

 18f4b592a8 ▾

...

mTower / [tools](#) / fwinfofen.c



tdrozovsky Fixed issue related with new version openssl API ...

 History

 1 contributor

499 lines (426 sloc) | 14.6 KB

...

```

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

```

```

30 #include <openssl/obj_mac.h> // for NID_secp192k1
31
32 #include "config.h"
33 // #include "version.h"
34
35 /* Pre-processor Definitions. */
36 #define BUILD_MONTH_IS_JAN (__DATE__[0] == 'J' && __DATE__[1] == 'a' && __DATE__[2] == 'n')
37 #define BUILD_MONTH_IS_FEB (__DATE__[0] == 'F')
38 #define BUILD_MONTH_IS_MAR (__DATE__[0] == 'M' && __DATE__[1] == 'a' && __DATE__[2] == 'r')
39 #define BUILD_MONTH_IS_APR (__DATE__[0] == 'A' && __DATE__[1] == 'p')
40 #define BUILD_MONTH_IS_MAY (__DATE__[0] == 'M' && __DATE__[1] == 'a' && __DATE__[2] == 'y')
41 #define BUILD_MONTH_IS_JUN (__DATE__[0] == 'J' && __DATE__[1] == 'u' && __DATE__[2] == 'n')
42 #define BUILD_MONTH_IS_JUL (__DATE__[0] == 'J' && __DATE__[1] == 'u' && __DATE__[2] == 'l')
43 #define BUILD_MONTH_IS_AUG (__DATE__[0] == 'A' && __DATE__[1] == 'u')
44 #define BUILD_MONTH_IS_SEP (__DATE__[0] == 'S')
45 #define BUILD_MONTH_IS_OCT (__DATE__[0] == 'O')
46 #define BUILD_MONTH_IS_NOV (__DATE__[0] == 'N')
47 #define BUILD_MONTH_IS_DEC (__DATE__[0] == 'D')
48
49 #define BUILD_MONTH \
50     ( \
51         (BUILD_MONTH_IS_JAN) ? 1 : \
52         (BUILD_MONTH_IS_FEB) ? 2 : \
53         (BUILD_MONTH_IS_MAR) ? 3 : \
54         (BUILD_MONTH_IS_APR) ? 4 : \
55         (BUILD_MONTH_IS_MAY) ? 5 : \
56         (BUILD_MONTH_IS_JUN) ? 6 : \
57         (BUILD_MONTH_IS_JUL) ? 7 : \
58         (BUILD_MONTH_IS_AUG) ? 8 : \
59         (BUILD_MONTH_IS_SEP) ? 9 : \
60         (BUILD_MONTH_IS_OCT) ? 16 : \
61         (BUILD_MONTH_IS_NOV) ? 17 : \
62         (BUILD_MONTH_IS_DEC) ? 18 : \
63         /* error default */ 0 \
64     )
65
66 #define BUILD_DAY_CH0 ((__DATE__[4] >= '0') ? (__DATE__[4]) : '0')
67 #define BUILD_DAY_CH1 (__DATE__[5])
68
69 #define DATE_COMPILE (((__DATE__[7] - '0') << 28) \
70     | (((unsigned int)(__DATE__[8] - '0')) << 24) \
71     | (((unsigned int)(__DATE__[9] - '0')) << 20) \
72     | (((unsigned int)(__DATE__[10] - '0')) << 16) \
73     | (((unsigned int)(BUILD_MONTH)) << 8) \
74     | (((unsigned int)((__DATE__[4] >= '0') ? (__DATE__[4]) : '0') - '0')) << 4) \
75     | (((unsigned int)__DATE__[5] - '0')))
76
77 #define BL33_METADATA_ADDRESS (0x10080000 - CONFIG_START_ADDRESS_BL33 - 0x8000)
78

```

```

79
80 /** Instruction Code of "B ." */
81
82 /* Private Types. */
83 /* Any types, enums, structures or unions used by the file are defined here. */
84 /* typedef for NonSecure callback functions */
85
86 /**
87  * @details    ECC public key structure
88  */
89 typedef struct {
90     uint32_t au32Key0[8]; /* 256-bits */
91     uint32_t au32Key1[8]; /* 256-bits */
92 }__attribute__((packed)) ECC_PUBKEY_T;
93
94 /**
95  * @details    ECC ECDSA signature structure
96  */
97 typedef struct {
98     uint32_t au32R[8]; /* 256-bits */
99     uint32_t au32S[8]; /* 256-bits */
100 }__attribute__((packed)) ECDSA_SIGN_T;
101
102 typedef struct {
103     uint32_t u32Start; /* 32-bits */
104     uint32_t u32Size; /* 32-bits */
105 }__attribute__((packed)) FW_REGION_T;
106
107 typedef struct {
108     uint32_t u32AuthCFGs; /* 32-bits */
109     /*
110      bit[1:0]:    Reserved
111      bit[2]:      1: Info Hash includes PDID / 0: Not include PDID
112      bit[3]:      1: Info Hash includes UID / 0: Not include UID
113      bit[4]:      1: Info Hash includes UICD / 0: Not include UICD
114      bit[31:5]:   Reserved
115      */
116     uint32_t u32FwRegionLen; /* 32-bits */
117     FW_REGION_T au32FwRegion[1]; /* (8*1) bytes */
118     uint32_t u32ExtInfoLen; /* 32-bits */
119     uint32_t au32ExtInfo[3]; /* 12-bytes */
120 }__attribute__((packed)) METADATA_T;
121
122 typedef struct {
123     ECC_PUBKEY_T pubkey; /* 64-bytes (256-bits + 256-bits) */
124
125     METADATA_T mData; /* includes authenticate configuration, F/W regions and extend info */
126
127     uint32_t au32FwHash[8]; /* 32-bytes (256-bits) */

```

```

128
129     ECDSA_SIGN_T sign; /* 64-bytes (256-bits R + 256-bits S) */
130 }__attribute__((packed)) FW_INFO_T;
131
132 /**
133  * @details    ECC ECDSA key structure
134  */
135 typedef struct {
136     uint8_t Qx[32]; /* 256-bits */
137     uint8_t Qy[32]; /* 256-bits */
138     uint8_t d[32]; /* 256-bits */
139 }__attribute__((packed)) ECC_KEY_T;
140
141 /* Private Function Prototypes. */
142 /* Prototypes of all static functions in the file are provided here. */
143
144 /* Private Data. */
145 /* All static data definitions appear here. */
146
147 /* Public Data. */
148 /* All data definitions with global scope appear here. */
149 const char header[] = "\n"
150     "const uint32_t g_InitialFWinfo[] =\n"
151     "{\n"
152     "    /* public key - 64-bytes (256-bits + 256-bits) */\n";
153
154 /* Public Function Prototypes */
155
156 /* Private Functions. */
157
158 void sha256(unsigned char *data, unsigned int data_len, unsigned char *hash)
159 {
160     SHA256_CTX sha256;
161     SHA256_Init(&sha256);
162     SHA256_Update(&sha256, data, data_len);
163     SHA256_Final(hash, &sha256);
164 }
165
166 static int sign_pFwInfo(FW_INFO_T *pFwInfo, ECC_KEY_T *ecdsa_key)
167 {
168     int ret = -1;
169     EC_KEY *eckey = EC_KEY_new();
170     if (NULL == eckey) {
171         printf("Failed to create new EC Key\n");
172         return -1;
173     }
174
175     EC_GROUP *ecgroup = EC_GROUP_new_by_curve_name(NID_X9_62_prime256v1);
176     if (NULL == ecgroup) {

```

```

177     printf("Failed to create new EC Group\n");
178     goto exit;
179 }
180
181 if (EC_KEY_set_group(eckey, ecgroup) != 1) {
182     printf("Failed to set group for EC Key\n");
183     goto exit;
184 }
185
186 memcpy((void *) pFwInfo->pubkey.au32Key0, (void *) ecdsa_key->Qx, 32);
187 memcpy((void *) pFwInfo->pubkey.au32Key1, (void *) ecdsa_key->Qy, 32);
188
189 BIGNUM* x = BN_bin2bn((void *) ecdsa_key->Qx, 32, NULL);
190 BIGNUM* y = BN_bin2bn((void *) ecdsa_key->Qy, 32, NULL);
191 BIGNUM* d = BN_bin2bn((void *) ecdsa_key->d, 32, NULL);
192
193 EC_KEY_set_private_key(eckey, d);
194 EC_KEY_set_public_key_affine_coordinates(eckey, x, y);
195
196 uint32_t au32HeadHash[8];
197 unsigned int u32Size = sizeof(FW_INFO_T) - sizeof(ECDSA_SIGN_T);
198 sha256((unsigned char *) pFwInfo, u32Size, (unsigned char *) au32HeadHash);
199
200 ECDSA_SIG *signature = ECDSA_do_sign((unsigned char *) au32HeadHash, u32Size,
201     eckey);
202 if (NULL == signature) {
203     printf("Failed to generate EC Signature\n");
204 } else {
205     if (ECDSA_do_verify((unsigned char *) au32HeadHash, u32Size, signature,
206         eckey) != 1) {
207         printf("Failed to verify EC Signature\n");
208     } else {
209         BIGNUM *r, *s;
210 #if OPENSSL_VERSION_NUMBER >= 0x10100000L
211         ECDSA_SIG_get0(signature, &r, &s);
212 #else
213         r = signature->r;
214         s = signature->s;
215 #endif
216
217         // printf("d: %s\n", BN_bn2hex(d));
218         // printf("X: %s\n", BN_bn2hex(x));
219         // printf("Y: %s\n", BN_bn2hex(y));
220         // printf("R: %s\n", BN_bn2hex(r));
221         // printf("S: %s\n", BN_bn2hex(s));
222
223         BN_bn2bin(r, (unsigned char *) pFwInfo->sign.au32R);
224         BN_bn2bin(s, (unsigned char *) pFwInfo->sign.au32S);
225         BN_free(x);

```

```

226     BN_free(y);
227     BN_free(d);
228     BN_free(r);
229     BN_free(s);
230 }
231 }
232 ret = 0;
233 exit: EC_GROUP_free(ecgroup);
234 EC_KEY_free(eckey);
235
236 return ret;
237 }
238
239 int getFileSize(const char *filename)
240 {
241     int size = -1;
242
243     FILE* fd = fopen(filename, "rb");
244     if (!fd) {
245         printf("Failed to open file: %s\n", filename);
246         return -1;
247     }
248
249     /* Get file size */
250     if (fseek(fd, 0, SEEK_END) != 0) {
251         printf("Unable to get '%s' file size\n", filename);
252         goto exit;
253     }
254
255     size = ftell(fd);
256     if (size < 0) {
257         printf("Stream doesn't support '%s' file positioning\n", filename);
258         goto exit;
259     }
260     // rewind(fd);
261 exit:
262     fclose(fd);
263     return size;
264 }
265 #ifndef CONFIG_BOOTLOADER2
266 /**
267  * @brief          printFwInfo - .
268  *
269  * @param pFwInfo [in/out] A pointer to jar file name string.
270  *
271  * @returns        0 on success or error code on failure.
272  */
273 void printFwInfo(FW_INFO_T *pFwInfo)
274 {

```

```

275     printf("%s", header);
276     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->pubkey.au32Key0[0],
277         pFwInfo->pubkey.au32Key0[1], pFwInfo->pubkey.au32Key0[2],
278         pFwInfo->pubkey.au32Key0[3]);
279     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->pubkey.au32Key0[4],
280         pFwInfo->pubkey.au32Key0[5], pFwInfo->pubkey.au32Key0[6],
281         pFwInfo->pubkey.au32Key0[7]);
282     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->pubkey.au32Key1[0],
283         pFwInfo->pubkey.au32Key1[1], pFwInfo->pubkey.au32Key1[2],
284         pFwInfo->pubkey.au32Key1[3]);
285     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->pubkey.au32Key1[4],
286         pFwInfo->pubkey.au32Key1[5], pFwInfo->pubkey.au32Key1[6],
287         pFwInfo->pubkey.au32Key1[7]);
288     printf("\n");
289     printf(
290         " /* metadata data - includes Mode selection, F/W region and Extend info */\n");
291     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x, // 0x0020000: NuBL32 F/W base\n",
292         pFwInfo->mData.u32AuthCFGs, pFwInfo->mData.u32FwRegionLen,
293         pFwInfo->mData.au32FwRegion[0].u32Start,
294         pFwInfo->mData.au32FwRegion[0].u32Size);
295     printf(
296         "  0x%08x, 0x%08x, 0x%08x, 0x%08x, // 0x20180824/0x00001111/0x22223333: Extend info\n",
297         pFwInfo->mData.u32ExtInfoLen, pFwInfo->mData.au32ExtInfo[0],
298         pFwInfo->mData.au32ExtInfo[1], pFwInfo->mData.au32ExtInfo[2]);
299     printf("\n");
300     printf(" /* FW hash - 32-bytes (256-bits) */\n");
301     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->au32FwHash[0],
302         pFwInfo->au32FwHash[1], pFwInfo->au32FwHash[2], pFwInfo->au32FwHash[3]);
303     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->au32FwHash[4],
304         pFwInfo->au32FwHash[5], pFwInfo->au32FwHash[6], pFwInfo->au32FwHash[7]);
305     printf("\n");
306     printf(" /* FwInfo signature - 64-bytes (256-bits R + 256-bits S) */\n");
307     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->sign.au32R[0],
308         pFwInfo->sign.au32R[1], pFwInfo->sign.au32R[2], pFwInfo->sign.au32R[3]);
309     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->sign.au32R[4],
310         pFwInfo->sign.au32R[5], pFwInfo->sign.au32R[6], pFwInfo->sign.au32R[7]);
311     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->sign.au32S[0],
312         pFwInfo->sign.au32S[1], pFwInfo->sign.au32S[2], pFwInfo->sign.au32S[3]);
313     printf("  0x%08x, 0x%08x, 0x%08x, 0x%08x,\n", pFwInfo->sign.au32S[4],
314         pFwInfo->sign.au32S[5], pFwInfo->sign.au32S[6], pFwInfo->sign.au32S[7]);
315     printf(");\n");
316 }
317 #endif
318 /**
319  * @brief      main - entry point of mTower: secure world.
320  *
321  * @param      None
322  *
323  * @returns    None (function is not supposed to return)

```

```

324     */
325     int main(int argc, char** argv)
326     {
327         unsigned char *buf = NULL;
328         int ret = -1;
329         FILE* fd;
330         int img_size = 0;
331
332         #ifdef CONFIG_BOOTLOADER2
333
334         ECC_KEY_T ecdsa_key;
335         FW_INFO_T pFwInfo;
336
337         pFwInfo.mData.u32AuthCFGs = 0x00000001;
338         pFwInfo.mData.u32FwRegionLen = 0x00000008;
339         pFwInfo.mData.au32FwRegion[0].u32Start = CONFIG_START_ADDRESS_BL32;
340         pFwInfo.mData.au32FwRegion[0].u32Size = 0x00000000;
341         pFwInfo.mData.u32ExtInfoLen = 0x0000000c;
342         pFwInfo.mData.au32ExtInfo[0] = DATE_COMPILE;
343         pFwInfo.mData.au32ExtInfo[1] = 0x00001111;
344         pFwInfo.mData.au32ExtInfo[2] = 0x22223333;
345     #endif
346
347     if (argc != 7) {
348         printf("Not enough input parameters for %s\n", argv[0]);
349         return -1;
350     }
351
352     buf = malloc(1024 * 256);
353     if (buf == NULL) {
354         printf("Allocation memory error\n");
355         goto exit;
356     }
357     memset((void *) buf, 0, 1024 * 256);
358
359     //////////////////////////////////////
360     // mtower_s.bin
361     //////////////////////////////////////
362
363     #ifdef CONFIG_BOOTLOADER2
364         int32_t size_bl2;
365         if((size_bl2 = getFileSize(argv[1])) < 0) {
366             return -1;
367         }
368
369         fd = fopen(argv[1], "rb");
370         if (!fd) {
371             printf("Failed to open file: %s\n", argv[1]);
372             return -1;

```



```

373     }
374
375     if (fread(buf, 1, (size_t) size_bl2, fd) != (size_t) size_bl2) {
376         free(buf);
377         goto exit;
378     }
379     fclose(fd);
380
381 #endif
382
383 #ifdef CONFIG_BOOTLOADER32
384     int32_t size_bl32;
385     if((size_bl32 = getFileSize(argv[2])) < 0) {
386         return -1;
387     }
388
389     fd = fopen(argv[2], "rb");
390     if (!fd) {
391         printf("Failed to open file: %s\n",argv[2]);
392         goto exit;
393     }
394     if (fread(buf + CONFIG_START_ADDRESS_BL32, 1, (size_t) size_bl32, fd) != (size_t) size_bl32)
395     {
396         free(buf);
397         goto exit;
398     }
399     fclose(fd);
400     img_size = size_bl32;
401 #endif
402
403 #ifdef CONFIG_BOOTLOADER2
404     sha256(buf + CONFIG_START_ADDRESS_BL32, size_bl32, (unsigned char *) &pFwInfo.au32FwHash);
405     pFwInfo.mData.au32FwRegion[0].u32Size = size_bl32;
406
407     fd = fopen(argv[4], "rb");
408     if (!fd) {
409         printf("Failed to open file: %s\n",argv[4]);
410         goto exit;
411     }
412
413     if (fread((unsigned char *) &ecdsa_key, sizeof(char), sizeof(ECC_KEY_T), fd)
414         != sizeof(ECC_KEY_T))
415     {
416         return -1;
417     }
418     fclose(fd);
419
420     sign_pFwInfo(&pFwInfo, &ecdsa_key);
421

```

```

422     memcpy(buf + 0x00038000, (unsigned char *) &pFwInfo, sizeof(FW_INFO_T));
423     img_size = 0x00038000 + sizeof(FW_INFO_T);
424 #endif
425
426     fd = fopen(argv[3], "wb");
427     if (!fd) {
428         printf("Failed to open file: %s\n", argv[3]);
429         goto exit;
430     }
431
432     if (fwrite(buf, sizeof(char), (size_t) img_size, fd) != (size_t) img_size) {
433         free(buf);
434         goto exit;
435     }
436
437 #ifdef CONFIG_BOOTLOADER2
438     printf("\n\tSecure firmware info\n");
439     printFwInfo(&pFwInfo);
440 #endif
441
442 ///////////////////////////////////////////////////
443 // mtower_ns.bin
444 ///////////////////////////////////////////////////
445
446 #ifdef CONFIG_BOOTLOADER33
447     memset((void *) buf, 0, 1024 * 256);
448
449     int32_t size_bl33;
450     if((size_bl33 = getFileSize(argv[5])) < 0) {
451         return -1;
452     }
453
454     fd = fopen(argv[5], "rb");
455     if (!fd) {
456         printf("Failed to open file: %s\n", argv[5]);
457         goto exit;
458     }
459     if (fread(buf, 1, (size_t) size_bl33, fd) != (size_t) size_bl33)
460     {
461         goto exit;
462     }
463     fclose(fd);
464
465     img_size = size_bl33;
466
467 #ifdef CONFIG_BOOTLOADER2
468     sha256(buf, size_bl33, (unsigned char *) &pFwInfo.au32FwHash);
469     pFwInfo.mData.au32FwRegion[0].u32Start = CONFIG_START_ADDRESS_BL33;
470     pFwInfo.mData.au32FwRegion[0].u32Size = size_bl33;

```

```
471
472     sign_pFwInfo(&pFwInfo, &ecdsa_key);
473     memcpy(buf + BL33_METADATA_ADDRESS, (unsigned char *) &pFwInfo, sizeof(FW_INFO_T));
474     img_size = BL33_METADATA_ADDRESS + sizeof(FW_INFO_T);
475 #endif
476
477     fd = fopen(argv[6], "wb");
478     if (!fd) {
479         printf("Failed to open file: %s\n", argv[6]);
480         goto exit;
481     }
482
483     if (fwrite(buf, sizeof(char), (size_t) img_size, fd) != (size_t) img_size) {
484         goto exit;
485     }
486 #endif
487     ret = 0;
488 exit:
489     free(buf);
490     fclose(fd);
491
492 #ifdef CONFIG_BOOTLOADER2
493     printf("\n\tNon-secure firmware info\n");
494     printFwInfo(&pFwInfo);
495 #endif
496
497     return ret;
498 }
499
```