QDL’s Crypto Module

Introduction

This is QDL’s module for RSA and symmetric key cryptography. It lets you create RSA and symmetric keys and encrypt or decrypt them. It also allows for importing and exporting JSON webkeys.

Function reference

export_jwks

Description

Export a key or set of keys to RFC 7517 format and write to storage.

Usage

```
export_jwks(keys., file_path)
```

Arguments

keys. - either a single key or a set of keys.

file_path - the fully qualified path for the output.

Output

This returns true if the operation successfully wrote the file.

Examples

```
  kk. := rsa_create_key(1024, 3)
  export_jwks(kk., '/tmp/keys.jwk')
true
```

This means that the set of keys was written in the correct format to the given file.

import_jwks

Description

Read a JSON webkey (as per RFC 7517)

QDL's Crypto Module

Usage

```
import_jwks(file_path)
```

Arguments

file_path - the fully qualified path to the file.

Output

This returns a stem of of keys.

Examples

```
   keys. := import_jwks('/tmp/keys.jwk');
```

Since there were no errors, the set of keys in RFC 7517 format was successfully imported and converted to a stem. Note that if there was one single key in the file, a single key would result.

rsa_create_key

Description

Create and RSA key either with the default size of 1024 bits or a custom size (that is larger than 1024 bits and a multiple of 256). Optionally, create several. Note that the alg returned in the key set is for compliance with RFC 7515 and is always set to RS256. This can be changed to another algorithm since it is intended to inform consumers of this key which algorithm to use.

Usage

```
rsa_create_key()
rsa_create_key(key_size)
rsa_create_key(key_size, count)
```

Arguments

key_size - the size of the key to create. 1024 < key_size and key_size&256 == 0.

count - the number of such keys to create

Output

The single key or a stem of keys. The keys of the stem are the key id (kid) field.

QDL's Crypto Module

Examples

```
 key. := rsa_create_key(4096); // create 4096 bit key pair
```

Note the depending on your system, this may take a while.

```
  key. := rsa_create_key(say(256*64))
16384
```

This dutifully reports it is creating a an RSA key that is 16,384 bits long. Note that this will probably take some time (several minutes at least) on most systems.

rsa_decrypt

Description

Decrypt the string or stem of strings using the key. By default, the public key is used but you may specify to use the private key if available.

Usage

```
  rsa_decrypt(key., arg|arg.)
  rsa_decrypt(key., arg|arg., use_private)
```

Arguments

key. - the key to use

arg | arg. - either a string or stem of strings.

use_private - use the private key for this if available (or fail if no private key)

Output

The decrypted string or stem of strings.

Examples

In this example, create a key with 2048 bits, encrypt a stem of strings, then decrypt it.

```
   key. := rsa_create_key(2048); // create 2048 bit key pair
   rsa_decrypt(key., rsa_encrypt(key., ['a','b']))
[a,b]
```

QDL's Crypto Module

rsa_encrypt

Description

encrypt a string or stem of them with the private key if use_private is true

Usage

```
rsa_encrypt(key., arg|arg.)
rsa_encrypt(key., arg|arg., use_private)
```

Arguments

key. - the key to use.

arg | arg. - the string or stem of strings to encrypt

use_private - (default is true) use the private key for this operation.

Note that as per the specification any string must be smaller than the key length, so if the key length is 1024, you are restricted to a string of length 1024=128 characters.

Output

The encrypted string or stem of strings.

Examples

Create a key, encrypt a string.

```
  key. := rsa_create_key(2048); // create 2048 bit key pair
  rsa_encrypt(key., 'the quick brown fox')
PdiGsfQU3dHxf8I3ozFPIta-
AsKg61lVCmX5oi4QziyBItLAlKXJ2Oi3TV1Wg69KbwCJk5UifohpWfhNYK203dEubOcEzOcH86cS_VsYFZf
fLpSBim0KLIc60lOieJbMMxtjYAfAJyKt-
wA6u7gi7QmHxOntELeWeTnJlfugyzc8Dtb_BoiLX83YxygLmrGAZYNjFjzaOt_UQyODwrAZpWb_7o61zRW_
IN5mNSyCBn-RHS5_r_621fa9p6G8NR7XlFH1a2ltx5G5Hep4HbhWwALJHapfjuZea9hFfwbaaVt-
8rkbEhTcARrio8h6HkO7V6MR7LiqWIUrfuLMMHd7sw
```

rsa_public_key

Description

get the public part(s) of the key(s)

Usage

```
 rsa_public_key(key.)
```

QDL’s Crypto Module

Arguments

key. - either a key or a stem of them.

Output

A public key or (if the argument was a stem of keys) a stem of public keys.

Examples

```
   key. := rsa_create_key(2048); // create 2048 bit key pair
   rsa_public_key(key.)
{
 kty:RSA,
 e:AQAB,
 use:sig,
 kid:GoZzh2pbLE4,
 alg:RS256,
 n:AITR9WBryOy-
MF02UowhDrqsDvHS9un0DKbqoMv3Brq3no3s0Cr5OO7D1OXw3z6Rtn6fqh7NeFb3XlHpJ1mv5xc1apQbC81
m8LrCHq_QaK60hJQ5Jz5ZrD-6kjv9dt8-qBwI4lgcWzu_gsbus8G7-O5EEXKOcGdCPj-094xKMO-
eunSfK_xngdFGAa-
ukFtJGd9FLSm0Nro08vrV7NV3JQBa_ipswIbaWS3zBaImzSgzglbfkrpu5zeSwgOhoqiMKNFS8qbVxUtynK
JVlWpbwU_a42PvFvMQu7PEdPJAbzvl8P1uuFzNeGjniwE1y6pG1v77v2s-bhaDKKLUiZ9dmFM
}
```

s_decrypt

Description

Perform symmetric decryption on the argument.

Usage

```
s_decrypt(key, target | target.)
```

Arguments

key - a base 64 encoded bit string.

target|target. - a string or stem of strings to decrypt.

Output

The decrypted string or stem of them.

QDL's Crypto Module

Examples

Given the key in the s_create_key section, unencrypt the example in the s_encrypt example.

```
  s_decrypt(k, 'Ef7BDvfdiz5lzTlDt5rmtdFemA')
the quick brown fox
```

s_encrypt

Description

Symmetrically encrypt the argument using the key.

Usage

```
  s_encrypt(key, target|target.)
```

Arguments

key - The base 64 encoded bit string to use for the key. Easily created with the random_string function.

target|target. - either a string or stem of them to encrypt

Output

The encrypted string or stem of encrypted strings.

Examples

```
 k := random_string(64); // (512 bits)/8 = 64 bytes
 s_encrypt(k, 'the quick brown fox')
Ef7BDvfdiz5lzTlDt5rmtdFemA
```

Note that this will vary of you do it, since the key is truly random. See the s_decrypt example with this key to verify yours works.