

# NTRUEncrypt SDK for C/C++ User's Notes

---

*Security Innovation, Inc., September 2013*

## Introduction

The NTRUEncrypt SDK for C/C++ implements the NTRUEncrypt algorithm, a replacement for legacy public-key encryption algorithms such as RSA. NTRUEncrypt gives incredible performance gains at no loss in security.

The September 2013 release of this SDK includes documentation in the form of sample code, source code, and fully documented header files to let you get up and running with NTRUEncrypt as quickly as possible. The notes in this short document supplement the sample code and header files, and provide extra background information to help you use NTRUEncrypt in the way best suited to your needs.

## Choosing a Parameter Set

The NTRUEncrypt SDK contains 16 different “parameter sets” which allow the user to select the appropriate security level. The parameter sets are specified using the `NTRU_ENCRYPT_PARAM_SET_ID` given in the header file `crypto_ntru.h`. The parameter set chosen is specified at key generation time; when the private and public keys are generated, the parameter set is encoded into the key blob.

The parameter set you choose depends on the following considerations:

- What security level do you need?
- Do you require compatibility with ASC Standard X9.98-2010 (for example, are you responding to an RFP that requires this compatibility?)
- Is it most important to minimize operation time, bandwidth, or both?

Once you have the answers to the questions above, Table 1 will tell you what parameter set to use. Note that this SDK does not provide a parameter set at the commonly used RSA-1024 security level, equivalent to 80-bit symmetric ciphers or 163-bit Elliptic Curve cryptography. This is because RSA-1024 is deprecated by NIST, Microsoft, and the CA/Browser Forum, and certificates at that security level will no longer be issued after December 31, 2010 (see, for example, <http://technet.microsoft.com/en-us/library/cc751157.aspx>).

Security Level	Best bandwidth and speed, no X9.98 compatibility	Need X9.98 compatibility		
		Best speed	Best bandwidth	Balance of speed and bandwidth
112-bit symmetric / RSA-2048 / ECC-224 ("commercial strength")	NTRU_ EES401EP2	NTRU_ EES659EP1	NTRU_ EES401EP1	NTRU_ EES541EP1
128-bit symmetric / RSA-4096 / ECC-256 ("industrial strength")	NTRU_ EES439EP1	NTRU_ EES761EP1	NTRU_ EES449EP1	NTRU_ EES613EP1
192-bit symmetric / RSA-7680 / ECC-384	NTRU_ EES593EP1	NTRU_ EES1087EP1	NTRU_ EES677EP1	NTRU_ EES887EP1
256-bit symmetric / RSA-15360 / ECC-512 ("military strength")	NTRU_ EES743EP1	NTRU_ EES1499EP1	NTRU_ EES1087EP2	NTRU_ EES1171EP1

Table 1: Choosing NTRUEncrypt Parameter Sets

Table 2 gives the bandwidth and performance at each security level for NTRUEncrypt keys for the appropriate parameter sets, and for Elliptic Curve Cryptography (ECC) and RSA. Running times were obtained on one core of a two-core 2 GHz Intel Duo T7250 processor running Linux. The RSA and ECC benchmarks were obtained using Openssl with the GNU multi-precision library. Recommended NTRUEncrypt parameter sets for each security level are highlighted.

Parameter Set	Public key blob size (bytes)	DER-encoded public key size	Private key blob size (bytes)	Encrypt / Decrypt speed (ops/sec)	Equivalent security RSA		Equivalent security ECC	
					Key size	Encrypt/Decrypt ops/sec <sup>1</sup>	Key size	Encrypt / Decrypt ops/sec (ECIES)
<b>EES401EP2</b>	<b>557</b>	<b>591</b>	<b>607</b>	<b>10638/8064</b>	256 (2048 bits)	3050 / 109	28 (224 bits)	686 / 951
EES401EP1	557	591	638	4032/2392				
EES541EP1	749	783	858	5988/3703				
EES659EP1	912	946	1007	6060/3937				
<b>EES439EP1</b>	<b>609</b>	<b>643</b>	<b>659</b>	<b>9900/7299</b>	512 (4096 bits)	788 / 16	32 (256 bits)	439 / 650
EES449EP1	623	657	713	3278/1872				
EES613EP1	848	882	971	5050/3030				
EES761EP1	1052	1086	1157	5000/3174				
<b>EES593EP1</b>	<b>821</b>	<b>855</b>	<b>891</b>	<b>6849/4694</b>	960 (7680 bits)	219 / 5 <sup>2</sup>	48 (384 bits)	184 / 285
EES677EP1	936	970	1072	1984/1101				
EES887EP1	1225	1259	1403	2732/1540				
EES1087EP1	1500	1534	1674	2717/1600				
<b>EES743EP1</b>	<b>1027</b>	<b>1061</b>	<b>1120</b>	<b>5000/3215</b>	1920 (15360 bits)	60 / 12	64 (512 bits)	84 / 116
EES1087EP2	1500	1534	1718	1633/894				
EES1171EP1	1616	1650	1851	1709/937				
EES1499EP1	2067	2101	2285	1724/972				

Table 2: Performance of NTRUEncrypt Parameter Sets

<sup>1</sup> Uses public exponent = 65537; public exponent = 3 would be faster

<sup>2</sup> extrapolation from smaller key sizes

## Random Number Generation

### Deterministic Random Byte Generator (DRBG)

Before invoking NTRUEncrypt key-generation or encryption, you must instantiate a deterministic random byte generator (DRBG). The DRBG in this release implements the ANS X9.82 Part 3-2007 standard, using HMAC\_DRBG. The DRBG instantiation function returns a handle which you can pass to the key-generation and encryption functions.

When instantiating a DRBG, you specify a security level. This must be equal to or greater than the security level of the NTRUEncrypt parameter set that will be used. See Table 1 for the security levels associated with the various NTRUEncrypt parameter sets. You may have up to 4 DRBGs simultaneously, which allows for one at each of the security levels: 112 bits, 128 bits, 192 bits, and 256 bits. Once instantiated, a single DRBG may be used for all operations at a given security level.

### Seeding the DRBG: Entropy source

Any DRBG must be seeded, and the security of the DRBG depends on the randomness of the seed. The X9.82 DRBG included in this release obtains the seed using an *entropy function*. The caller must supply the entropy function, and pass a pointer to it to the DRBG instantiation function. The sample code in this release gives a sample entropy function.

The entropy function prototype is in `ntru_crypto_drbg.h`. It has two arguments: a one-byte command, and an address for one byte of data to be passed back. The function returns one byte of status, with 1 indicating success and 0 indicating failure. The three commands that must be implemented are `INIT`, `GET_NUM_BYTES_PER_BYTE_OF_ENTROPY`, and `GET_BYTE_OF_ENTROPY`.

The `INIT` command is used to perform whatever initialization the entropy function requires.

The `GET_NUM_BYTES_PER_BYTE_OF_ENTROPY` command passes back the number of bytes needed to obtain 8 bits of entropy; this value must be from 1 to 8 bytes. This value is dependent on the randomness of the entropy source. Note that the amount of entropy required to instantiate a DRBG is 1.5 times the DRBG's security level. For example, to instantiate a DRBG with 112-bit security requires 21 bytes of entropy.

The `GET_BYTE_OF_ENTROPY` command simply passes back a byte containing some entropy; the amount of entropy will be from 1 to 8 bits as indicated in the response to the `GET_NUM_BYTES_PER_BYTE_OF_ENTROPY` command.

The caller-supplied entropy function may be a true random source, an already properly-seeded pseudo-random number generator, or a seed that has been obtained from a random source. The sample code included in this release fixes a seed to show how the function works and to provide working sample code. It is obviously *not* an acceptable way to provide entropy for a real application.

The caller may supply a personalization string to provide data in addition to the entropy for the instantiation of a DRBG. The personalization string may be a maximum of 32 octets.