universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Verifiable Election with SGX Enclaves

Cybersecurity MCs: Secure Execution Environments

David Gomes Morais, 93147

Junho, 2022
DETI, Universidade de Aveiro

Introdução

O presente relatório tem como objetivo explicar e documentar as decisões tomadas na implementação de um sistema de votação verificável, composto por uma *Ballot application* e uma *Voter application*, ambas empregando *SGX enclaves*, desenvolvido no âmbito do segundo projeto da cadeira de Ambientes de Execução Seguros.

Para tal foi desenvolvida uma enclave **Voter** (e consequentemente uma aplicação que comunica com esta) que permite a geração de credenciais, exportação da chave pública a partir das mesmas e fazer o *cast* de um voto, selado e assinado de forma a ser válido. Por outro lado, foi desenvolvida uma enclave **Ballot** (e consequente aplicação) que permite a geração de credenciais e exportação da respectiva chave pública, em semelhança à préviamente mencionada, assim como ações para executar uma eleição e para verificar os resultados com base nos votos dados pelos *voters*.

Além disso, foram também criados *scripts* de teste em *python* de forma a automatizar e simplificar o processo de testagem e geração de dados em peso, devido a interface de ambas as aplicações ser a linha de comandos, sem qualquer *input* por parte do utilizador. É de fazer notar que as aplicações que envolvem enclaves foram construídas a partir do *Sample code* de exemplo presente em no *sgxsdk* no diretório *SampleCode/SampleEnclave*.

Voter enclave

O objetivo da enclave *Voter* é permitir, através da ação GEN, a criação de credenciais, que consistem num par de chaves assimétricas - a *hash* da *public key* e a *private key* -, derivadas a partir de uma *password* dada pelo utilizador como parâmetro e que vão ser armazenadas no *filesystem* de forma selada para manterem a autenticidade e confidencialidade dos dados. Além disso, através da ação PUB, é possível exportar a *hash* da *public key*, a partir das credenciais dadas pelo utilizador como parâmetro, cujo processo passa pelo *unseal* das credenciais usando a *password* do utilizador, e armazenar a chave pública no sistema de ficheiros. Por último, permite que um *voter* faça o *cast* de um voto através da ação VOTE, isto é, uma *string*, encriptada com a *public key* do *ballot* e assinada com a chave pública do *voter* por forma a manter a confidencialidade do voto.

Design da interface

Após se terem ponderado sobre os requisitos da aplicação, a interface da enclave foi desenhada com base nas três ações que esta permitia, desta forma foram criadas três funções *ecalls* para se entrar para a enclave e se executarem operações dentro desta. Os protótipos das mesmas estão presente no ficheiro *Voter/Enclave/Enclave.edl* (Fig. 1) fazendo uso da sintaxe *EDL* (*Enclave Definition Language*) e inclui *ecall_gen_credentials* usada para despoletar o processo de geração de credenciais, que leva como argumento uma *string* representativa da *password*, e é chamada quando a ação GEN é escolhida pelo utilizador. Além disso, inclui também uma *ecall* para exportar a chave pública presente nas credenciais seladas (*ecall_unseal_and_export_pub*) que leva como argumentos um *buffer* que contém as *sealed credentials* e o seu tamanho, assim como uma *string* representativa da *password* para que estas possam ser *unsealed*, e é chamada quando o método PUB é executado. Por fim, existem *ecall_produce_vote* que dado uma *string* representativa do voto, o *buffer* e o tamanho das credenciais

seladas, a *string* contendo a *password* e o *buffer* e o tamanho contendo a *public key* do *voter* (exportada através da ação PUB), que permite que seja feito o *cast* de um voto pelo utilizador.

Em seguida foram definidas os protótipos das funções *ocalls* que saem da enclave e são chamadas dentro das *ecalls* em si, sendo que são utilizadas maioritariamente para extrair dados das mesmas e estão definidas no mesmo ficheiro *.edl*, incluindo funções para fazer *prints* na linha de comandos - *ecall_print_string* que tem como argumento a *string* a ser imprimida - e para escrever ficheiros no *filesystem* - ecall_write_file, que tem como argumentos o buffer a escrever no ficheiro, o seu tamanho, o diretório em qual o armazenar e a sua extensão (uma vez que o nome do ficheiro é um UUID gerado aleatoriamente para evitar colisões, não há necessidade de o passar como argumento). A única exceção a esta norma é a *ocall_derive_key*, que é usada para obter uma chave por derivação da *password* dada pelo utilizador (que faz uso do método *PKCS5_PBKDF2_HMAC_SHA1* da *untrusted library* ***openssl/crypt***) e leva como argumentos um *buffer* de *output* onde colocar a chave derivada, assim como o seu tamanho, o *salt* usado e a *password* proporcionada pelo utilizador.

```
53      trusted {
54          public void ecall_gen_credentials([in, string] char* str);
55          public void ecall_unseal_and_export_pub(
56              [in, size=sealedLen] uint8_t* sealed, size_t sealedLen,
57              [in, string] char* password
58          );
59          public void ecall_produce_vote(
60              [in, string] char* vote,
61              [in, size=sealed_size] uint8_t* sealed_data, size_t sealed_size,
62              [in, string] char* password,
63              [in, size=pubKeyLen] uint8_t* pubKey, size_t pubKeyLen
64          );
65      };
66
67      untrusted {
68          void ocall_print_string([in, string] const char *str);
69          void ocall_derive_key(
70              [in, out, size=key_len] uint8_t* key, size_t key_len,
71              char salt, [in, string] char* password
72          );
73          void ocall_write_file(
74              [in, size=size] uint8_t* data, size_t size,
75              [in, string] const char* directory, [in, string] const char* extension
76          );
77      };
```

Fig. 1. Snippet de Voter/Enclave/Enclave.edl, onde são definidos os protótipos das funções ecall e ocall que entram e saem da enclave, respetivamente, nas partes trusted e untrusted.

Em relação às estruturas de dados utilizados em ambas as enclaves, é de fazer notar que as credenciais geradas e armazenadas estão na estrutura representada na Fig. 2, e foram geradas usando *RSA-3072*, daí o tamanho em *bytes* do módulo e do expoente privado serem 384. Os primeiros 32 *bytes* são a *hash* da *public key*, enquanto que a *private key* está numa estrutura semelhante ao de *_sgx_rsa3072_key_t*, com o módulo N, seguido do expoente privado d e do expoente público e. As partes públicas e privadas da chave estão divididas pelo byte com valor '\n'.

As credenciais são seladas através da cifra de *rijndael-128* em modo *GCM*, e o seu formato está também representado na Fig.2, onde os primeiros 12 *bytes* são o *IV* usado, seguido dos dados encriptados e fechado com os 16 *bytes* do *MAC* para verificação. Por último, quando o voto é *casted*, este é guardado de tal forma que os primeiros 416 *bytes* são cifrados usando *RSA* com a chave pública do *ballot* (os primeiros 32 são o identificador do *voter*, e os restantes são o voto que tem um tamanho

de 352 *bytes* devido ao tamanho inerente do RSA). A assinatura de 384 *bytes* que o sucede é feita sob os dados cifrados, utilizando também *RSA-3072* mas com a chave privada do *voter*.

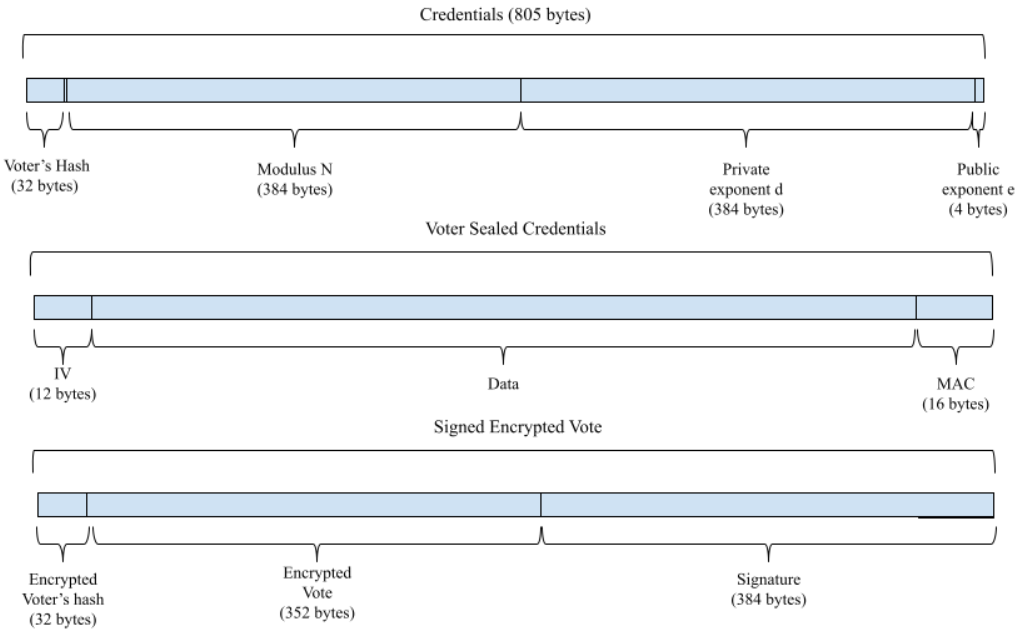

Fig. 2. Estruturas de dados usados em ambas as enclaves, representando, respetivamente, as credenciais usadas, as credenciais seladas para serem guardadas no sistema de ficheiros e o voto assinado e cifrado para ser armazenado.

Por fim, a interface da aplicação com a qual o utilizador interage não requer qualquer *inputs* em *run time*, apenas têm de ser passados os parâmetros corretos pela linha de comandos. A *flag* -a define a ação a ser executada, podendo tomar o valor de GEN, PUB e VOTE, a *flag* -p define a *password* e tem que ser usada em todas as ações por forma a permitir fazer o *seal* e *unseal* dos dados, a *flag* -c define o *filename* onde as credenciais estão armazenadas e é necessária para a ação PUB e VOTE e, por fim, a *flag* -v define a string do voto e é necessária para a ação VOTE.

Implementação da enclave

Algumas considerações que valham a pena fazer notar na implementação são que a criação do par de chaves que agem como credenciais foram gerados com o método *sgx_create_rsa_key_pair*, passando os tamanhos inerentes à cifra *RSA-3072*, o que inclui um tamanho do módulo e de expoente privado de 384 *bytes*, e o tamanho de um expoente público de 4 *bytes*. Quando as chaves foram obtidas, gerou-se a *hash* da chave pública para agir como identificador usando o método *sgx_sha256_init*, com dois *updates* (cada um com 194 *bytes*) uma vez o método *sgx_sha384_msg* só permitia mensagens de 256 bytes ou menor.

Por forma a que as credenciais pudessem ser armazenadas de forma segura e confiável no *filesystem*, houve a necessidade de as selar e, para isso, usou-se a cifra simétrica de *rijndael-128* no modo *GCM sgx_rijndael128GCM_encrypt*, usando uma *key* (derivada de uma *password* dada pelo utilizador e um *salt* 0), que foi depois armazenada no *filesystem* no ficheiro */creds/x.seal*, onde *x* representa um UUID aleatório com 36 *bytes*.

Para se fazer a exportação do identificado (*hash*) da chave pública, foi feito a decifra das credenciais armazenadas no sistema, através de *sgx_rijndael128GCM_decrypt*, da qual os primeiros 32 *bytes* que

continham o *hash* foram armazenados no ficheiro */keys/y*, onde *y* representa um UUID aleatório também com 36 *bytes*.

Por forma a produzir um voto, o mesmo descrito acima foi executado para fazer o *unseal* das credenciais. A chave pública do *ballot* foi também lida e esta foi posta numa estrutura de dados para ser usado pelo *encryptor* (através do método *sgx_create_rsa_pub1_key*). Foi definido um chamado "voto identificável" que consta no *hash* seguido do texto do voto em si, e este foi cifrado usando *RSA* e a chave pública do *ballot*, através do método *sgx_rsa_pub_encrypt_sha256.* Por último, esta cifra foi assinada usando a chave privada do *voter*, e a assinatura foi *appended* ao fim da cifra para servir de verificação, sendo depois armazenada localmente no sistema de ficheiros em */votes/z.vote*, onde *z* representa um UUID aleatório com 36 *bytes*. É de fazer notar que todos os métodos mencionados nesta seção (identificáveis pelo começo em sgx_) pertencem a trusted library da enclave responsável por operações criptográficas

Ballot enclave

A enclave *Ballot* tem duas ações muito semelhantes à enclave *Voter* no que toca à geração de credenciais e exportação da chave pública a partir da mesma, de forma que o processo feito nesta aplicação espelha em tudo o que foi mencionado previamente. O único desvio está explicado na subseção da implementação e recai sob a forma de fazer o *seal* das credenciais para serem armazenados que, em vez de ser *password-based*, foca-se no valor do *MRENCLAVE* inerente à enclave em si, que permite detectar mudanças nesta, ou seja só uma versão específica da enclave a correr com um *CPU* específico pode fazer o *unseal* de dados selados pela mesma.

Além disso, a enclave permite também, através da ação RUN, executar um processo de votação, onde são carregadas as credenciais do *ballot*, assim como as chaves públicas de voters autorizados (presentes no diretório */keys*) e dos *casted votes* (presentes no diretório */votes*). Posto isto, os votos são validados baseado na decifra dos mesmos com a chave pública do *ballot* e da verificação se estes contêm (ou não) o *identifier* de um *voter* autorizado nos primeiros 32 *bytes* do voto. A lista de votos válidos é depois contada e *randomly sorted* por forma a não se saber a ordem de que *voter* fez que vote.

Por fim, a ação CHECK permite verificar se um voto de um dado *voter v* foi ou não contado para o resultado final, onde o processo é semelhante ao da ação RUN, mas em vez de se fazer o *random sort* da lista de votos, esta é percorrida e é verificado se existe um voto identificado com *v*, e se *v* está contido na lista de voters autorizados. O resultado pode ser um de três possíveis - o voto não foi *casted*, o voto foi *casted* mas não foi verificado e o voto foi *casted* e verificado.

Design da interface

As *ocalls* presentes nesta enclave são semelhantes à das mencionadas para a enclave anterior, assim como as primeiras duas *ecalls* (*ecall_gen_credentials* e *ecall_unseal_and_export_pub*) que permitem fazer a geração e exportação de credenciais e chaves públicas respetivamente. A única diferença reside nos argumentos destas duas últimas, onde não existe a necessidade de passar a *password* devido ao facto de o processo de selagem não ser *password-based*. Existe também uma *ecall* para despoletar o processo de correr uma *election* através de *ecall_run_election*, que tem como argumentos o *buffer* e

o tamanho das credenciais seladas do *ballot*, um *buffer* a conter as *public keys* de *voters* autorizados e o seu tamanho e por fim um *buffer* que contém a lista de *casted votes* e o seu tamanho.

A função *ecall_check_voter* está encarregue de começar o processo associado com a ação CHECK, levando os mesmos argumentos que a *ecall* mencionada acima (visto as suas funcionalidades serem relativamente semelhantes), mas tem como argumentos adicionais um *buffer* que contém a chave pública do *voter* que se pretende verificar, assim como o seu tamanho.

Em termos das estruturas de dados usadas, são as mesmas mencionadas na seção acima, visto que grande parte das operações de ambas as enclaves têm dados (e tipos de dados) em comum. Para terminar, a aplicação tem também uma interface na linha de comandos sem *input* do utilizador em *runtime*, sendo apenas passar os argumentos corretos através de *flags* quando se inicia a aplicação. A *flag* -a define a ação a ser executada, podendo tomar o valor de GEN, PUB, VOTE e CHECK, a *flag* -p define o ficheiro na qual está armazenada a *public key* do *voter* que pretende verificar o estado do seu voto.

Implementação da enclave

Para a geração de um par de chaves que vão servir de credenciais, uma estratégia semelhante à enclave *Voter* foi usada, como já tinha sido mencionado, a única diferença está no facto de, em vez de serem seladas através de uma cifra simétrica de *rijndael-128*, é feito através da função *sgx_seal_data* (Fig. 3), que usa o valor do *MRENCLAVE* para selar os dados presentes no *buffer cred*, sendo primeiro necessário a criação de uma estrutura de dados apropriada (*sgx_sealed_data_t*), usando o auxilio de *sgx_calc_sealed_data_size* para calcular o tamanho apropriado a alocar. Após isto, as credenciais são armazenadas ficheiro Ballot/ballot.seal, através do uso da *ocall_write_file*.

```
123     // seal the ballot credentials (using MRSIGNER)
124     size_t sealed_size = sgx_calc_sealed_data_size(0, (uint32_t) credLen);
125     uint8_t* sealed_data = (uint8_t*) malloc(sealed_size);
126
127     sgx_status = sgx_seal_data(0, NULL, (uint32_t) credLen, cred, (uint32_t) sealed_size, (sgx_sealed_data_t*) sealed_data);
128     free(cred);
129     if (sgx_status != SGX_SUCCESS) {
130         printf("ERROR while sealing the credentials %d\n", sgx_status);
131         return;
132     }
...
143     size_t macLen = sgx_get_add_mac_txt_len((sgx_sealed_data_t*) sealed);
144     size_t credLen = sgx_get_encrypt_txt_len((sgx_sealed_data_t*) sealed);
145
146     uint8_t* cred = (uint8_t*) malloc(credLen);
147     sgx_status_t sgx_status = sgx_unseal_data((sgx_sealed_data_t*) sealed, NULL, (uint32_t*)&macLen, cred, (uint32_t*)&credLen);
148     if (sgx_status != SGX_SUCCESS) {
149         printf("Failed to unseal data\n");
150         return;
151     }
```

Fig. 3. Snippet do código de Ballot/Enclave/Enclave.cpp, onde é feito o seal e unseal das credenciais da ballot box, respetivamente.

Para fazer a exportação da chave pública, é primeiro alocado espaço necessário para o *MAC* e para o *plaintext* através de *sgx_get_add_mac_txt_len* e *sgx_get_encrypt_txt_len* respectivamente, sendo posteriormente feito uma chamada à função *sgx_unseal_data*. Esta função faz a verificação da integridade através do *MAC*, sendo que não há necessidade de nós a termos de fazer manualmente e, após ser extraída a *public key* das credenciais da *ballot box*, estas são armazenadas no ficheiro *Ballot/ballot*.

Para se correr uma eleição existe a necessidade de obter as chaves publicas de todos os *voters*, assim como os votos armazenados e a credenciais da *ballot box*, sendo utilizado sempre o mesmo método para fazer o *unseal* destas últimas. De seguida, por cada voto que se encontra na lista, este é decifrado usando *sgx_rsa_priv_decrypt_sha256* e a chave privada da *ballot box*, sendo depois necessário

verificar se o *voter* é ou não autorizado, através da função *isAuthorizedVoter* (Fig.4). Esta função o que faz é comparar o identificador que se encontra nos primeiros 32 *bytes* do voto com todos as *public keys* pertencentes a voters autorizados, sendo que quando encontra um *match*, retorna *True*.

```
73
74  int isAuthorizedVoter(uint8_t* publicKeyInVote, size_t keyLen, uint8_t* listOfAuthorizedVoters, size_t listLen) {
75      if (publicKeyInVote == NULL || keyLen <= 0 || listOfAuthorizedVoters == NULL || listLen <= 0) return 0;
76
77      int numAuthVoters = (int) listLen / SGX_SHA256_HASH_SIZE;
78      for (int i = 0; i<numAuthVoters; i++) {
79
80          if (comparePublicKeys(publicKeyInVote, SGX_SHA256_HASH_SIZE,
81                  listOfAuthorizedVoters + i*SGX_SHA256_HASH_SIZE, SGX_SHA256_HASH_SIZE)) {
82              return 1;
83          }
84
85      }
86      return 0;
87  }
88
```

Fig. 4. Função que verifica se um dado voter está entre a lista de authorized voters através do seu identificador.

Caso não seja autorizado, o voto é excluído sem qualquer *feedback*. Posteriormente os votos autorizados são contados e a lista de votos autorizados é depois *shuffled* com base num *array r* com tantos *bytes* random como votos verificados, gerados por *sgx_read_rand*. O *shuffled* é feito de tal forma que por cada posição da lista de votos, o voto é trocado com o na posição dada por cada *byte* random, sendo que esta lista shuffled faz com que os resultados estejam sempre numa ordem diferente e é posteriormente imprimida no terminal através da *ocall* responsável por fazer *printf* fora da enclave.

```
227     // provide a list of votes in a random order
228     if (validVoteCount > 1) {
229         uint8_t* tmp = (uint8_t*) malloc(size);
230         size_t i;
231         uint16_t* r = (uint16_t*) malloc(validVoteCount);
232         sgx_status = sgx_read_rand((uint8_t*) r, validVoteCount*sizeof(uint16_t));
233
234         for (i=0; i<(size_t) validVoteCount; ++i) {
235             size_t rnd = (size_t) r[i];
236             size_t j = i + rnd / (UINT16_MAX / (validVoteCount-i) + 1);
237
238             memcpy(tmp, shuffledVotes + j * size, size);
239             memcpy(shuffledVotes + j *size, shuffledVotes + i * size, size);
240             memcpy(shuffledVotes + i * size, tmp, size);
241         }
242     }
```

Fig. 5. Snippet do código de Ballot/Enclave/Enclave.cpp, responsável pelo shuffle dos resultados da votação.

O *flow* da ação CHECK é igual ao de RUN mas, antes de se verificar se o voto é válido verifica-se se a *public key* dada como parâmetro é igual à presente nos primeiros 32 *bytes* do voto. Se nenhum voto tiver esta *public key*, então o voto não foi *casted*, caso contrário depende do resultado da função *isAuthorizedVoter*, que se retornar *True* significa que o voto foi contado para o total. Um dos três possíveis resultados é então imprimido para o *stdout* através da *ocall* associada ao *printf*.

Testes e limitações

Para testar as enclaves desenvolvidas foram criados dois *scripts* em *python* de testagem, um focado no *voter* e outro focado no *ballot* (*tests/test_Voter.py* e *tests/test_Ballot.py*, respectivamente) onde se exploram alguns *use cases* que possam levar a erros, assim como criação de *script* de batch para criar 100 credenciais, exportar as suas chaves privadas e criar 100 votos, assim como para correr uma eleição 20 vezes.

Os testes da enclave *voter* contêm casos tais como a criação de credenciais com *passwords* válidas e inválidas, exportação de chaves públicas a partir de certificados válidos e *tampered*, exportação de chaves públicas quando a *password* incorreta, fazer votos com credenciais corretas, assim como com credenciais *tampared* e *passwords* erradas/inválidas. Este script pode ser corrido com *pytests* (Fig. 6) onde os testes são corridos consecutivamente, usando funções teardown para eliminar a lista de credenciais, chaves e votos. Além disso o *script* pode ser corrido também através da *command line* para despoletar a criação de 100 credenciais, com as respectivas chaves públicas e votos.

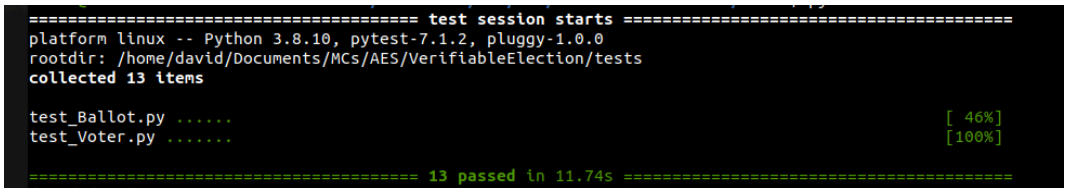

Fig. 6. Output dos testes usando os scripts no diretório /tests, em conjunto com pytest.

Os testes da enclave ballot são semelhantes, com *use cases* de criação de credenciais, exportação de chaves públicas com credenciais válidas e *tampered*, assim como correr uma votação com os mesmos *inputs* e garantir que a contagem de votos é sempre a mesma, porém que a ordem da lista de *output* varia conforme a randomização oferecida pela enclave. No entanto é de fazer notar que este teste pode falhar devido a haver uma pequena probabilidade de a ordem ser igual, visto haver um número finito de combinações possíveis, especialmente com apenas duas opções de voto. A mesma coisa pode ser dita da execução deste *script*, ou seja pode ser executado através do *pytest*, onde os testes correm em consecutivo, sem funções de *tear down* (não se justificaram), porém estes testes devem ser feitos num ambiente próprio, uma vez que as credenciais e a chave pública do ballot box vai ser *overwritten*, e se houver votos cifrados com a mesma já dentro do diretório */votes*, estes passaram a ser inválido no ambiente de "produção". O script pode ser também corrido diretamente, o que irá criar 100 credenciais, exportar 100 chaves públicas e fazer o *cast* de 100 votos, e irá correr uma eleição 20 vezes para garantir que a contagem se mantenha, mas a ordem relativa muda.

Limitações possíveis incluem, talvez e devido à sua importância, a execução da derivação de uma *key* a partir de uma *password* proporcionada pelo utilizador fora da enclave, uma vez que o objetivo da enclave era não confiar no CPU nem na memória, fazer esta operação fora da mesma pode ter potenciais consequências, no entanto nenhuma das *libraries trusted* de criptografia tinha algo adequado para a geração de chaves *password-based*.