

10. SGX攻撃編②

Ao Sakurai

2025年度セキュリティキャンプ全国大会
L3 - TEEビルド&スクラップゼミ

本セクションの目標



- Controlled-Channel攻撃とAEPIC Leak攻撃についての解説を行う
- mprotectシステムコールを用いる事により、ごく簡単かつ擬似的なControlled-Channel攻撃を実践する
- SGX-Vaultの実装の一部に対し、Controlled-Channel攻撃に対する軽減策の導入を行う

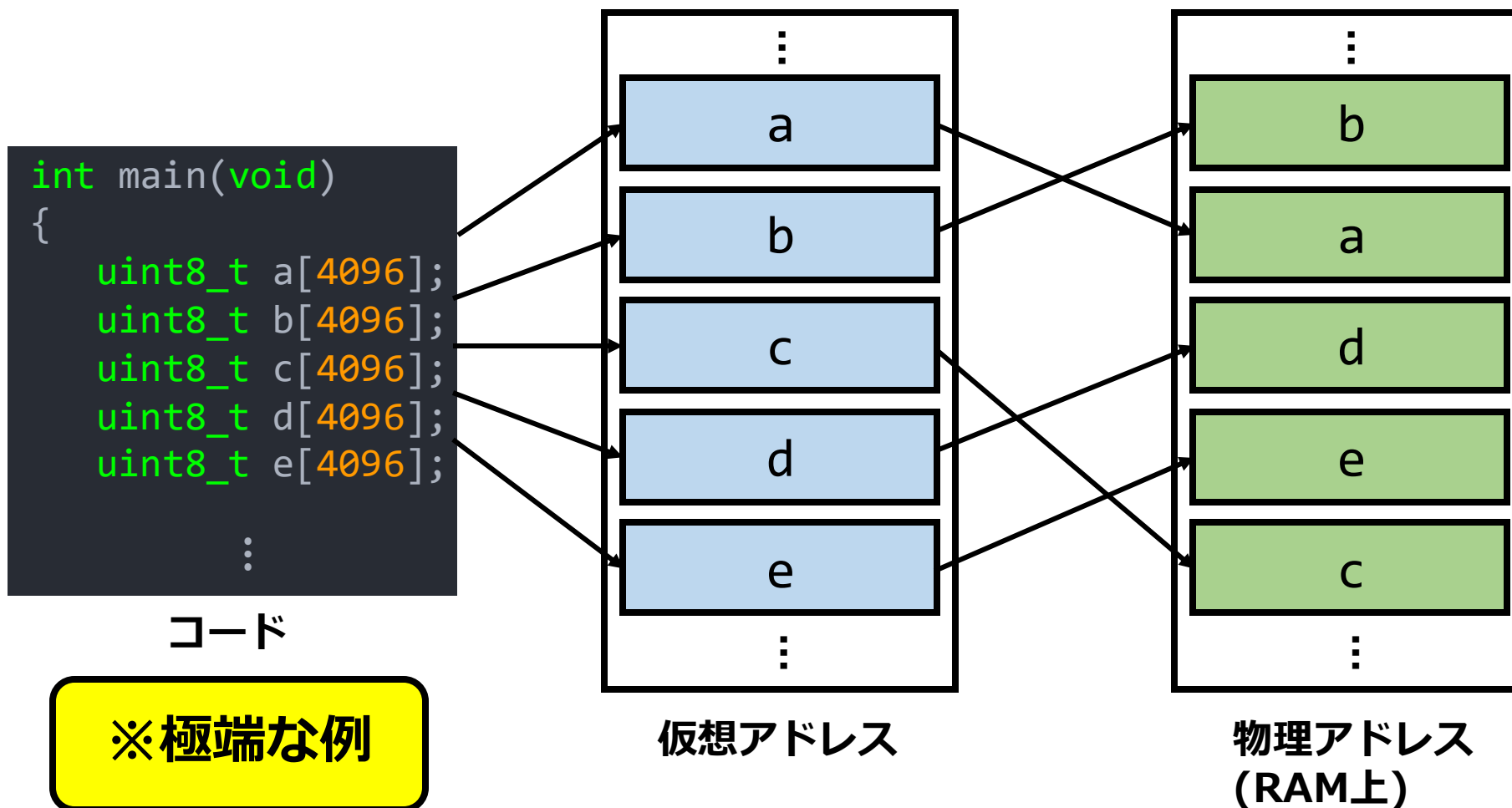
Controlled-Channel攻撃



- **SGX**のような**防護システム**に対して**非常に有効なサイドチャネル攻撃**の一つ
- この攻撃では**ページフォルト**を悪用する
- 以下、OSは**攻撃者**により**制御権を掌握されている**と仮定する
- この攻撃では**3つ**のフェーズが存在する：
 1. コード・実行バイナリに対する**オフライン解析**
 2. **オンライン解析** (①実行バイナリを実行; ②ページフォルト起動; ③観測)
 3. 観測結果から**秘密を推定**

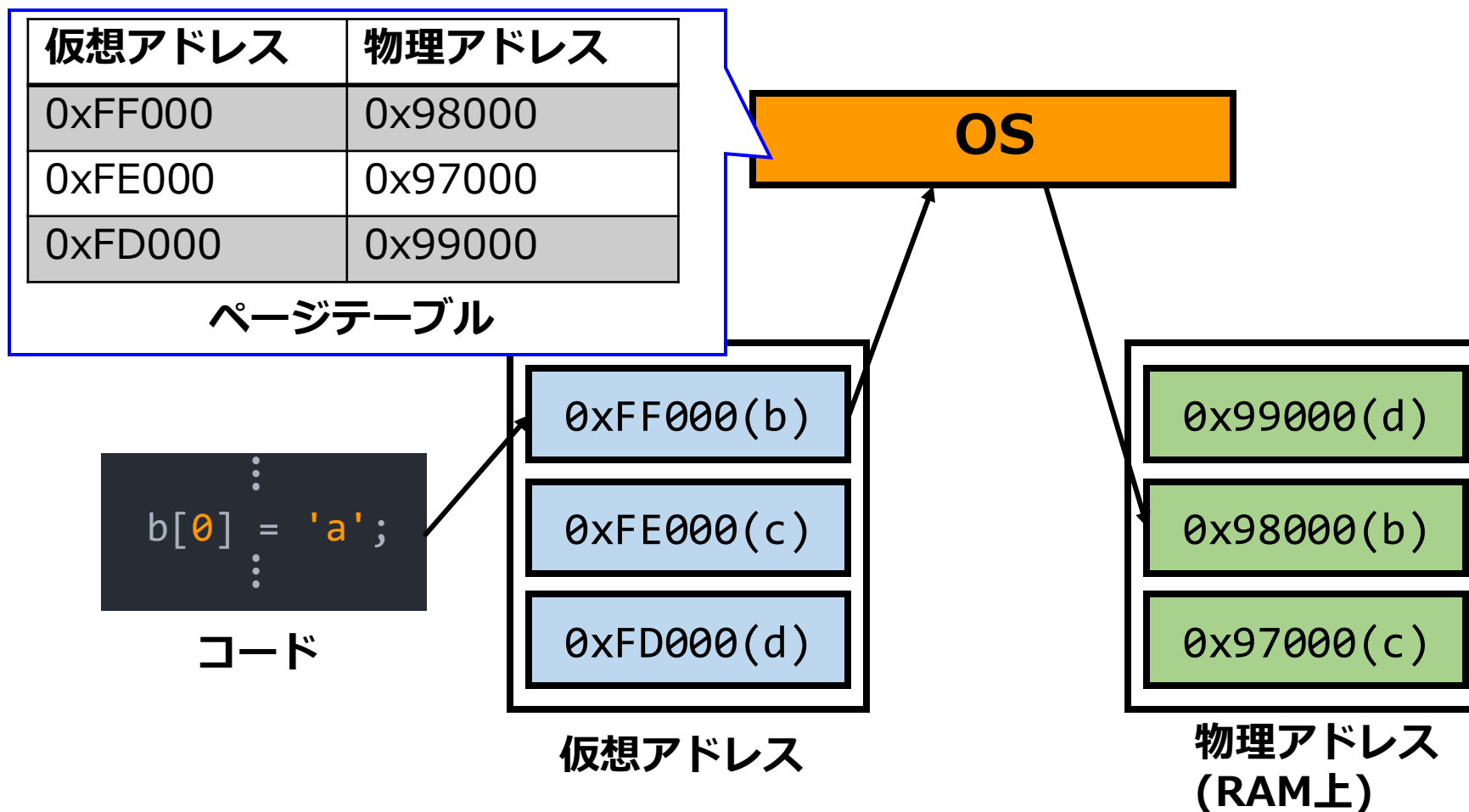


- 仮想記憶を実装するためのアルゴリズム



ページテーブル

- OSは**ページテーブル**を用いて**仮想アドレス**に対応する**物理アドレス**を取得する

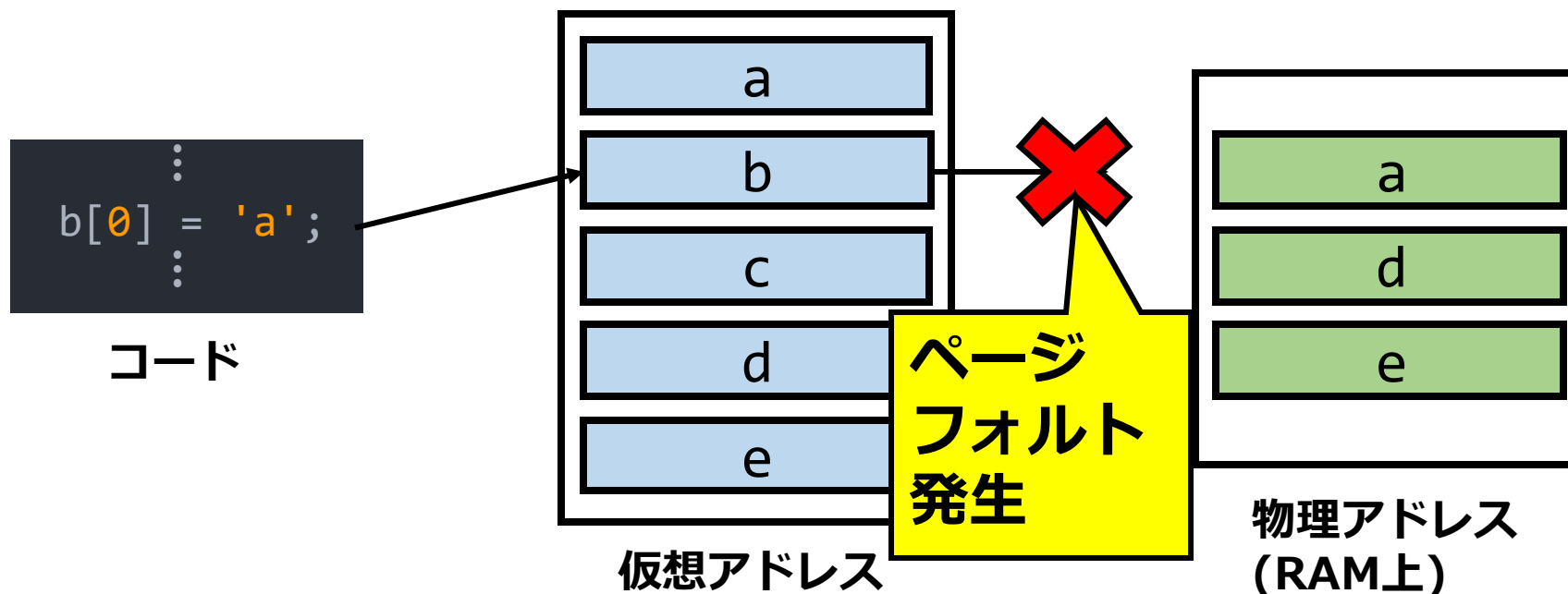


ページフォルト



- ページフォルト: 以下の状況が発生した場合に発生する、必ずしも致命的ではないエラーの事

1. プログラムが**仮想ページ**にアクセスする
2. OSが何らかの原因 (ページアウト、**アクセス制限**等)により**物理アドレス**を**仮想アドレス**から**導出できない**



入力依存処理



- ある**処理実行**が**特定の入力データ**によって**条件的に決定される**時、それを**入力依存処理**と呼ぶ
 - 例: ifブロックに囲まれた処理
- 入力依存処理には**2種類**存在する：
 - 入力依存**制御転送**
 - 入力依存**データアクセス**

入力依存制御転送



- 変数**s**によって**どちらの関数**にアクセスされるかが決定される時、それを「**入力依存制御転送**」と呼ぶ

```
char* WelcomeMessage(GENDER s) {  
    char *mesg;  
  
    //GENDER is an enum of MALE and FEMALE  
    if(s == MALE) {  
        mesg = WelcomeMessageForMale();  
    }  
    else {  
        mesg = WelcomeMessageForFemale();  
    }  
  
    return mesg;  
}
```

入力依存制御転送

入力依存制御転送

入力依存データアクセス



- 変数 **s** によって **どちらのデータ** にアクセスされるかが決定される時、それを「**入力依存データアクセス**」と呼ぶ

```
void* CountLogin(GENDER s) {  
  
    if(s == MALE) {  
        gMaleCount++;  
    }  
    else {  
        gFemaleCount++;  
    }  
  
    return;  
}
```

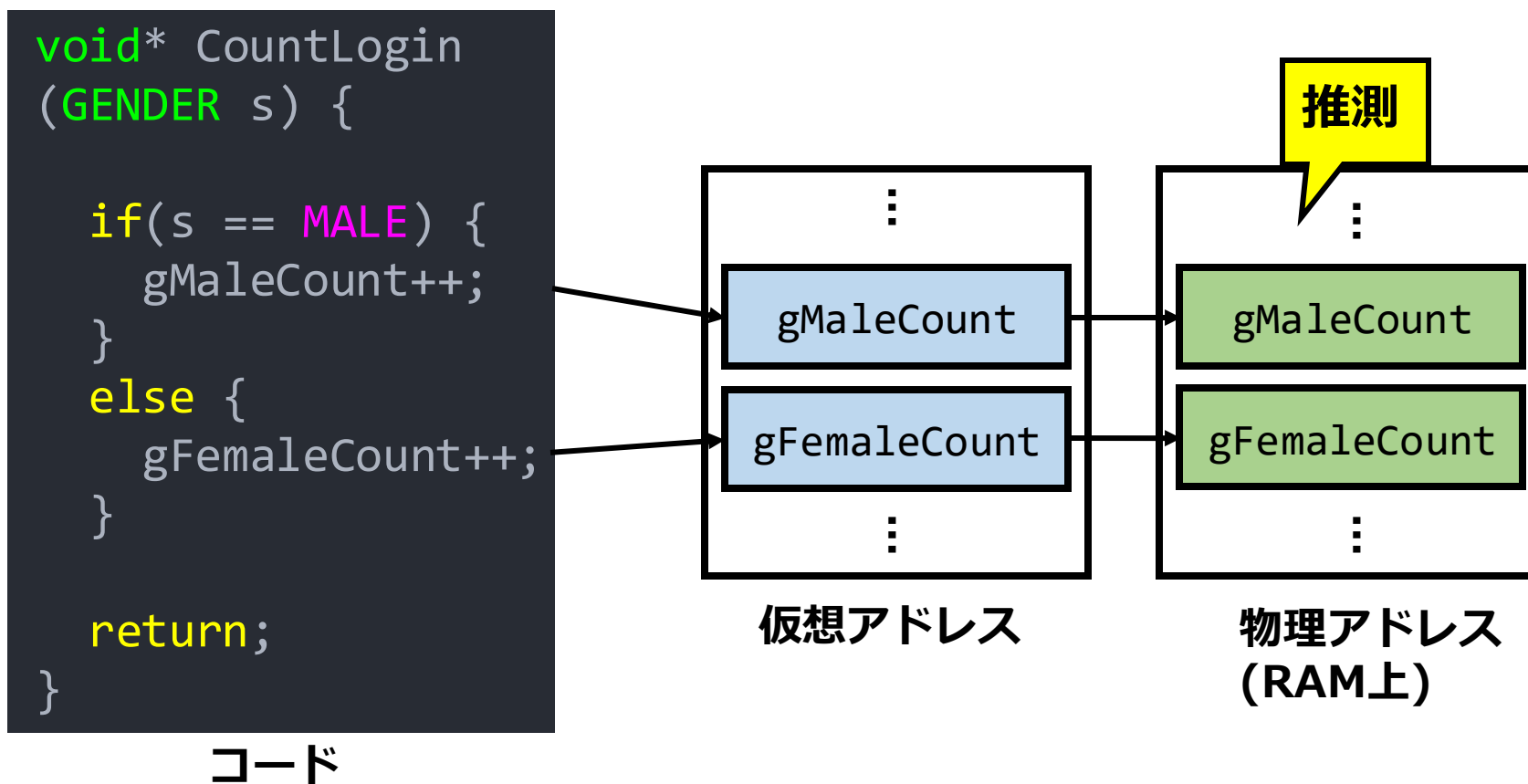
入力依存データアクセス

入力依存データアクセス

Phase 1. オフライン解析



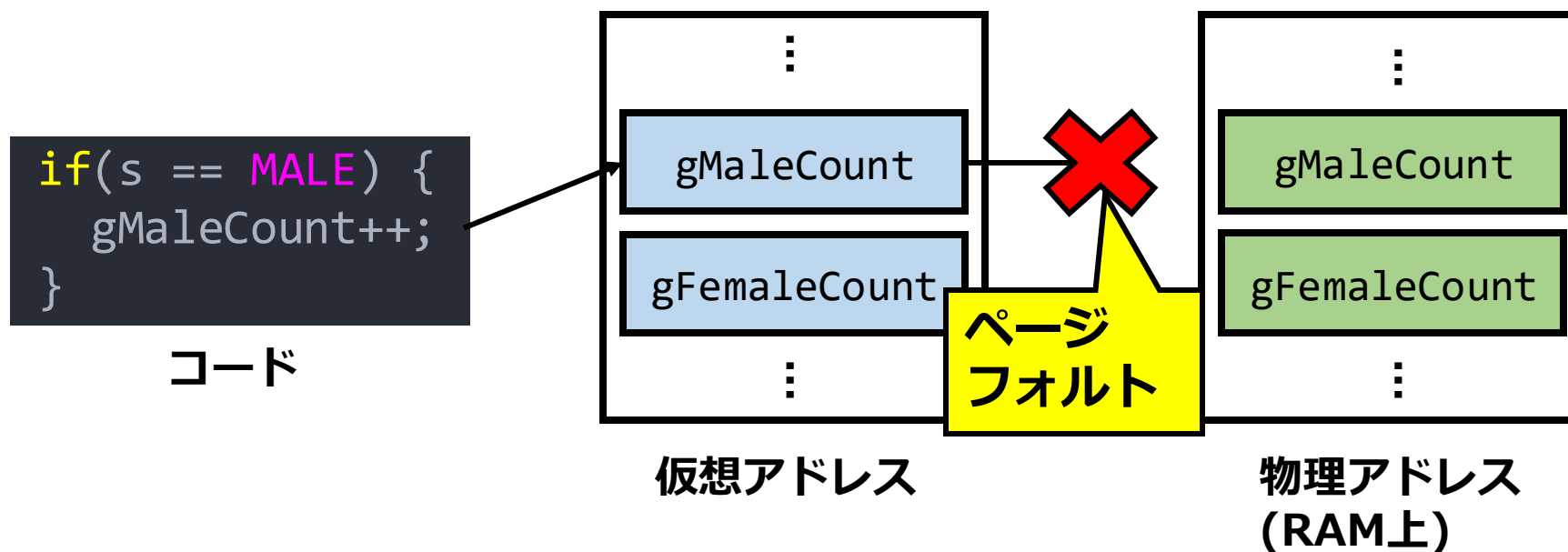
- 実行バイナリやソースコードを解析し、 gMaleCountや gFemaleCountがロードされる**アドレスを推測**する



Phase 2. ページフォルト起動 (2/2)



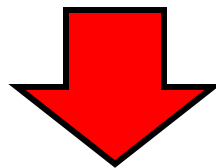
- ページテーブルエントリの**予約ビット**を立てる等により、gMaleCount及びgFemaleCountの載る**ページへのアクセスを禁止**する
- **禁止したページ**に載る**データへのアクセス**が発生すると、**ページフォルト**が発生する



Phase 3. 秘密情報の推定



- ページフォルトが発生すると、OSのページフォルトハンドラにページフォルトアドレスが伝達される
- 攻撃者は、攻撃対象の入力依存要素が載っているアドレス(あるいはページ)をオフライン解析にて取得済みである



ページフォルトアドレスをOSから取得する事により、gMaleCountへのアクセスが発生した事、ひいては“s”がMALEだった事も漏洩してしまう

この攻撃を実行する上での困難



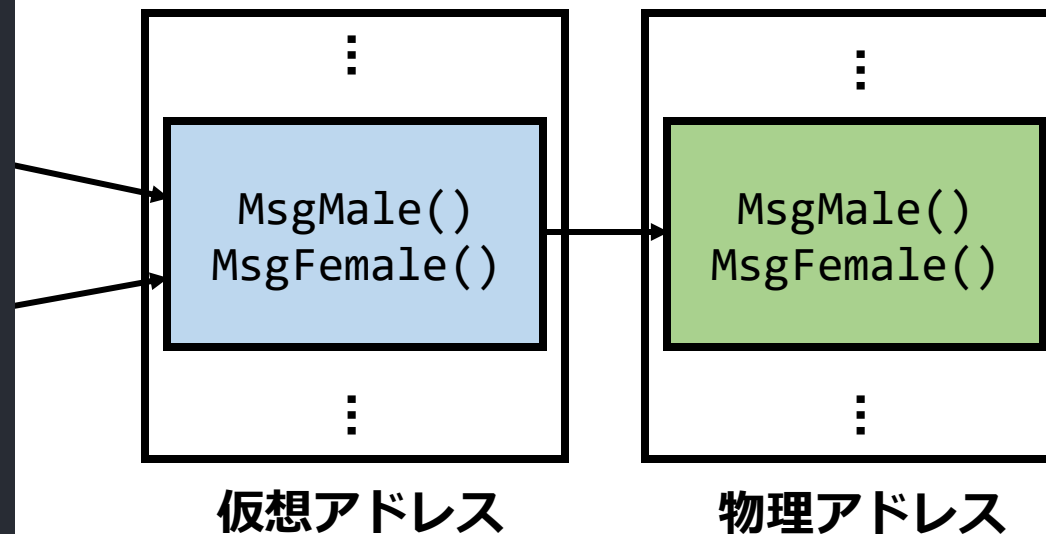
- 複数のコードやデータが同一ページ上に載る可能性が多分に存在するため、その場合単にフォールトの発生したページだけを見るだけではどちらのコードやデータなのか**識別できない**

```
char* WelcomeMessage
(GENDER s) {
    char *mesg;

    if(s == MALE) {
        mesg = MsgMale();
    }
    else {
        mesg = MsgFemale();
    }

    return mesg;
}
```

コード



SGXから返されるページフォールトアドレス



OSが制御可能

通知されるフォールトアドレスは
MsgMale()の完全なアドレスでは
なく、下位12ビットがゼロ化
されている

OS

ページフォールト
ハンドラ

OSはページフォールトの
発生したページ番号の粒度
でしかアドレスを取得できない

OSは制御不能 (Enclave)

MsgMale()
MsgFemale()

MsgMale()
MsgFemale()

仮想アドレス

物理アドレス

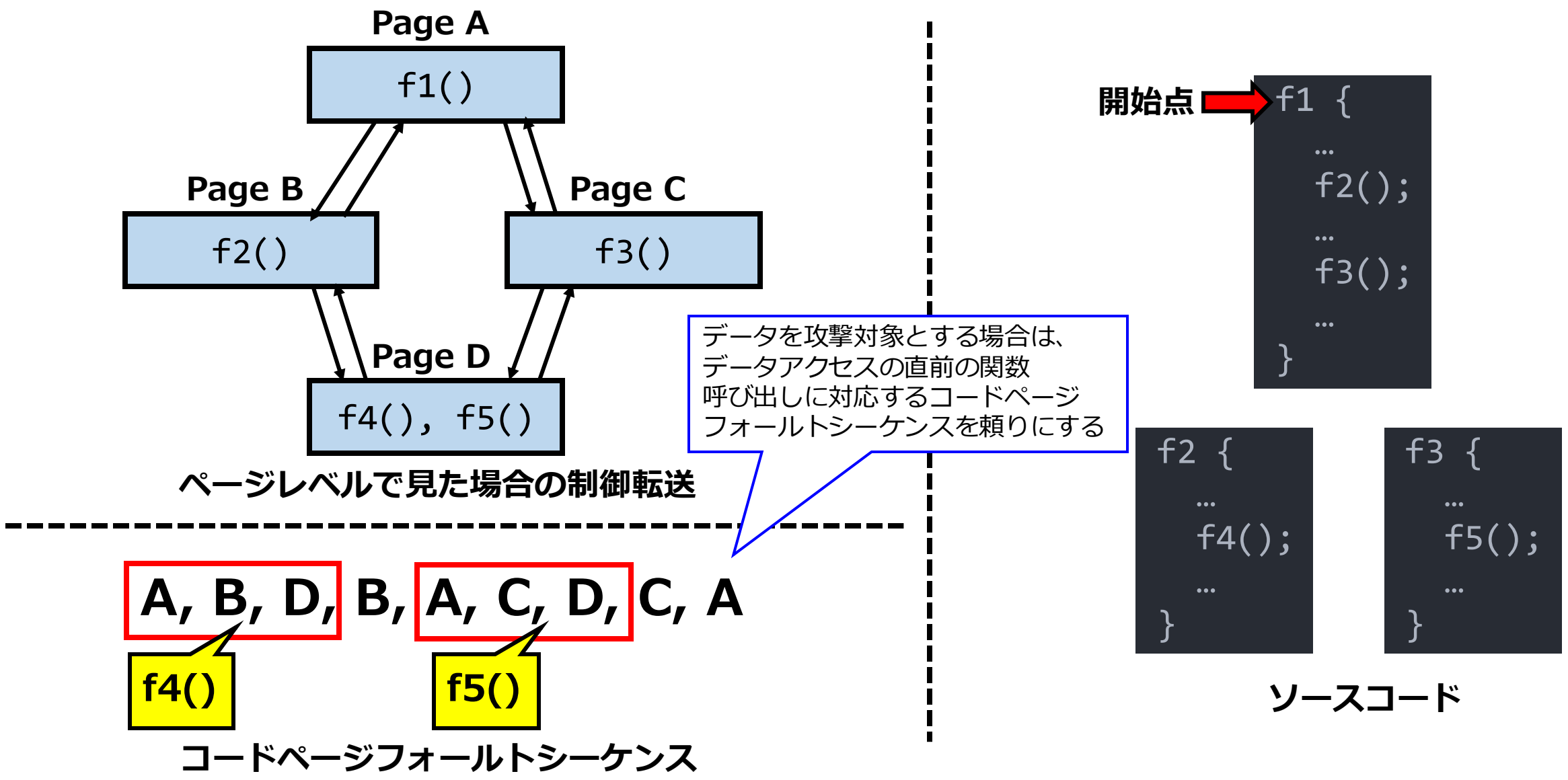


この攻撃を実行する上での困難



- **ページレベルの粒度**でしかページフォールトアドレスからは情報を得られないため、**複数の秘密情報が同一ページ上に載っている**と、このままでは**識別が出来ない**
- この問題を解決するには、次のページで説明する**ページフォールトシーケンス**という概念を用いる

ページフォールトシーケンス



攻撃実践例 – Hunspell（スペルチェッカ）



- Hunspellにより参照される辞書ハッシュテーブルに対して攻撃し、秘密情報である文章を抽出

Folklore, legends, myths and fairy tales have followed childhood through the ages, for every healthy youngster has a wholesome and instinctive love for stories fantastic, marvelous and manifestly unreal. The winged fairies of Grimm and Andersen have brought more happiness to childish hearts than all other human creations.

元の文章

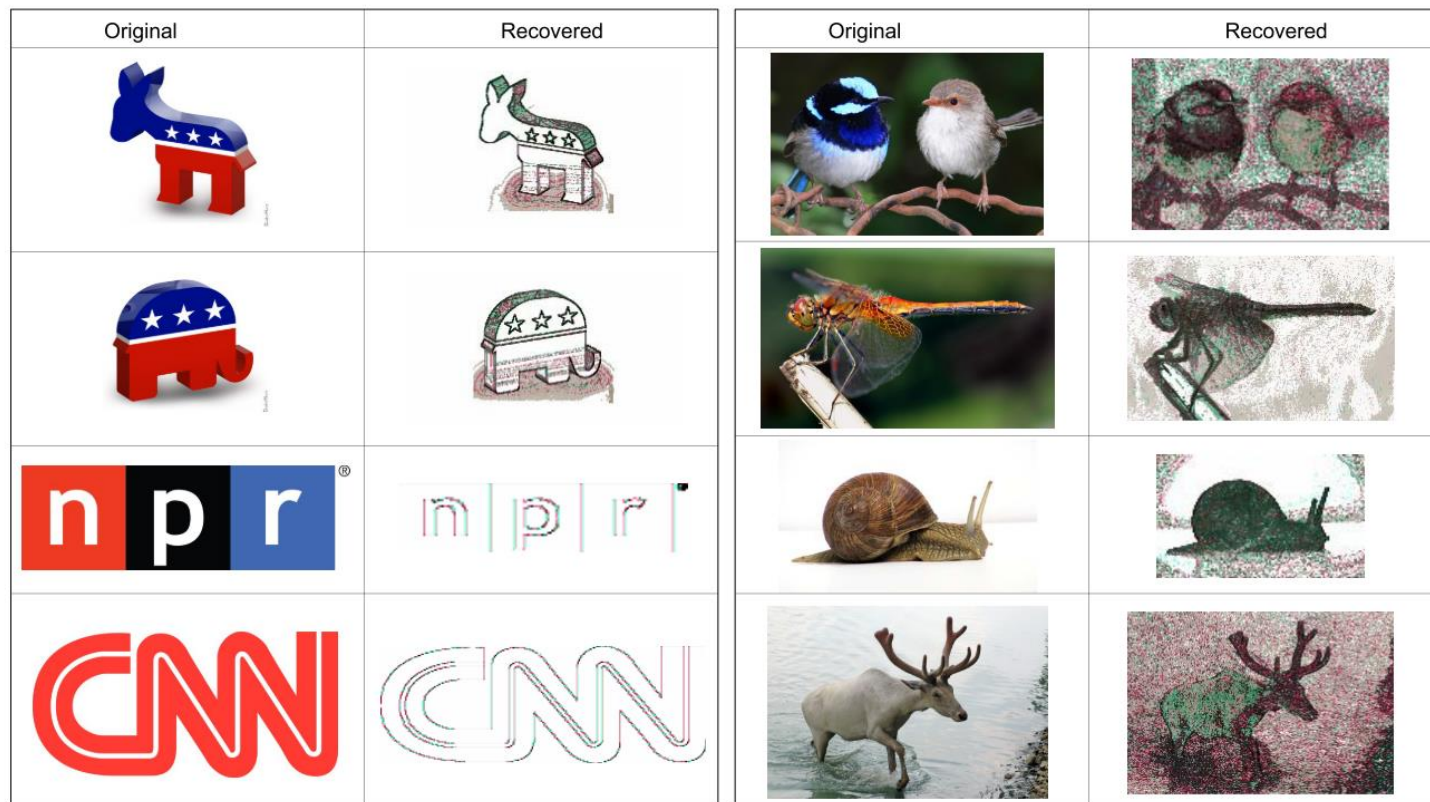
folklore *legend* myths and fairy *tale* have *follow* childhood through the *age* for every healthy youngster has a wholesome and instinctive love for [store] fantastic marvelous and *manifest* unreal the [wine] *fairy* of [grill] and Andersen have brought more happiness to childish *heart* than all other human *create*

抽出した文章

攻撃実践例 – libjpeg



- JPEGファイルのロード時（デコード時）に呼び出される、逆離散コサイン変換を行う関数に対して攻撃し、秘密情報である画像データを抽出





■ Controlled-Channel攻撃実践課題

入力依存データアクセスを行うEnclaveにおいて、分岐先データの載るページのアクセス権を剥奪し、フォールトを発生させて秘密情報の断定を行う実験コードを実装せよ。

但し、OSのページフォールトハンドラを改竄したり中身を見たりするのは難易度が高いため、あくまでも**実験に最適化したEnclave**に対して攻撃を行う。



■ Controlled-Channel攻撃実践の要件

- 攻撃対象Enclaveは、`uint8_t*`型のポインタ**2つ**をグローバル空間に有している
- また、攻撃対象Enclaveは同じく**グローバル空間に1バイトの**`uint8_t`型変数である**秘密情報**を有する
- 攻撃対象Enclaveは、**初期化処理**を行うECALLと、**本処理**を行うECALLの**2つのECALL**を提供している



■ Controlled-Channel攻撃実践の要件

- 攻撃対象Enclaveの初期化処理では、**2つのグローバルポインタ**に対しそれぞれ**ページサイズ**分のバッファを割り当て、バッファを（非ゼロの）**適当な値**（'a'など）で**埋め尽くす**
- また、**sgx_read_rand**関数を用いて**uint8_t型**の**乱数**を生成する。
uint8_tの性質上、この値は**0~255**のいずれかになる



■ Controlled-Channel攻撃実践の要件

- **乱数が128未満**である場合、**秘密情報に0を代入**する。**128以上**である場合は**秘密情報に1を代入**する
- 初期化処理ECALL時には、引数として**ページサイズ**を渡しても良い。同時に、Enclave外に**グローバルバッファの仮想アドレス**を**リターン**しても良い



■ Controlled-Channel攻撃実践の要件

- 攻撃対象Enclaveの**本処理**では、**秘密情報**が**0**であれば**一方のバッファ**に、**1**であれば**もう一方のバッファ**に**アクセス**する
(=入力依存データアクセス)



■ Controlled-Channel攻撃実践のヒント

- ページアクセスの禁止は、**mprotect関数**をEnclave外で
用いる事によって比較的容易に実現できる
 - mprotectに渡すアドレスは、**ページの境界のアドレスに一致して**
いなければならない点に注意
- mprotectでアクセスを禁じたページにアクセスすると**セグフォ**が
発生する。この時Enclaveから返ってくるsgx_status_tの値は
SGX_ERROR_ENCLAVE_CRASHEDである
 - つまり、ページフォールトアドレスを見る必要はなく、**クラッシュしたか**
否かで判定できるため、**片方の秘密バッファの載るページだけアクセスを**
禁じれば十分



- Controlled-Channel攻撃は、攻撃対象コードに**条件分岐さえ存在しなければ無力**である
 - 条件分岐の排除を含め、タイミング攻撃を含むサイドチャネル攻撃の余地を排する実装方法を**定数時間実装**（Constant-time Implementation）と呼ぶ
- SGX-VaultのEnclave内コードにおいて、どれでも良いので**条件分岐を1つ選び、制御フローが単一**になるように**修正**せよ。

ÆPIC Leak



- **APIC** : Advanced Programmable Interrupt Controllerの略で、割り込み処理の高度な制御を行う**割り込みコントローラ**
- **各CPUコア内部**に実装され、**CPU**に対し**配送**された**割り込み**を処理する**Local APIC**と、**システムに1つのみ存在し**、**外部デバイス**からの**割り込み**を適切な**CPU**に**転送**する**I/O APIC**がある
- SGX攻撃の補助として使われるシングルステップ実行フレームワークである**SGX-Step**も、Local APICのAPICタイマを用いてシングルステップ処理を実現している



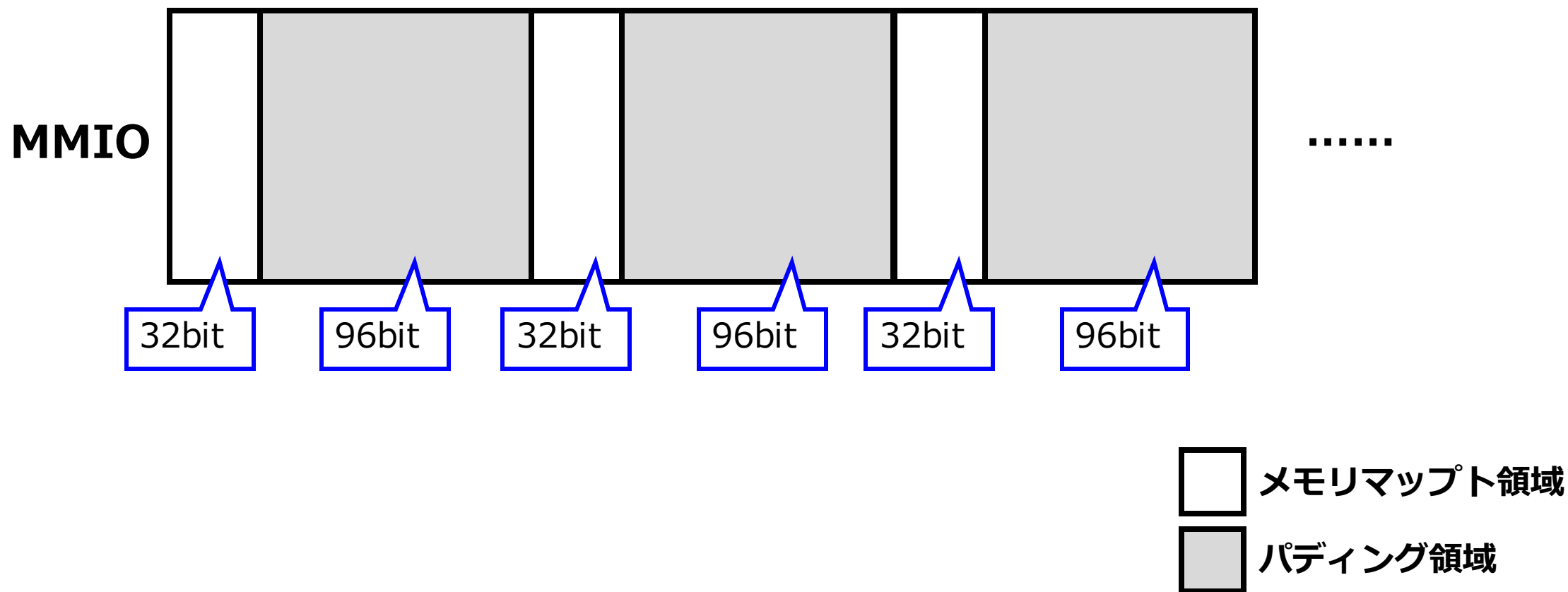
- 最近のAPICは、デフォルトでは**xAPIC**と呼ばれるモードで動作している。xAPICでは、Local APICとのやり取りを行うための**APICレジスタ**を、物理アドレス空間上の4kBの**メモリマップトI/O (MMIO)**の形で公開している
 - **MMIO**：メインメモリ上のある特定のアドレスにアクセスすると、対応する入出力機器とのやり取りが出来る仕組み
- より新しいモードとして、**x2APIC**というモードも存在する。割り込み配送の性能向上が図られており、またAPICレジスタには完全に**MSRを通してアクセス**する（**MMIOアクセスの廃止**）
- どちらのモードを用いるかは**OS (root権限)**によって**設定**できる



- MMIOのベースアドレスはデフォルトでは物理アドレス 0xFEE00000に設定されているが、MSRのIA32_APIC_BASEの値を変更する事でコアごとに対応するMMIOアドレスを変更できる
- APICレジスタは32bit、64bit、または256bitのいずれかの大きさの値を取り扱うが、**xAPICモード**における**メモリマップト領域**では**32bit単位に分割**してマッピングされ、さらに**128bit単位に整列**される



- **32bit**のメモリマップト領域を**128bit**単位に整列するために、**96bit**の**パディング領域**を挟む事によってこれを実現している





- このパディング領域は文字通りパディング用であり、それ以外に何らかの**アーキテクチャ的な定義**がなされていない領域である
- Intelは、この**パディング領域へのアクセス**は**未定義の動作**を引き起こす可能性があるため**行ってはならない**と言及している
 - 普通は、0x00または0xFFの読み出し、システムハング、そしてトリプルフォールト（例外ハンドラの失敗の対応にも失敗した際のフォールト）のいずれかが発生する
 - あくまでもメモリマップト領域である32bitの部分にのみアクセスしAPICとのやり取りを行うのが本来の想定

キャッシュ階層 (1/4)

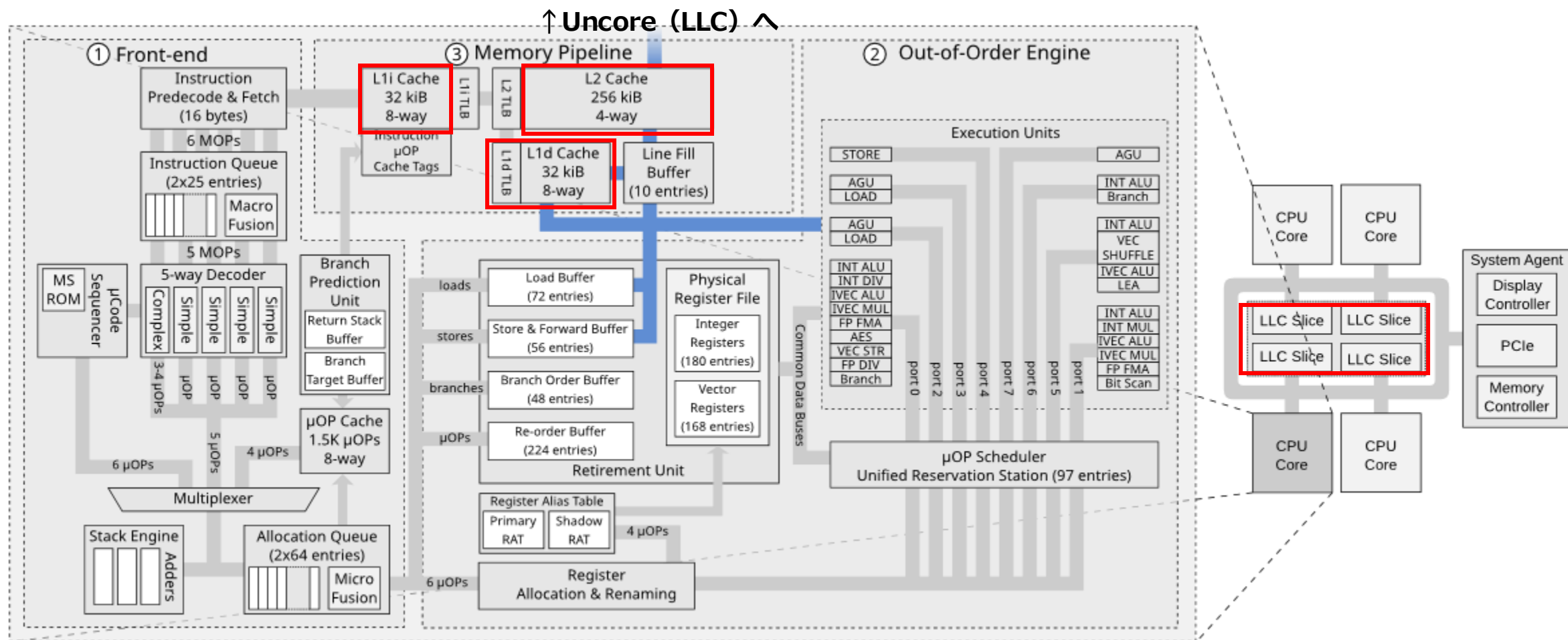


- 頻繁にアクセスするデータや命令を保持する役割を持つ、CPUパッケージ内に存在しメインメモリよりもアクセス時間の速い**キャッシュメモリ**は階層構造になっている
- 最近のCPUでは、CPUの主要部分に近い順に**L1, L2, L3キャッシュ**の3つのキャッシュにより成り立っている
- **L1キャッシュ**には、命令を保持しておく**L1Iキャッシュ**と、データを保持しておく**L1Dキャッシュ**が存在する

キャッシュ階層 (2/4)



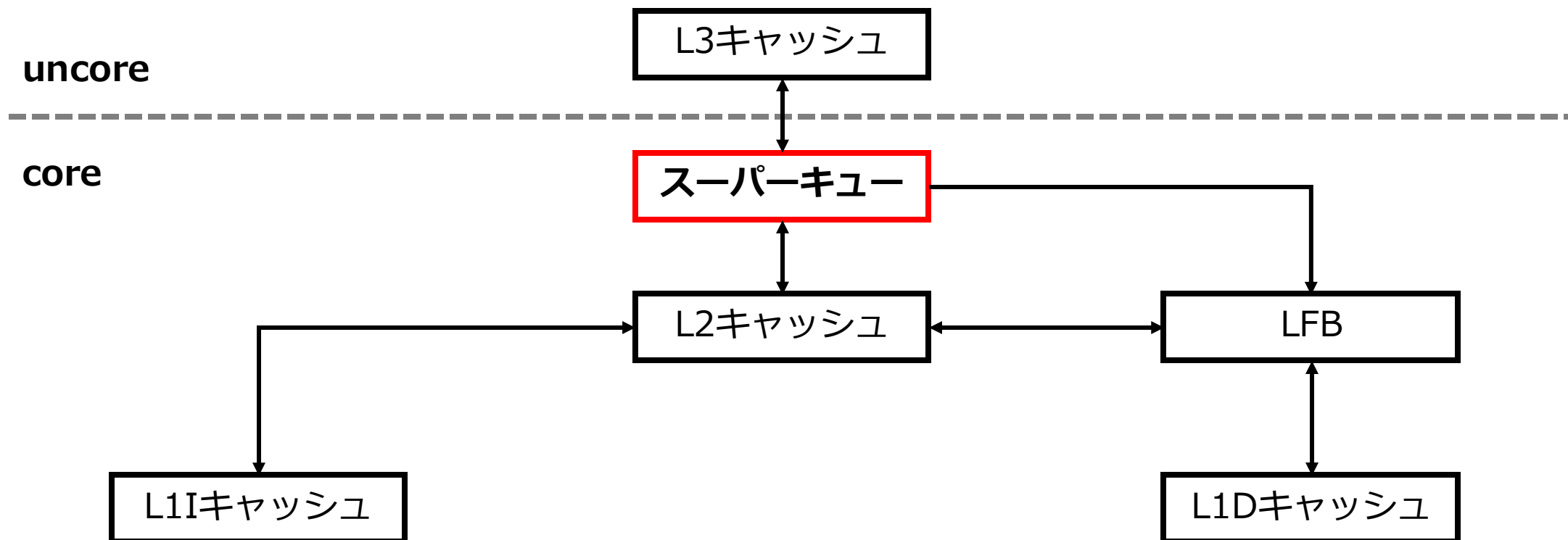
- 以下の図はSkylake CPUの構造を示すものである ([4]より引用)



キャッシュ階層 (3/4)



- さらに、L1D-L2間のLFBに相当するものとして、L2とL3の間に**スーパーキュー (Superqueue)** という架け橋的なバッファも存在する

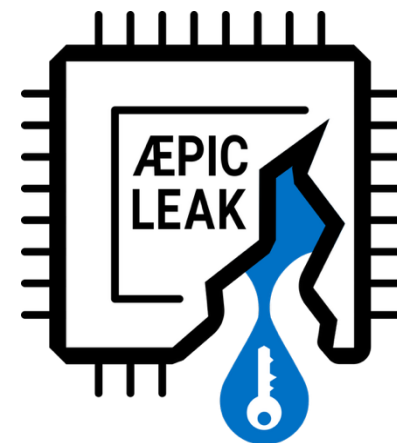




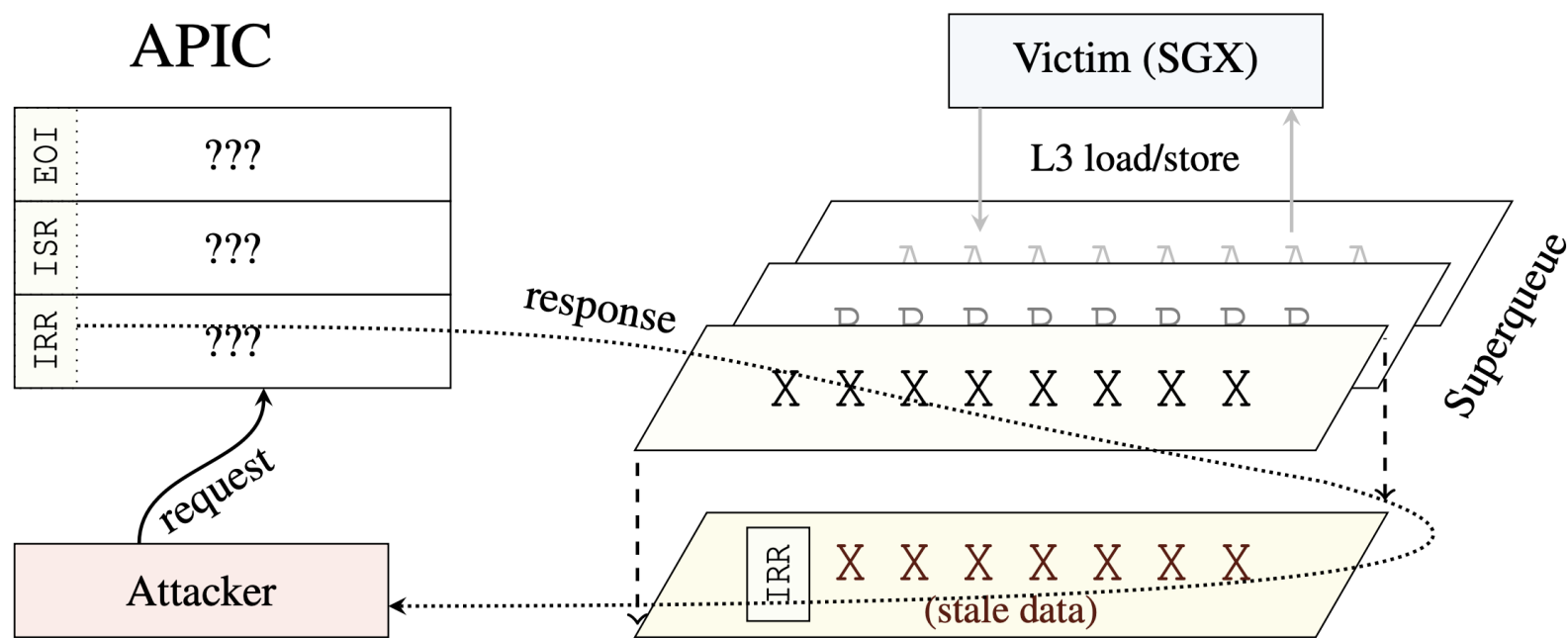
- **APIC**は、**L3キャッシュ**から**L2キャッシュ**に値を持ってくる際に、この**スーパーキュー**を使用する
- スーパーキューはキャッシュ間のデータの転送に用いられるものであるため、ユーザ空間、カーネル空間、そして**Enclaveのデータ**等、**あらゆる出処のデータ**が格納される可能性がある



- **Sunny Coveマイクロアーキテクチャ**を搭載するCPUにおいて、**xAPICのMMIOのパディング領域**を読み取ると、**スーパーキューに残留している値が漏洩する**バグがある事が判明した
 - Sunny Cove μ -Archを搭載するCPUとしては、第10～13世代Core シリーズCPUや、第3世代Xeon-SP CPUが挙げられる
- このバグを悪用し、**Enclave由来のスーパーキュー内の秘密情報を漏洩させる**攻撃が**ÆPIC Leak**である
 - 根本原因は、APICレジスタ (MMIO) のパディング部分が**正しく初期化されない**という**アーキテクチャ上のバグ**である
 - 現在発表されているSGX攻撃の中では最新に近い攻撃の1つ



- ÆPIC Leakの概要図 ([5]より引用)



- 具体的には、**パディング領域に1~4バイト単位**でアクセスを行うと**前述のような漏洩が発生**する
 - 8バイト以上単位でのアクセスでは必ず0xFFが返却される
- APICレジスタは複数存在するが、**アクセスするAPICレジスタとその内容に相関は存在しない**
- 一方、APICアドレスに対応する**キャッシュライン上のオフセットは、漏洩させるデータのキャッシュライン上のオフセットに一致**する
 - **キャッシュライン**：キャッシュの構成単位（通常**64バイト**）
 - 例えばAPICベースアドレスから0x10のオフセットの場所にアクセスすると、攻撃対象キャッシュラインの0x10のオフセットの値が漏洩するイメージ

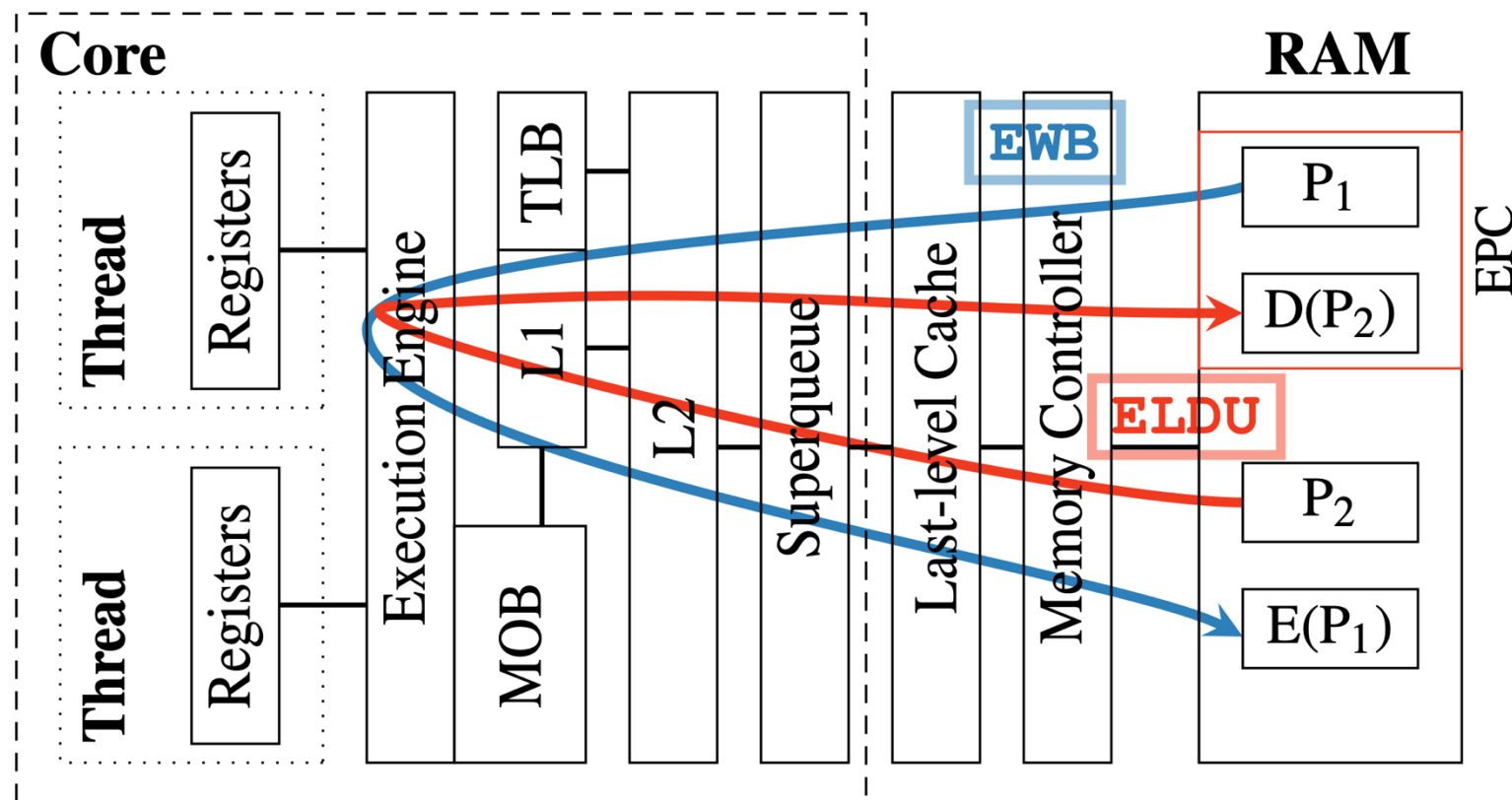
- 128bitの内パディング部分でない**32bitのメモリマップト領域**は正常に初期化されるため、**64バイトのキャッシュライン**であれば**後半48バイト**（後半**3/4**）のみが漏洩する
- さらに、**128の倍数**のアドレスから始まる**キャッシュライン**からのみ漏洩させられる
 - キャッシュラインは通常**2つペアで転送**されるため、**2つ目**のキャッシュライン（アドレス：64+128×n）がまず**転送**され、その後**1つ目**（アドレス：128×n）で**上書きする形になる**からであると考えられる
- 結果的に、ÆPIC Leakでは $\frac{96}{128} \times \frac{1}{2} = 0.375$ 、即ち**任意のページの37.5%**を漏洩させる事が出来る

- Foreshadow攻撃（後述）でも使用されている手法であるが、**秘密情報が載るEPCページをEWBとELDUでページ退避&ロードを繰り返す事で、強制的に秘密情報の載るページの内容をキャッシュ階層（スーパーキュー含む）に持ち込む事が出来る**
- Foreshadow攻撃の論文ではこの手法に対する名前をつけていないが、ÆPIC Leakの論文で**Enclave Shaking**と名付けられている

ÆPIC Leakの補助手法 (2/3)



- Enclave Shakingの概要図 ([5]より引用)。Enclave Shakingにより、秘密情報の載るページの内容がキャッシュ階層に浸透する様子を示している



- 別の補助手法として、ある**攻撃対象**の載るものと**並行して実行**されている、**同一物理コア内の論理コア**（=**ハイパースレッド**）、即ち**兄弟スレッド**（Sibling Thread）を**悪用**する手法がある
- **攻撃対象に並行して実行**される**兄弟スレッド**にて、攻撃対象と**無関係なページ**の**ページオフセット x** に**連続的にアクセス**すると、**攻撃対象ページ**の**オフセット x** の値が**漏洩**する**可能性を上げられる**
- 論文では、この手法を**Cache Line Freezing**と呼んでいる

- ÆPIC Leakの攻撃例として、SGX Failのセクションでも取り上げた **SECRET Networkのコンセンサスシードの抽出**を行う攻撃を取り上げる
 - 当該セクションで説明した通り、SGX Failの論文で実験として実際に行われた攻撃である
- 早い話、コンセンサスシードを暗号化しているSGX_FILEのAES鍵 (**IPFSL鍵**) に対して**ÆPIC Leak**を仕掛け、IPFSL鍵を抽出し**コンセンサスシードを復号**する事を考える

(復習) SECRETにおけるコンセンサスシード (1/4)



- SECRETでは、前述の様々な保護機能で使用する鍵は、全て親玉（**マスター秘密鍵**）である**コンセンサスシード**から導出される
- スマートコントラクトへのメッセージ送信時は、ユーザは**コンセンサスシード**から導出した**公開鍵**を用いて**暗号文**を**トランザクション**に含める
 - 一方、**秘密鍵**は**MRSIGNERポリシーのシーリングデータ**として、チェーン全体に複製（デプロイ）される
- また、口座残高等の現在の状態を暗号化する鍵もまた**コンセンサスシード**から導出される

(復習) SECRETにおけるコンセンサスシード (2/4)



- 当然、この**コンセンサスシード**が**万が一にも漏洩**すると、SECRETが売りにしている**あらゆる保護機能が無力化**される
- さらに、このコンセンサスシードは原則として**永続的かつ不変**のものであるため、もし漏洩すると**極めて面倒な事になる**
 - **ブロックチェーンそのものを分裂**させなければならない**ハードフォーク**でのみ対応する事が出来る

- コンセンサスシードは、独自に用意した鍵により
Intel Protected File System Library (以下、IPFSL) を
用いて128bit AES/GCMで暗号化され**Enclave外にストア**される
- **IPFSL : SGX_FILE**型という**Enclave内**で直接扱えるような
FILE構造体に基づき、**暗号化した状態でのファイル入出力**
(`sgx_fopen`等) を提供する機能
 - Enclave内から**直接Enclave外に保存**できる、**暗号化鍵を指定できる**等の
部分でシーリングと異なる
 - 本ゼミでは使用していないが、`include`とリンクさえ行えば普通に
デフォルトのSGXSDKで利用可能

(復習) SECRETにおけるコンセンサスシード (4/4)

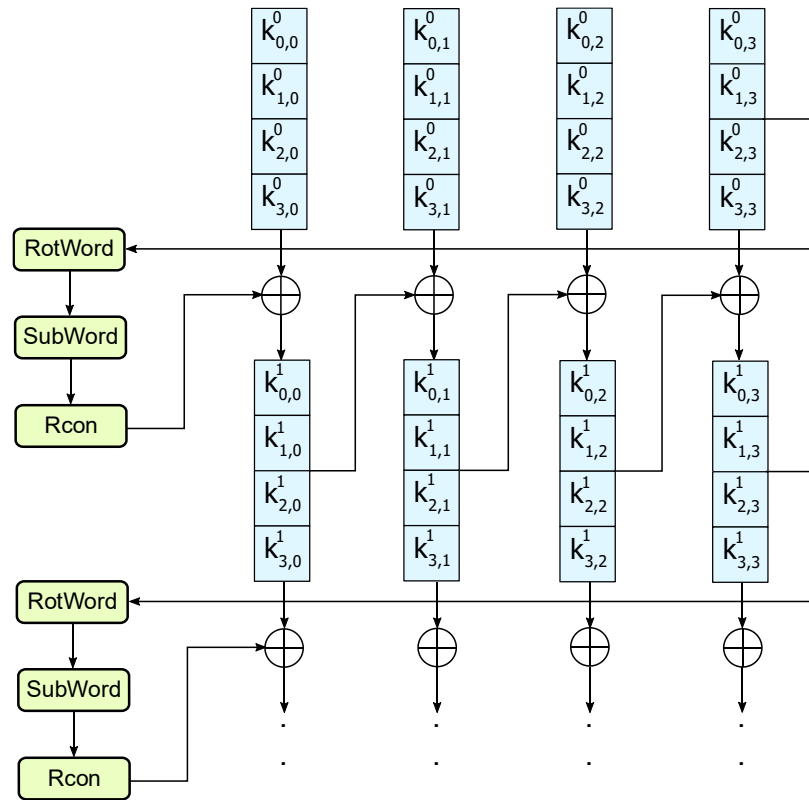


- IPFSLでコンセンサスシードを暗号化する際に使用した鍵は
シーリングでストアされる
 - 鍵自体はAESで使う普通の128bitの共通鍵
- SGX Failでは、この**IPFSL用の鍵**がEnclave内で**コンセンサスシードの暗号化及び復号に使用されている最中**に**攻撃を仕掛け**、IPFSL用鍵を抽出する手法を選択した
 - **IPFSL鍵が漏れると簡単にコンセンサスシードが復号出来てしまい、SECRET上のあらゆる秘密情報が究極的には暴かれてしまう**

- まず、Controlled-Channel攻撃の**ページフォールトシーケンス**手法を利用し、IPFSLがAES鍵により**コンセンサスシードを復号**する関数の所まで**実行を進める**
 - 厳密には、復号にあたり鍵伸長処理を行う
k0_aes_DeckKeyExpansion_NI()関数の実行まで進める
- この状態で**Enclaveを中断**し、**ÆPIC Leak攻撃を実行**する

- ÆPIC Leakを仕掛けると、**N**ラウンド目と**N+1**ラウンド目のラウンド鍵の、それぞれ**後半3ワード**が**復元**できる
 - 1ワード：サイズは4バイトで、AESブロックである4×4行列の1行または1列に相当する。ワード換算では128bitは4ワードになる
 - 先頭1ワードが抽出できないのは、前述の37.5%のリーク率が原因である

- ここで、AESの鍵伸長では、**N+1**ラウンド鍵の**第2ワード**を導出するために、**N+1**ラウンド鍵の**先頭ワード**と**N**ラウンド鍵の**第2ワード**とでXORを取っている (図は[6]より引用)



- ここで、AESの鍵伸長では、**N+1**ラウンド鍵の**第2ワード**を導出するために、**N+1**ラウンド鍵の**先頭ワード**と**N**ラウンド鍵の**第2ワード**とでXORを取っている

- つまり、nラウンド鍵の第iワードを W_i^n と表現すると、上記は
$$W_2^{N+1} = W_1^{N+1} \oplus W_2^N$$

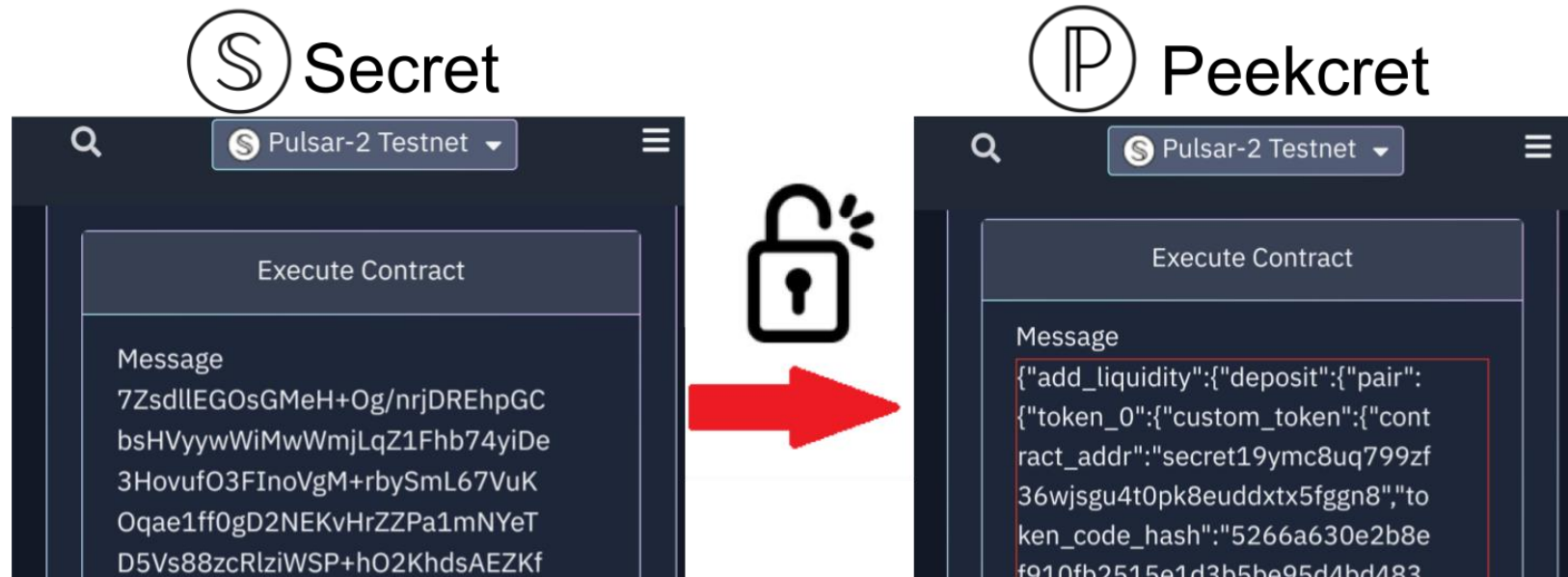
となり、XORの対称性により

$$W_1^{N+1} = W_2^N \oplus W_2^{N+1}$$

が成立する

- これは即ち、**N**ラウンド鍵と**N+1**ラウンド鍵のそれぞれ**第2ワード同士をXOR**すれば、**N+1**ラウンド目の**先頭ワードが復元**でき、**N+1ラウンド鍵が完全に抽出**できる事になる
- 使用する**AES関数の仕様** (S-BoxやRcon等) は**公開情報**であるため、あるラウンド鍵から**1つ前のラウンド鍵を導出する材料**はこれで**全て揃う**事になる
- どのラウンド鍵を抽出したかは分からないため、128bit AESのラウンド数である**10通りのブルートフォース**を行い、最終的に**IPFSLのAES鍵を復元しコンセンサスシードを復号**できてしまう

- 解読したコンセンサスシードを使用し、以下のように**SECRETのブロックのトランザクションの復号に成功**している
(図は[7]より引用)



参考文献 (1/2)



- [1] "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems", Yuanzhong Xu et al., <https://ieeexplore.ieee.org/document/7163052>
- [2] "Intel SGX - Controlled-Channel Attacks解説", 自己引用, <https://qiita.com/Clifford/items/f527fd210e3f7866e803>
- [3] "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control", Jo Van Bulck et al., <https://jovanbulck.github.io/files/systex17-sgxstep.pdf>
- [4] "MDS: Microarchitectural Data Sampling", 2023/7/20閲覧, <https://mdsattacks.com/>
- [5] "ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture", Pietro Borrello et al., <https://aepicleak.com/aepicleak.pdf>
- [6] "AES key schedule - Wikipedia", 2023/7/27閲覧, https://en.wikipedia.org/wiki/AES_key_schedule
- [7] "How Stuff Gets eXposed", 2023/7/27閲覧, <https://sgx.fail/>

参考文献 (2/2)



[8]“Local APICについて - 睡分不足”, 2023/7/25閲覧,
<https://mmi.hatenablog.com/entry/2017/03/27/202656>

[9]“What is the semantics for Super Queue and Line Fill buffers?”, 2023/7/25閲覧,
<https://stackoverflow.com/questions/45783251/what-is-the-semantics-for-super-queue-and-line-fill-buffers>

[10]“SoK: SGX.Fail: How Stuff Gets eXposed, by Stephan van Schaik et al.,
<https://sgx.fail/files/sgx.fail.pdf>