AES256_KEY_SIZE 32

extern void
aes256_encrypt (const unsigned char *key,
                unsigned char *out,
                const unsigned char *in);
```

Figure 12

```
#include <openssl/evp.h>

#include "aes256_ecb.h"

void
aes256_encrypt (const unsigned char *key,
                unsigned char *out,
                const unsigned char *in)
{
  EVP_CIPHER_CTX *ctx;
  int len;

  if(!(ctx = EVP_CIPHER_CTX_new()))
    abort();

  if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_ecb(), NULL, key, NULL))
    abort();

  if(1 != EVP_EncryptUpdate(ctx, out, &len, in, AES_BLOCK_SIZE))
    abort();

  EVP_CIPHER_CTX_free(ctx);
}
```

Figure 13

7. Streamlined NTRU Prime: Test Vectors

For brevity, below is a reduced number of test vectors, for full examples see [sntrup761kat].

```
count = 0

seed = "061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479"
       "D09D86DC9ABCFDE7056A8C266F9EF97ED08541DBD2E1FFA1"

pk   = "36C969CF1008A6AA9551A784941C65A9BF68C2DC33FA36B5"
       "D266B25171B346679F2D22BF3123A79C790D6DEC68E1BC44"
       "420A6824F5357C78E3C336FEE0551E620DCB975F563682A3"
       "12A3353B521C727F57CABED0C3228F09317CAE8B58158EBF"
       "5B26BDC6E6365AA601ACAD2ABD37F5830D0BBFE355705C0A"
       "62B76A5C910AD04E55E5DAD749C7393D2E2E8AB643E62E47"
       "57AD2201CAA33203F53B4A9D4757E7274D72BDB036A31D7D"
       "E11E5D1C66CF3059F33B6A2972C1E1D9C9FB2AEBC78B2C05"
       "5D48D79C3A7C996C08B7DEF0791CDE895053885D8DA1D254"
       "EE19C090AF34F4720B0B77139108D8498982FEAB0B54934C"
       "EE3DCD24B049F981A84C928028A64A26CDF87052313C3E50"
       "B2E1F539394502433C0962996C3599189B15174281B6595B"
       "567B8C4CD80902860E613AE1906A55607CBF0E11AD6C0C0D"
       "F38C006A5F535FA6E6FB49D10B78B9CB6473BF0630518DB6"
       "FDECFD70DED201C7E35FFB3BB78D8AEC181F6DE960C316FE"
       "354B7CC69E8048201965AAFEF4EA3F808737FF45255A1779"
       "DE57A4B68DB587263D7B7F6CB07D8B01224DD291237EFD9C"
       "01676E30154A2A7D60174536580FE64EFEDDA5FB42C13ED8"
       "C5768A3CBCAE7A2343B3128CD5C1C663A07C7DC0E1E2642C"
       "0D9A02F349C964154A6C4308D8ECF30F47E9A81EED2A32A2"
       "BB44DFC36A28A66AE77BB139FA416B6327EEFE33632469CC"
       "C21229587573C4F7752CE1CC79CA0C6BDA08CD79E25720D2"
       "D9092EA2AB13F31E9A3AF1C69FC6379D8E2AD2B87B514C81"
       "7A087338B7B0736B8954DE9223225B40079C92601248C14B"
       "2104901E74F849D9EEE9F5636C1DAD6031AB477C573197E7"
       "EB6CE535D9F0F69183DD9DF5521595E5C9E98D846CE08AA6"
       "55B70CC0D8041401929690298F645D9112DBB03B189CDB1F"
       "CF4D512F6874B409BF55EE1CC284A05B698B8818F043C259"
       "1C9F4ABA29CF4259915D6A0BE71B6B93963C618CBB567838"
       "E5EADE6500DE9AD250083A01EB3EB44C00EECB2B0F874CDA"
       "165FAA6524CA13D435C938DB9469292FE97283C69222107E"
       "A2F9EFCA1B6D41DCA5B149B2A8CC2244A1BC54261CC11742"
       "ACD27B7F2352C33FB83FE143386479D78D6D5C3E51684B60"
       "A56714182449C94327139B0289B9BDE0AE3FEF319FE2C605"
       "AB5507C894D2C2761A3BD3EA30F4BA928F9F23303061617E"
       "2F042DCF229AFF2C9345A6B3CDCF90D3E3BCC3A1110C8561"
       "6BD585C31CDE3E69ADC12A18BFA5797DF0543435A7C874A8"
       "ACF3786FCCBA4A6CEAA65E5666230AA7206AE78EB1B98CB5"
       "236508E6357E18CCCFAEC5693532BE4EE38022C48FE94F62"
       "B0293A088BF4D737F48748F23BCB338B58B4D666AD3C64E9"
       "ECD07265F971B07AF716D4A5B719C3F5FE35744734CBA543"
       "0381661E372B6F6510D61B11E4697A1A961589949DBA53B5"
       "BC5BDB76FA09324387D799536506182EFF7078034B34E1DD"
       "D2612B0A40E7F1A294FBA869DA2E46C4C36BB08C7E9BEF09"
```

```
        "A94459E8FC2D3CC579D15284DF1F19EF77E03041511ACC39"
        "BF9CA8AC8DE6A0E5EEA0BBD4289B6DAE38E9E82EB50A0397"
        "B3EE6C52FBDC7DEEA2C376825E89016C859B09488548BF76"
        "CA62D696DF94A61E8F0E13C69EE816CE5AA2768987B0A782"
        "D74C673EC005"

sk   = "55565515155545556569159955A255545545616145059455"
        "65569150558956155488529554565500985449955555AA546"
        "5400A5555558956510A6515596559665996A449154154144"
        "65565555551105624156954944626940555555165665955 5"
        "55A5159155595659156565154240408919545456955859 0"
        "64258656285054569505A6A95694944666141558011556 50"
        "55485955259049A5545525455504526A195455554185525A"
        "899625525556582484644514556589656258515459150185"
        "08154A51A0448959112A286212608A52911A949119A28956"
        "64165900A861541A200A52104029544600425A5A1198101A"
        "1A50A094A551425A484959A1506486A618555A8990561502"
        "9622A2A46696A2A6595A4668262252505048 9A8040562808"
        "168156854185026540A1059A94AAA291811948A259816991"
        "110951802409AA5A2A0629009245144665560840409689 8A"
        "A562205622851508919140661 10A0905A841044865042A26"
        "88AA12441AAA46064669155555A2109659299922820036C9"
        "69CF1008A6AA9551A784941C65A9BF68C2DC33FA36B5D266"
        "B25171B346679F2D22BF3123A79C790D6DEC68E1BC44420A"
        "6824F5357C78E3C336FEE0551E620DCB975F563682A312A3"
        "353B521C727F57CABED0C3228F09317CAE8B58158EBF5B26"
        "BDC6E6365AA601ACAD2ABD37F5830D0BBFE355705C0A62B7"
        "6A5C910AD04E55E5DAD749C7393D2E2E8AB643E62E4757AD"
        "2201CAA33203F53B4A9D4757E7274D72BDB036A31D7DE11E"
        "5D1C66CF3059F33B6A2972C1E1D9C9FB2AEBC78B2C055D48"
        "D79C3A7C996C08B7DEF0791CDE895053885D8DA1D254EE19"
        "C090AF34F4720B0B77139108D8498982FEAB0B54934CEE3D"
        "CD24B049F981A84C928028A64A26CDF87052313C3E50B2E1"
        "F539394502433C0962996C3599189B15174281B6595B567B"
        "8C4CD80902860E613AE1906A55607CBF0E11AD6C0C0DF38C"
        "006A5F535FA6E6FB49D10B78B9CB6473BF0630518DB6FDEC"
        "FD70DED201C7E35FFB3BB78D8AEC181F6DE960C316FE354B"
        "7CC69E8048201965AAFEF4EA3F808737FF45255A1779DE57"
        "A4B68DB587263D7B7F6CB07D8B01224DD291237EFD9C0167"
        "6E30154A2A7D60174536580FE64EFEDDA5FB42C13ED8C576"
        "8A3CBCAE7A2343B3128CD5C1C663A07C7DC0E1E2642C0D9A"
        "02F349C964154A6C4308D8ECF30F47E9A81EED2A32A2BB44"
        "DFC36A28A66AE77BB139FA416B6327EEFE33632469CCC212"
        "29587573C4F7752CE1CC79CA0C6BDA08CD79E25720D2D909"
        "2EA2AB13F31E9A3AF1C69FC6379D8E2AD2B87B514C817A08"
        "7338B7B0736B8954DE9223225B40079C92601248C14B2104"
        "901E74F849D9EEE9F5636C1DAD6031AB477C573197E7EB6C"
        "E535D9F0F69183DD9DF5521595E5C9E98D846CE08AA655B7"
        "0CC0D8041401929690298F645D9112DBB03B189CDB1FCF4D"
```

```
          "512F6874B409BF55EE1CC284A05B698B8818F043C2591C9F"
          "4ABA29CF4259915D6A0BE71B6B93963C618CBB567838E5EA"
          "DE6500DE9AD250083A01EB3EB44C00EECB2B0F874CDA165F"
          "AA6524CA13D435C938DB9469292FE97283C69222107EA2F9"
          "EFCA1B6D41DCA5B149B2A8CC2244A1BC54261CC11742ACD2"
          "7B7F2352C33FB83FE143386479D78D6D5C3E51684B60A567"
          "14182449C94327139B0289B9BDE0AE3FEF319FE2C605AB55"
          "07C894D2C2761A3BD3EA30F4BA928F9F23303061617E2F04"
          "2DCF229AFF2C9345A6B3CDCF90D3E3BCC3A1110C85616BD5"
          "85C31CDE3E69ADC12A18BFA5797DF0543435A7C874A8ACF3"
          "786FCCBA4A6CEAA65E5666230AA7206AE78EB1B98CB52365"
          "08E6357E18CCCFAEC5693532BE4EE38022C48FE94F62B029"
          "3A088BF4D737F48748F23BCB338B58B4D666AD3C64E9ECD0"
          "7265F971B07AF716D4A5B719C3F5FE35744734CBA5430381"
          "661E372B6F6510D61B11E4697A1A961589949DBA53B5BC5B"
          "DB76FA09324387D799536506182EFF7078034B34E1DDD261"
          "2B0A40E7F1A294FBA869DA2E46C4C36BB08C7E9BEF09A944"
          "59E8FC2D3CC579D15284DF1F19EF77E03041511ACC39BF9C"
          "A8AC8DE6A0E5EEA0BBD4289B6DAE38E9E82EB50A0397B3EE"
          "6C52FBDC7DEEA2C376825E89016C859B09488548BF76CA62"
          "D696DF94A61E8F0E13C69EE816CE5AA2768987B0A782D74C"
          "673EC0059FA532AE97F8BDF90F90130926654FB8B469D772"
          "049D72A8375CD8459D06CC1B90633EB3A899685D21491B50"
          "62A9FC73FB6E878D7A73198EFA0B569D9CE665CE126FFDC9"
          "862EB00D11F457A7995555F7C92011C24E1CEC45C270FB5F"
          "121F08177F97FC3F631C0EF86A92E99D557FB69A0F6FCD8B"
          "1C0EF94AA7429B3A8A11614D847202D3D04A98313FC0D63A"
          "BF9FD75CA321C879458BF8837B7E74CCF5179C7714FA9800"
          "D1821EE3F9639D28136B7910872631F85AE7B6DD289E0103"
          "4217210859D4E53C65487CE38AFF621DA76BA1C4E2E77ED5"
          "380B47B8D983CCB5BB793E"

ct    = "84F327D38929039EF84366AB796A96E6CC268454D5AD8D05"
          "410D3E969B6D228B6CED2CBAD7BAB3B4B7EE453747D3331D"
          "43B21CD2ECC3BF557D696E81773EE69F687FE1F61E63742A"
          "F913A35418575467118409D4FFDBBAC5CA062DFF3F04D761"
          "54D17EC34212DEBFCA967A91E461F73786D949D6A9765140"
          "B9DC7849639A487058F45DDA919576090D35E783804BA7B8"
          "B33B5C6A8C1EBB7C39F042696CF9F2B624D26EA5D6D10148"
          "9E242CC35126B573928B6B8BF9945269AD22E7DA70CCB7DC"
          "374F75641978D5F6D5F540A1169EB098570A7CFD65C9257C"
          "4C196F42C6DA4F4466D8F11A17C6A75A9BDD45B5AC77A836"
          "7B005ED22141A81B995A19E4DC9E6743B4B45E0329E7A00D"
          "3FF90C7A917CDC9CA75A34FFFDD9A321751B5C8B2AB9E1EA"
          "037CF45C04B412A97F52E7C6D66508456D9117961257B4F5"
          "0BB9F500DE99B6F061C377D44F495D711AB2ED3E88CCF14D"
          "F10F8C60BD00E29CD83AD160FAEF2984FAC7E897CBB8CDA1"
          "44D50E4C1AD20238A2617598DA6EFE786AA82E0931B0B5D1"
          "A7FA024B856353AFAEDB0A1150A42DA182353CE5A00403BB"
```

```
        "6B7A13EFFC14B0FD97E1117302ED8A66FAAED88B43B34CFC"
        "73CB33E49A4C34D0BFDEE2FEADC2B4C8BB1EC41215586FC6"
        "6550D496373305B99CEE254EAC71D3CDAC689A777062ACAE"
        "65B031B5A1A973835BE6A4BB061F178531C45DE3B33E7763"
        "B483621821E037FE5EF48FD971D8D2BEFDA63250E9B1181D"
        "8B8086BED9CDA8CDE6644CDE8EBF7C1F1038AFE798A2A4C4"
        "92B603B7F4F8805D0FDF19D8274888CEBEC3043CC9ACF024"
        "BD30D8B0D28D3119859301103865 98775CF84E0848B11E09"
        "0905A65F56142FC17D2573536FC51E7D5B2E0FF7B0947115"
        "5008561156FBAEC215031ED0D844EB14D6C609173355FFF8"
        "AE22C6E5BCCD19B2832E5FBE625C2C0E516E1369D1FCF49A"
        "5FDBAF8CF572F04317DB8ED70EE1605B96D077B4F054C72B"
        "139F5BCAA5BB627E39BE5BCC97AF922C5E70835BD918281B"
        "95E557FC86A43C75980A0D9F0F9162F230014CD867DD07DB"
        "23260934546C04499D6B8627B8D5DEC73FA45327F8498352"
        "9BAF3AA10E885724FE76055099D136CCAACC159B4CE38784"
        "404CC9B04723910F990AF77C64D42A29D784FC897955758F"
        "1D2A12BB7C6F6A8E5B94EFA740F662CF0DA2D099C61EE1CC"
        "6B661A64F0887B805AF53C7B64C4456DAF228582C1B2212A"
        "8F718E56DF6B72D7EFA0C5D26582CE96AEFCE13CB2830C3F"
        "B32E5115AACC20D3A335F7A4C12E0CD1DE532D8D9D42D69B"
        "C2D0BE0AC7DF3B8F416B9A53DB261C47272725A8A8829867"
        "8351B42103219092076B6ECE744D993388A8F2C475079CB8"
        "BC4DC492A4797147C01EA3508225826F7A3D32BF502B0F45"
        "52F36FC9F6FB91E82DB1949338B45436EC0FF63B53F900F2"
        "7F8147D420F93D33C83794980E94143FD361F944DDB56D31"
        "14EE974C96A155"

ss   = "344CA5E25F6DA5EA95E4A695B1C5446ECA9859334532E4A9"
       "537669F012C743A2"

count = 1

seed = "D81C4D8D734FCBFBEADE3D3F8A039FAA2A2C9957E835AD55"
       "B22E75BF57BB556AC81ADDE6AEEB4A5A875C3BFCADFA958F"

pk   = "D2530F125EE5F208B1976A66BCBC917161F6929E636BA8C7"
       "3470DE18065F6057528D718744E9248DFFF6BB55C188CEAC"
       "B9419863C3C456B46A21354834ADA6B2132C67747C9EE70D"
       "02560268EE650E56C84BCE6A6700C5E612999110E53A866A"
       "E6B4F778230367B8B886C4FEC089C6267F91C7F24D6CE53C"
       "754D9CCCDD25756D76CD211D5954F0E8A11343679C1F692C"
       "E5C0D0E42A02381144E1E0201B6F49F00628A86E09918488"
       "BC3E1E1071561700544C2CE50F4B7BAE4757CA7A0DD0A221"
       "260E0574D1F8F81AF072FCBA8061B5B8BB450FDCC6732D35"
       "CDBBBC4EF1798EA7E263BCF369059EED3A86E2C61C72CBFC"
       "D69521C3A1FD4868AF5638148043162B8B6E39F82B56D6C9"
       "C7724807F623C06CC08FF619F7961EC972B1BD2856F87D5E"
       "6940DC15B9575264A4AB0CA229C5B02C9AA5BB8DA2BD8EB0"
```

```
        "9A7A5E824C2D666A17C4845CA9530F3A5E45BE5A380B83C8"
        "1B3965180C706E2CC9ABF4DC8A1559CCF176EFD0FAAD1B8E"
        "184AAA08E24CA6F5D070E040D64CD65A1628E99F3BACAB5A"
        "7206B5C7D1F7D282651448E259A839E128774530C931C4E4"
        "E6F60AB9CDEB645AA24012C3A0983750C04914C59E34AF67"
        "B9128CE906AD162D70E582C42E21898EF8023A9D1BC3B5F5"
        "BDE6415CD168F1DE726A2ECB27C48A73356FBBEE8F405A6F"
        "8C34F8F5A864C109045618B03FBFDCBAFD8BC93465863407"
        "0D40E5C62A03F9B544C325BCC56D8C70D67DA39879356E98"
        "71B016A0CCF3FE1C6AB941B69417FF514B616C9CFE9EA066"
        "FBB4B081B25D584E40430B0950A7EBFBFA9B4E4EE003F297"
        "BA668F7139F26F218026DC9BFDBBBE6FD1CDE03CAC981429"
        "9126E8CBDAFFD54101E07C09B65B88C9743B85C464E44C72"
        "56F506F07BC7D3877E2578D589CD2001E5124C9E647FAC21"
        "34B34E74DC188478E538BBFB750B47600191C51B71F18460"
        "CECD5FBDABAA48D6DD6DBE7E26CEF9498220379F1FF25AFA"
        "CA60CEFFA408CEB677E4CAFB3A43B3E5FAA5B731EA608A94"
        "5EDF645136ADC77B07DB34E191D9E786ED7CFD97E2330548"
        "FFE021997E2E8B774E4A5F1666A91F05D471874E765DF42C"
        "7C3D9EF37CD946E0D69C9EA83CEC9B1AA41A25F1A3A458C9"
        "935CBC7D294B64D628DA7F1B0FA4F24E6375DF31AE82815F"
        "5DFF4ACD0DD9BD8A8740CB92633D1E000191B837021F143F"
        "64B08388A78B9A0F55FE7B824FA9D4E85709EAC46EAB5B24"
        "C46BBC2D1D8D39FEC8BA130EB68A7F55769606F07CB7B8CA"
        "0AD99739C68A365E415983B964F1F2D261145057B7A76D72"
        "300AD49D9FFFFFFE9C41BD48B82F8450274C6C25DCECE61D"
        "4C443FEE0D3C359FDD4E4409AD607AE7A707B83B8629134D"
        "ACCBD54C618EEF0B2F04E848F7B62C494DEC89F2830BACED"
        "B3B876670309B36F7D70BE0219F5D05A3C1E5BB58B1CF053"
        "FE92D2E3F934AA2047F963172E04B7457FED3956C08A705C"
        "9C441F1FEE07E05344C63817F5DA7F298D5323BF88FB490E"
        "1C628015ECD09C5D89978EB42A2DFD32E0820B005BDB60FF"
        "FB15EBE1A123C6107530FE48C72C925FB85465749EF1A46C"
        "6BA7F0F31D911D9E78608DAA64EC87CB3C82ABEC988FA365"
        "6734A038FB5A3F072B7E9E9F383227EA3D5ACE37A6A5536B"
        "9E3AA926E900"

sk    = "64515999655565914856965695410550601945189559554 9"
        "84866951658555588521521601525454456 9A65216995559"
        "85415514151155925466465965555565555561461655950 8"
        "955596965965656456995 55A1596555A966566614655A409"
        "5155955466959061565 9A5515551255565955519555699 15"
        "A5614089A15501582951945A155586595555556541104911"
        "69A011565664915624555924415 5A584145650516445A555"
        "55595548565545568116646951 1A59642155555501990285"
        "891A5140624165801502984000628400 21A908660880A85A"
        "28008958212528A0A604218026444A2AA5452A9522051201"
        "99055041A99129298A989A895206621800648A98A8806986"
        "4862A4651468A14114268209A55249440A89519065056895"
```

"8622984881626685219 8A2A2096459194A55040291048662"
"2949A09448210A94429A444126A4A66A0AA18921A4A54514"
"595A81248446A86601905118A28504929101850418 22A524"
"22565641A19596906885024058864840 8A14990A6801D253"
"0F125EE5F208B1976A66BCBC917161F6929E636BA8C73470"
"DE18065F6057528D718744E9248DFFF6BB55C188CEACB941"
"9863C3C456B46A21354834ADA6B2132C67747C9EE70D0256"
"0268EE650E56C84BCE6A6700C5E612999110E53A866AE6B4"
"F778230367B8B886C4FEC089C6267F91C7F24D6CE53C754D"
"9CCCDD25756D76CD211D5954F0E8A11343679C1F692CE5C0"
"D0E42A02381144E1E0201B6F49F00628A86E09918488BC3E"
"1E1071561700544C2CE50F4B7BAE4757CA7A0DD0A221260E"
"0574D1F8F81AF072FCBA8061B5B8BB450FDCC6732D35CDBB"
"BC4EF1798EA7E263BCF369059EED3A86E2C61C72CBFCD695"
"21C3A1FD4868AF5638148043162B8B6E39F82B56D6C9C772"
"4807F623C06CC08FF619F7961EC972B1BD2856F87D5E6940"
"DC15B9575264A4AB0CA229C5B02C9AA5BB8DA2BD8EB09A7A"
"5E824C2D666A17C4845CA9530F3A5E45BE5A380B83C81B39"
"65180C706E2CC9ABF4DC8A1559CCF176EFD0FAAD1B8E184A"
"AA08E24CA6F5D070E040D64CD65A1628E99F3BACAB5A7206"
"B5C7D1F7D282651448E259A839E128774530C931C4E4E6F6"
"0AB9CDEB645AA24012C3A0983750C04914C59E34AF67B912"
"8CE906AD162D70E582C42E21898EF8023A9D1BC3B5F5BDE6"
"415CD168F1DE726A2ECB27C48A73356FBBEE8F405A6F8C34"
"F8F5A864C109045618B03FBFDCBAFD8BC934658634070D40"
"E5C62A03F9B544C325BCC56D8C70D67DA39879356E9871B0"
"16A0CCF3FE1C6AB941B69417FF514B616C9CFE9EA066FBB4"
"B081B25D584E40430B0950A7EBFBFA9B4E4EE003F297BA66"
"8F7139F26F218026DC9BFDBBBE6FD1CDE03CAC9814299126"
"E8CBDAFFD54101E07C09B65B88C9743B85C464E44C7256F5"
"06F07BC7D3877E2578D589CD2001E5124C9E647FAC2134B3"
"4E74DC188478E538BBFB750B47600191C51B71F18460CECD"
"5FBDABAA48D6DD6DBE7E26CEF9498220379F1FF25AFACA60"
"CEFFA408CEB677E4CAFB3A43B3E5FAA5B731EA608A945EDF"
"645136ADC77B07DB34E191D9E786ED7CFD97E2330548FFE0"
"21997E2E8B774E4A5F1666A91F05D471874E765DF42C7C3D"
"9EF37CD946E0D69C9EA83CEC9B1AA41A25F1A3A458C9935C"
"BC7D294B64D628DA7F1B0FA4F24E6375DF31AE82815F5DFF"
"4ACD0DD9BD8A8740CB92633D1E000191B837021F143F64B0"
"8388A78B9A0F55FE7B824FA9D4E85709EAC46EAB5B24C46B"
"BC2D1D8D39FEC8BA130EB68A7F55769606F07CB7B8CA0AD9"
"9739C68A365E415983B964F1F2D261145057B7A76D72300A"
"D49D9FFFFFFE9C41BD48B82F8450274C6C25DCECE61D4C44"
"3FEE0D3C359FDD4E4409AD607AE7A707B83B8629134DACCB"
"D54C618EEF0B2F04E848F7B62C494DEC89F2830BACEDB3B8"
"76670309B36F7D70BE0219F5D05A3C1E5BB58B1CF053FE92"
"D2E3F934AA2047F963172E04B7457FED3956C08A705C9C44"
"1F1FEE07E05344C63817F5DA7F298D5323BF88FB490E1C62"
"8015ECD09C5D89978EB42A2DFD32E0820B005BDB60FFFB15"

```
        "EBE1A123C6107530FE48C72C925FB85465749EF1A46C6BA7"
        "F0F31D911D9E78608DAA64EC87CB3C82ABEC988FA3656734"
        "A038FB5A3F072B7E9E9F383227EA3D5ACE37A6A5536B9E3A"
        "A926E900B736895EF729F2159BA82A0D090B0D4A26FFC467"
        "511911F39D3EC248467A576EE1F8DCA10DD0C3BE961080D9"
        "25B823AD9538477F258DD445AF4872DFDEB8E4A819DDE314"
        "766683246379B03AA738907ED3671359999FF298C4A44133"
        "417821912013E792C90D939815BC3AEBB565E1D6B42BB356"
        "CC6A6C79EB6D640001C9D0ED847B4D39B2C38FC2123609EF"
        "94608B766EBEB91DD12D228123D29D14C1B4169D8417D260"
        "54304B5C900E5CF78159735D0FFA15B691369BC66811A9F1"
        "1ADB3ED280E3151830BAC71B6E5EE77238F632911067ED8C"
        "525887CF983CFE4A2572D3"

ct    = "8CE9955F6DC23A0E49D9B263C43026609612696FD84D37DF"
        "A1192BA0D9412D2125E25A7C64209A92E5F1C6F170D5C4C8"
        "91D5050E336F628544620D6B0A9C5058DBBB51C4F540AE3C"
        "19BD941E7CD3105A7BCD1774FF05B0B65E310A1D6F88253C"
        "FA36C2E7F34130B14613B188D8B9D6C9BE040CE2446F33F7"
        "5E2A61AF7D7C11FFF54505F1E1EEE49172234915D722599C"
        "6BD77D0BCEBEC69435BAC571A194939861CE3ED3960C0FD0"
        "FDB8D7CBD272659BBB881C29A70ECB62388062C4ABB2A565"
        "A0D9CE73E23E9CA7D589DE524E8289762499F867697003DB"
        "B87E1657313D05E5E8E7C7FF407DF7931EB6842C82521936"
        "9E92D2E65E29CFCF31DCD0F40706AF8B4F70D9E8AF6147AC"
        "7D8B1900773D9EE2CE6C4EEC38C4FF44F2E397D95684CE5E"
        "E6FD5FE4975216819BAE496D2CE888A6F630FE448A4872BB"
        "C5164A70A33A9988A3C4AEA17A8688346191CD4BE2F8D021"
        "CC97D5A20BFFFFC628ED6F267145C41BCFB83161C2326294"
        "3B530DF4BD4B6BDD85D78A2DAF0499B8DE97AFE6191AFD3A"
        "35E8268A4D08EC55DBA8BEC15BDE4F823D4B3C62F7990EF7"
        "A655D03DD4897F640ABDD60D58A3244F745CEA09ACCDD56D"
        "E4D8F08F43025E084E91634E25FC7B39BCAFFC1803E8D480"
        "6786548263D6BF8D8ACC84FA9A01D786F438FEDB811DF9FC"
        "0EFFCF9C6A787B17228B260D6ACCE43075E7F80CC57C6821"
        "01CD9CDF12B2D3F931AAAE1A2434AAE772C11CCEE49E2503"
        "DAE5CBC98E41F6AD6E885B1F011E9883CE078B33D99FC414"
        "71CCA14FC0DC12789464CE92B93A08E8F3D25822827A0827"
        "3FA2E5114909F0D4A378A482F0EBDF781991E1E28B3F72CA"
        "D3886575FF7E59F2B8607CE314D8292E27C1D6187EC5C76C"
        "B1347185D760D69F91B7F455DDE52257B03CBB9845B78867"
        "EABAA28535B6DEA69D14693666B1E978503F65412C5AFF4A"
        "FEC1F537C416EAC390D5EEA0C3B4694DB4FAB95CAC381D12"
        "F42F176F27A4BEE0C0A0A458B51E5C71D82724593014C154"
        "30BECDBB472659D91B093192590A43A6E56BB6AD3C6C5BAA"
        "7AA49D888E1CFA656200680A31ECDB99A4B0109133523653"
        "D9769A945ADEC3890734841B94ACC62A8AB0805F0E9E9E3B"
        "4471552DA0B5E78935C149E3D4DFCB20F75BDA04A86611B4"
        "192BC6AF2B5C3C1C4424E40E8AF9E1A7A3E57B9AACA0B730"
```

```
        "585931386329C19846D1BF091084C12380D7B2FB3348E7BC"
        "25EF9BC39FEBA174B8FC9A191A11C345E0E3CB3170593928"
        "429F0729BB96D771BAFCB84633B80377658EBB5866631AE2"
        "EBAE05940AFDBA13F34A0341E923EA2C31F11E2A1A421C45"
        "22AB50153A77B7950834CBF23A16D01462A0FCD270BCF66B"
        "9F6E0A5A39F5989D75643DFC8EC4D5B051AE94C6DB74D777"
        "9C54016EAD109D10AC190B07816CDCD2B8EC9BC596090DF0"
        "0FE23B4E862FE7E0D7EBCBF6B2A1A565435DA662CB34489B"
        "B570286BA114F8"

ss   = "16C15126F734E51268BA916CE3B39A72E171AE79B8C2B6A6"
       "8B34AB0DC5621B7E"
```

Figure 14

8. Acknowledgements

The majority of this document was derived from [NTRUPrime] [NTRUPrimePQCS].

9. IANA Considerations

This document has no IANA actions.

10. Security Considerations

See [NTRUPrime] and [NTRUPrimePQCS] for discussions.

Streamlined NTRU Prime sntrup761 is aiming for the standard goal of IND-CCA2 security, is widely implemented with good performance on a wide range of architectures, and has been studied by researchers for several years. However new cryptographic primitives should be introduced and trusted conservatively, and new research findings may be published at any time that may warrant implementation reconsiderations.

The increase in communication size and computational requirements may be a concern for restricted computational devices, which would then not be able to take advantage of the improved security properties offer by this work.

11. References

11.1. Normative References

[NTRUPrimePQCS]
Bernstein, D.J., Brumley, B. B., Chen,, M., Chuengsatiansup, C., Lange, T., Marotzke, A., Peng, B., Tuveri, N., Vredendaal, C. V., and B. Yang, "NTRU Prime: round 3", WWW https://ntruprime.cr.yp.to/nist/ntruprime-20201007.pdf, October 2020.

11.2. Informative References

[NTRUPrime] Bernstein, D.J., Chuengsatiansup, C., Lange, T., and C. van Vredendaal, "NTRU Prime: reducing attack surface at

low cost", WWW https://ntruprime.cr.yp.to/
ntruprime-20170816.pdf, August 2017.

[SageMath] "SageMath", WWW https://www.sagemath.org/.

[sntrup761kat] "sntrup761 kat_kem.rsp", WWW https://
ntruprime.cr.yp.to/nist/ntruprime-20201007/KAT/kem/
sntrup761/kat_kem.rsp.html.

Author's Address

Simon Josefsson

Email: simon@josefsson.org
URI: https://blog.josefsson.org/