tdrozdovsky Added more info about mTower binaries and fix some warnings History

1 contributor

385 lines (327 sloc) | 9.91 KB ...

```
1  /**
2   * @file        arch/arm/m2351/src/numaker_pfm_m2351/secure/main.c
3   * @brief       Provides functionality to start secure world, initialize secure
4   *              and normal worlds, pass to execution to normal world.
5   *
6   * @copyright   Copyright (c) 2019 Samsung Electronics Co., Ltd. All Rights Reserved.
7   * @author      Taras Drozdovskyi t.drozdovsky@samsung.com
8   *
9   * Licensed under the Apache License, Version 2.0 (the "License");
10  * you may not use this file except in compliance with the License.
11  * You may obtain a copy of the License at
12  *
13  * http://www.apache.org/licenses/LICENSE-2.0
14  *
15  * Unless required by applicable law or agreed to in writing, software
16  * distributed under the License is distributed on an "AS IS" BASIS,
17  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
18  * See the License for the specific language governing permissions and
19  * limitations under the License.
20  */
21
22 /* Included Files. */
23 #include <stdio.h>
24 #include <stdint.h>
25 #include <stdlib.h>
26 #include <string.h>
27
28 #include <openssl/sha.h>
29 //#include "config.h"
```

```
//#include "version.h"

#include <openssl/ec.h>      // for EC_GROUP_new_by_curve_name, EC_GROUP_free, EC_KEY_new, EC_KEY_
#include <openssl/ecdsa.h>   // for ECDSA_do_sign, ECDSA_do_verify
#include <openssl/obj_mac.h> // for NID_secp192k1

#include <stdlib.h>
#include <string.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/rand.h>
#include <openssl/err.h>

/* Pre-processor Definitions. */
#define AES_BLOCK_SIZE 16
#define AES_KEY_SIZE 32

/** Start address for non-secure boot image */

/* Private Types. */
/* Any types, enums, structures or unions used by the file are defined here. */
/* typedef for NonSecure callback functions */

/**
 * @details    ECC ECDSA key structure
 */
typedef struct {
  uint8_t Qx[32]; /* 256-bits */
  uint8_t Qy[32]; /* 256-bits */
  uint8_t d[32]; /* 256-bits */
}__attribute__((packed)) ECC_KEY_T;

typedef struct _AES_DATA {
  unsigned char key[AES_KEY_SIZE];
  unsigned char iv[AES_BLOCK_SIZE];
} AES_DATA;

typedef struct Message_Struct {
  unsigned char *body;
  int *length;
  AES_DATA *aes_settings;
} Message;

/* Private Function Prototypes. */
/* Prototypes of all static functions in the file are provided here. */

/* Private Data. */
/* All static data definitions appear here. */
uint32_t key[] =
```

```
  { 0x78563412, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
      0x00000000, 0xefcdac00 };

uint32_t iv[] =
  { 0x78563412, 0x00000000, 0x00000000, 0xefcdac00 };

/* Public Data. */
/* All data definitions with global scope appear here. */

/* Public Function Prototypes */
Message *message_init(int);
int aes256_init(Message *);
Message *aes256_encrypt(Message *);
void aes_cleanup(AES_DATA *);
void message_cleanup(Message *);

void sha256(unsigned char *data, unsigned int data_len, unsigned char *hash)
{
  SHA256_CTX sha256;
  SHA256_Init(&sha256);
  SHA256_Update(&sha256, data, data_len);
  SHA256_Final(hash, &sha256);
}

Message *message_init(int length)
{
  Message *ret = malloc(sizeof(Message));
  ret->body = malloc(length);
  ret->length = malloc(sizeof(int));
  *ret->length = length;
  //initialize aes_data
  aes256_init(ret);
  return ret;
}

int aes256_init(Message * input)
{
  AES_DATA *aes_info = malloc(sizeof(AES_DATA));
  //point to new data
  input->aes_settings = aes_info;
  //get rand bytes
  memcpy(input->aes_settings->key, key, AES_KEY_SIZE);
  memcpy(input->aes_settings->iv, iv, AES_KEY_SIZE / 2);

  return 0;
}

Message *aes256_encrypt(Message * plaintext)
{
```

```
128   EVP_CIPHER_CTX *enc_ctx;
129   Message * encrypted_message;
130   int enc_length = *(plaintext->length)
131       + (AES_BLOCK_SIZE - *(plaintext->length) % AES_BLOCK_SIZE);
132
133   encrypted_message = message_init(enc_length);
134   //set up encryption context
135   enc_ctx = EVP_CIPHER_CTX_new();
136   EVP_EncryptInit(enc_ctx, EVP_aes_256_cfb(), plaintext->aes_settings->key,
137       plaintext->aes_settings->iv);
138   //encrypt all the bytes up to but not including the last block
139   if (!EVP_EncryptUpdate(enc_ctx, encrypted_message->body, &enc_length,
140       plaintext->body, *plaintext->length))
141   {
142     EVP_CIPHER_CTX_cleanup(enc_ctx);
143     printf("EVP Error: couldn't update encryption with plain text!\n");
144     return NULL;
145   }
146   //update length with the amount of bytes written
147   *(encrypted_message->length) = enc_length;
148   //EncryptFinal will cipher the last block + Padding
149   if (!EVP_EncryptFinal_ex(enc_ctx, enc_length + encrypted_message->body,
150       &enc_length))
151   {
152     EVP_CIPHER_CTX_cleanup(enc_ctx);
153     printf("EVP Error: couldn't finalize encryption!\n");
154     return NULL;
155   }
156   //add padding to length
157   *(encrypted_message->length) += enc_length;
158   //no errors, copy over key & iv rather than pointing to the plaintext msg
159   memcpy(encrypted_message->aes_settings->key, plaintext->aes_settings->key,
160       AES_KEY_SIZE);
161   memcpy(encrypted_message->aes_settings->iv, plaintext->aes_settings->iv,
162       AES_KEY_SIZE / 2);
163   //Free context and return encrypted message
164   EVP_CIPHER_CTX_cleanup(enc_ctx);
165   return encrypted_message;
166 }
167
168 void aes_cleanup(AES_DATA *aes_data)
169 {
170 //    free(aes_data -> iv);
171 //    free(aes_data -> key);
172 //    free(aes_data);
173 }
174
175 void message_cleanup(Message *message)
176 {
```

```
  //free message struct
  aes_cleanup(message->aes_settings);
  free(message->length);
  free(message->body);
  free(message);
}

/**
 * @brief         main - entry point of mTower: secure world.
 *
 * @param         None
 *
 * @returns       None (function is not supposed to return)
 */
int main(int argc, char * argv[])
{
  // Initialize openSSL
//  ERR_load_crypto_strings();
//  OpenSSL_add_all_algorithms();
//  OPENSSL_config(NULL);

  EC_KEY *eckey = EC_KEY_new();
  if (NULL == eckey) {
    printf("Failed to create new EC Key\n");
    return -1;
  }

  EC_GROUP *ecgroup = EC_GROUP_new_by_curve_name(NID_X9_62_prime256v1);
  if (NULL == ecgroup) {
    printf("Failed to create new EC Group\n");
    return -1;
  }

  if (EC_KEY_set_group(eckey, ecgroup) != 1 ) {
    printf("Failed to set group for EC Key\n");
    return -1;
  }

  if (EC_KEY_generate_key(eckey) != 1) {
    printf("Failed to generate EC Key\n");
    return -1;
  }

  const BIGNUM* d = EC_KEY_get0_private_key(eckey);
  const EC_POINT* Q = EC_KEY_get0_public_key(eckey);
  const EC_GROUP* group = EC_KEY_get0_group(eckey);

  BIGNUM* x = BN_new();
  BIGNUM* y = BN_new();
```

```
226
227   if (!EC_POINT_get_affine_coordinates_GFp(group, Q, x, y, NULL)) {
228     return -1;
229   }
230
231 //  printf("d: %s\n", BN_bn2hex(d));
232 //  printf("X: %s\n", BN_bn2hex(x));
233 //  printf("Y: %s\n", BN_bn2hex(y));
234
235   ECC_KEY_T ecc_ecdsa_key;
236
237   BN_bn2bin(d, &ecc_ecdsa_key.d[0]);
238   BN_bn2bin(x, ecc_ecdsa_key.Qx);
239   BN_bn2bin(y, ecc_ecdsa_key.Qy);
240
241 //  for (int i = 0; i != 32; i++)
242 //    printf(" %02X",ecc_ecdsa_key.d[i]);
243 //  printf("\n");
244
245   char* buff;
246   buff = malloc(strlen(argv[1]) + strlen("ecdsa_keys.bin") + 1);
247   buff[0] = 0;
248   strcat(buff, argv[1]);
249   strcat(buff, "ecdsa_keys.bin");
250
251   FILE* fd = fopen(buff, "wb");
252   if (!fd) {
253     printf("Failed to open file\n");
254     free(buff);
255     return -1;
256   }
257   free(buff);
258
259   if (fwrite((void *) &ecc_ecdsa_key, sizeof(char),
260       (size_t) (sizeof(ECC_KEY_T)), fd) != (size_t) (sizeof(ECC_KEY_T)))
261   {
262     fclose(fd);
263     return -1;
264   }
265   fclose(fd);
266
267   Message *message, *enc_msg;
268
269   message = message_init(64);
270   memcpy((char *) message->body, (unsigned char *) ecc_ecdsa_key.Qx, 64);
271
272   enc_msg = aes256_encrypt(message);
273   //clean up ssl;
274   //  EVP_cleanup();
```

```
275   //  CRYPTO_cleanup_all_ex_data(); //Stop data leaks
276   //  ERR_free_strings();
277
278   buff = malloc(strlen(argv[1]) + strlen("NuBL32PubKeyEncrypted.bin") + 1);
279   buff[0] = 0;
280   strcat(buff, argv[1]);
281   strcat(buff, "NuBL32PubKeyEncrypted.bin");
282
283   fd = fopen(buff, "wb");
284   if (!fd) {
285     printf("Failed to open file\n");
286     free(buff);
287     return -1;
288   }
289   free(buff);
290
291   if (fwrite((void *) enc_msg->body, sizeof(char), 64, fd) != 64) {
292     fclose(fd);
293     return -1;
294   }
295   fclose(fd);
296 //  free(buff);
297 //  message_cleanup(message);
298 //  message_cleanup(enc_msg);
299
300   buff = malloc(strlen(argv[1]) + strlen("NuBL32PubKeyEncryptedHash.bin") + 1);
301   buff[0] = 0;
302   strcat(buff, argv[1]);
303   strcat(buff, "NuBL32PubKeyEncryptedHash.bin");
304
305   fd = fopen(buff, "wb");
306   if (!fd) {
307     printf("Failed to open file\n");
308     free(buff);
309     return -1;
310   }
311   free(buff);
312   unsigned char PubKeyEncryptedHash[64];
313   sha256((void *) enc_msg->body, 64, (unsigned char *) PubKeyEncryptedHash);
314
315   if (fwrite((void *) PubKeyEncryptedHash, sizeof(char), 32, fd) != 32) {
316     fclose(fd);
317     return -1;
318   }
319   fclose(fd);
320
321   message = message_init(64);
322   memcpy((char *) message->body, (unsigned char *) ecc_ecdsa_key.Qx, 64);
323
```

```
  enc_msg = aes256_encrypt(message);
  //clean up ssl;
  //  EVP_cleanup();
  //  CRYPTO_cleanup_all_ex_data(); //Stop data leaks
  //  ERR_free_strings();

  buff = malloc(strlen(argv[1]) + strlen("NuBL33PubKeyEncrypted.bin") + 1);
  buff[0] = 0;
  strcat(buff, argv[1]);
  strcat(buff, "NuBL33PubKeyEncrypted.bin");

  fd = fopen(buff, "wb");
  if (!fd) {
    printf("Failed to open file\n");
    free(buff);
    return -1;
  }
  free(buff);

  if (fwrite((void *) enc_msg->body, sizeof(char), 64, fd) != 64) {
    fclose(fd);
    return -1;
  }
  fclose(fd);
//  free(buff);
//  message_cleanup(message);
//  message_cleanup(enc_msg);

  buff = malloc(strlen(argv[1]) + strlen("NuBL33PubKeyEncryptedHash.bin") + 1);
  buff[0] = 0;
  strcat(buff, argv[1]);
  strcat(buff, "NuBL33PubKeyEncryptedHash.bin");

  fd = fopen(buff, "wb");
  if (!fd) {
    printf("Failed to open file\n");
    free(buff);
    return -1;
  }
  free(buff);
//  unsigned char PubKeyEncryptedHash[64];
  sha256((void *) enc_msg->body, 64, (unsigned char *) PubKeyEncryptedHash);

  if (fwrite((void *) PubKeyEncryptedHash, sizeof(char), 32, fd) != 32) {
    fclose(fd);
    return -1;
  }
  fclose(fd);

```

```
373  exit:
374    BN_free(x);
375    BN_free(y);
376    EC_GROUP_free(ecgroup);
377    EC_KEY_free(eckey);
378
379    // clean up ssl;
380  //  EVP_cleanup();
381  //  CRYPTO_cleanup_all_ex_data(); //Stop data leaks
382  //  ERR_free_strings();
383
384    return 0;
385  }
```