C2000 RSA Library

USER'S GUIDE

Contents

1. Introduction

This RSA library contains the implementation of RSASSA-PSS and the RSASSA-PKCS#1 v1.5 signature scheme.

The RSA library consists of the below parts:

- **RSASSA-PSS APIs** **:** Generation and verification of digital signatures
- **RSASSA-PKCS#1 v1.5** **:** Generation and verification of digital signatures
- **EMSA-PKCS#1 v1.5** **:** EMSA-PKCS1 encoding and decoding
- **EMSA-PSS APIs** **:** EMSA-PSS encoding and decoding
- **Yarrow PRNG APIs** **:** Pseudo-random generator for random salt generation

The library makes use of a Multiple Precision Integer math library adapted for C28 platform from the open source *libtommath* library. The corresponding files are *mpi.c* and *mpi.h*.

Example projects, *rsassapss_example* and *rsassapkcs1_v1_5* are provided along with the library with 2048 bit and 3072-bit message length examples that can act as a usage reference to the users.

NOTE: There is no theoretical upper bound on the modulus length to be used i.e., the libraries support modulus lengths that are greater than 3072 bits as well. However, to ensure correct operation, users should closely monitor the heap usage as the modulus length increases.

---

2. RSASSA-PSS APIs

2.1. rsa_err rsa_init_pubkey()

Prototype
rsa_err rsa_init_pubkey(rsa_PublicKey *pub_key)

Description
Initialize RSA Public Key data structure

Parameters
pub_key is the pointer to the rsa_PublicKey data structure

Returns
MP_OKAY if successful, MP_MEM if memory could not be allocated for the structure.

2.2. rsa_err rsa_init_pvtkey()

Prototype

rsa_err rsa_init_pvtkey(rsa_PrivateKey *pvt_key)

Description

Initialize RSA Private Key data structure

Parameters

pvt_key is the pointer to the rsa_PrivateKey data structure

Returns

MP_OKAY if successful, MP_MEM if memory could not be allocated for the structure.

2.3. rsa_err rsa_init_modulus()

Prototype

rsa_err rsa_init_modulus(rsa_Modulus *modulus)

Description

Initialize RSA Modulus data structure

Parameters

modulus is the pointer to the rsa_Modulus data structure

Returns

MP_OKAY if successful, MP_MEM if memory could not be allocated for the structure

2.4. rsa_err rsa_set_pubkey_from_hex()

Prototype

rsa_err rsa_set_pubkey_from_hex(rsa_PublicKey pub_key, unsigned char *pub_key_hex)

Description

Set RSA public key value in the data structure from a hex string

Parameters

pub_key is the pointer to the rsa_PublicKey data structure
pub_key_hex is the public key hex string (without the '0x' prefix)

Returns
MP_OKAY if successful. One of the failure macros defined in *mpi.h* depending on the type of failure.

2.5. rsa_err rsa_set_pvtkey_from_hex()

Prototype
rsa_err rsa_set_pvtkey_from_hex(rsa_PrivateKey *pvt_key, unsigned char *pvt_key_hex)

Description
Set RSA private key value in the data structure from a hex string

Parameters
pvt_key is the pointer to the rsa_PrivateKey data structure
pvt_key_hex is the private key hex string (without the '0x' prefix)

Returns
MP_OKAY if successful. One of the failure macros defined in *mpi.h* depending on the type of failure.

2.6. rsa_err rsa_set_modulus_from_hex()

Prototype
rsa_err rsa_set_modulus_from_hex(rsa_Modulus *modulus, unsigned char *modulus_hex)

Description
Set RSA modulus value in the data structure from a hex string

Parameters
modulus is the pointer to the rsa_Modulus data structure
modulus_hex is the modulus hex string (without the '0x' prefix)

Returns
MP_OKAY if successful. One of the failure macros defined in *mpi.h* depending on the type of failure.

2.7. rsa_err rsa_signature_to_mem()

Prototype
rsa_err rsa_signature_to_mem(rsa_Signature *sign, uint16_t *sign_str, uint16_t sign_len)

Description
Extract RSA signature value to a byte array

Parameters
sign is the pointer to the rsa_Signature data structure
sign_str is the pointer to the string where signature has to be written
sign_len is the length of the signature in bytes

Returns
MP_OKAY if successful. MP_MEM if there's a memory failure.

2.8. rsa_err rsa_hex_to_data()

Prototype
rsa_err rsa_hex_to_data(unsigned char *hex_str, uint16_t *data)

Description
Function to write data represented in a hex string into the memory bytewise

Parameters
hex_str is the hex string (without the '0x' prefix)
data is the pointer to the memory location where the data will be written

Returns
0 if successful. 3 if hex string is invalid.

2.9. rsassa_pss_sign()

Prototype
```
rsa_err rsassa_pss_sign(uint16_t *hash, uint16_t hashlen, uint16_t saltlen,
                rsa_PrivateKey *key_pvt, rsa_Modulus *modulus,
                uint16_t modulus_bitlen, yarrow_prng *prng,
                rsa_Signature *sign)
```

Description
Function to generate a signature of the data using RSASSA-PSS algorithm

Parameters
hash is the pointer to the bytewise hash digest of the message

hashlen is the length of the hash digest of the message in bytes
saltlen is the length of the random salt in bytes
key_pvt is the pointer to the rsa_PrivateKey data structure holding the RSA private key value
modulus is the pointer to the rsa_Modulus data structure holding the RSA modulus value
modulus_bitlen is the length of the RSA modulus in bits (not bytes)
prng is the pointer to the yarrow prng handler to generate random salt
sign is the pointer to the rsa_Signature data structure which will hold the generated the signature

Returns

Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pss.h* depending on the type of failure.

2.10.rsassa_pss_verify()

Prototype

```
rsa_err rsassa_pss_verify( rsa_Signature *sign, uint16_t *hash, uint16_t hashlen,
                    uint16_t saltlen, rsa_PublicKey *key_pub,
                    rsa_Modulus *modulus, uint16_t modulus_bitlen,
                    int16_t *result)
```

Description

Function to verify the signature of a data using RSASSA-PSS algorithm

Parameters

sign is the pointer to the rsa_Signature data structure holding the signature to be verified
hash is the pointer to the bytewise hash digest of the message
hashlen is the length of the hash digest of the message in bytes
saltlen is the length of the random salt in bytes
key_pub is the pointer to the rsa_PublicKey data structure holding the RSA public key value
modulus is the pointer to the rsa_Modulus data structure holding the RSA modulus value
modulus_bitlen is the length of the RSA modulus in bits (not bytes)
result is the pointer to the verification result

- 1 - Verification successful
- 0 - Incorrect signature. Verification failed
- -1 - Verification incomplete

Returns

Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pss.h* depending on the type of failure.

3. RSASSA-PKCS#1 v1.5 APIs

3.1. rsassa_pkcs1_v1_5_sign ()

Prototype

```
rsa_err rsassa_pkcs1_v1_5_sign(rsa_PrivateKey* key_pvt, rsa_Modulus* modulus,
   rsa_Signature* sign, uint16_t *hash, uint16_t hash_size)
```

Description

Function to generate a signature of the data using RSASSA- PKCS#1 v1.5 algorithm

Parameters

Sign is the pointer to the rsa_Signature data structure which will hold the generated the signature
hash is the pointer to the byte wise hash digest of the message
hash_size is the size of the hash based on the SHA encoding type (32 for SHA256 and 48 for SHA384)
key_pub is the pointer to the rsa_PublicKey data structure holding the RSA private key value
modulus is the pointer to the rsa_Modulus data structure holding the RSA modulus value
result is the pointer to the verification result with the below possible values:

1 - *Verification successful*
0 - *Incorrect signature. Verification failed*
-1 - *Verification incomplete*

Returns

Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pkcs1_v1_5.h* depending on the type of failure.

3.2. rsassa_pkcs1_v1_5_verify ()

Prototype

```
rsa_err rsassa_pkcs1_v1_5_verify(rsa_Signature* sign, uint16_t* hash, uint16_t hash_size,
   uint16_t mod_len, rsa_PublicKey* key_pub, rsa_Modulus* modulus, int16_t* result)
```

Description

Function to verify a signature of the data using RSASSA- PKCS#1 v1.5 algorithm

Parameters

key_pvt is the pointer to the rsa_PrivateKey data structure holding the RSA private key value
modulus is the pointer to the rsa_Modulus data structure holding the RSA modulus value
sign is the pointer to the rsa_Signature data structure which will hold the generated the signature
hash is the pointer to the byte wise hash digest of the message
hash_size is the size of the hash based on the SHA encoding type

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pkcs1_v1_5.h* depending on the type of failure.

---

4. EMSA-PKCS#1 v1.5 APIs

4.1. emsa_pkcs1_v1_5_encode ()

Prototype
int emsa_pkcs1_v1_5_encode(uint16_t* emsa_out, uint16_t* digest, uint16_t hash_size)

Description
EMSA-PKCS#1 v1.5 encode message

Parameters
emsa_out is the pointer to hold the encoded data
emsa_outlen is the output length of the encoded data
digest is the serialized hash data based on the SHA encoding type
hash_size is the length of the hash

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pkcs1_v1_5.h* depending on the type of failure.

4.2. emsa_pkcs1_v1_5_decode ()

Prototype
int emsa_pkcs1_v1_5_decode(uint16_t* EM_Str, uint16_t *hash, uint16_t hash_size, int16_t* result)

Description
EMSA-PKCS#1 v1.5 decode message

Parameters
EM_Str is the pointer to the EMSA padding bytes
hash is the pointer to the hash to be compared (serialized digest)
hash_size is the size of the hash based on the SHA encoding type
result is a pointer to the variable to hold the verification result
1 - Verification successful

0 - Incorrect signature. Verification failed
-1 - Verification incomplete

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pkcs1_v1_5.h* depending on the type of failure.

---

5. EMSA-PSS APIs

5.1. pkcs_1_mgf()

Prototype
int pkcs_1_mgf(uint16_t *seed, int16_t seedlen, uint16_t *mask, int16_t masklen);

Description
PKCS1_MGF mask generation using SHA-256

Parameters
seed is the pointer to the input seed
seedlen is the length of the seed in bytes
mask is the pointer to the output mask
masklen is the length of the output mask in bytes

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pss.h* depending on the type of failure.

5.2. emsa_pss_encode()

Prototype
```
int emsa_pss_encode(uint16_t *msghash, uint16_t msghashlen, uint16_t saltlen,
                    uint16_t modulus_bitlen, yarrow_prng *prng,
                    uint16_t *emsa_out, uint16_t emsa_outlen)
```

Description
EMSA-PSS encode message

Parameters

msghash is the pointer to the bytewise hash digest of the message
hashlen is the length of the hash digest of the message in bytes
saltlen is the length of the random salt in bytes
modulus_bitlen is the length of the RSA modulus in bits (not bytes)
prng is the pointer to the yarrow prng handler to generate random salt
emsa_out is the pointer to hold the encoded data
emsa_outlen is the output length of the encoded data

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pss.h* depending on the type of failure.

5.3. emsa_pss_decode()

Prototype
int emsa_pss_decode(uint16_t *sig, uint16_t siglen, uint16_t *msghash, uint16_t msghashlen,
    uint16_t saltlen, uint16_t modulus_bitlen, int16_t *res)

Description
EMSA-PSS decode message

Parameters
sig is the pointer to the message encoding to be decoded
siglen is the length in bytes of the message encoding to be decoded
msghash is the pointer to the bytewise hash digest of the message
hashlen is the length of the hash digest of the message in bytes
saltlen is the length of the random salt in bytes
modulus_bitlen is the length of the RSA modulus in bits (not bytes)
res is a pointer to the variable to hold the verification result
    1 - Verification successful
    0 - Incorrect signature. Verification failed
    -1 - Verification incomplete

Returns
Returns PKCS_STATUS_SUCCESS if successful. One of the PKCS failure macros defined in *emsa_pss.h* depending on the type of failure.

6. Yarrow PRNG APIs

6.1. yarrow_init_with_nounce()

Prototype

int yarrow_init_with_nounce(yarrow_prng *prng, const uc *nounceCounter)

Description

Initialize yarrow handle with nounce counter values

Parameters

prng is the pointer to the yarrow PRNG handle
nounceCounter is the pointer to the array which contains the counter values to be initialized with

Returns

Returns YARROW_STATUS_SUCCESS if successful. YARROW_STATUS_ERROR_NULL if argument is invalid.

6.2. yarrow_add_entropy()

Prototype

int yarrow_add_entropy(uc *in, const uint32_t inlen, yarrow_prng *prng)

Description

Introduce entropy in the algorithm to generate pseudo-random values

Parameters

in is the random input string which will be used to introduce entropy
inlen is the length of the random input string
prng is the pointer to the yarrow PRNG handle

Returns

Returns YARROW_STATUS_SUCCESS if successful. YARROW_STATUS_ERROR_NULL if argument is invalid.

6.3. yarrow_ready()

Prototype

int yarrow_ready(yarrow_prng *prng)

Description
Prepare the yarrow PRNG handle to generate random values

Parameters
prng is the pointer to the yarrow PRNG handle

Returns
Returns YARROW_STATUS_SUCCESS if successful. YARROW_STATUS_ERROR_NULL if argument is invalid.

6.4. yarrow_read()

Prototype
int yarrow_read(yarrow_prng *prng, uc *out, uc outlen)

Description
Read random values using yarrow PRNG

Parameters
prng is the pointer to the yarrow PRNG handle
out is the output string of random values
outlen is the length of the output string of random values

Returns
Returns *outlen* if successful. 0 if failure.

---

7. Examples

7.1. rsassapss_example project

This example project is provided along with the RSA library and acts as a reference to the users on how we use the APIs within the library to generate and verify digital signatures using the RSASSA-PSS signature scheme.

This project contains:

1.) RSASSA-PSS 2048 bit example
2.) RSASSA-PSS 3072 bit example

Note: Predefined symbol 'RSA_2048' or 'RSA_3072' must be defined by the user to choose between the 2048 bit and 3072 bit examples within the project.

Example Flows:

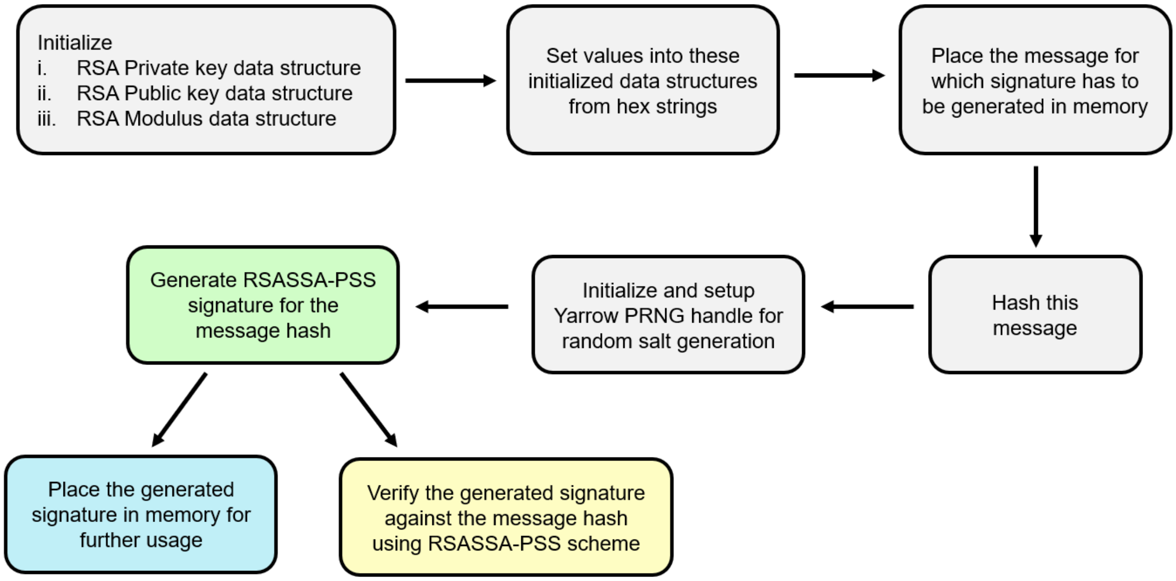

Figure 1: RSASSA-PSS Example Flow

7.2. rsassapkcs1_v1_5_example project

This example project is provided along with the RSA library and acts as a reference to the users on how we use the APIs within the library to generate and verify digital signatures using the RSASSA- PKCS#1 v1.5 signature scheme.

This project contains:

3.) RSASSA-PKCS#1 v1.5 2048 bit example
4.) RSASSA-PKCS#1 v1.5 3072 bit example

Note: Predefined symbol 'RSA_2048' or 'RSA_3072' must be defined by the user to choose between the 2048 bit and 3072 bit examples within the project.

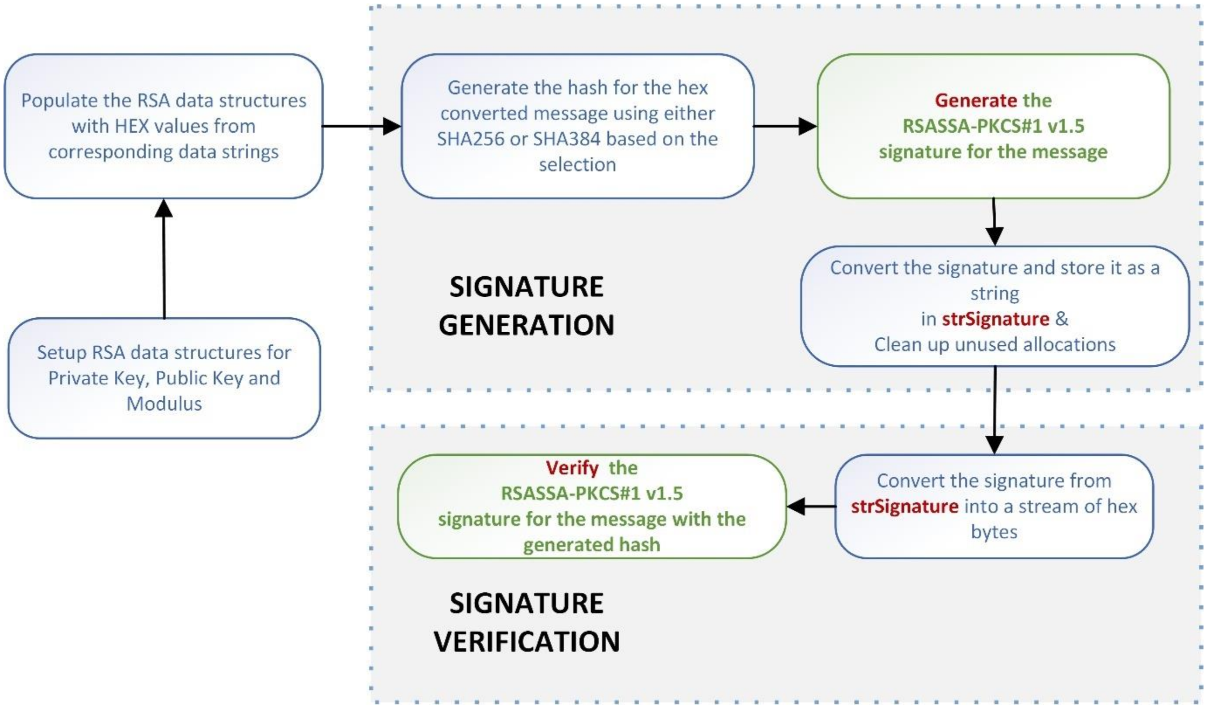

Figure 2: RSASSA-PKCS#1 v1.5 Example Flow

8. Performance and memory details

8.1. RSASSA-PSS

The table below lists the performance numbers of the library when run in an F28004x device at 100MHz.

DIGITAL SIGNATURE GENERATION (Execution time in seconds)						
Message length	Optimization Level					
	-Ooff	-O0	-O1	-O2	-O3	-O4
2048 bit	70.4	50.4	52.3	28.4	29.8	26.9
3072 bit	227.2	163.9	169.7	84.3	93.4	87.2

DIGITAL SIGNATURE VERIFICATION (Execution time in seconds)						
Message length	Optimization level					
	-Ooff	-O0	-O1	-O2	-O3	-O4
2048 bit	0.83	0.58	0.60	0.31	0.34	0.33
3072 bit	1.59	1.10	1.16	0.59	0.65	0.62

Note: The above measurements were made at an –opt_for_speed=5 setting

The table below lists the code size of the files that are included in the library.

module	Code size in 16-bit words	Data size in 16-bit words
mpi	3384	0
rsassa_pss	311	0
emsa_pss	1172	0
yarrow	176	84
aes_common	618	517
aes_ctr	266	0
sha_256	710	128

Note: The above code sizes are at an –opt_for_speed=5, --opt_level=4 setting

The table below lists the heap and stack usage of the library.

	Heap usage in 16-bit words	Stack usage in 16-bit words
RSASSA-PSS 2048 bit	1850	232
RSASSA-PSS 3072 bit	2651	232

Note: The above heap and stack usage measurements are at an –opt_for_speed=5, --opt_level=4 setting

8.2. RSASSA-PKCS#1 v1.5

The table below lists the performance numbers of the library when run in an F28P65x device at 100MHz.

RSASSA-PKCS1 v1.5 SIGNATURE GENERATION (execution time in seconds)						
Message length and hash type	Optimization level					
	-Ooff	-O0	-O1	-O2	-O3	-O4
2048 bit (256 bit hash)	25.29	3.28	3.52	29.35	30.21	28.97
3072 bit (256 bit hash)	13.41	25.06	26.03	10.23	13.22	9.37
2048 bit (384 bit hash)	28.95	3.44	2.69	32.5	28.82	29
3072 bit (384 bit hash)	25.93	25.59	23.15	21.38	8.69	9.41

RSASSA-PKCS1 v1.5 SIGNATURE VERIFICATION (execution time in seconds)						
Message length and hash type	Optimization level					
	-Ooff	-O0	-O1	-O2	-O3	-O4
2048 bit (256 bit hash)	0.82	0.56	0.57	0.37	0.38	0.36
3072 bit (256 bit hash)	1.57	1.07	1.08	0.69	0.71	0.68
2048 bit (384 bit hash)	0.86	0.56	0.56	0.4	0.36	0.36
3072 bit (384 bit hash)	1.65	1.08	1.07	0.76	0.68	0.69

Note: The above measurements were made at an –opt_for_speed=5 setting

The table below lists the code size of the files that are included in the library.

module	Code size in 16-bit words	RO data size in 16-bit words
mpi	3384	0
rsassa_pkcs1_v1_5	268	0
emsa_pkcs1_v1_5	0	60
sha_256	1263	128
sha_384	1274	320

Note: The above code sizes are at an –opt_for_speed=5, --opt_level=4 setting

The table below lists the heap and stack usage of the library.

	Heap usage in 16-bit words	Stack usage in 16-bit words
2048 bits (384-bit hash)	2835	768 bytes
3072 bits (384-bit hash)	3691	768 bytes

Note: The above heap and stack usage measurements are at an –opt_for_speed=5, --opt_level=4 setting