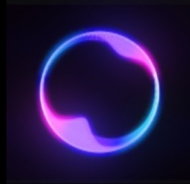
4. Sealing

Ao Sakurai

2023年度セキュリティキャンプ全国大会
L5 - TEEの活用と攻撃実践ゼミ

本セクションの目標

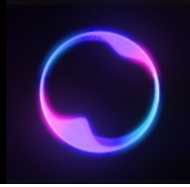


- Enclave内のデータを安全な状態でEnclave外（EPC外メモリや補助記憶装置）にストアする**シーリング機能**について習得する

- ここまでで学習した内容をフルに活用し、**SGXを利用したパスワード管理システム**として「**SGX-Vault**」を実装する

シーリング/アンシーリング

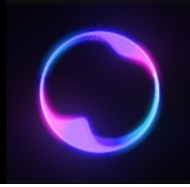
Enclaveは揮発性（1/3）



- Enclaveは**メインメモリ（主記憶装置）**上に存在するため、
  あくまでもその内容は**揮発性**である

- **電源OFF**に加え、様々なイベントにより**Enclaveは破壊される**
  - sgx_destroy_enclave()の呼び出しによる明示的な破壊
  - Enclaveを使用したプログラムの終了
  - 電源状態の遷移（スリープや休止状態）
  - コードのバグ等に伴うEnclaveのクラッシュ
  - etc...

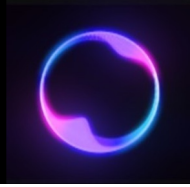
Enclaveは揮発性（2/3）



- Enclave**実行中**はもちろん、Enclaveが**終了・破壊された後**も**秘密情報**を**SGXマシンで安全に保持したい**ケースはかなり多い
  - メモリ節約の為一時的にハードディスクにストアしたい
  - SGXマシン上でパスワードを安全に管理したい
  - SGXマシン上のSQLサーバに安全な形でデータをストアしたい

- どうすればEnclaveの秘密情報を**暗号化して長期的に保存**できる？

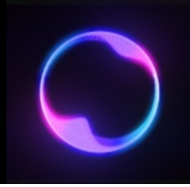
Enclaveは揮発性（3/3）



- Enclave内で鍵を生成して暗号化する？
  - **sgx_rijndael128GCM_encrypt()**関数等が提供されている
  - ではその**暗号化に使用した鍵**は**どうやって保持する？**

- このように、通常のやり方では**暗号鍵の堂々巡り問題**が発生し、秘密情報を**長期的にEnclave外に保存する事が難しい**

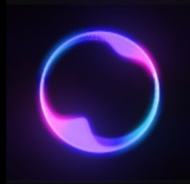
シーリング/アンシーリング（1/2）



- この問題を解決するため、SGXでは**シーリング**（Sealing）と呼ばれる**暗号化機能**が用意されている
  - シーリングされたデータの**復号**を**アンシーリング**（Unsealing）という

- シーリング及びアンシーリングでは、CPU製造時にCPU内部で生成される**Root Sealing key**（RSK）等と共に、**2通り**ある**ポリシ**のいずれかを用いて導出した**シーリング鍵**で暗号処理を行う
  - RSKはCPU内部の**e-fuse**と呼ばれるICにストアされており、**Intelすらも知らない鍵**である
  - RSKについてはProvisioningのセッションでも解説

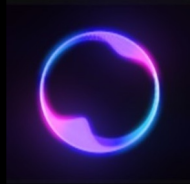
シーリング/アンシーリング（2/2）



- シーリング鍵はCPUのe-fuseに格納されているRSKを用いて導出されるため、**マシンが変わる**と**恒久的にアンシーリング出来なくなる**点に注意

- 他にも、**モデル固有レジスタ（MSR）**から変更できる**OWNEREPOCH**（現在の**マシン所有者**に**紐づくバージョン**のようなもの）にも依存するため、**これを変更**しても**互換性が失われる**

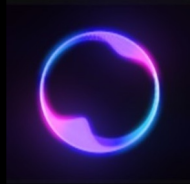
シーリングのポリシ（1/2）



- シーリングポリシには、Enclaveの**コード同一性**に基づく**MRENCLAVEポリシ**と、Enclave**署名者の同一性**に基づく**MRSIGNERポリシ**が存在する

- **MRENCLAVEポリシ**では、シーリングを行うEnclaveの**MRENCLAVE**やRSK等を材料にシーリング鍵を導出する
  - MRENCLAVEが変わるとシーリング鍵も変わるので、**Enclaveコード等に変更が入る**と**以前のシーリングデータはアンシーリングできなくなる**

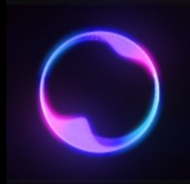
シーリングのポリシ（2/2）



- **MRSIGNERポリシ**では、シーリングを行うEnclaveの**MRSIGNER**やRSK等を材料にシーリング鍵を導出する
  - **Enclave署名鍵が同一**であれば、Enclaveコードが変わっても以前のシーリングデータを**アンシーリング出来る**

- この事から、**MRSIGNERポリシ**の方が**使い勝手は良い**一方で、**安全性を重視**するのであれば**MRENCLAVEポリシ**の方が良い
  - 危殆化したEnclaveを修正しMRENCLAVEを更新すれば、以前のデータはアンシーリング出来なくなるため

シーリング処理の実装（1/4）

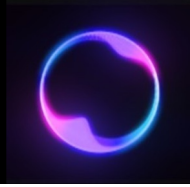


- **シーリングを行うAPI**を呼び出す際、引数として**シーリング結果のバイナリ**（sgx_sealed_data_t型変数）の**サイズをsize_t型で渡す**必要がある

- シーリングにも使用されている**AES/GCM**は、その**暗号文の長さ**は**平文の長さと一致**する

- しかし、シーリング結果のバイナリは**暗号文の他にも様々なメタデータを含む**ため、そのバイナリのサイズは**平文よりも大きくなり**、**かつ場合によって異なる**
  - 大抵約500バイトほど大きくなる
  - ではどうやってシーリングしたバイナリサイズを特定するのか？

シーリング処理の実装（2/4）

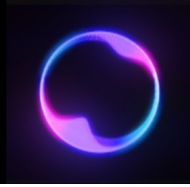


- 暗号化したい平文に対応するsgx_sealed_data_tのサイズを計算し返してくれる**sgx_calc_sealed_data_size**という**ヘルパー関数**を呼び出す
  - 第1引数はGCMの追加認証データのサイズ、第2引数はシーリングしたいデータ（平文）のサイズ。前者は通常0で良い

```
size_t sealed_data_size;
sealed_data_size = sgx_calc_sealed_data_size(0, data_length);
```

- この二度手間があるので、Enclaveで何度もシーリングを行う場合はこれを含む一連の処理を行う専用の関数を用意した方が実装上楽になる

シーリング処理の実装（3/4）

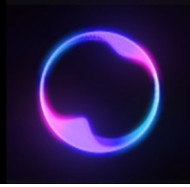


- シーリング結果のサイズを計算したら、シーリング本処理を行う **sgx_seal_data**を呼び出す
  - 戻り値はsgx_status_t（SGX関連処理の実行結果ステータス）
  - 引数は順にGCMの追加認証データの長さ、追加認証データ、シーリングするデータ（平文）のサイズ、シーリングするデータ、予め取得したシーリング結果サイズ、シーリング結果が格納されるポインタ
  - 追加認証データを使用しない場合は第1引数は0、その次はNULLで良い

```
status = sgx_seal_data(0, NULL, data_length, data_plain,
      sealed_data_size, (sgx_sealed_data_t*)sealed_data);
```

- 後はEnclave外にシーリング結果を出し、メモリで保持するなりファイルシステムにストアするなりする
  - 生のバイナリ値なので、Base64エンコード等しておくと扱いやすい

シーリング処理の実装（4/4）

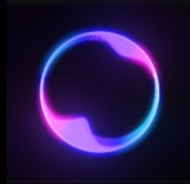


- sgx_seal_dataは**MRSIGNERポリシで固定**なので、MRENCLAVEポリシを使用したい場合は**sgx_seal_data_ex**を使用する
  - 以下の通りsgx_seal_dataに加え、引数先頭に**3変数**が追加される
  - 第1引数を**0x0001**にすると**MRENCLAVEポリシ**、**0x0002**にすると**MRSIGNERポリシ**となる
  - 第2・3引数はどうでもいいので以下の例のものを丸ごとコピペする

```
sgx_misc_select_t misc = 0xF0000000; //uint32_t
sgx_attributes_t attr;
attr.flags = 0xFF0000000000000B; //uint64_t
attr.xfrm = 0; //uint64_t

status = sgx_seal_data_ex(0x0001, attr, misc, 0, NULL,
     data_length, data_plain,sealed_data_size,
     (sgx_sealed_data_t*)sealed_data);
```

アンシーリング処理の実装



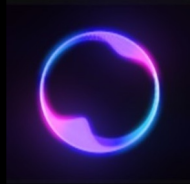
- まずはアンシーリングに向けて平文用のバッファを確保したいので、**sgx_get_encrypt_txt_len**というヘルパー関数で**平文の長さを取得**しておく

```
decrypt_buf_length = sgx_get_encrypt_txt_len((sgx_sealed_data_t*)sealed_data);
uint8_t decrypt_buf = new uint8_t[decrypt_buf_length]();
```

- あとは**sgx_unseal_data**を呼び出せば**アンシーリング完了**
  - 引数は順にシーリングデータ、追加認証データ、追加認証データの長さ、アンシーリング結果の平文が格納されるポインタ、アンシーリング結果の平文の長さが格納されるポインタ
  - アンシーリング関数では**ポリシを指定する関数**（sgx_unseal_data_ex のようなもの）は**存在しない**（**自動的にシーリングデータから識別される**）

```
status = sgx_unseal_data((sgx_sealed_data_t*)sealed_data, NULL, 0,
            decrypt_buf, &plaintext_length);
```

ちなみに



- シーリング (暗号化)を行う関数

```
sgx_seal_data(0, NULL, plain_len, plain,
    sealed_size, sealed_data);
```

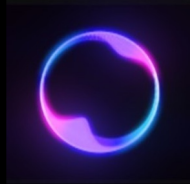
- アンシーリング (復号)を行う関数

```
sgx_unseal_data(sealed_data, NULL, 0, plain,
    plain_buf_size);
```

当たり前のように順序が逆

SGX-Vault

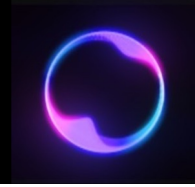
SGX-Vault



- SGXの保護機能をフルに活用した**パスワード管理ソフトウェア**として**SGX-Vault**の実装を行う
  - 名前は勝手に決めてあるだけなので、自分でつけたい場合改名しても全く問題ない

- 本ゼミではこのSGX-Vaultをセクションに応じて改良していく形で実装を進めていく

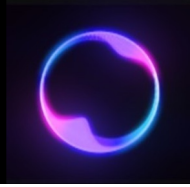
SGX-Vaultの要件



- SGX-Vaultは、パスワードの安全な取り扱いに関する以下の機能を提供する：
  - パスワード登録機能
  - パスワード新規作成機能
  - 登録したパスワードの取得機能
  - 登録したパスワードの変更機能
  - キー一覧取得機能
  - パスワード削除機能

- パスワードは**key-value**の形で保持し、keyをクエリする事でvalue（=対応するパスワード）を取得出来る
- 対応可能なパスワードの仕様（文字数、文字種）は**基本自由**だが、**最低限英数字には対応**する事

SGX-Vaultの各機能の要件（1/7）



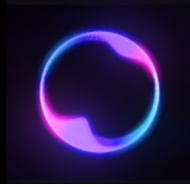
■マスターパスワードの設定・検証

- SGX-Vault初使用時は、以降の使用時に必須となる**マスターパスワード**の登録を実施する

- マスターパスワードは**シーリングにより暗号化**し**ファイルシステム**に保存する

- マスターパスワードが登録済みの場合、**後続の処理を行う前にその検証を行う**
  - UntrustedなAppで入力させ、それをEnclave内に読み込んで検証する

SGX-Vaultの各機能の要件（2/7）

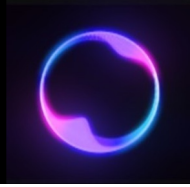


■パスワード登録機能

- **UntrustedなApp**から**パスワード**と対応する**キー**をEnclaveに読み込み、**シーリング**で暗号化し**ファイルシステムに保存**する

- 既存のパスワードに対する**キーとの重複が発生**した場合は、**登録せずにエラーを返す**

SGX-Vaultの各機能の要件（3/7）



■パスワード新規作成機能

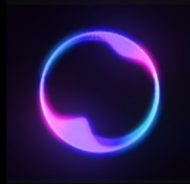
- 引数として渡されたキーに対し、パスワードを**Enclave内で乱数的に生成**してそのキーに対応させる

- **乱数的な生成は必ずEnclave内で完結しなければならない**

- 生成したキーとパスワードはシーリングしてファイル保管しておく

- 既存のパスワードに対する**キーとの重複が発生**した場合は、**新規作成せずにエラーを返す**

SGX-Vaultの各機能の要件（4/7）

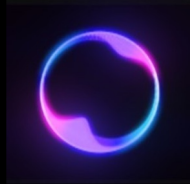


■**登録したパスワードの取得機能**

- 引数として渡されたキーに対応するパスワードを**シーリングしたデータから検索**し、その**パスワード**を**UntrustedなAppに返す**

- キーに**対応するパスワードが存在しなかった場合**は適切な例外（エラー）処理を行う

SGX-Vaultの各機能の要件（5/7）

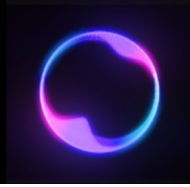


■**登録したパスワードの変更機能**

- 引数として渡されたキーに対応するパスワードを、同じく
  引数として渡された**新パスワードで更新する**

- キーに**対応するパスワードが存在しなかった場合**の処理は任意
  - エラーにするか、新規作成と同様の挙動にするかは自由

SGX-Vaultの各機能の要件（6/7）

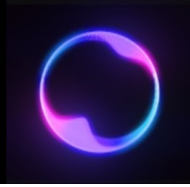


■キー一覧取得機能

- パスワードの登録に際してどのようなキーで登録したかを失念した場合に備え、登録した**全パスワードに対応する各キー**を**一覧化して取得**する

- あくまで**キーの取得**であり、**パスワード自体**は**ここでは取得しない**点に注意

SGX-Vaultの各機能の要件（7/7）

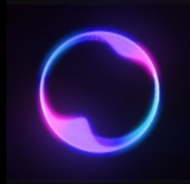


■パスワードの削除機能

- 引数として渡された**キー**と、それに対応する**パスワード**を**SGX-Vault（≒シーリングデータ）**から**削除する**

- キーに**対応するパスワードが存在しなかった場合**は適切に対処する

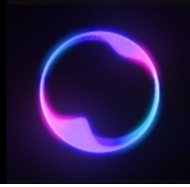
SGX-Vault実装のヒント



- 改行区切りのstd::string型にて、さらにカンマなどをデリミタとしてkey-valueを管理するのが楽

- Enclave内におけるパスワードの乱数的生成には、sgx_read_rand関数とアスキー文字の組み合わせが使えるはずである

本セクションのまとめ



- SGXにおいて長期的に安全にデータを保持するために欠かせない機能であるシーリング機能について学習した

- これまで学習したSGXの知識を活用し、ローカルで動作するSGXベースのパスワード管理システムであるSGX-Vaultを実装した

参考文献

[1]”Intel® Software Guard Extensions Programming Reference”, Intel, https://www.intel.com/content/dam/develop/external/us/en/documents/329298-002-629101.pdf