PSKC Library (libpskc) Manual

Simon Josefsson

COLLABORATORS			
	TITLE : PSKC Library (libpskc) Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Simon Josefsson	January 11, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Abstract

The Portable Symmetric Key Container (PSKC) format is used to transport and provision symmetric keys to cryptographic devices or software. The PSKC Library (libpskc) allows you to parse, validate and generate PSKC data. This manual documents the interfaces of PSKC library and contains a tutorial to get you started working with the library and the command-line tool (pskctool).

Part I

PSKC Tutorial

Chapter 1

PSKC Introduction

The Portable Symmetric Key Container (PSKC) format is used to transport and provision symmetric keys to cryptographic devices or software. The PSKC Library allows you to parse, validate and generate PSKC data. The PSKC Library is written in C, uses LibXML, and is licensed under LGPLv2+. A companion to the library is a command line tool (pskctool) to interactively manipulate PSKC data.

To get a feeling of the PSKC data format we show the shortest possible valid PSKC content.

```
<?xml version="1.0"?>
<KeyContainer xmlns="urn:ietf:params:xml:ns:keyprov:pskc" Version="1.0">
  <KeyPackage/>
</KeyContainer>
```

Of course, since the intent with PSKC is to transport cryptographic keys, the example above is of little use since it does not carry any keys. The next example is more realistic, and show PSKC data used to transport a key used for a OATH HOTP implementation. The interesting values are the DeviceInfo values to identify the intended hardware, the Key Id "12345678", and the base64-encoded shared secret "MTIzNA==".

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
        xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
  <KeyPackage>
    <DeviceInfo>
      <Manufacturer>Manufacturer</Manufacturer>
      <SerialNo>987654321</SerialNo>
    </DeviceInfo>
    <Key Id="12345678"
        Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
      <AlgorithmParameters>
        <ResponseFormat Length="8" Encoding="DECIMAL"/>
      </AlgorithmParameters>
      <Data>
        <Secret>
          <PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
          </PlainValue>
        </Secret>
        <Counter>
          <PlainValue>0</PlainValue>
        </Counter>
      </Data>
    </Key>
  </KeyPackage>
</KeyContainer>
```

Chapter 2

Library libpskc

To illustrate how the library works, let's give an example on how to parse the data above and print the device serial number (SerialNo field). The code below is complete and working but performs minimal error checking.

```c
#include <stdio.h>
#include <pskc/pskc.h>

/*
 * $ cc -o serialno serialno.c $(pkg-config --cflags --libs libpskc)
 * $ ./serialno pskc-hotp.xml
 * SerialNo: 987654321
 * $
 */

#define PSKC_CHECK_RC             \
  if (rc != PSKC_OK) {            \
    printf ("%s (%d): %s\n", pskc_strerror_name (rc),     \
      rc, pskc_strerror (rc));          \
    return 1;              \
  }

int
main (int argc, const char *argv[])
{
  char buffer[4096];
  FILE *fh;
  size_t len;
  pskc_t *container;
  pskc_key_t *keypackage;
  int rc;

  if (argc != 2)
    {
      printf ("Usage: %s <PSKCFILE>\n", argv[0]);
      return 1;
    }
  fh = fopen (argv[1], "r");
  if (!fh)
    {
      perror ("fopen");
      return 1;
    }
  len = fread (buffer, 1, sizeof (buffer), fh);
  fclose (fh);
```

```
  rc = pskc_global_init ();
  PSKC_CHECK_RC;

  rc = pskc_init (&container);
  PSKC_CHECK_RC;
  rc = pskc_parse_from_memory (container, len, buffer);
  PSKC_CHECK_RC;

  keypackage = pskc_get_keypackage (container, 0);

  if (keypackage)
    printf ("SerialNo: %s\n", pskc_get_device_serialno (keypackage));

  pskc_done (container);
  pskc_global_done ();
}
```

Compiling and linking code with the PSKC Library requires that you specify correct compilation flags so that the header include file and the shared library is found. There is only one include file and it should be used like #include <pskc/pskc.h>. The library is called libpskc.so on GNU systems and libpskc.dll on Windows systems. To build the previous file, assuming the code is stored in a file called "serialno.c", invoke the following command.

```
cc -o serialno serialno.c -I/path/to/pskc/include/path -L/path/to/pskc/lib/path -Wl,-rpath/ ←
    path/to/pskc/lib/path -lpskc
```

A pkg-config file is provided, so that you may use pkg-config to select proper compilation flags if you want.

```
cc -o serialno serialno.c $(pkg-config --cflags --libs libpskc)
```

After building the tool you would invoke it passing the name of the file with the PSKC input above, and it will print the serial number.

```
jas@latte:~$ ./serialno pskc.xml
SerialNo: 987654321
jas@latte:~$
```

2.1 Converting PSKC data to CSV format

We conclude with a larger example illustrating how to read a PSKC file, parse it and print a human readable summary of the PSKC data to stderr, validate it against the PSKC XML Schema (this is normally not needed) and print the validation outcome to stderr, and iterate through all keys in the file and print to stdout a comma-separated list with three fields: the key id, the device serialno, and the hex encoded cryptographic key. This code example check error codes and releases resources.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <pskc/pskc.h>

/*
 * $ cc -o pskc2csv pskc2csv.c $(pkg-config --cflags --libs libpskc)
 * $ ./pskc2csv pskc.xml 2> /dev/null
 * 12345678,12345678,MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
 * $
 */

int
main (int argc, const char *argv[])
```

```
{
  struct stat st;
  FILE *fh = NULL;
  char *buffer = NULL, *out;
  size_t i;
  pskc_t *container = NULL;
  pskc_key_t *keypackage;
  int exit_code = EXIT_FAILURE, rc, isvalid;

  rc = pskc_global_init ();
  if (rc != PSKC_OK)
    {
      fprintf (stderr, "pskc_global_init: %s\n", pskc_strerror (rc));
      goto done;
    }

  if (argc != 2)
    {
      fprintf (stderr, "Usage: %s PSKCFILE\n", argv[0]);
      goto done;
    }

  /* Part 1: Read file. */

  fh = fopen (argv[1], "r");
  if (fh == NULL)
    {
      perror ("fopen");
      goto done;
    }

  if (fstat (fileno (fh), &st) != 0)
    {
      perror ("fstat");
      goto done;
    }

  buffer = malloc (st.st_size);
  if (buffer == NULL)
    {
      perror ("malloc");
      goto done;
    }

  i = fread (buffer, 1, st.st_size, fh);
  if (i != (size_t) st.st_size)
    {
      fprintf (stderr, "short read\n");
      goto done;
    }

  /* Part 2: Parse PSKC data. */

  rc = pskc_init (&container);
  if (rc != PSKC_OK)
    {
      fprintf (stderr, "pskc_init: %s\n", pskc_strerror (rc));
      goto done;
    }

  rc = pskc_parse_from_memory (container, i, buffer);
  if (rc != PSKC_OK)
```

```
    {
      fprintf (stderr, "pskc_parse_from_memory: %s\n", pskc_strerror (rc));
      goto done;
    }

  /* Part 3: Output human readable variant of PSKC data to stderr. */

  rc = pskc_output (container, PSKC_OUTPUT_HUMAN_COMPLETE, &out, &i);
  if (rc != PSKC_OK)
    {
      fprintf (stderr, "pskc_output: %s\n", pskc_strerror (rc));
      goto done;
    }

  fprintf (stderr, "%.*s\n", (int) i, out);

  pskc_free (out);

  /* Part 4: Validate PSKC data. */

  rc = pskc_validate (container, &isvalid);
  if (rc != PSKC_OK)
    {
      fprintf (stderr, "pskc_validate: %s\n", pskc_strerror (rc));
      goto done;
    }

  fprintf (stderr, "PSKC data is Schema valid: %s\n", isvalid ? "YES" : "NO");

  /* Part 5: Iterate through keypackages and print key id, device
     serial number and base64 encoded secret. */

  for (i = 0; (keypackage = pskc_get_keypackage (container, i)); i++)
    {
      const char *key_id = pskc_get_key_id (keypackage);
      const char *device_serialno = pskc_get_key_id (keypackage);
      const char *b64secret = pskc_get_key_data_b64secret (keypackage);

      printf ("%s,%s,%s\n", key_id ? key_id : "",
        device_serialno ? device_serialno : "",
        b64secret ? b64secret : "");
    }

  exit_code = EXIT_SUCCESS;

done:
  pskc_done (container);
  if (fh && fclose (fh) != 0)
    perror ("fclose");
  free (buffer);
  pskc_global_done ();
  exit (exit_code);
}
```

Below we'll illustrate how to build the tool and run it on the same PSKC data as above. The tool prints different things to stdout and stderr, which you can see below.

```
jas@latte:~$ cc -o pskc2csv pskc2csv.c $(pkg-config --cflags --libs libpskc)
jas@latte:~$ ./pskc2csv pskc.xml 2> /dev/null
12345678,12345678,MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
jas@latte:~$ ./pskc2csv pskc.xml > /dev/null
Portable Symmetric Key Container (PSKC):
```

```
  Version: 1.0
  KeyPackage 0:
    DeviceInfo:
      Manufacturer: Manufacturer
      SerialNo: 987654321
    Key:
      Id: 12345678
      Algorithm: urn:ietf:params:xml:ns:keyprov:pskc:hotp
      Key Secret (base64): MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
      Key Counter: 0
      Response Format Length: 8
      Response Format Encoding: DECIMAL

PSKC data is Schema valid: YES
jas@latte:~$
```

2.2 Digitally sign PSKC data

The library can also digitally sign PSKC data using a X.509 private key and certificate, both stored in files. Below is a minimal example illustring how to read a PSKC file, digitally sign it and then print the signed XML to stdout.

```c
#include <stdio.h>
#include <pskc/pskc.h>

/*
 * $ cc -o pskcsign pskcsign.c $(pkg-config --cflags --libs libpskc)
 * $ ./pskcsign pskc-hotp.xml pskc-ee-key.pem pskc-ee-crt.pem > signed.xml
 */

#define PSKC_CHECK_RC               \
  if (rc != PSKC_OK) {              \
    printf ("%s (%d): %s\n", pskc_strerror_name (rc),     \
      rc, pskc_strerror (rc));           \
    return 1;               \
  }

int
main (int argc, const char *argv[])
{
  char buffer[4096];
  FILE *fh;
  size_t len;
  pskc_t *container;
  char *out;
  int rc;

  if (argc != 4)
    {
      printf ("Usage: %s <PSKCFILE> <X509KEY> <X509CERT>\n", argv[0]);
      return 1;
    }
  fh = fopen (argv[1], "r");
  if (!fh)
    {
      perror ("fopen");
      return 1;
    }
  len = fread (buffer, 1, sizeof (buffer), fh);
  fclose (fh);
```

```
  rc = pskc_global_init ();
  PSKC_CHECK_RC;

  rc = pskc_init (&container);
  PSKC_CHECK_RC;
  rc = pskc_parse_from_memory (container, len, buffer);
  PSKC_CHECK_RC;

  rc = pskc_sign_x509 (container, argv[2], argv[3]);
  PSKC_CHECK_RC;

  rc = pskc_output (container, PSKC_OUTPUT_XML, &out, &len);
  PSKC_CHECK_RC;
  fwrite (out, 1, len, stdout);
  pskc_free (out);

  pskc_done (container);
  pskc_global_done ();

  return 0;
}
```

You would compile and use the example like this.

```
jas@latte:~$ cc -o pskcsign pskcsign.c $(pkg-config --cflags --libs libpskc)
jas@latte:~$ ./pskcsign pskc-hotp.xml pskc-ee-key.pem pskc-ee-crt.pem > signed.xml
jas@latte:~$
```

The next section illustrate how to verify the content of "signed.xml". For more background and information on how to generate the necessary private key and certificates, see the "pskctool" command line tool documentation.

2.3 Verify signed PSKC data

To verify XML digital signatures in PSKC data, you may use the pskc_verify_x509crt function.

```
#include <stdio.h>
#include <pskc/pskc.h>

/*
 * $ cc -o pskcverify pskcverify.c $(pkg-config --cflags --libs libpskc)
 * $ ./pskcverify signed.xml pskc-root-crt.pem
 * OK
 * $
 */

#define PSKC_CHECK_RC                  \
  if (rc != PSKC_OK) {                 \
    printf ("%s (%d): %s\n", pskc_strerror_name (rc),    \
      rc, pskc_strerror (rc));          \
    return 1;               \
  }

int
main (int argc, const char *argv[])
{
  char buffer[4096];
  FILE *fh;
  size_t len;
  pskc_t *container;
```

```
  int rc, valid_sig;

  if (argc != 3)
    {
      printf ("Usage: %s <PSKCFILE> <X509CERT>\n", argv[0]);
      return 1;
    }
  fh = fopen (argv[1], "r");
  if (!fh)
    {
      perror ("fopen");
      return 1;
    }
  len = fread (buffer, 1, sizeof (buffer), fh);
  fclose (fh);

  rc = pskc_global_init ();
  PSKC_CHECK_RC;

  rc = pskc_init (&container);
  PSKC_CHECK_RC;
  rc = pskc_parse_from_memory (container, len, buffer);
  PSKC_CHECK_RC;

  rc = pskc_verify_x509crt (container, argv[2], &valid_sig);
  PSKC_CHECK_RC;
  puts (valid_sig ? "OK" : "FAIL");

  pskc_done (container);
  pskc_global_done ();
}
```

You would compile and use the example like this.

```
jas@latte:~$ cc -o pskcverify pskcverify.c $(pkg-config --cflags --libs libpskc)
jas@latte:~$ ./pskcverify signed.xml pskc-root-crt.pem
OK
jas@latte:~$
```

For more background and information on how to generate the necessary private key and certificates, see the "pskctool" command line tool documentation.

2.4   Create PSKC data

To create PSKC data you should first get a handle to a container using pskc_init. Add one or more keypackages to the container using pskc_add_keypackage. For each keypackage, set the relevant values you want using the "pskc_set_*" functions, for example pskc_set_device_serialno.

The XML output is created as usual with pskc_build_xml.

Here follows an example that would generate PSKC data that could be used to personalize an imaginary HOTP token.

```
#include <stdio.h>
#include <pskc/pskc.h>

/*
 * $ cc -o pskccreate pskccreate.c $(pkg-config --cflags --libs libpskc)
 * $ ./pskccreate
 */
```

```
#define PSKC_CHECK_RC                      \
  if (rc != PSKC_OK) {                     \
    printf ("%s (%d): %s\n", pskc_strerror_name (rc),      \
      rc, pskc_strerror (rc));             \
    return 1;                   \
  }

int
main (void)
{
  size_t len;
  pskc_t *container;
  pskc_key_t *keypackage;
  char *out;
  int rc;

  rc = pskc_global_init ();
  PSKC_CHECK_RC;
  rc = pskc_init (&container);
  PSKC_CHECK_RC;

  rc = pskc_add_keypackage (container, &keypackage);
  PSKC_CHECK_RC;

  pskc_set_device_manufacturer (keypackage, "Acme");
  pskc_set_device_serialno (keypackage, "42");

  pskc_set_key_id (keypackage, "4711");
  pskc_set_key_algorithm (keypackage,
        "urn:ietf:params:xml:ns:keyprov:pskc:hotp");

  pskc_set_key_algparm_resp_encoding (keypackage, PSKC_VALUEFORMAT_DECIMAL);
  pskc_set_key_algparm_resp_length (keypackage, 8);

  pskc_set_key_data_counter (keypackage, 42);

  rc = pskc_set_key_data_b64secret (keypackage, "Zm9v");
  PSKC_CHECK_RC;

  rc = pskc_build_xml (container, &out, &len);
  PSKC_CHECK_RC;
  fwrite (out, 1, len, stdout);
  pskc_free (out);

  pskc_done (container);
  pskc_global_done ();

  return 0;
}
```

You would compile and use the example like this.

```
jas@latte:~$ cc -o pskccreate pskccreate.c $(pkg-config --cflags --libs libpskc)
jas@latte:~$ ./pskccreate
<?xml version="1.0"?>
<KeyContainer xmlns="urn:ietf:params:xml:ns:keyprov:pskc" Version="1.0"><KeyPackage>< ←
   DeviceInfo><Manufacturer>Acme</Manufacturer><SerialNo>42</SerialNo></DeviceInfo><Key Id ←
   ="4711" Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp"><AlgorithmParameters>< ←
   ResponseFormat Encoding="DECIMAL" Length="8"/></AlgorithmParameters><Data><Secret>< ←
   PlainValue>Zm9v</PlainValue></Secret><Counter><PlainValue>42</PlainValue></Counter></ ←
   Data></Key></KeyPackage></KeyContainer>
jas@latte:~$
```

For more background and information what each field mean and which ones are required, you should read the PSKC specification (RFC 6030). You may pretty print the XML generate using "xmllint --pretty 1" which may simplify reading it. You may use "pskctool --info" to print a human summary of some PSKC data and validate the XML syntax using "pskctool --validate".

Chapter 3

Command line pskctool

To simplify working with PSKC data a command line tool is also provided, called "pskctool". When invoked without parameters, it will print some instructions describing what it does and the parameters it accepts.

```
Usage: pskctool [OPTION]... [FILE]...
Manipulate Portable Symmetric Key Container (PSKC) data.

This tool allows you to parse, print, validate, sign and verify PSKC data.  The
input is provided in FILE or on standard input.

  -h, --help              Print help and exit
  -V, --version           Print version and exit
      --strict            Fail hard on PSKC parse error  (default=off)
  -d, --debug             Show debug messages on stderr  (default=off)
  -q, --quiet             Quiet operation  (default=off)
  -v, --verbose           Produce more output  (default=off)


Selecting one of the following modes is required:

 Mode: info
  -i, --info              Parse and print human readable summary of PSKC input
                            (default=off)


 Mode: validate
  -e, --validate          Validate PSKC input against XML Schema  (default=off)


 Mode: sign
  Digitally sign PSKC data
      --sign              Sign PSKC input  (default=off)
      --sign-key=FILE     Private key to sign with
      --sign-crt=FILE     X.509 certificate to sign with


 Mode: verify
  Verify digitally signed PSKC data
      --verify            Verify signed PSKC input  (default=off)
      --verify-crt=FILE   Trusted X.509 certificate for verification


Report bugs to: oath-toolkit-help@nongnu.org
pskctool home page: <https://www.nongnu.org/oath-toolkit/>
General help using GNU software: <https://www.gnu.org/gethelp/>
```

As you can see, the pskctool have a few different modes: info, validate, sign and verify. We describe each of them in the next few sections.

3.1 Parse and print summary of PSKC data

The most common parameter to use is --info (-i) to parse and print a human readable summary of PSKC data. This step is also known as "pretty printing" the PSKC data. A filename can be supplied to have the tool read PSKC data from that file, or if no filename is supplied, the tool will read from standard input. To illustrate how the tool works, we will assume the following PSKC data is available in a file "pskc.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
        xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
  <KeyPackage>
    <DeviceInfo>
      <Manufacturer>Manufacturer</Manufacturer>
      <SerialNo>987654321</SerialNo>
    </DeviceInfo>
    <Key Id="12345678"
         Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
      <AlgorithmParameters>
        <ResponseFormat Length="8" Encoding="DECIMAL"/>
      </AlgorithmParameters>
      <Data>
        <Secret>
          <PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
          </PlainValue>
        </Secret>
        <Counter>
          <PlainValue>0</PlainValue>
        </Counter>
      </Data>
    </Key>
  </KeyPackage>
</KeyContainer>
```

Running the tool with the --info parameter, i.e., "pskctool --info pskc.xml" will produce a human readable variant of the PSKC data.

```
Portable Symmetric Key Container (PSKC):
  Version: 1.0
  Signed: NO
  KeyPackage 0:
    DeviceInfo:
      Manufacturer: Manufacturer
      SerialNo: 987654321
    Key:
      Id: 12345678
      Algorithm: urn:ietf:params:xml:ns:keyprov:pskc:hotp
      Key Secret (base64): MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
      Key Counter: 0
      Response Format Length: 8
      Response Format Encoding: DECIMAL
```

If the --verbose (-v) parameter is given, the tool will also print an indented version of the XML structure. Note that this will invalidate any digital signatures on the PSKC data. Thus, this is normally only useful to simplify human reading of the XML code of an PSKC file. The output will also contain the human readable summary, but you may use --quiet (-q) to suppress that part. Together, the combination of --verbose and --quiet can be used in batch jobs to indent PSKC data (but beware that this breaks any signatures).

In some situations when using pskctool --info the tool may print a warning about unsupported elements. The --debug parameter can be used in these situations to get more information about the source of the problem. For example, running "pskctool --info --debug --quiet" on the data in figure 6 of RFC 6030 will currently yield the following output on stderr.

```
debug: unknown <KeyContainer> element <EncryptionKey>
debug: unknown <KeyContainer> element <MACMethod>
debug: non-compliant Manufacturer value: Manufacturer
debug: unknown <Secret> element <EncryptedValue>
debug: unknown <Secret> element <ValueMAC>
warning: parse error (use -d to diagnose), output may be incomplete
```

Even when noticing a problem, the tool continue with the parsing and will eventually print the information it managed to parse. In some situations (e.g., batch jobs) you would prefer the tool to signal this error. The --strict parameter can be used to make the tool fail when there is a parse error.

3.2 Validate PSKC against XML Schema

The --validate (-e) parameter can be used to validate PSKC data according to the XML Schema specified in RFC 6030. This performs a deep analysis and syntax check of the data and will print either "OK" or "FAIL" depending on validation outcome.

```
$ pskctool -e pskc-ocra.xml
OK
$
```

Note that the exit code from pskctool --validate is 0 (indicating success) even when FAIL is printed. Use --quiet to suppress output and let the exit code correspond to validation result.

Note: If this command always results in errors, the XML catalog on your system needs to be updated to point to the installed PSKC schema files.

3.3 Digitally sign PSKC data

PSKC files can be integrity protected and authenticated using XML Digital Signatures. We support using a X.509 end-entity certificate together with a private key. To verify the signature, you will need to supply the issuer of the end-entity certificate as a trusted root. To illustrate this, we first show how to generate example root and end-entity private keys and certificates using GnuTLS. First generate the root private key and certificate:

```
jas@latte:~$ certtool --generate-privkey --outfile pskc-root-key.pem
Generating a 2432 bit RSA private key...
jas@latte:~$ certtool --generate-self-signed --load-privkey pskc-root-key.pem --outfile  ↩
    pskc-root-crt.pem
Generating a self signed certificate...
Please enter the details of the certificate's distinguished name. Just press enter to  ↩
    ignore a field.
Country name (2 chars):
Organization name:
Organizational unit name:
Locality name:
State or province name:
Common name: My PSKC root
UID:
This field should not be used in new certificates.
E-mail:
Enter the certificate's serial number in decimal (default: 1350939670):


Activation/Expiration time.
The certificate will expire in (days): 100


Extensions.
```

```
Does the certificate belong to an authority? (y/N): y
Path length constraint (decimal, -1 for no constraint):
Is this a TLS web client certificate? (y/N):
Will the certificate be used for IPsec IKE operations? (y/N):
Is this also a TLS web server certificate? (y/N):
Enter the e-mail of the subject of the certificate:
Will the certificate be used to sign other certificates? (y/N): y
Will the certificate be used to sign CRLs? (y/N):
Will the certificate be used to sign code? (y/N):
Will the certificate be used to sign OCSP requests? (y/N):
Will the certificate be used for time stamping? (y/N):
Enter the URI of the CRL distribution point:
X.509 Certificate Information:
  Version: 3
  Serial Number (hex): 5085b416
  Validity:
    Not Before: Mon Oct 22 21:01:11 UTC 2012
    Not After: Wed Jan 30 21:01:13 UTC 2013
  Subject: CN=My PSKC root
  Subject Public Key Algorithm: RSA
  Certificate Security Level: Normal
    Modulus (bits 2432):
      00:d3:cf:07:f9:75:df:61:91:a4:a9:e2:a6:54:fa:48
      b1:70:8c:a1:83:4e:ce:fa:01:d7:01:96:7a:5f:57:27
      1a:5a:fb:02:f4:50:b5:40:b6:67:8a:63:e3:60:8f:ed
      6e:9d:40:df:46:0d:8c:42:31:d9:74:08:f9:7d:48:fc
      e2:21:2e:fe:fd:e1:02:55:54:b5:5e:57:f8:5f:a0:8c
      81:5e:ca:5c:bd:64:41:5d:71:b5:81:84:1b:dc:36:75
      cc:19:62:19:f1:36:ed:00:98:13:5c:ce:3b:8c:ba:f9
      7f:9f:21:20:c2:0d:08:4e:e5:08:ad:5c:83:4e:c3:7c
      2a:4d:e0:7c:45:d2:b6:b9:42:8b:de:48:5f:60:2d:2e
      18:a7:f5:da:81:cf:24:d6:de:6d:31:07:63:20:d9:5e
      7c:ba:88:fa:1b:d8:98:3c:ab:05:4e:ca:a8:60:8d:6e
      9c:13:35:01:23:82:53:36:5b:e1:01:62:7f:ce:41:d1
      74:67:1b:f8:60:4b:87:e4:2c:52:6a:0a:67:4c:0d:27
      80:2d:6d:f7:2e:6f:2e:12:fb:d2:09:dc:d9:11:b1:b8
      c0:a4:34:00:3b:a0:87:c7:f2:2f:7f:30:6a:b6:c7:f1
      96:fc:6f:de:df:40:ac:2b:1a:d7:24:18:ae:1a:d7:8a
      4b:6b:a8:93:36:af:72:0e:93:15:30:47:fa:58:8a:4e
      97:86:14:a0:ef:84:46:5f:b4:a1:cd:98:d5:eb:97:fb
      4e:94:10:08:ba:c6:3f:57:0d:ef:1b:1b:21:af:4a:bd
      e7
    Exponent (bits 24):
      01:00:01
  Extensions:
    Basic Constraints (critical):
      Certificate Authority (CA): TRUE
    Key Usage (critical):
      Certificate signing.
    Subject Key Identifier (not critical):
      1f2507c525358817404c90b7f36e3b97dbbec098
Other Information:
  Public Key Id:
    1f2507c525358817404c90b7f36e3b97dbbec098

Is the above information ok? (y/N): y


Signing certificate...
jas@latte:~$
```

Next we generate a private key and certificate for the end-entity that will sign the PSKC data.

```
jas@latte:~$ certtool --generate-privkey --outfile pskc-ee-key.pem
Generating a 2432 bit RSA private key...
jas@latte:~$ certtool --generate-certificate --load-ca-privkey pskc-root-key.pem --load-ca- ↩
    certificate pskc-root-crt.pem --load-privkey pskc-ee-key.pem --outfile pskc-ee-crt.pem
Generating a signed certificate...
Please enter the details of the certificate's distinguished name. Just press enter to  ↩
    ignore a field.
Country name (2 chars):
Organization name:
Organizational unit name:
Locality name:
State or province name:
Common name: My PSKC end entity
UID:
This field should not be used in new certificates.
E-mail:
Enter the certificate's serial number in decimal (default: 1350939833):


Activation/Expiration time.
The certificate will expire in (days): 50


Extensions.
Does the certificate belong to an authority? (y/N):
Is this a TLS web client certificate? (y/N):
Will the certificate be used for IPsec IKE operations? (y/N):
Is this also a TLS web server certificate? (y/N):
Enter the e-mail of the subject of the certificate:
Will the certificate be used for signing (required for TLS)? (y/N): y
Will the certificate be used for encryption (not required for TLS)? (y/N):
X.509 Certificate Information:
  Version: 3
  Serial Number (hex): 5085b4b9
  Validity:
    Not Before: Mon Oct 22 21:03:54 UTC 2012
    Not After: Tue Dec 11 21:03:57 UTC 2012
  Subject: CN=My PSKC end entity
  Subject Public Key Algorithm: RSA
  Certificate Security Level: Normal
    Modulus (bits 2432):
      00:c4:4c:2b:8d:33:29:14:0f:4b:49:f5:8e:0c:f6:5b
      9f:0f:e3:17:aa:c5:77:8d:d4:64:16:c4:d4:4d:7d:04
      2d:0d:14:78:77:ba:4c:3c:bd:5c:46:9e:d0:24:b9:bb
      3d:92:2c:21:29:c3:e6:ea:5f:4e:e7:2e:60:c6:0e:0e
      fe:a3:ac:94:e9:0e:bf:84:8f:3b:db:97:45:2b:72:58
      07:0b:1f:5a:4e:b3:c6:e4:99:32:8a:56:a7:40:6e:a5
      93:62:99:9d:eb:5e:64:20:8a:bc:de:4d:9e:e3:62:22
      b4:6f:c8:50:c1:09:42:a8:90:c1:76:75:57:05:ab:b0
      f9:f6:e8:26:73:23:45:c4:3e:31:2b:3a:d0:23:db:42
      d7:1b:d2:57:be:16:cc:71:4d:2b:b1:4f:59:88:0f:29
      9f:ff:b8:05:4a:f7:8f:c6:c4:cb:a0:77:6d:0b:35:5b
      35:7a:ad:d3:d7:1b:b4:dd:dc:d8:a0:8d:ab:fb:c0:ab
      ec:1b:37:47:0b:06:d9:14:1f:f2:fc:bb:3d:ed:2d:5e
      b4:a5:cb:ec:4e:ab:ba:52:02:40:21:a6:8e:3e:3b:78
      0f:a7:73:62:30:4b:05:72:2a:71:1a:81:31:d5:e4:c4
      12:e9:7e:95:a2:9c:1f:53:2f:bb:f0:33:ce:37:c4:58
      fc:da:35:2b:09:18:3c:94:21:d3:7d:d9:d9:b0:ce:d0
      b9:c8:77:b5:e1:ce:9b:83:7c:e5:84:7d:4e:64:5f:c0
      2b:db:1a:0e:06:47:e4:24:44:ed:14:05:49:6f:17:78
      e3
```

```
      Exponent (bits 24):
         01:00:01
   Extensions:
     Basic Constraints (critical):
       Certificate Authority (CA): FALSE
     Key Usage (critical):
       Digital signature.
     Subject Key Identifier (not critical):
       0d8aed9f4ed4e2c3e12f7ca45fc6e8c8f56bb9c2
     Authority Key Identifier (not critical):
       1f2507c525358817404c90b7f36e3b97dbbec098
Other Information:
   Public Key Id:
     0d8aed9f4ed4e2c3e12f7ca45fc6e8c8f56bb9c2

Is the above information ok? (y/N): y


Signing certificate...
jas@latte:~$
```

At this point, we have the following files:

- "pskc-root-key.pem" root private key;

- "pskc-root-crt.pem" root certificate;

- "pskc-ee-key.pem" end entity private key;

- "pskc-ee-crt.pem" end entity certificate.

Let's use these files to digitally sign the following PSKC data, stored in a file "pskc-hotp.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
        xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
  <KeyPackage>
    <DeviceInfo>
      <Manufacturer>Manufacturer</Manufacturer>
      <SerialNo>987654321</SerialNo>
    </DeviceInfo>
    <Key Id="12345678"
         Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
      <AlgorithmParameters>
        <ResponseFormat Length="8" Encoding="DECIMAL"/>
      </AlgorithmParameters>
      <Data>
        <Secret>
          <PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
          </PlainValue>
        </Secret>
        <Counter>
          <PlainValue>0</PlainValue>
        </Counter>
      </Data>
    </Key>
  </KeyPackage>
</KeyContainer>
```

The --sign mode flag requires the --sign-key and --sign-crt which specify the private key and certificate to use for signing.

```
$ pskctool --sign --sign-key pskc-ee-key.pem --sign-crt pskc-ee-crt.pem pskc-hotp.xml > ↩
    pskc-hotp-signed.xml
$
```

Below is the signed XML output. As you can see, due to the signature it becomes rather unreadable. You may use "pskctool --info" to analyse it, or "pskctool --info --verbose --quiet" to print indented XML (however that will invalidate signature).

```
<?xml version="1.0"?>
<KeyContainer xmlns="urn:ietf:params:xml:ns:keyprov:pskc" Version="1.0"><KeyPackage>< ↩
    DeviceInfo><Manufacturer>Manufacturer</Manufacturer><SerialNo>987654321</SerialNo></ ↩
    DeviceInfo><Key Id="12345678" Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">< ↩
    AlgorithmParameters><ResponseFormat Encoding="DECIMAL" Length="8"/></AlgorithmParameters ↩
    ><Data><Secret><PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</PlainValue></Secret><Counter>< ↩
    PlainValue>0</PlainValue></Counter></Data></Key></KeyPackage><Signature xmlns="http:// ↩
    www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference>
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>scw48LN8ec/vu7/f7F1AGcfjDpI=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>HYDZFC205862s+zoas+Ny6h0ckDJmqDGz81lEPjvjGcN1AYzT7PATsIUVure0QNl
Kvt2TxdSDgnYlWwAJWjAtmp0UHRzF6hsmDl7WiHpeCkfxpwvdz8K469rbLPUwB6I
Zyfx/msTwJGbycPek9SFoaEqn8G7oNU59UH1HjDO0ERyKXhkiIrRaIWfGdqy4v0z
xYbPnAvzdHcEBdVOVQ3d+zeR/3nWGINjmxPnYGiCrY4YoktKm/VPNw3yuo3CNTIs
N4Vs4rjNVr7NcplFKLOmBBsQwKRg3JXnVW7kQu9ZonJyJEeDoNXdrG8uCa7EYT+s
eh6486o/Wvb7oUVbUN3JW5VRTnVK8YNOwAnxB1fTa92pJwffLB+knBlzVNteWCyA
BciIcboYbMdxLVmNKcF5pA==</SignatureValue>
<KeyInfo>
<X509Data>
<X509Certificate>MIIDdzCCAi+gAwIBAgIEUOYFHTANBgkqhkiG9w0BAQsFADAXMRUwEwYDVQQDEwxN
eSBQU0tDIHJvb3QwIhgPMjAxMzAxMDMyMjI0MzBaGA8yMjg2MTAxOTIyMjQzMlow
HTEbMBkGA1UEAxMTSXkgUFNLQyBlbmQgZW50aXR5MIIBUjANBgkqhkiG9w0BAQEF
AAOCAT8AMIIBOgKCATEAxEwrjTMpFA9LSfWODPZbnw/jF6rFd43UZBbE1E19BC0N
FHh3ukw8vVxGntAkubs9kiwhKcPm6l9O5y5gxg4O/qOslOkOv4SPO9uXRStyWAcL
H1pOs8bkmTKKVqdAbqWTYpmd615kIIq83k2e42IitG/IUMEJQqiQwXZ1VwWrsPn2
6CZzI0XEPjErOtAj20LXG9JXvhbMcU0rsU9ZiA8pn/+4BUr3j8bEy6B3bQs1WzV6
rdPXG7Td3Nigjav7wKvsGzdHCwbZFB/y/Ls97S1etKXL7E6rulICQCGmjj47eA+n
c2IwSwVyKnEagTHV5MQS6X6VopwfUy+78DPON8RY/No1KwkYPJQh033Z2bDO0LnI
d7XhzpuDfOWEfU5kX8Ar2xoOBkfkJETtFAVJbxd44wIDAQABo2EwXzAMBgNVHRMB
Af8EAjAAMA8GA1UdDwEB/wQFAwMHgAAwHQYDVR0OBBYEFHYGbZIa17d44czfdCkT
Mn+rWSBNMB8GA1UdIwQYMBaAFNLIhrjU/J0jWFX4rjsfsUkz1PQcMA0GCSqGSIb3
DQEBCwUAA4IBMQCxI1JOMqwgi/mj9KNutqGbTHdgKptt9lBylilwjMaNaY2lZe8S
5XNg9SoupGr1xBbMsDwWLILSuwPiedbn50fBpAAUW31WKKio6xRCJVmWeo0iY0Cr
rIXbwqKhnBP943U4Ch31oEbZtbo+XRbiq11wv6dLNsi76TNGDqsjTKgEcSIYI6Vd
rMxnil6ChoIBvSSPGHhJuj1bW1EPW92JtIa6byrAj1m4RwSviQy2i65YoIdtrhRt
CWekj2zuL/0szv5rZMCCvxioOCA8znqELEPMfs0Aa/cACD2MZcC4gGXehNCvzYJr
TmB6lFpxP6f0g6eO7PVcqYN9NCwECxb5Cvx2j2uNlereY35/9oPR6YJx+V7sL+DB
n6F0mN8OUAFxDamepKdGRApU8uZ35624o/I4
</X509Certificate>
</X509Data>
</KeyInfo>
</Signature></KeyContainer>
```

3.4   Verify digitally signed PSKC data

To verify signed PSKC data you use the --verify parameter. It requires another parameter, --verify-crt, which should contain a trusted X.509 certificate. The signature will be validated against the end-entity X.509 certificate inside the PSKC file, and the end-entity certificate will be verified against the indicated --verify-crt trust root. Using the files "pskc-hotp-signed.xml" and "pskc-root-crt.pem" prepared in the previous section, below we illustrate how verifying signatures work.

```
jas@latte:~$ pskctool --verify --verify-crt pskc-root-crt.pem pskc-hotp-signed.xml
OK
jas@latte:~$
```

If verification fails, it prints "FAIL" to standard output. Note that the exit code from pskctool --verify is 0 (indicating success) even when FAIL is printed. Use --quiet to suppress output and let the exit code correspond to validation result.

Part II

API Reference

This part contains the complete API reference for the PSKC Library. There is a separate section for each include file, which contains related functions grouped together, but applications should include the top-level <pskc/pskc.h> file.

Chapter 4

pskc

pskc — Top-level include file.

Types and Values

typedef	pskc_t
typedef	pskc_key_t

Description

The top-level <pskc/pskc.h> include file is responsible for declaring top-level types and including all other header files. The pskc_t type is used for the high-level PSKC container type and the pskc_key_t type represent each key package within the container.

Functions

Types and Values

pskc_t

```
typedef struct pskc pskc_t;
```

All PSKC data is represented through the pskc_t container type, which is a high-level structure that only carries a version indicator (see pskc_get_version()), an optional identity field (see pskc_get_id()) and any number of pskc_key_t types, each containing one key (see pskc_get_keypackage()).

pskc_key_t

```
typedef struct pskc_key pskc_key_t;
```

PSKC keys are represented through the pskc_key_t type. Each key is part of a higher level pskc_t container type. The pskc_get_keypackage() function is used to retrieve the pskc_key_t values from the pskc_t structure.

Chapter 5

version

version — Library version handling.

Functions

const char *	pskc_check_version ()

Types and Values

#define	PSKC_VERSION
#define	PSKC_VERSION_NUMBER

Description

The pskc_check_version() function can be used to discover the library version and to test that the shared library during run-time is recent enough.

Functions

pskc_check_version ()

```
const char~*
pskc_check_version (const char *req_version);
```

Check PSKC library version.

See PSKC_VERSION for a suitable *req_version* string.

This function is one of few in the library that can be used without a successful call to pskc_global_init().

Parameters

req_version	version string to compare with, or NULL.

Returns

Check that the version of the library is at minimum the one given as a string in *req_version* and return the actual version string of the library; return NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

Types and Values

PSKC_VERSION

```
# define PSKC_VERSION "2.6.11"
```

Pre-processor symbol with a string that describe the header file version number. Used together with pskc_check_version() to verify header file and run-time library consistency.

PSKC_VERSION_NUMBER

```
# define PSKC_VERSION_NUMBER 0x02060b00
```

Pre-processor symbol with a hexadecimal value describing the header file version number. For example, when the header version is 1.2.3 this symbol will have the value 0x01020300. The last two digits are only used between public releases, and will otherwise be 00.

Chapter 6

global

global — Global functions.

Functions

int	pskc_global_init ()
void	pskc_global_done ()
void	(*pskc_log_func) ()
void	pskc_global_log ()
void	pskc_free ()

Description

The library is initialized using pskc_global_init() which is a thread-unsafe function that should be called when the code that needs the PSKC library functionality is initialized. When the application no longer needs to use the PSKC Library, it can call pskc_global_done() to release resources.

The pskc_free() function is used to de-allocate memory that was allocated by the library earlier and returned to the caller.

For debugging, you can implement a function of the pskc_log_func signature and call pskc_global_log() to make the library output some messages that may provide additional information.

Functions

pskc_global_init ()

```
int
pskc_global_init (void);
```

This function initializes the PSKC library. Every user of this library needs to call this function before using other functions. You should call pskc_global_done() when use of the PSKC library is no longer needed.

Returns

On success, PSKC_OK (zero) is returned, otherwise an error code is returned.

pskc_global_done ()

```
void
pskc_global_done (void);
```

This function deinitializes the PSKC library, which were initialized using pskc_global_init(). After calling this function, no other PSKC library function may be called except for to re-initialize the library using pskc_global_init().

pskc_log_func ()

```
void
(*pskc_log_func) (const char *msg);
```

Function prototype for the application to be able to log PSKC messages, see pskc_global_log().

Parameters

msg	message to be logged.

pskc_global_log ()

```
void
pskc_global_log (pskc_log_func log_func);
```

Enable global debug logging function. The function will be invoked by the library when it wishes to print various debug information.

pskc_log_func is of the form, void (*pskc_log_func) (const char *msg);

The application may use this to provide a debug mode with trace information for the user.

Parameters

log_func	new global pskc_log_func log function to use.

pskc_free ()

```
void
pskc_free (void *ptr);
```

Deallocates memory region by calling free(). If *ptr* is NULL no operation is performed.

This function is necessary on Windows, where different parts of the same application may use different memory heaps.

Parameters

ptr	memory region to deallocate, or NULL.

Chapter 7

errors

errors — Error handling.

Functions

const char *	pskc_strerror ()
const char *	pskc_strerror_name ()

Types and Values

enum	pskc_rc

Description

Most library functions uses an int return value to indicate success or failure, using pskc_rc values. The values can be converted into human readable explanations using pskc_strerror(). The symbolic error codes can be converted into strings using pskc_strerror_name().

Functions

pskc_strerror ()

```
const char~*
pskc_strerror (int err);
```

Convert return code to human readable string explanation of the reason for the particular error code.

This string can be used to output a diagnostic message to the user.

This function is one of few in the library that can be used without a successful call to pskc_init().

Parameters

err	error code, a pskc_rc value.

Returns

Returns a pointer to a statically allocated string containing an explanation of the error code *err* .

pskc_strerror_name ()

```
const char~*
pskc_strerror_name (int err);
```

Convert return code to human readable string representing the error code symbol itself. For example, pskc_strerror_name(PSKC_OK) returns the string "PSKC_OK".

This string can be used to output a diagnostic message to the user.

This function is one of few in the library that can be used without a successful call to pskc_init().

Parameters

err	error code, a pskc_rc value.

Returns

Returns a pointer to a statically allocated string containing a string version of the error code *err* , or NULL if the error code is not known.

Types and Values

enum pskc_rc

Return codes for PSKC functions. All return codes are negative except for the successful code PSKC_OK which are guaranteed to be

1. Positive values are reserved for non-error return codes.

Note that the pskc_rc enumeration may be extended at a later date to include new return codes.

Members

PSKC_OK	Successful return.

PSKC_MALLOC_ERROR	Memory allocation failed.
PSKC_XML_ERROR	Error returned from XML library.
PSKC_PARSE_ERROR	Error parsing PSKC data.
PSKC_BASE64_ERROR	Error decoding base64 data.
PSKC_UNKNOWN_OUTPUT_FORMAT	Unknown output format.
PSKC_XMLSEC_ERROR	Error returned from XMLSec library.

PSKC_LAST_ERROR	Meta-error indicating the last error code, for use when iterating over all error codes or similar.

Chapter 8

enums

enums — PSKC value enumerations and related functions.

Functions

const char *	pskc_pinusagemode2str ()
pskc_pinusagemode	pskc_str2pinusagemode ()
const char *	pskc_valueformat2str ()
pskc_valueformat	pskc_str2valueformat ()
const char *	pskc_keyusage2str ()
pskc_keyusage	pskc_str2keyusage ()

Types and Values

enum	pskc_pinusagemode
enum	pskc_valueformat
enum	pskc_keyusage

Description

The pskc_pinusagemode type describes PIN Policy Usage Modes. You can convert between string representation and integer values using pskc_pinusagemode2str() and pskc_str2pinusagemode().

The pskc_valueformat type describes PSKC data value encodings. You can convert between string representation and integer values using pskc_valueformat2str() and pskc_str2valueformat().

The pskc_keyusage type describes what PSKC keys may be used for. You can convert between string representation and integer values using pskc_keyusage2str() and pskc_str2keyusage(). Note that often multiple pskc_keyusage values are ORed together to form set of values.

Functions

pskc_pinusagemode2str ()

```
const char~*
pskc_pinusagemode2str (pskc_pinusagemode pinusagemode);
```

Convert pskc_pinusagemode to a string. For example, pskc_pinusagemode2str(PSKC_PINUSAGEMODE_LOCAL) will return "Local". The returned string must not be deallocated.

Parameters

| pinusagemode | an pskc_pinusagemode enumeration type | |

Returns

String corresponding to pskc_pinusagemode.

pskc_str2pinusagemode ()

```
pskc_pinusagemode
pskc_str2pinusagemode (const char *pinusagemode);
```

Convert a string to a pskc_pinusagemode type. For example, pskc_str2pinusagemode("Local") will return PSKC_PINUSAGEMODE_LO

Parameters

| pinusagemode | an string describing a key usage. | |

Returns

The corresponding pskc_pinusagemode value.

pskc_valueformat2str ()

```
const char~*
pskc_valueformat2str (pskc_valueformat valueformat);
```

Convert pskc_valueformat to a string. For example, pskc_valueformat2str(PSKC_VALUEFORMAT_DECIMAL) will return "DECIMAL". The returned string must not be deallocated.

Parameters

| valueformat | an pskc_valueformat enumeration type | |

Returns

String corresponding to pskc_valueformat.

pskc_str2valueformat ()

```
pskc_valueformat
pskc_str2valueformat (const char *valueformat);
```

Convert a string to a pskc_valueformat type. For example, pskc_str2valueformat("DECIMAL") will return PSKC_VALUEFORMAT_DE

Parameters

valueformat	an string describing a key usage.

Returns

The corresponding pskc_valueformat value.

pskc_keyusage2str ()

```
const char~*
pskc_keyusage2str (pskc_keyusage keyusage);
```

Convert pskc_keyusage to a string. For example, pskc_keyusage2str(PSKC_KEYUSAGE_OTP) will return "OTP". The returned string must not be deallocated.

Parameters

keyusage	an pskc_keyusage enumeration type

Returns

String corresponding to pskc_keyusage.

pskc_str2keyusage ()

```
pskc_keyusage
pskc_str2keyusage (const char *keyusage);
```

Convert a string to a pskc_keyusage type. For example, pskc_str2keyusage("KeyWrap") will return PSKC_KEYUSAGE_KEYWRAP.

Parameters

keyusage	an string describing a key usage.

Returns

The corresponding pskc_keyusage value.

Types and Values

enum pskc_pinusagemode

Enumeration of PIN Policy Usage Modes. This indicate the way the PIN is used.

Members

PSKC_PINUSAGEMODE_UNKNOWN	Unknown mode.
PSKC_PINUSAGEMODE_LOCAL	PIN is checked locally on the device.
PSKC_PINUSAGEMODE_PREPEND	PIN is prepended to the OTP and checked by OTP validating party.
PSKC_PINUSAGEMODE_APPEND	PIN is appended to the OTP and checked by OTP validating party.

PSKC_PINUSAGEMODE_ALGORITHMIC	The PIN is used as part of the algorithm computation.
PSKC_PINUSAGEMODE_LAST	Meta-value corresponding to the highest value, for use in iterating over all usage mode values.

enum pskc_valueformat

Enumeration of PSKC value encoding formats.

Members

PSKC_VALUEFORMAT_UNKNOWN	Unknown format.

PSKC_VALUEFORMAT_DECIMAL	Only numerical digits.
PSKC_VALUEFORMAT_HEXADECIMAL	Hexadecimal response.
PSKC_VALUEFORMAT_ALPHANUMERIC	All letters and numbers (case sensitive).
PSKC_VALUEFORMAT_BASE64	Base-64 encoded.
PSKC_VALUEFORMAT_BINARY	Binary data.
PSKC_VALUEFORMAT_LAST	Meta-value corresponding to the highest value, for use in iterating over all encoding format values.

enum pskc_keyusage

Enumeration of PSKC key usage values. These values puts constraints on the intended usage of the key. The recipient of the PSKC document MUST enforce the key usage. The values are assigned to numbers so that they can be ORed together to form a set of values.

Members

PSKC_KEYUSAGE_UNKNOWN	Unknown format.
PSKC_KEYUSAGE_OTP	The key MUST only be used for OTP generation.
PSKC_KEYUSAGE_CR	The key MUST only be used for Challenge/Response purposes.
PSKC_KEYUSAGE_ENCRYPT	The key MUST only be used for data encryption purposes.

PSKC_KEYUSAGE_INTEGRITY	The key MUST only be used to generate a keyed message digest for data integrity or authentication purposes.

PSKC_KEYUSAGE_VERIFY	The key MUST only be used to verify a keyed message digest for data integrity or authentication purposes (this is the opposite key usage of 'Integrity').

PSKC_KEYUSAGE_UNLOCK	The key MUST only be used for an inverse Challenge/Response in the case where a user has locked the device by entering a wrong PIN too many times (for devices with PIN-input capability).

PSKC_KEYUSAGE_DECRYPT	The key MUST only be used for data decryption purposes.
PSKC_KEYUSAGE_KEYWRAP	The key MUST only be used for key wrap purposes.
PSKC_KEYUSAGE_UNWRAP	The key MUST only be used for key unwrap purposes.
PSKC_KEYUSAGE_DERIVE	The key MUST only be used with a key derivation function to derive a new key.

PSKC_KEYUSAGE_GENERATE	The key MUST only be used to generate a new key based on a random number and the previous value of the key.
PSKC_KEYUSAGE_LAST	Meta-value corresponding to the highest value, for use in iterating over all key usage values.

Chapter 9

container

container — High-level PSKC container handling.

Functions

int	pskc_init ()
void	pskc_done ()
int	pskc_parse_from_memory ()
int	pskc_get_signed_p ()
int	pskc_validate ()
int	pskc_build_xml ()
int	pskc_sign_x509 ()
int	pskc_verify_x509crt ()
const char *	pskc_get_version ()
void	pskc_set_version ()
const char *	pskc_get_id ()
void	pskc_set_id ()
pskc_key_t *	pskc_get_keypackage ()
int	pskc_add_keypackage ()
int	pskc_output ()

Types and Values

enum	pskc_output_formats_t

Description

PSKC data is represented through the pskc_t type which is created by calling pskc_init() and destroyed by calling pskc_done(). You may parse PSKC data in XML form from a buffer by calling pskc_parse_from_memory(). To convert PSKC data to human readable form you may use pskc_output(). To validate PSKC data against the XML Schema, you may use pskc_validate(). To generate PSKC based on the internal parsed representation you may use pskc_build_xml() which takes a pskc_output_format enumeration to indicate output form.

The PSKC data structure is a high-level structure that only carries a version indicator (see pskc_get_version()), an optional identity field (see pskc_get_id()) and any number of pskc_key_t types, each containing one key (see pskc_get_keypackage()).

Functions

pskc_init ()

```
int
pskc_init (pskc_t **container);
```

This function initializes the PSKC *container* handle. The memory allocate can be released by calling pskc_done().

Parameters

| container | pointer to a pskc_t handle to initialize. | |

Returns

On success, PSKC_OK (zero) is returned, on memory allocation errors PSKC_MALLOC_ERROR is returned.

pskc_done ()

```
void
pskc_done (pskc_t *container);
```

This function releases the resources associated with the PSKC *container* handle.

Parameters

| container | a pskc_t handle, from pskc_init(). | |

pskc_parse_from_memory ()

```
int
pskc_parse_from_memory (pskc_t *container,
                        size_t len,
                        const char *buffer);
```

This function will parse the XML data in *buffer* of *len* size into *container* . If PSKC_PARSE_ERROR is returned, parsing of some elements have failed but the *container* is still valid and contain partially parsed information. In this situation, you may continue but raise a warning.

Parameters

container	a pskc_t handle, from pskc_init().	
len	length of *buffer* .	
buffer	XML data to parse.	

Returns

On success, PSKC_OK (zero) is returned, on memory allocation errors PSKC_MALLOC_ERROR is returned, on XML library errors PSKC_XML_ERROR is returned, on PSKC parse errors PSKC_PARSE_ERROR is returned.

pskc_get_signed_p ()

```
int
pskc_get_signed_p (pskc_t *container);
```

Check whether the container is signed or not (note that it does not validate the signature, merely checks whether there is one).

Parameters

container	a pskc_t handle, from pskc_init().

Returns

a non-0 value if the container contains a Signature element, 0 if there is no Signature element.

pskc_validate ()

```
int
pskc_validate (pskc_t *container,
               int *isvalid);
```

This function validate the PSKC *container* handle the PSKC XML Schema.

Parameters

container	a pskc_t handle, from pskc_init().	
isvalid	output variable holding validation result, non-0 for valid.	

Returns

On success, PSKC_OK (zero) is returned, or an error code.

pskc_build_xml ()

```
int
pskc_build_xml (pskc_t *container,
                char **out,
                size_t *len);
```

This function builds a XML file from the data in *container* . As a convenience, it also converts the XML into a string placed in the newly allocated *out* of length *len* using pskc_output() with PSKC_OUTPUT_XML.

Parameters

container	a pskc_t handle, from pskc_init().	
out	pointer to output variable to hold newly allocated string.	
len	output variable holding length of *out .	

Returns

On success, PSKC_OK (zero) is returned, on memory allocation errors PSKC_MALLOC_ERROR is returned.

pskc_sign_x509 ()

```
int
pskc_sign_x509 (pskc_t *container,
                const char *key_file,
                const char *cert_file);
```

Sign PSKC data using X.509 certificate and private key.

Parameters

container	a pskc_t handle, from pskc_init().	
key_file	filename of file containing private key.	
cert_file	filename of file containing corresponding X.509 certificate.	

Returns

On success, PSKC_OK (zero) is returned, or an error code.

pskc_verify_x509crt ()

```
int
pskc_verify_x509crt (pskc_t *container,
                     const char *cert_file,
                     int *valid_signature);
```

Verify signature in PSKC data against trusted X.509 certificate.

Parameters

container	a pskc_t handle, from pskc_init().	
cert_file	filename of file containing trusted X.509 certificate.	

valid_signature	output variable with result of verification.	

Returns

On success, PSKC_OK (zero) is returned, or an error code.

pskc_get_version ()

```
const char~*
pskc_get_version (pskc_t *container);
```

Get the PSKC KeyContainer Version attribute. Normally this string is always "1.0" and a missing field is a syntax error according to the PSKC schema.

Parameters

container	a pskc_t handle, from pskc_init().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_version ()

```
void
pskc_set_version (pskc_t *container,
                  const char *version);
```

Set the PSKC KeyContainer Version attribute. Normally this string is always "1.0" and a missing field is a syntax error according to the PSKC schema.

The pointer is stored in `container` , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using `container` .

Parameters

container	a pskc_t handle, from pskc_init().	
version	pointer to version string to set.	

Since: 2.2.0

pskc_get_id ()

```
const char~*
pskc_get_id (pskc_t *container);
```

Get the PSKC KeyContainer Id attribute.

Parameters

container	a pskc_t handle, from pskc_init().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_id ()

```
void
pskc_set_id (pskc_t *container,
             const char *id);
```

Set the PSKC KeyContainer Id attribute.

The pointer is stored in `container` , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using `container` .

Parameters

container	a pskc_t handle, from pskc_init().	
id	pointer to id string to set.	

Since: 2.2.0

pskc_get_keypackage ()

```
pskc_key_t~*
pskc_get_keypackage (pskc_t *container,
                     size_t i);
```

Get a PSKC keypackage pskc_key_t handle for the `i` 'th key package in `container` . `i` is zero-based, i.e., 0 refer to the first key package, 1 refer to the second key package, and so on.

Parameters

container	a pskc_t handle, from pskc_init().	
i	number of keypackage to get.	

Returns

NULL if there is no `i` 'th key package, or a valid pskc_key_t pointer.

pskc_add_keypackage ()

```
int
pskc_add_keypackage (pskc_t *container,
                     pskc_key_t **key);
```

Add a new a PSKC keypackage to the *container* and give back a pskc_key_t handle.

Parameters

container	a pskc_t handle, from pskc_init().	
key	pointer to pskc_key_t key package handle.	

Returns

PSKC_MALLOC_ERROR on memory allocation errors, or PSKC_OK on success.

Since: 2.2.0

pskc_output ()

```
int
pskc_output (pskc_t *container,
             pskc_output_formats_t format,
             char **out,
             size_t *len);
```

Convert PSKC data to a serialized string of the indicated type. This is usually used to convert the PSKC data to some human readable form.

Parameters

container	a pskc_t handle, from pskc_init().	
format	an pskc_output_formats_t enumeration type indicating format.	
out	pointer to output variable holding newly allocated string.	
len	pointer to output variable hold length of *out .	

Returns

PSKC_OK on success, or an error code.

Types and Values

enum pskc_output_formats_t

Enumeration of different PSKC output formats.

Members

PSKC_OUTPUT_HUMAN_COMPLETE	All information in human-readable format.
PSKC_OUTPUT_XML	Output container in XML format.
PSKC_OUTPUT_INDENTED_XML	Output container in intended XML format (will invalidate any XML Digital Signatures).

Chapter 10

keypackage

keypackage — PSKC keypackage handling.

Functions

const char *	pskc_get_device_manufacturer ()
void	pskc_set_device_manufacturer ()
const char *	pskc_get_device_serialno ()
void	pskc_set_device_serialno ()
const char *	pskc_get_device_model ()
void	pskc_set_device_model ()
const char *	pskc_get_device_issueno ()
void	pskc_set_device_issueno ()
const char *	pskc_get_device_devicebinding ()
void	pskc_set_device_devicebinding ()
const struct tm *	pskc_get_device_startdate ()
void	pskc_set_device_startdate ()
const struct tm *	pskc_get_device_expirydate ()
void	pskc_set_device_expirydate ()
const char *	pskc_get_device_userid ()
void	pskc_set_device_userid ()
const char *	pskc_get_cryptomodule_id ()
void	pskc_set_cryptomodule_id ()
const char *	pskc_get_key_id ()
void	pskc_set_key_id ()
const char *	pskc_get_key_algorithm ()
void	pskc_set_key_algorithm ()
const char *	pskc_get_key_issuer ()
void	pskc_set_key_issuer ()
const char *	pskc_get_key_algparm_suite ()
void	pskc_set_key_algparm_suite ()
pskc_valueformat	pskc_get_key_algparm_chall_encoding ()
void	pskc_set_key_algparm_chall_encoding ()
uint32_t	pskc_get_key_algparm_chall_min ()
void	pskc_set_key_algparm_chall_min ()
uint32_t	pskc_get_key_algparm_chall_max ()
void	pskc_set_key_algparm_chall_max ()
int	pskc_get_key_algparm_chall_checkdigits ()
void	pskc_set_key_algparm_chall_checkdigits ()

pskc_valueformat	pskc_get_key_algparm_resp_encoding ()
void	pskc_set_key_algparm_resp_encoding ()
uint32_t	pskc_get_key_algparm_resp_length ()
void	pskc_set_key_algparm_resp_length ()
int	pskc_get_key_algparm_resp_checkdigits ()
void	pskc_set_key_algparm_resp_checkdigits ()
const char *	pskc_get_key_profileid ()
void	pskc_set_key_profileid ()
const char *	pskc_get_key_reference ()
void	pskc_set_key_reference ()
const char *	pskc_get_key_friendlyname ()
void	pskc_set_key_friendlyname ()
const char *	pskc_get_key_userid ()
void	pskc_set_key_userid ()
const char *	pskc_get_key_data_secret ()
int	pskc_set_key_data_secret ()
const char *	pskc_get_key_data_b64secret ()
int	pskc_set_key_data_b64secret ()
uint64_t	pskc_get_key_data_counter ()
void	pskc_set_key_data_counter ()
uint32_t	pskc_get_key_data_time ()
void	pskc_set_key_data_time ()
uint32_t	pskc_get_key_data_timeinterval ()
void	pskc_set_key_data_timeinterval ()
uint32_t	pskc_get_key_data_timedrift ()
void	pskc_set_key_data_timedrift ()
const struct tm *	pskc_get_key_policy_startdate ()
void	pskc_set_key_policy_startdate ()
const struct tm *	pskc_get_key_policy_expirydate ()
void	pskc_set_key_policy_expirydate ()
const char *	pskc_get_key_policy_pinkeyid ()
void	pskc_set_key_policy_pinkeyid ()
pskc_pinusagemode	pskc_get_key_policy_pinusagemode ()
void	pskc_set_key_policy_pinusagemode ()
uint32_t	pskc_get_key_policy_pinmaxfailedattempts ()
void	pskc_set_key_policy_pinmaxfailedattempts ()
uint32_t	pskc_get_key_policy_pinminlength ()
void	pskc_set_key_policy_pinminlength ()
uint32_t	pskc_get_key_policy_pinmaxlength ()
void	pskc_set_key_policy_pinmaxlength ()
pskc_valueformat	pskc_get_key_policy_pinencoding ()
void	pskc_set_key_policy_pinencoding ()
int	pskc_get_key_policy_keyusages ()
void	pskc_set_key_policy_keyusages ()
uint64_t	pskc_get_key_policy_numberoftransactions ()
void	pskc_set_key_policy_numberoftransactions ()

Description

The pskc_key_t structure hold data for one key package in a high-level PSKC pskc_t structure. The pskc_get_keypackage() and pskc_add_keypackage() functions can be used to get a pskc_key_t handle, which is always related to one pskc_t structure. This section describes all the functions that are used to access and modify information stored in pskc_key_t PSKC key packages.

Functions

pskc_get_device_manufacturer ()

```
const char~*
pskc_get_device_manufacturer (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo Manufacturer value. This element indicates the manufacturer of the device.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_manufacturer ()

```
void
pskc_set_device_manufacturer (pskc_key_t *key,
                              const char *devmfr);
```

Set the PSKC KeyPackage DeviceInfo Manufacturer value. This element indicates the manufacturer of the device.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle from, e.g., pskc_add_keypackage().	
devmfr	string with device manufacturer name to set.	

Since: 2.2.0

pskc_get_device_serialno ()

```
const char~*
pskc_get_device_serialno (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo SerialNo value. This element contains the serial number of the device.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_serialno ()

```
void
pskc_set_device_serialno (pskc_key_t *key,
                          const char *serialno);
```

Set the PSKC KeyPackage DeviceInfo SerialNo value. This element indicates the serial number of the device.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle from, e.g., pskc_add_keypackage().
serialno	string with serial number to set.

Since: 2.2.0

pskc_get_device_model ()

```
const char~*
pskc_get_device_model (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo Model value. This element describes the model of the device (e.g., "one-button-HOTP-token-V1").

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_model ()

```
void
pskc_set_device_model (pskc_key_t *key,
                           const char *model);
```

Set the PSKC KeyPackage DeviceInfo Model value. This element describes the model of the device (e.g., "one-button-HOTP-token-V1").

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle from, e.g., pskc_add_keypackage().	
model	a string with model name to set.	

Since: 2.2.0

pskc_get_device_issueno ()

```
const char~*
pskc_get_device_issueno (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo IssueNo value. This element contains the issue number in case there are devices with the same serial number so that they can be distinguished by different issue numbers.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_issueno ()

```
void
pskc_set_device_issueno (pskc_key_t *key,
                           const char *issueno);
```

Set the PSKC KeyPackage DeviceInfo IssueNo value. This element contains the issue number in case there are devices with the same serial number so that they can be distinguished by different issue numbers.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle from, e.g., pskc_add_keypackage().	
issueno	a string with issue number to set.	

Since: 2.2.0

pskc_get_device_devicebinding ()

```
const char~*
pskc_get_device_devicebinding (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo DeviceBinding value. This element allows a provisioning server to ensure that the key is going to be loaded into the device for which the key provisioning request was approved. The device is bound to the request using a device identifier, e.g., an International Mobile Equipment Identity (IMEI) for the phone, or an identifier for a class of identifiers, e.g., those for which the keys are protected by a Trusted Platform Module (TPM).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_devicebinding ()

```
void
pskc_set_device_devicebinding (pskc_key_t *key,
                               const char *devbind);
```

Set the PSKC KeyPackage DeviceInfo DeviceBinding value. This element allows a provisioning server to ensure that the key is going to be loaded into the device for which the key provisioning request was approved. The device is bound to the request using a device identifier, e.g., an International Mobile Equipment Identity (IMEI) for the phone, or an identifier for a class of identifiers, e.g., those for which the keys are protected by a Trusted Platform Module (TPM).

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
devbind	a string with device binding to set.	

Since: 2.2.0

pskc_get_device_startdate ()

```
const struct tm~*
pskc_get_device_startdate (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo StartDate. This element denote the start date of a device (such as the one on a payment card, used when issue numbers are not printed on cards).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant struct (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_startdate ()

```
void
pskc_set_device_startdate (pskc_key_t *key,
                           const struct tm *startdate);
```

Set the PSKC KeyPackage DeviceInfo StartDate. This element denote the start date of a device (such as the one on a payment card, used when issue numbers are not printed on cards).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().
startdate	pointer to a tm struct with device starting date to set.

Since: 2.2.0

pskc_get_device_expirydate ()

```
const struct tm~*
pskc_get_device_expirydate (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo ExpiryDate. This element denote the end date of a device (such as the one on a payment card, used when issue numbers are not printed on cards).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant struct (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_expirydate ()

```
void
pskc_set_device_expirydate (pskc_key_t *key,
                            const struct tm *expirydate);
```

Set the PSKC KeyPackage DeviceInfo ExpiryDate. This element denote the end date of a device (such as the one on a payment card, used when issue numbers are not printed on cards).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
expirydate	pointer to a tm struct with device expiry date to set.	

Since: 2.2.0

pskc_get_device_userid ()

```
const char~*
pskc_get_device_userid (pskc_key_t *key);
```

Get the PSKC KeyPackage DeviceInfo Userid value. This indicates the user with whom the device is associated.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_device_userid ()

```
void
pskc_set_device_userid (pskc_key_t *key,
                        const char *userid);
```

Set the PSKC KeyPackage DeviceInfo Userid value. This indicates the user with whom the device is associated.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
userid	a string with user identity to set.	

Since: 2.2.0

pskc_get_cryptomodule_id ()

```
const char~*
pskc_get_cryptomodule_id (pskc_key_t *key);
```

Get the PSKC KeyPackage CryptoModule Id value. This element carries a unique identifier for the CryptoModule and is implementation specific. As such, it helps to identify a specific CryptoModule to which the key is being or was provisioned.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_cryptomodule_id ()

```
void
pskc_set_cryptomodule_id (pskc_key_t *key,
                          const char *cid);
```

Set the PSKC KeyPackage CryptoModule Id value. This element carries a unique identifier for the CryptoModule and is implementation specific. As such, it helps to identify a specific CryptoModule to which the key is being or was provisioned.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
cid	the cryptomodule id to set	

Since: 2.2.0

pskc_get_key_id ()

```
const char~*
pskc_get_key_id (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Id attribute value. It is a syntax error for this attribute to not be available.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_id ()

```
void
pskc_set_key_id (pskc_key_t *key,
                 const char *keyid);
```

Set the PSKC KeyPackage Key Id attribute value. It is a syntax error for this attribute to not be available.

The pointer is stored in `container` , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using `container` .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyid	the key identity string to set.	

Since: 2.2.0

pskc_get_key_algorithm ()

```
const char~*
pskc_get_key_algorithm (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Algorithm attribute value. This may be an URN, for example "urn:ietf:params:xml:ns:keyprov:pskc:hotp

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_algorithm ()

```
void
pskc_set_key_algorithm (pskc_key_t *key,
                        const char *keyalg);
```

Set the PSKC KeyPackage Key Algorithm attribute value. This may be an URN, for example "urn:ietf:params:xml:ns:keyprov:pskc:hotp'

The pointer is stored in `container` , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using `container` .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyalg	the key algorithm string to set.	

Since: 2.2.0

pskc_get_key_issuer ()

```
const char~*
pskc_get_key_issuer (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Issuer value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_issuer ()

```
void
pskc_set_key_issuer (pskc_key_t *key,
                     const char *keyissuer);
```

Set the PSKC KeyPackage Key Issuer value.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyissuer	a key issuer string to set.	

Since: 2.2.0

pskc_get_key_algparm_suite ()

```
const char~*
pskc_get_key_algparm_suite (pskc_key_t *key);
```

Get the PSKC KeyPackage Key AlgorithmParameters Suite value.

Parameters

| key | a pskc_key_t handle, from pskc_get_keypackage(). | |

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_algparm_suite ()

```
void
pskc_set_key_algparm_suite (pskc_key_t *key,
                            const char *keyalgparmsuite);
```

Set the PSKC KeyPackage Key AlgorithmParameters Suite value.

The pointer is stored in `container` , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using `container` .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyalgparmsuite	the key algorithm parameter suite string to set.	

Since: 2.2.0

pskc_get_key_algparm_chall_encoding ()

```
pskc_valueformat
pskc_get_key_algparm_chall_encoding (pskc_key_t *key,
                                     int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Encoding value. This attribute defines the encoding of the challenge accepted by the device.

If `present` is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an pskc_valueformat value

pskc_set_key_algparm_chall_encoding ()

```
void
pskc_set_key_algparm_chall_encoding (pskc_key_t *key,
                                     pskc_valueformat vf);
```

Set the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Encoding value. This attribute defines the encoding of the challenge accepted by the device.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
vf	the pskc_valueformat encoding type to set.	

Since: 2.2.0

pskc_get_key_algparm_chall_min ()

```
uint32_t
pskc_get_key_algparm_chall_min (pskc_key_t *key,
                                int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Min value. This attribute defines the minimum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the minimum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the minimum number of bytes of the unencoded value.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_algparm_chall_min ()

```
void
pskc_set_key_algparm_chall_min (pskc_key_t *key,
                                uint32_t challmin);
```

Set the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Min value. This attribute defines the minimum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the minimum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the minimum number of bytes of the unencoded value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
challmin	the minimum challenge length to set.	

Since: 2.2.0

pskc_get_key_algparm_chall_max ()

```
uint32_t
pskc_get_key_algparm_chall_max (pskc_key_t *key,
                                int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Max value. This attribute defines the maximum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the maximum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the maximum number of bytes of the unencoded value.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_algparm_chall_max ()

```
void
pskc_set_key_algparm_chall_max (pskc_key_t *key,
                                uint32_t challmax);
```

Set the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat Max value. This attribute defines the maximum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the maximum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the maximum number of bytes of the unencoded value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
challmax	the maximum challenge length to set.	

Since: 2.2.0

pskc_get_key_algparm_chall_checkdigits ()

```
int
pskc_get_key_algparm_chall_checkdigits
                                (pskc_key_t *key,
                                 int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat CheckDigits value. This attribute indicates whether a device needs to check the appended Luhn check digit, as defined in [ISOIEC7812], contained in a challenge. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. A value of TRUE indicates that the device will check the appended Luhn check digit in a provided challenge. A value of FALSE indicates that the device will not check the appended Luhn check digit in the challenge.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

1 to indicate a CheckDigits value of true, or 0 to indicate false.

pskc_set_key_algparm_chall_checkdigits ()

```
void
pskc_set_key_algparm_chall_checkdigits
                                (pskc_key_t *key,
                                 int checkdigit);
```

Set the PSKC KeyPackage Key AlgorithmParameters ChallengeFormat CheckDigits value. This attribute indicates whether a device needs to check the appended Luhn check digit, as defined in [ISOIEC7812], contained in a challenge. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. A value of TRUE indicates that the device will check the appended Luhn check digit in a provided challenge. A value of FALSE indicates that the device will not check the appended Luhn check digit in the challenge.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
checkdigit	non-zero to indicate setting true CheckDigit, 0 otherwise.	

Since: 2.2.0

pskc_get_key_algparm_resp_encoding ()

```
pskc_valueformat
pskc_get_key_algparm_resp_encoding (pskc_key_t *key,
                                    int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ResponseFormat Encoding value. This attribute defines the encoding of the response generated by the device, it MUST be included.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an pskc_valueformat value

pskc_set_key_algparm_resp_encoding ()

```
void
pskc_set_key_algparm_resp_encoding (pskc_key_t *key,
                                    pskc_valueformat vf);
```

Set the PSKC KeyPackage Key AlgorithmParameters ResponseFormat Encoding value. This attribute defines the encoding of the response generated by the device, it MUST be included.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
vf	the pskc_valueformat encoding type to set.	

Since: 2.2.0

pskc_get_key_algparm_resp_length ()

```
uint32_t
pskc_get_key_algparm_resp_length (pskc_key_t *key,
                                  int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ResponseFormat Length value. This attribute defines the length of the response generated by the device and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or ALPHANUMERIC, this value indicates the number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_algparm_resp_length ()

```
void
pskc_set_key_algparm_resp_length (pskc_key_t *key,
                                  uint32_t length);
```

Set the PSKC KeyPackage Key AlgorithmParameters ResponseFormat Length value. This attribute defines the length of the response generated by the device and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or ALPHANUMERIC, this value indicates the number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
length	length of response to set.	

Since: 2.2.0

pskc_get_key_algparm_resp_checkdigits ()

```
int
pskc_get_key_algparm_resp_checkdigits (pskc_key_t *key,
                                       int *present);
```

Get the PSKC KeyPackage Key AlgorithmParameters ResponseFormat CheckDigits value. This attribute indicates whether the device needs to append a Luhn check digit, as defined in [ISOIEC7812], to the response. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. If the value is TRUE, then the device will append a Luhn check digit to the response. If the value is FALSE, then the device will not append a Luhn check digit to the response.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

1 to indicate a CheckDigits value of true, or 0 to indicate false.

pskc_set_key_algparm_resp_checkdigits ()

```
void
pskc_set_key_algparm_resp_checkdigits (pskc_key_t *key,
                                       int checkdigit);
```

Set the PSKC KeyPackage Key AlgorithmParameters ResponseFormat CheckDigits value. This attribute indicates whether the device needs to append a Luhn check digit, as defined in [ISOIEC7812], to the response. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. If the value is TRUE, then the device will append a Luhn check digit to the response. If the value is FALSE, then the device will not append a Luhn check digit to the response.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
checkdigit	non-zero to indicate setting true CheckDigit, 0 otherwise.	

Since: 2.2.0

pskc_get_key_profileid ()

```
const char~*
pskc_get_key_profileid (pskc_key_t *key);
```

Get the PSKC KeyPackage Key KeyProfileId value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_profileid ()

```
void
pskc_set_key_profileid (pskc_key_t *key,
                        const char *profileid);
```

Set the PSKC KeyPackage Key KeyProfileId value.

The pointer is stored in *container*, not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container*.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
profileid	pointer to profileid string to set.	

Since: 2.2.0

pskc_get_key_reference ()

```
const char~*
pskc_get_key_reference (pskc_key_t *key);
```

Get the PSKC KeyPackage Key KeyReference value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_reference ()

```
void
pskc_set_key_reference (pskc_key_t *key,
                        const char *keyref);
```

Set the PSKC KeyPackage Key KeyReference value.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyref	pointer to key reference string to set.	

Since: 2.2.0

pskc_get_key_friendlyname ()

```
const char~*
pskc_get_key_friendlyname (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Friendlyname value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_friendlyname ()

```
void
pskc_set_key_friendlyname (pskc_key_t *key,
                           const char *fname);
```

Set the PSKC KeyPackage Key Friendlyname value.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
fname	pointer to friendly name string to set.	

Since: 2.2.0

pskc_get_key_userid ()

```
const char~*
pskc_get_key_userid (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Userid value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_userid ()

```
void
pskc_set_key_userid (pskc_key_t *key,
                     const char *keyuserid);
```

Set the PSKC KeyPackage Key Userid value.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyuserid	pointer to key userid string to set.	

Since: 2.2.0

pskc_get_key_data_secret ()

```
const char~*
pskc_get_key_data_secret (pskc_key_t *key,
                          size_t *len);
```

Get the PSKC KeyPackage Key Data Secret value. If `len` is not set, the caller can only use the returned value for comparison against NULL to check whether the field is present or not.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
len	pointer to output variable with length of returned data.	

Returns

a constant string (must not be deallocated) holding the content of length *`len` , or NULL if not set.

pskc_set_key_data_secret ()

```
int
pskc_set_key_data_secret (pskc_key_t *key,
                          const char *data,
                          size_t len);
```

Set the PSKC KeyPackage Key Data Secret value. The `data` data is copied into the `key` handle, so you may modify or deallocate the `data` pointer after calling this function. The data is base64 encoded by this function. On errors, the old secret is not modified.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
data	the byte array with the key to set, of `len` length.	
len	length of `data` byte array.	

Returns

PSKC_BASE64_ERROR on base64 encoding errors, PSKC_MALLOC_ERROR on memory allocation errors, or PSKC_OK on success.

Since: 2.2.0

pskc_get_key_data_b64secret ()

```
const char~*
pskc_get_key_data_b64secret (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Data Secret value in base64 as a zero-terminated string.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant string (must not be deallocated) holding the content of length *`len`, or NULL if not set.

pskc_set_key_data_b64secret ()

```
int
pskc_set_key_data_b64secret (pskc_key_t *key,
                             const char *b64secret);
```

Set the PSKC KeyPackage Key Data Secret value in base64 as a zero-terminated string. The `b64secret` data is copied into the `key` handle, so you may modify or deallocate the `b64secret` pointer after calling this function. The data is base64 decoded by this function to verify data validity. On errors, the old secret is not modified.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().
b64secret	the base64 encoded secret to set.

Returns

PSKC_BASE64_ERROR on base64 decoding errors, PSKC_MALLOC_ERROR on memory allocation errors, or PSKC_OK on success.

Since: 2.2.0

pskc_get_key_data_counter ()

```
uint64_t
pskc_get_key_data_counter (pskc_key_t *key,
                           int *present);
```

Get the PSKC KeyPackage Key Data Counter value. This element contains the event counter for event-based OTP algorithms.

If `present` is non-NULL, it will be 0 if the Counter field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_data_counter ()

```
void
pskc_set_key_data_counter (pskc_key_t *key,
                           uint64_t counter);
```

Set the PSKC KeyPackage Key Data Counter value. This element contains the event counter for event-based OTP algorithms.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
counter	counter value to set.	

Since: 2.2.0

pskc_get_key_data_time ()

```
uint32_t
pskc_get_key_data_time (pskc_key_t *key,
                        int *present);
```

Get the PSKC KeyPackage Key Data Time value. This element contains the time for time-based OTP algorithms. (If time intervals are used, this element carries the number of time intervals passed from a specific start point, normally it is algorithm dependent).

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_data_time ()

```
void
pskc_set_key_data_time (pskc_key_t *key,
                        uint32_t datatime);
```

Set the PSKC KeyPackage Key Data Time value. This element contains the time for time-based OTP algorithms. (If time intervals are used, this element carries the number of time intervals passed from a specific start point, normally it is algorithm dependent).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
datatime	the data time value to set.	

Since: 2.2.0

pskc_get_key_data_timeinterval ()

```
uint32_t
pskc_get_key_data_timeinterval (pskc_key_t *key,
                                int *present);
```

Get the PSKC KeyPackage Key Data TimeInterval value. This element carries the time interval value for time-based OTP algorithms in seconds (a typical value for this would be 30, indicating a time interval of 30 seconds).

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_data_timeinterval ()

```
void
pskc_set_key_data_timeinterval (pskc_key_t *key,
                                uint32_t timeinterval);
```

Set the PSKC KeyPackage Key Data TimeInterval value. This element carries the time interval value for time-based OTP algorithms in seconds (a typical value for this would be 30, indicating a time interval of 30 seconds).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
timeinterval	time interval value to set.	

Since: 2.2.0

pskc_get_key_data_timedrift ()

```
uint32_t
pskc_get_key_data_timedrift (pskc_key_t *key,
                             int *present);
```

Get the PSKC KeyPackage Key Data TimeDrift value. This element contains the device clock drift value for time-based OTP algorithms. The integer value (positive or negative drift) that indicates the number of time intervals that a validation server has established the device clock drifted after the last successful authentication. So, for example, if the last successful authentication established a device time value of 8 intervals from a specific start date but the validation server determines the time value at 9 intervals, the server SHOULD record the drift as -1.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_data_timedrift ()

```
void
pskc_set_key_data_timedrift (pskc_key_t *key,
                             uint32_t timedrift);
```

Set the PSKC KeyPackage Key Data TimeDrift value. This element contains the device clock drift value for time-based OTP algorithms. The integer value (positive or negative drift) that indicates the number of time intervals that a validation server has established the device clock drifted after the last successful authentication. So, for example, if the last successful authentication established a device time value of 8 intervals from a specific start date but the validation server determines the time value at 9 intervals, the server SHOULD record the drift as -1.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
timedrift	the time drift value to set.	

Since: 2.2.0

pskc_get_key_policy_startdate ()

```
const struct tm~*
pskc_get_key_policy_startdate (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Policy StartDate. This element denote the start of the validity period of a key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant struct (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_policy_startdate ()

```
void
pskc_set_key_policy_startdate (pskc_key_t *key,
                               const struct tm *startdate);
```

Set the PSKC KeyPackage Key Policy StartDate. This element denote the start of the validity period of a key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().
startdate	pointer to a tm struct with key policy starting date to set.

Since: 2.2.0

pskc_get_key_policy_expirydate ()

```
const struct tm~*
pskc_get_key_policy_expirydate (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Policy ExpiryDate. This element denote the expiry of the validity period of a key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().

Returns

a constant struct (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_policy_expirydate ()

```
void
pskc_set_key_policy_expirydate (pskc_key_t *key,
                                const struct tm *expirydate);
```

Set the PSKC KeyPackage Key Policy ExpiryDate. This element denote the expiry of the validity period of a key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
expirydate	pointer to a tm struct with key policy expiry date to set.	

Since: 2.2.0

pskc_get_key_policy_pinkeyid ()

```
const char~*
pskc_get_key_policy_pinkeyid (pskc_key_t *key);
```

Get the PSKC KeyPackage Key Policy PINPolicy PINKeyId value. This attribute carries the unique 'Id' attribute vale of the "Key" element held within this "KeyContainer" that contains the value of the PIN that protects the key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	

Returns

a constant string (must not be deallocated) holding the content, or NULL if not set.

pskc_set_key_policy_pinkeyid ()

```
void
pskc_set_key_policy_pinkeyid (pskc_key_t *key,
                              const char *pinkeyid);
```

Set the PSKC KeyPackage Key Policy PINPolicy PINKeyId value. This attribute carries the unique 'Id' attribute vale of the "Key" element held within this "KeyContainer" that contains the value of the PIN that protects the key.

The pointer is stored in *container* , not a copy of the data, so you must not deallocate the data before another call to this function or the last call to any function using *container* .

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
pinkeyid	pin key id value to set.	

Since: 2.2.0

pskc_get_key_policy_pinusagemode ()

```
pskc_pinusagemode
pskc_get_key_policy_pinusagemode (pskc_key_t *key,
                                  int *present);
```

Get the PSKC KeyPackage Key Policy PINPolicy PINUsageMode value. This mandatory attribute indicates the way the PIN is used during the usage of the key.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an pskc_pinusagemode value

pskc_set_key_policy_pinusagemode ()

```
void
pskc_set_key_policy_pinusagemode (pskc_key_t *key,
                                  pskc_pinusagemode pinusagemode);
```

Set the PSKC KeyPackage Key Policy PINPolicy PINUsageMode value. This mandatory attribute indicates the way the PIN is used during the usage of the key.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
pinusagemode	the pskc_pinusagemode value to set	

Since: 2.2.0

pskc_get_key_policy_pinmaxfailedattempts ()

```
uint32_t
pskc_get_key_policy_pinmaxfailedattempts
                                (pskc_key_t *key,
                                 int *present);
```

Get the PSKC KeyPackage Key Policy PINPolicy MaxFailedAttempts value. This attribute indicates the maximum number of times the PIN may be entered wrongly before it MUST NOT be possible to use the key anymore (typical reasonable values are in the positive integer range of at least 2 and no more than 10).

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_policy_pinmaxfailedattempts ()

```
void
pskc_set_key_policy_pinmaxfailedattempts
                                (pskc_key_t *key,
                                 uint32_t attempts);
```

Set the PSKC KeyPackage Key Policy PINPolicy MaxFailedAttempts value. This attribute indicates the maximum number of times the PIN may be entered wrongly before it MUST NOT be possible to use the key anymore (typical reasonable values are in the positive integer range of at least 2 and no more than 10).

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
attempts	number of attempts to set.	

Since: 2.2.0

pskc_get_key_policy_pinminlength ()

```
uint32_t
pskc_get_key_policy_pinminlength (pskc_key_t *key,
                                  int *present);
```

Get the PSKC KeyPackage Key Policy PINPolicy MinLength value. This attribute indicates the minimum length of a PIN that can be set to protect the associated key. It MUST NOT be possible to set a PIN shorter than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/ characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_policy_pinminlength ()

```
void
pskc_set_key_policy_pinminlength (pskc_key_t *key,
                                  uint32_t minlength);
```

Set the PSKC KeyPackage Key Policy PINPolicy MinLength value. This attribute indicates the minimum length of a PIN that can be set to protect the associated key. It MUST NOT be possible to set a PIN shorter than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/ characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
minlength	the length to set.	

Since: 2.2.0

pskc_get_key_policy_pinmaxlength ()

```
uint32_t
pskc_get_key_policy_pinmaxlength (pskc_key_t *key,
                                  int *present);
```

Get the PSKC KeyPackage Key Policy PINPolicy MaxLength value. This attribute indicates the maximum length of a PIN that can be set to protect this key. It MUST NOT be possible to set a PIN longer than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/ characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_policy_pinmaxlength ()

```
void
pskc_set_key_policy_pinmaxlength (pskc_key_t *key,
                                  uint32_t maxlength);
```

Set the PSKC KeyPackage Key Policy PINPolicy MaxLength value. This attribute indicates the maximum length of a PIN that can be set to protect this key. It MUST NOT be possible to set a PIN longer than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/ characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
maxlength	the length to set.	

Since: 2.2.0

pskc_get_key_policy_pinencoding ()

```
pskc_valueformat
pskc_get_key_policy_pinencoding (pskc_key_t *key,
                                 int *present);
```

Get the PSKC KeyPackage Key Policy PINPolicy PINEncoding value. This attribute indicates the encoding of the PIN and MUST be one of the pskc_valueformat values.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an pskc_valueformat value

pskc_set_key_policy_pinencoding ()

```
void
pskc_set_key_policy_pinencoding (pskc_key_t *key,
                                 pskc_valueformat pinencoding);
```

Set the PSKC KeyPackage Key Policy PINPolicy PINEncoding value. This attribute indicates the encoding of the PIN and MUST be one of the pskc_valueformat values.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
pinencoding	the pskc_valueformat encoding to set.	

Since: 2.2.0

pskc_get_key_policy_keyusages ()

```
int
pskc_get_key_policy_keyusages (pskc_key_t *key,
                               int *present);
```

Get the PSKC KeyPackage Key Policy KeyUsage values. The element puts constraints on the intended usage of the key. The recipient of the PSKC document MUST enforce the key usage.

If *present* is non-NULL, it will be 0 if the field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding a set of pskc_keyusage values ORed together.

pskc_set_key_policy_keyusages ()

```
void
pskc_set_key_policy_keyusages (pskc_key_t *key,
                               int keyusages);
```

Set the PSKC KeyPackage Key Policy KeyUsage values. The element puts constraints on the intended usage of the key. The recipient of the PSKC document MUST enforce the key usage.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
keyusages	integer with pskc_keyusage values ORed together.	

Since: 2.2.0

pskc_get_key_policy_numberoftransactions ()

```
uint64_t
pskc_get_key_policy_numberoftransactions
                                (pskc_key_t *key,
                                 int *present);
```

Get the PSKC KeyPackage Key Policy NumberOfTransactions value. The value in this element indicates the maximum number of times a key carried within the PSKC document can be used by an application after having received it. When this element is omitted, there is no restriction regarding the number of times a key can be used.

Note that while the PSKC specification uses the XML data type "nonNegativeInteger" for this variable, this implementation restricts the size of the value to 64-bit integers.

If *present* is non-NULL, it will be 0 if the Counter field is not present or 1 if it was present.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
present	output variable indicating whether data was provided or not.	

Returns

an integer holding the content.

pskc_set_key_policy_numberoftransactions ()

```
void
pskc_set_key_policy_numberoftransactions
                                (pskc_key_t *key,
                                 uint64_t uses);
```

Set the PSKC KeyPackage Key Policy NumberOfTransactions value. The value in this element indicates the maximum number of times a key carried within the PSKC document can be used by an application after having received it. When this element is omitted, there is no restriction regarding the number of times a key can be used.

Note that while the PSKC specification uses the XML data type "nonNegativeInteger" for this variable, this implementation restricts the size of the value to 64-bit integers.

Parameters

key	a pskc_key_t handle, from pskc_get_keypackage().	
uses	the number of transactions to set.	

Since: 2.2.0

Chapter 11

Index