5. TDX Moduleとメタデータ

Ao Sakurai

2026年度セキュリティキャンプ全国大会
L1 – 暗号・CVMビルド&スクラップゼミ

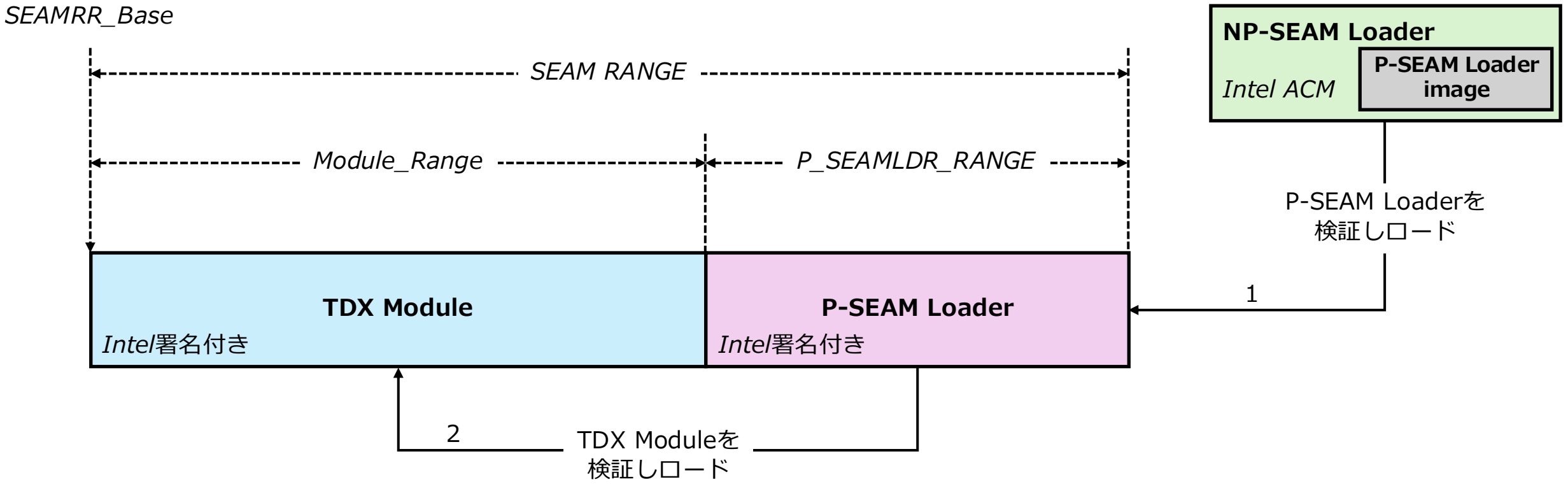
本セクションの目標

- 本スライドでは、TDX Module及びTDの各種メタデータに関する説明を詳細に行っていく

TDX Moduleのロード

TDX Moduleのロード概要

- Demystified論文[1]の図を参考にする形で、TDX Moduleをロードするための2段階のプロセスを以下の図に示す：



TDX Moduleのロード概要

- TDX Moduleのロードには、以下の3つのコンポネントが大きく関わる：
  - **NP-SEAM Loader**
  - **P-SEAM Loader**
  - TDX Module自身

- これらは**全てIntelにより提供・署名**されており、Intelを全面的に信頼するTDXの脅威モデルには反していない

- が、これらがどのようにロードされるかの手順も重要であるため、それについて説明する

NP-SEAM Loaderのロード

- TDX Moduleをロードするには、まず**NP-SEAM Loader**（Non-Persistent SEAM Loader）のロードを行う必要がある

- NP-SEAM Loaderは、**Intel ACM**の1つである
  - ACM：Authenticated Code Module

- ACMはプロセッサ内部RAMで動作し、Intelによる署名が付与されており、その検証に成功したら起動される
  - このACMの内容は、署名検証に成功した場合には正しい前提を取る
  - つまり、このACMはTDXにおけるHW-RoTであるCPU内に存在するため、TDX（TEE）の信頼モデルには違反しない

NP-SEAM Loaderのロード

- 具体的には、**GETSEC[ENTERACCS]**という命令を通じて、Intel TXTによりこの認証及びロードを実施してもらう
  - TXT：Trusted Execution Technology

- Intel TXTは、システムが信頼可能な状態で起動した事を保証するための技術であり、それこそSGXよりも前に登場している技術である

- TXTは以下のようにして信頼の連鎖を形成する：
  - Intel署名付きのACMを対応する署名検証鍵で検証しそれをTPMのPCR0に拡張
  - ACMが次の層（BIOS初期コード等）のPCRを測定・拡張
  - 以降、同様にn番目のコンポネントがn+1番目のコンポネントを測定・拡張

NP-SEAM Loaderのロード

- 別の言い方をすれば、TXTはトラストチェーンにより各ロード段階・コンポネントレイヤごとに**階層化されたMeasured Bootを実現**するHW機能であると言える
  - ACMはそのようなチェーンのRoTとして機能する

- TDX Moduleを真にIntelによるお墨付きのある状態で起動するために、TXTの提供する機能の内この**ACMという機能のみを流用**したという経緯である
  - よって、BIOS設定画面ではTDXの利用のためにTXT機能の有効化は不要。内部で勝手にACM部分だけが使われる
  - 同様に、TXTの設定からの有効化に必須なTPMの装着も不要

NP-SEAM Loaderのロード

- NP-SEAM Loader含むACMの具体的なロード方法として、GETSEC[ENTERACCS]が呼ばれたら、Intel TXTの**プラットフォームチップセット公開鍵**により**署名検証**を行う
- この署名はHASH(ACRAM, ACBASE, ACSIZE)に対して行われている
- これは、ACRAM（ACMがロードされるプロセッサ内部RAM）上のACBASE（そのACMのベースアドレス）からACSIZE分のコンテンツに対する署名という事を意味する
  - 要は**ACM丸ごとそのものに対する署名**である
- 検証成功後、実際にプロセッサ内部RAMにNP-SEAM Loaderがロードされる

P-SEAM Loaderのロード

- プロセッサ内部のIntel鍵（HW-RoT鍵）による検証に成功し無事NP-SEAM Loaderがロードされたら、今度は**P-SEAM Loader**（Persistent SEAM Loader）のロードを実施する

- NP-SEAM Loader内には予めP-SEAM Loaderのイメージが同梱されている

- よって、**NP-SEAM Loader**がP-SEAM Loaderイメージを**検証しロード**を実施する

P-SEAM Loaderのロード

- P-SEAM Loaderは、TDX Moduleをロードするための、TDX Moduleと同じく**SEAM VMX rootで動作**するコンポネントである

- NP-SEAM Loaderに直接TDX Moduleを含めてしまうと、特にアップデートに関しては**融通が利かなくなる**事が容易に想像できるため、この**P-SEAM Loaderが挟まる事は重要**である

- TDX Module同様、P-SEAM LoaderもSEAMCALLやSEAMRETを介して外部のSWとのやり取りを行う

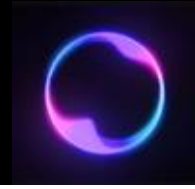
P-SEAM Loaderのロード

- P-SEAM LoaderとTDX Moduleは、**双方共にSEAM Rangeにロード**される

- SEAM Rangeの復習：UEFI/BIOSによって予約されたシステムメモリの一部で、これ専用のTME-MK鍵によって暗号化され他とは隔離される

- SEAM RangeはIA32_SEAMRR_PHYS_BASEとIA32_SEAMRR_PHYS_MASKという2つのMSR（モデル固有レジスタ）によって指定される
  - さらに、SEAM RangeはTDX Module用の**Module_Range**と、P-SEAM Loader用の**P_Seamldr_Range**に分割される

TDX Moduleのロード



- このP-SEAM Loaderは、TDX Moduleをロードするための
  インタフェースであるseamldr_installを提供する
  - 具体的にはこれは**SEAMCALL[SEAMLDR_INSTALL]**である[4]
  - ちなみに、RAXの第63bitを1にしてSEAMCALLを発行すると、TDX Module
    ではなくP-SEAM Loaderへの指示として解釈される

- P-SEAM Loaderによってロードされる TDX Moduleイメージは、
  SEAM Range**ではない**メモリバッファにプリロードされる

- seamldr_installはパラメータとして、TDX Moduleイメージが
  プリロードされているバッファの物理アドレスと、
  SEAM_SIGSTRUCT（TDX Moduleの署名構造）が渡される

TDX Moduleのロード

- SEAM_SIGSTRUCTには、一例として以下の情報が含まれる（完全な一覧に関しては[4]を参照）：
  - TDX Moduleのハッシュ値
  - TDX ModuleのSVN
  - LP（論理プロセッサ）単位のスタックページ数
  - LP単位のデータページ数
  - グローバルデータページ数
- 特に下3つのページ数は、seamldr_installがTDX Moduleの様々なメモリ領域の物理/仮想（リニア）アドレスやサイズを決定するのに使用される
- ちなみに、SIGSTRUCTという用語はSGXにおいても登場する

TDX Moduleのロード

- seamldr_installを呼び出す際の作法として、呼び出しの際には全ての論理プロセッサ（LP）で連続して呼び出す必要がある

- 具体的には、**最初のLP**でこの命令が呼び出されると、TDX Moduleのインストール（ロード）セッションが**開始**される

- その後、続く他の各LPは、この最初のLPにより開始されたインストールセッションに**連動**する形で、LPのVMCSキャッシュをクリアする
  - この連動の際、既に別の既存のインストールセッション中でないかについてもチェックする

TDX Moduleのロード

- 最後のLPでseamldr_installが呼び出されると以下の処理を実施する事で、TDX Moduleをロード・検証し利用可能にする：
  1. seamldr_installへのパラメータをチェックする
  2. TDX Moduleの署名をチェックする
  3. ロードされるイメージのSVNをチェック。そして、常駐しているTDX ModuleのSVNと比較する(※)
  4. SEAM Range内のTDX Moduleの各種メモリ領域（コード、データ、スタック、ページテーブル、**Sysinfo_Table**、Keyhole、Keyhole-Edit）の物理・仮想アドレス、サイズを決定する（各種用語については後述）

※例えば再起動後はSEAM Rangeは揮発性につき当然ワイプされているので、「常駐しているTDX Module」というのは存在しないと考えられる。そして、参考文献[5]にも、メモリ上のTDX Moduleが存在しない場合の「Reload Scenario」というものについて言及している。つまり、上記の手順3のSVNチェックは、あくまでもアップデートをする場合の話であると推察される。

TDX Moduleのロード

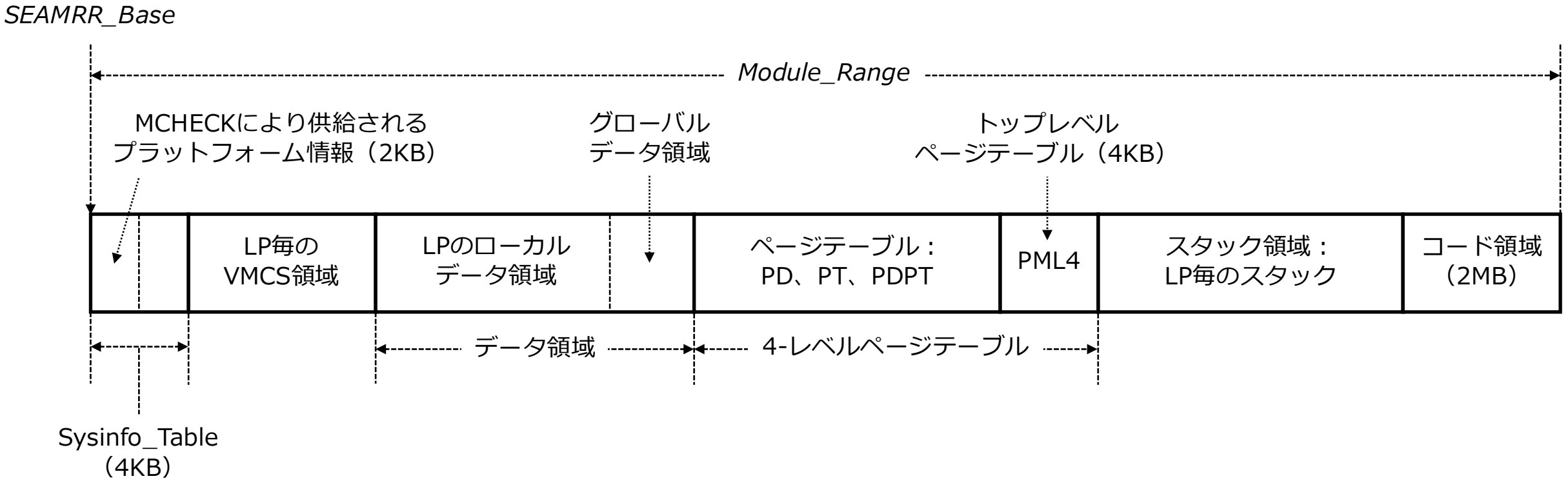
- （続き）：
  5. TDX ModuleのバイナリイメージをSEAM Rangeにロードし、イメージを測定し、TDX Moduleのハッシュ値の計算と検証を行う
  6. TDX ModuleのSysinfo_Tableを設定する。
  7. 各LPに**SEAM Transfer VMCS**（前回記事で詳述済み）を設定する
  8. TDX Moduleのハッシュ値とSVNをP-SEAM Loaderのデータ領域に記録する
     - 前ページの注意書きにも関連するが、アップデート時にはこの領域に保存したメタデータを参照して手順3等を実施できると思われる。

- P-SEAM Loaderはこのロード用インタフェース以外にも、P-SEAM Loader自身のシャットダウンや、P-SEAM Loaderのシステム情報を取得するための他のインタフェース機能も提供する

TDX Moduleのメモリレイアウト

TDX Moduleの物理メモリレイアウト

- TDX Moduleの物理メモリレイアウトの図を以下に示す（図は[1]を参考に作成）：



TDX Moduleの物理メモリレイアウト

- TDX Moduleの物理メモリレイアウトは、まずTDX Moduleの**Sysinfo_Table**を保持する4kBテーブルから始まる
- Sysinfo_Tableには、まずNP-SEAM Loaderによって**MCHECK**で入力された、2kBのプラットフォーム情報が含まれる
  - MCHECKについての説明は後述
- 同時に、Sysinfo_TableはP-SEAM Loaderによって入力された、以下の構成のTDX Module情報を含む：
  - SEAM Rangeのベースアドレスとサイズ
  - 各種メモリ領域それぞれに対するベース仮想アドレス（複数）
  - LP数
  - プライベートHKIDの範囲

TDX Moduleの物理メモリレイアウト

- 次に、**LP毎のVMCS領域**が配置される
  - 以前説明の通り、各LPには4kBのSEAM Transfer VMCSが割り当てられる
- その次には**データ領域**が配置される
  - これはLP毎のローカルデータ領域と、グローバルデータ領域とに分割される
- さらに次には**4-レベルページテーブル**（PT, PD, PDPT, PML4）が配置される
  - 4-レベルページテーブルについては前回スライドを参照

TDX Moduleの物理メモリレイアウト

- そして、その後ろには**LP毎のスタック領域**が配置される

- 最後に、**TDX Moduleの実行コード用のコード領域**が配置される
  という形で、TDX Moduleの物理メモリレイアウトは一式が揃う

MCHECKとXuCode

- **MCHECK**：Intel TXTのACMに似た、Intel署名付きのファームウェアモジュール
  - ただしACMとは別物である
- MCHECK自体も、参考文献[6]によれば**XuCode**というものの一部（Non-Persistent XuCode）として実装されている
  - ただし、[6]は一次情報ではないため真偽は不明
- もしこれが真であれば、MCHECKはP-SEAM Loaderに対するNP-SEAM Loaderと同様の立ち位置の存在であると言える
  - 通常の（Persistent）XuCodeに対するNP-XuCodeという立ち位置

MCHECKとXuCode

- MCHECKはBIOSブートシーケンスの初期段階で実行され、以下のような処理を実施する：
  - CPU構成の確認
  - プラットフォーム固有のIntel SGX初期化
  - メモリ暗号化エンジンのメモリ暗号化鍵のプログラミング（生成）
  - CPUインターコネクト暗号化エンジンのリンクキーのプログラミング。XuCodeや、SGXのメモリー貫性トラフィックを保護するために使用される
  - XuCodeインストール前のメモリ初期化
  - XuCodeへの関連情報受け渡し

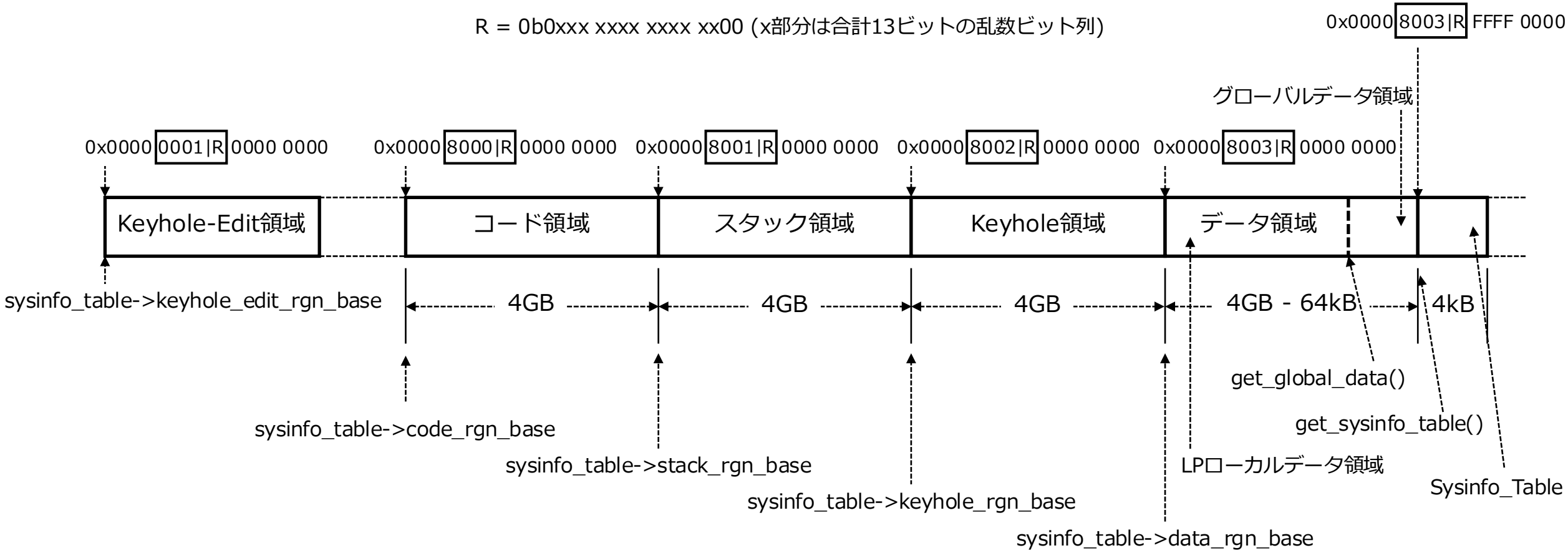
MCHECKとXuCode

- **XuCode**：SGXのメモリ管理やEnclave操作をはじめとした一部のSGX命令を実装する拡張マイクロコード（eXtended uCode）
  - 通常のx86マイクロコードからは独立している

- これは筆者の憶測でしか無いが、上記MCHECKの説明はSGX時代のものであるため、TDXを導入するにあたりさらに機能追加されている可能性もある
  - 例えばSysinfo_Tableへのプラットフォーム情報注入等

TDX Moduleの仮想メモリレイアウト

- 以下にTDX Moduleの仮想メモリレイアウトの図を掲載する（図は[1]内の図を参考に作成）：



TDX Moduleの仮想メモリレイアウト

- TDX Moduleは独自の仮想アドレス空間を持ち、そのアドレス変換のためのページテーブルも自身のメモリレイアウト上に保持する
  - 物理メモリレイアウトで言えばModule_Range内
- TDX Moduleの仮想アドレス空間は、P-SEAM Loaderがこの TDX Module用ページテーブルを構築する事で確立される
- 前述の仮想メモリレイアウト図における「R」の部分の通り、P-SEAM LoaderはTDX Moduleに対する仮想アドレスの**第34～46bit部分をランダム化**する
  - これは何がどこにあるかを予め把握した上でBoF等でメモリを破壊し、目当ての場所のコードを攻撃に転用したりする**メモリ破壊攻撃を対策**するための仕様である

TDX Moduleの仮想メモリレイアウト

- 図3を見るとコード、スタック、Keyhole（後述）、そしてデータにそれぞれ4GB程を割り当てるというかなりのメモリの大盤振る舞いをしているように見える

- しかし、これは仮想アドレス空間上の話なので、実際のメモリ消費量はずっと少ない
  - 例えば仮想アドレス空間上で4GBのコード領域は、物理アドレス空間上では2MBしか消費しない

- このように仮想側が多めに取られているのは、P-SEAM Loaderによる仮想アドレスビットのランダム化により互いに衝突しないよう、十分なスペースを空けるためであると考えられる

TDX Moduleの仮想メモリレイアウト

- 各領域に対する（ベース）仮想アドレス及びその領域のサイズに関しては、Sysinfo_Tableに格納される

- TDX Moduleの仮想メモリ領域の中でも、コード、スタック、データ、Sysinfo_Tableの4つに対するPTE（ページテーブルエントリ）に関しては、事前に静的に入力しておく事が可能である
  - これにより、これらのPTEはランタイム中に変更を加える必要がなくなる

- 例外として、TDX Module実行中に外部SWから動的に渡されるデータをマッピングするための**Keyhole領域**はランタイム中に編集できなければならない
  - この実現のためには**Keyhole-Edit領域**というものが使われる。詳細は後述

TDX Moduleの初期化と設定

TDX Moduleの初期化と設定

- P-SEAM LoaderによりTDX Moduleがロードされると、その後の初期化等の処理はそのマシンのホスト側HV（ハイパーバイザ）が引き継いで実施する

- HVはまず、**SEAMCALL[TDH.SYS.INIT]**を実行し、TDX Moduleをグローバルにわたる形で初期化する

- 次に、HVは各LP上で**SEAMCALL[TDH.SYS.LP.INIT]**を実行する
  - このSEAMCALLは、後述するKeyhole領域の他、前述したLP毎のデータ/スタック領域といった、LP毎のパラメータのチェックと初期化を実施する

TDX Moduleの初期化と設定

- その後、HVはグローバルなプライベートHKIDを割り当て、**SEAMCALL[TDH.SYS.CONFIG]**を介してそれをTDX Moduleに渡す
  - これにより**TDMR**（Trust Domain Memory Region）の初期化も実施される
  - TDMRについての詳細に関する説明は後述
- このSEAMCALL[TDH.SYS.CONFIG]自体は、TDX Moduleのグローバル設定を行うための汎用的な命令である[5]
- 「グローバルなプライベートHKID」とは、次ページで説明するTDXグローバル秘密鍵に紐づけるHKIDを指す

TDX Moduleの初期化と設定

- そして、各プロセッサパッケージはそれぞれ**SEAMCALL[TDH.SYS.KEY.CONFIG]**を実行する

- これにより**TDXグローバル秘密鍵**が生成され、前の手順で割り当てられたグローバルなプライベートHKIDとバインドされる

- TDXグローバル秘密鍵は、**PAMT**や**TDR**のように文字通り特定のTDには完結しないかタイミングの問題でできない、TDX上のグローバルな要素を配置するメモリの暗号化に使われる
  - このバインドのため、内部で以前説明した**PCONFIG命令**が使われる
  - **PAMT**：Physical Address Metadata Table
  - **TDR**：Trust Domain Root

TDX Moduleの初期化と設定

- 最後に、HVは**SEAMCALL[TDH.SYS.TDMR.INIT]**を複数回呼び出し、各TDMRのPAMTを徐々に初期化していく

- この一連の手続きをもってTDX Moduleの初期化と設定が完了する

TDのメタデータ

TDのメタデータ

- TEEにおいては、TEE機能自体やTEEインスタンス等に関する**各種メタデータを安全に保持・管理**する事が**不可欠**である
  - これはTEEのライフサイクル管理で重要な役割を果たすだけでなく、RAによって検証する対象にもなり、無くてはならない要素である

- 例えばIntel SGXは、CPUマイクロコードによる処理のみがアクセスできるEPC（保護メモリページ）上のSECSというデータ構造に、MRENCLAVEやMRSIGNERといったEnclaveのメタデータを格納する

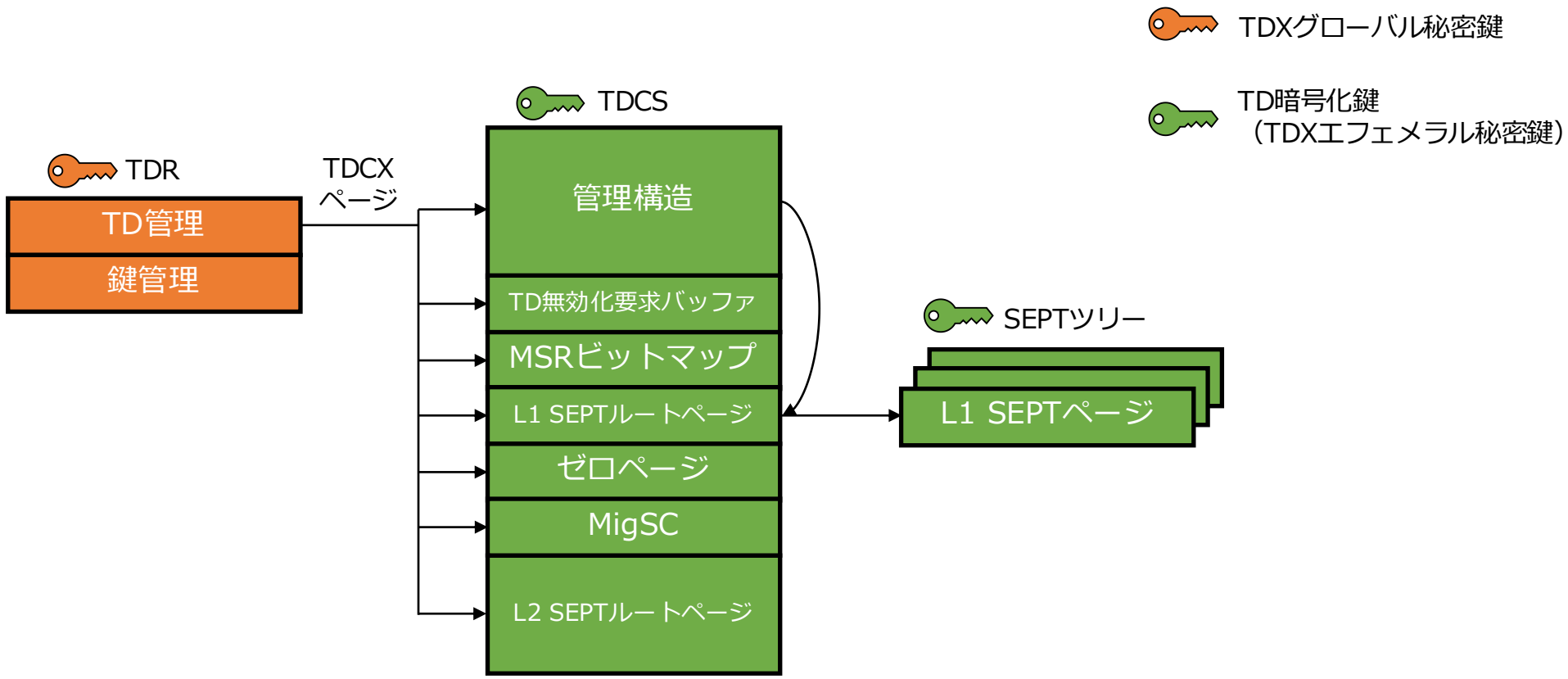
TDのメタデータ

- TDXにおいても様々なメタデータが存在し、TDのライフサイクル全体を管理する責任を持つ**TDX Module**が、**各TDインスタンスのメタデータを保持**する
  - これらはHV等によるアクセス・改竄を防ぐため、適切なメモリ暗号化が施された領域に配置され管理される

- TDXのメタデータは、以下の4つの制御構造によって構成される：
  - **TDR**（Trust Domain Root）
  - **TDCS**（Trust Domain Control Structure）
  - **TDVPS**（Trust Domain Virtual Processor State）
  - **SEPT**（Secure EPT）

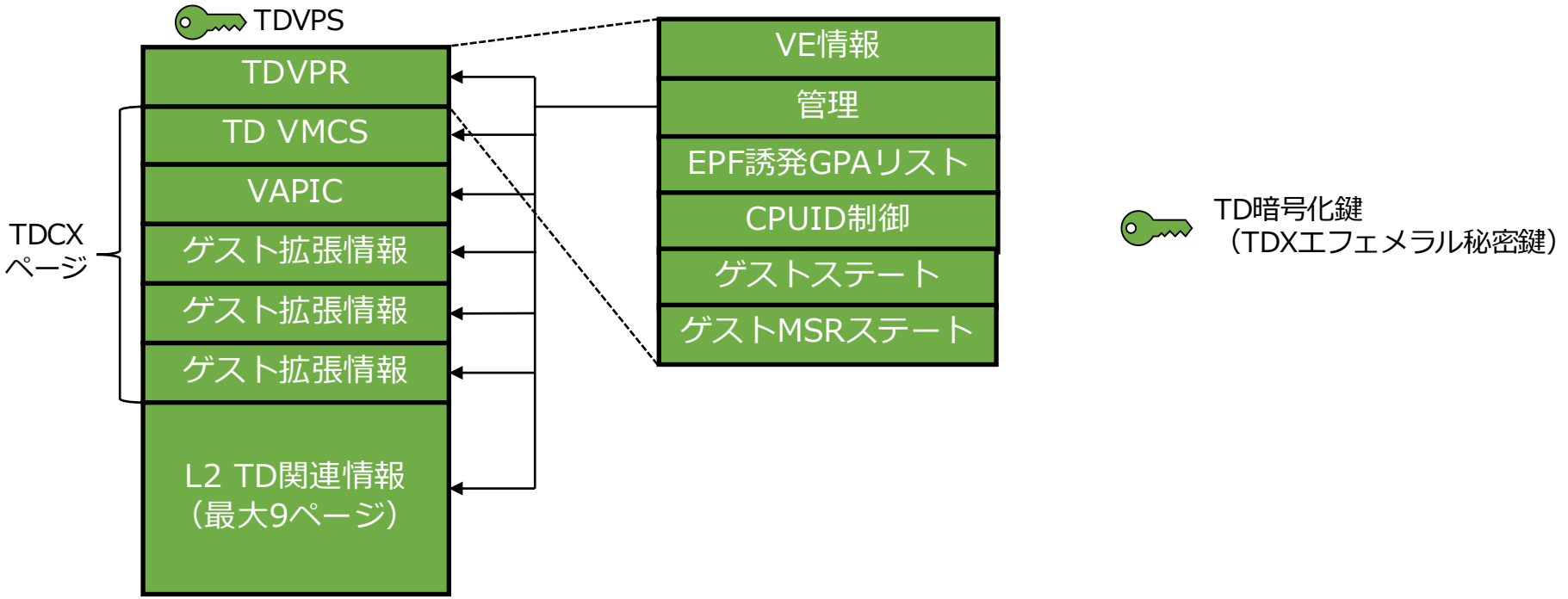
TDのメタデータ

- これらのメタデータの相互関係を、Demystified論文[1]内の図をベースに最新のTDX 2.0の情報に準じて修正した図を、まずはTDスコープのメタデータの分から掲載する：



TDのメタデータ

- 同じく、次はvCPUスコープのメタデータ（TDVPS）についての図を以下に示す：



TDのメタデータ

- これらのメタデータは全て**TDMR**というメモリ上に配置される

- 詳細は次回以降のスライドに譲るが、TDMRは簡単に言えば**TD関係の保護すべきデータ**（TDやTDメタデータ等）を**配置**するための**暗号化されたメモリページ**の事である
  - SGXで言えばEPCが限りなくこのTDMRに近い

- 直後に詳述するが、例えばTDCS等に使われるメモリページであるTDCX（後述）は、「TDMR内のTDCS用のメモリページ」という事になる
  - よって、TDCXも実体としてはこのTDMRである事になる

TDR

- TDRは、TD開始時に作成され、TD終了時にデストラクトされる「初期構造体」（Initial Structure）である

- 簡単に言い換えれば、**そのTDについての代表として機能**する、主に**参照の起点となる構造体**と言える

- SEAMCALLは、TDの全ライフサイクル中において、このTDRの物理アドレスを使用する事で対応するTDの参照を行う

TDR

- TDRはメモリ暗号化鍵の情報を含む部分と、**TDCX**ページへの参照の2つの部分により構成される

- TDCXはTDCS用の4kBサイズの物理ページであり、**TDCS**はこの**TDCXの集合で構成されている**事になる
  - そしてその実体は前述の通りTDMRである
  - **TDCX**：Trust Domain Control Extension[5]

TDR

- TDRはTDX Moduleが生成を要求する、TDのメモリを暗号化する秘密鍵（**TDエフェメラル秘密鍵**）が作成される前に作られるため、前述の通りTDXグローバル秘密鍵により暗号化される

- 他のTDCS、TDVPS、SEPTといったメタデータは、TDのメモリページと共にPAMTのowner属性を通じてTDRに関連付けられる
  - この辺りの議論は後述にてより詳しく説明する

TDCS

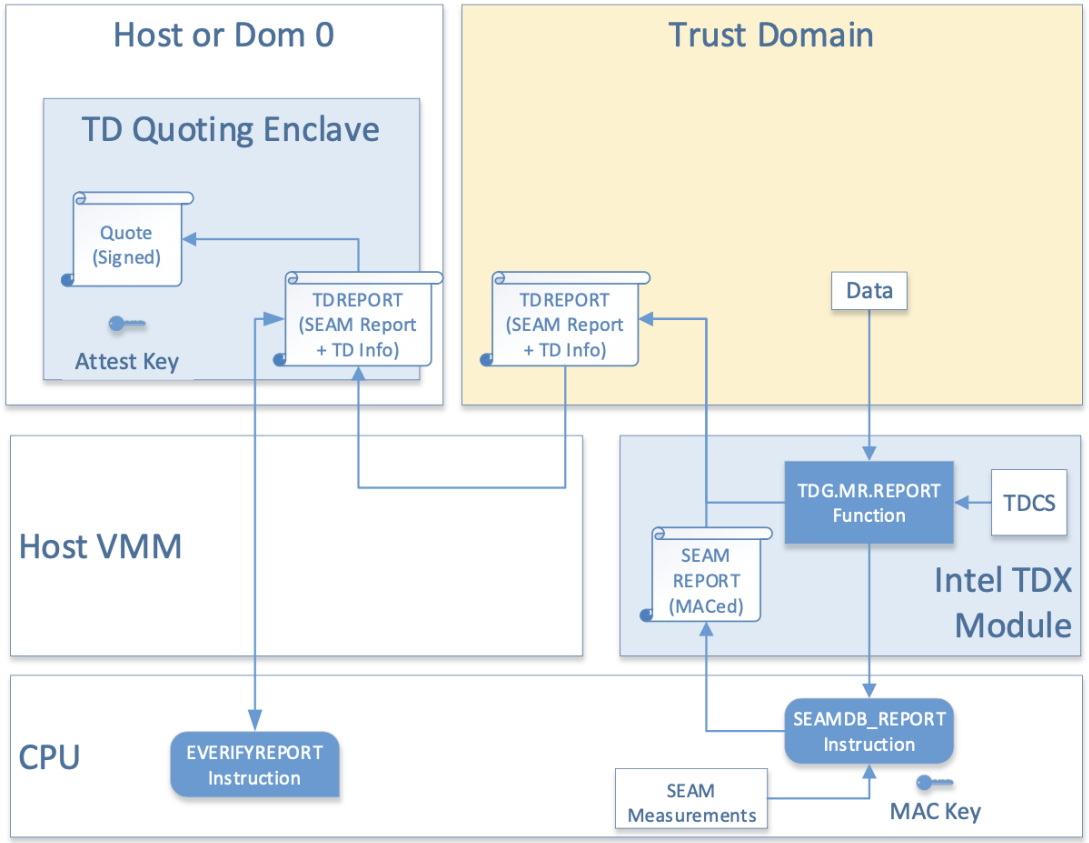
- TDCSは、**対応するTDの動作を管理し、状態を保持**するための制御構造である
  - これもSGXで言えばSECSに限りなく近い存在
- TDCSは前述の**TDCX**という**独立した領域**に配置される
  - これはIntel公式文書[5]においても「The TDCS is not intended to be directly accessible to software.」という具合に、独立の区画である事が示されている
- このTDCSは、TDR生成時に生成される、対応するTDと同じ**TDエフェメラル秘密鍵によって暗号化**される

TDCS

- TDX Moduleのソースコードから読み取れる情報に基づくと、以下の12個のTDCXページから構成されるデータ構造である：
  - **ページ0-3**：TD管理構造
    - 大まかに、TD全体管理情報、実行制御（SEPTルートページへの参照を持つEPTPを含む）、Attestation関係（測定値）、マイグレーション制御、CPUID仮想化、APIC関連情報、TDX-IO（TDISPのTDX側実装であるTDX Connect）設定が含まれるが、厳密にそれぞれが何ページ目に配置されるかは不明。詳細はソースコードを参照
  - **ページ4**：TD無効化（TD Invalidation）リクエストバッファ。詳細は不明だが、[10]を見るにTDX Connect関係か
  - **ページ5**：MSRビットマップ
  - **ページ6**：SEPTルートページ
  - **ページ7**：特別なゼロページ（オールゼロのページ）
  - **ページ8**：マイグレーションストリームコンテキスト（MigSC）
  - **ページ9-11**：L2 SEPTルートページ。TD Partitioning（Nested-VM機能）用と思われる

TDCS

- AttestationにおけるREPORT（TEEの身元証明書）に含まれる一部の各種測定値（ハッシュ）は、このTDCSからフェッチされる事からも、このTDCSは非常に重要な存在である



Intel公式文書[5]の12.5章内の図を引用

TDVPS

- TDVPSはTDの各vCPUについての制御構造であり、文字通りvCPU1個につき1つ（vCPUスコープ）のTDVPSが割り当てられる
  - 例えばTDが複数のvCPUを持つ場合は、それぞれのvCPUに対するTDVPSが払い出される

- TDVPSは2026年5月のTDX 2.0現在、**TDVPR**（Trust Domain Virtual Processor Root）を先頭とし、続く14のTDCXからなる、合計15のメモリページから構成される

TDVPS

- Demystified論文ではTDCXではなく「**5つのTDVPX**」としているが、これはTDX1.0の情報であり[9]、**TDX1.5以降は廃止**されている
  - スコープは違えどわざわざ名前を分ける程ではないという判断と思われる

- このTDVPSも、TDCS同様に対応するTDの**TDエフェメラル秘密鍵で暗号化**される

TDVPS

- TDVPSは、TDX Moduleのソースコードを見ると以下のようなレイアウトとなっている：
  - **ページ0**：TDVPR
    - これには、VE例外情報、他のTDVPSページ（TDCX）への参照を行う管理構造、EPF（詳細不明だが、EPT（SEPT）によるアドレス変換に失敗した際に発生するページフォールトの事らしい。Extended Page Faultの略か）を誘発したゲスト物理アドレス（GPA）リスト、CPUID制御、ゲストステート、ゲストMSRステートが含まれる
  - **ページ1**：TD Transfer VMCS
  - **ページ2**：VAPIC
  - **ページ3-5**：ゲスト拡張情報（ステート）
  - **ページ6-14**：L2 TD（TD Partitioning）関連情報
    - 2026年3月現在、TD Partitioningで生成できるL2（Nested）TDの数は3個で、それぞれに対し以下の3つのページが割り当てられるため、最大で合計9ページ。
      - **ページA**：L2 VMCS
      - **ページB**：MSRビットマップ
      - **ページC**：シャドウMSRビットマップ

SEPTと共有EPT

- 以前にも述べた通り、HVがEPTを使用してGPA（ゲスト物理アドレス）からHPA（ホスト物理アドレス）へ変換する従来型VMの作法は、HVを信頼対象外とするTDXの脅威モデルに違反する

- よって、TDXでは**SEPT**と**共有EPT**の2つのEPTが導入されている
  - いずれもEPTの一種であるため、GPA-HPA間の変換を行う役割を持つ[5]

- SEPTは**TDのプライベートメモリのアドレス変換**に使用され、**そのTDのTDエフェメラル秘密鍵によって保護**される
  - この**SEPTのルートページ**はTDCSに格納され、SEPTへの参照はこのルートページを経由して行われる
  - つまり**SEPTルートページ**はその**SEPTの代表となるようなページ**とも言える

SEPTと共有EPT

- 一方で共有EPTは、仮想化I/Oのケースのように、TDがHVと明示的に共有するようなメモリのアドレス変換に使用される
  - こちらはSEPTとは異なりTDX ModuleではなくHVの制御下に存在する

- TDのゲストカーネルは、GPAのSharedビットを1にする事で、共有メモリページとするページを決定する事ができる

Keyhole領域

Keyhole領域

- SEAMCALLが呼び出される際には、その処理に必要な情報も一緒に引数的に渡される事も多々ある

- この情報が格納されたホスト側メモリバッファは、TDX Moduleが自身の仮想アドレス空間にマッピングする事で、初めてアクセスできるようになる

- このSEAMCALL経由のバッファ割り当てを実現するために使用されるTDX Module用仮想アドレス範囲として、**Keyhole領域**と**Keyhole-Edit領域**というものが存在する
  - あくまでもホスト側メモリバッファを安全にTDX Moduleが参照するための仕組みであるため、これらはTDCALL時には使用されない

Keyhole領域

- **Keyhole領域**：SEAMCALLでTDX Moduleに渡されたデータ本体をマッピングするための領域本体

- Keyhole領域は、「Keyhole」と呼ばれる単位（**セグメント**）から構成される**単一の配列**という形で存在する

- この1配列は最大128個のKeyholeセグメントから構成され、各LPにはこの最大128個のKeyholeの1配列が1つ割り当てられる[18][19]
  - Demystified論文では1LPにつき1セグメントを割り当て、セグメント総数の上限が128個とあるが、これは**古い情報であるか単純に誤り**であるため注意
  - 現行のTDX2.0実装ではセグメント総数上限の定義は特に無い

Keyhole領域

- Keyholeとデータのマッピングは、Keyholeに対応するPTEが、SEAM Range外のデータ（**物理ページ**）とKeyhole（**仮想ページ**）を**マッピング**する事で実現される
  - つまりKeyholeという4kBにアラインされた仮想ページは、**仮想アドレスレイアウト上**は**TDX Moduleが有している**が**参照先はホスト側**という、境界をまたいだ参照を持つ事になる
- もし同一メモリページ内に複数のメモリバッファが存在し得る場合には、各Keyholeはページ上の参照バッファの総数を追跡するための参照カウントを保持する
- TDX ModuleはLP毎のデータ構造をセットアップする際、LRUリストを用いて空きKeyholeを整理する

Keyhole領域

- ややこしい点として、**Keyhole自体**も**元々の物理アドレス**を当然**有している**
  - これはTDX Moduleのロード時に初期化されるものであるため、SEAM Range内に存在する
- しかし、Keyhole自体は単なるマッピングインタフェースに過ぎないため、Keyhole自体の**元々の物理アドレスには用はない**
- よってKeyholeの仮想アドレスは、リーフレベルPTEにおいて、対応する物理アドレスとして**オール0な空の物理アドレスが割り当てられる**
  - リーフレベルPTE：4-レベルページテーブルのような階層型ページテーブルにおける一番下層に位置する構造のエントリ。つまりはいわゆる「ページテーブルエントリ（PTE）」[11]

Keyhole-Edit領域

- このようにKeyhole用PTEを駆使しながらホスト側バッファのKeyholeへの割り当てを行うが、当然この**Keyhole PTEを配置する場所が別途必要**になる

- このKeyhole PTE自体が載る物理アドレスがマッピングされるのが**Keyhole-Edit領域**である

- Keyhole-Editや前述の元々のKeyhole物理アドレスの所在の特定は難しいが、TDX Moduleロード時に初期化されるため、設計的にもSEAM Range内に存在すると考えるのが自然
  - よって、Keyhole-Edit領域はSEAM Range用TME-MK鍵で暗号化される事になる

Keyhole-Edit領域

- ホスト側メモリバッファを受け渡すようなSEAMCALLを処理する際には、TDX Moduleはバッファのメモリページが既にKeyholeによってマッピングされているかどうかを確認する
  - マッピングされている場合、Keyholeの参照カウントをインクリメントし、マッピングされた仮想アドレスを返す
  - マッピングされていない場合は、LRUリストから空きKeyholeを選択し、Keyhole-Edit領域上の対応するPTEをアップデートする事で、このKeyhole仮想アドレスにメモリバッファの物理アドレスを紐づける
- 各SEAMCALLの終了時に、そのSEAMCALLに対応するKeyhole参照カウントはデクリメントされ、参照されなくなったKeyholeはLRUリストに戻る

物理メモリの管理

TDMRとCMR

- TDX Moduleは、一連のTDMRとその制御構造であるPAMTを用いて物理メモリ管理を行う

- TDMRは、TDのプライベートメモリやメタデータに使用できるメモリ領域である**CMR**のリストに基づきHVにより構築される
  - **CMR**：Convertible Memory Regions
  - TDMRの構成設定は、SEAMRRのようなHWベースのレンジレジスタではなく、あくまでもHVやBIOSのようなSWにより管理されるという仕様となっている

- これまたSGXでは、TDMR相当のEPCがCMR相当のPRMから払い出されているのに類似した仕組みである
  - CMRもPRMもUEFI/BIOSによって用意されるという点も共通

TDMRとCMR

- TDMRは、TME-MKによる暗号化に加え、TDXのメモリ完全性保護機能（**Logical Integrity（Li）/Cryptographic Integrity（Ci）**）による保護の対象となる事ができる
  - LiやCiについては次のスライドで解説する

- 裏付けは取れていないが、TDMRに払い出される前の未使用CMRは、特に暗号化も完全性保護（Ci/Li）もされていないと推測される

TDMRとCMR

- それぞれのTDMRは**1GBの整数倍のサイズ**を持つ物理メモリ上の**単一の範囲**であり、**2つ以上のTDMR同士で重なる事はない**

- TDMRが複数存在し得るものであり、かつHVが構築するという事実から推測すると、典型的にはTDごとにTDMRを払い出すイメージであると思われる
  - SGXにおいて各Enclaveに対するEPCがPRMから払い出されるのと相似とも言える

- ただしSGXとの大きな違いとして、EPCとは異なり必ずしも**2のべき乗のサイズである必要はない**

TDMRとCMR

- TDMRには、TDX Moduleにより使用する事ができない予約領域が含まれている事がある
  - 予約領域は4kBページの配列として構成されており、例外的にCMR内に存在する必要すらない[5]

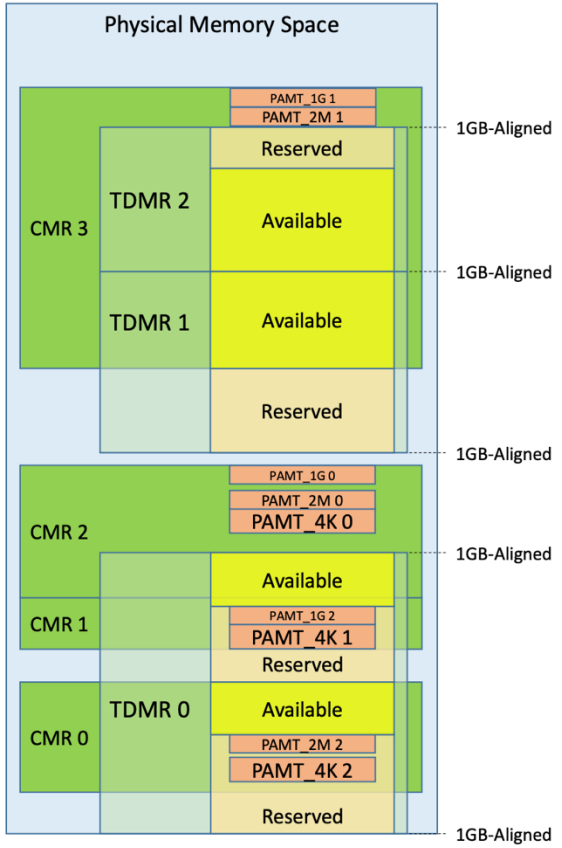


Figure 4.2: Example of Convertible Memory Ranges (CMRs) vs. Trust Domain Memory Regions (TDMRs)

Intel公式文書[5]内の図を引用

TDMRとCMR

- この予約領域はTDページに**変換不可能（not Convertible）**である
- 反対に、それ以外の全てのTDMR内のメモリはTDページに**変換可能（Convertible）**である必要がある
- Convertibleとは、TDXのセキュリティ特性の保護下でTDやTDメタデータに使える状態である事を指す
  - Convertibleの由来は、「共有ページからプライベートページに変換可能」[5]、転じて単にTDやTDメタデータのために使用（変換）できるから、という事であると推測できる

PAMT

- TDX Moduleは、**PAMT**を使用する事でTDMR内の各物理メモリページの**ページ属性**の追跡を行う

- この属性には、**ページ所有者**、**ページタイプ**、そして**ページサイズ**に関する情報が含まれる

- ページタイプ（PT）は、そのページが何に使われているかを示す
  - 例えばPT_TDR（TDR）、PT_REG（TDプライベートメモリ）といった具合である[5]

- ページ所有者（OWNER）にはそのページが割り当てられているTDのTDRが代入される[5]

PAMT

- このページ属性により、TDMR内の物理メモリページが**適切なタイプを持ち**、そして**最大1つのTDにのみ割り当てられる事**が**保証**される

- AMD SEV-SNPにおけるリバースマップテーブル（RMP）に似ているが、RMPと異なりGPAを保持しないため、完全性保護機能は有さない
  - TDMRの時点でTEE用なのが確定するため、RMPのようにHVかTEEかの識別子を持つ事も無い

- 最大1つのTDにのみ割り当てるという意味では、TDにとっては他の主体に妨害されていないページを使える事を確信できるという意味で、SEV-SNPにおけるPVALIDATE的な役割を持つとも言える

PAMT

- ページがTDのプライベートメモリに割り当てられたら、TDX ModuleはSEPT内におけるページの単位サイズと、PAMT内のそれとが一致しているかの確認を行う事もできる

- PAMTはブロックという単位に分割され、各ブロックは1GBサイズの範囲内でTDMRのページアドレスの追跡を行う

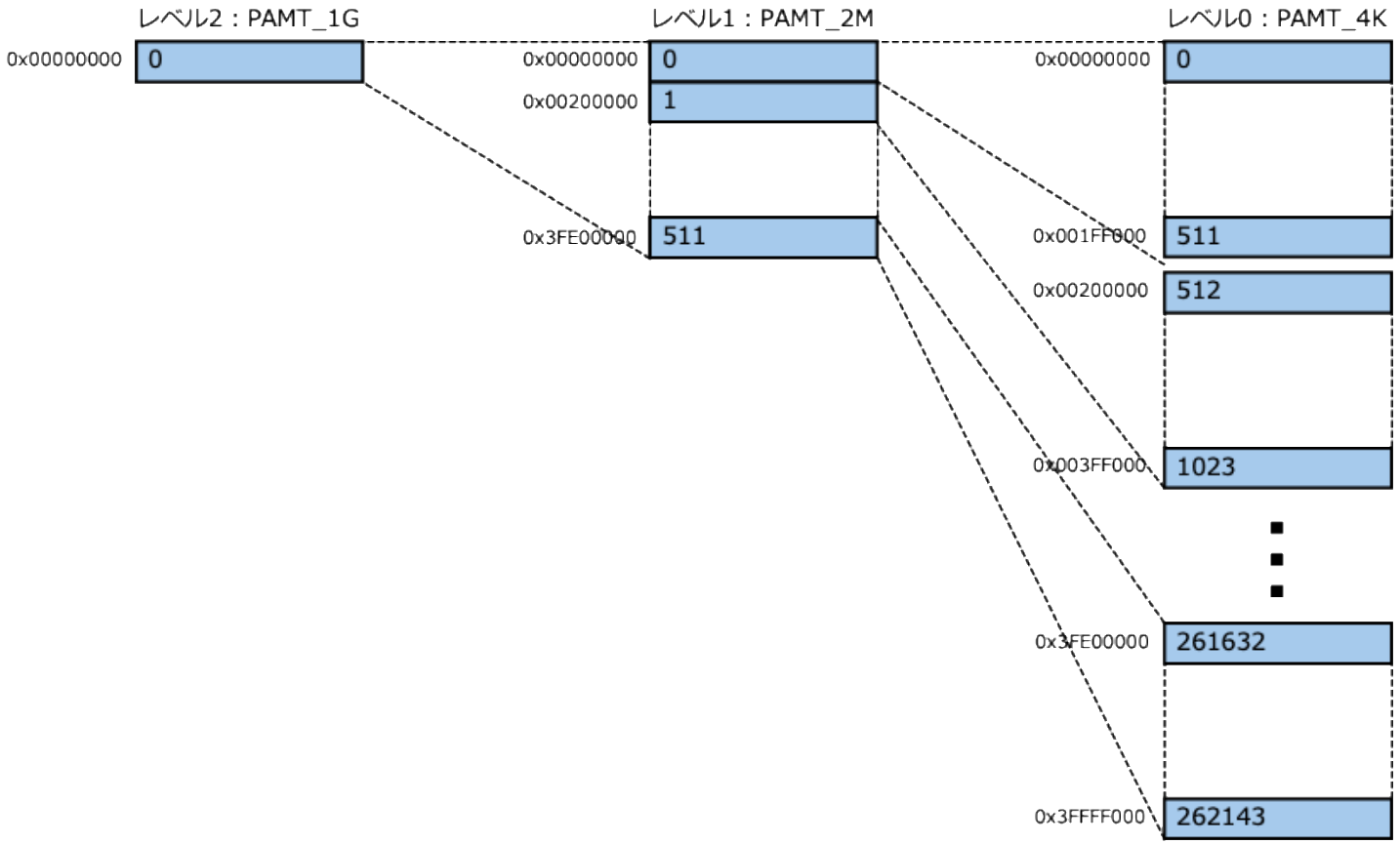
PAMT

- この各ブロックは、いわば3-レベルページングのような形で追跡を行うもので、**3つの内いずれかのレベルに振り分けられ**、各レベルはそれぞれ4kB、2MB、1GBページのメタデータ追跡を行う

- 各レベルの内訳は以下の通り：
  - 第1レベル：単一の1GBページ
  - 第2レベル：512個の2MBページ
  - 第3レベル：512×512個の4kBページ

PAMT

- このレベル構造の概要図を、Intel公式文書[5]内の図を参考にする形で以下に示す：



PAMT

- TDX Moduleは、物理アドレスを受け取るとPAMTのこの構造に対して階層的なページウォークを実行し、サニティチェックのために
  そのページ属性を取得できる

- この時、PAMTはTDMRのHPAで追跡を行う[5]
  - PAMTはブロックに分割されるとは言え、全体で見るとTDに対して横断的に管理するため、GPAで管理するモチベーションはそもそも無いと言える

- TDX Moduleは、TDX Moduleがランタイム時に使用する各ページのページ属性を更新する事で、データ構造を管理する
  - ページへのアクセス、削除、追加を伴う操作は全て、TDX ModuleがPAMTウォークを行いページ属性を調整し、対応するアクセス権をチェックする事に繋がる

PAMT

- PAMT用のメモリはHVにより割り当てられ、TDX Moduleの TDXグローバル秘密鍵により暗号化される

- SEPTとPAMTの違いがややこしいかも知れないが、**SEPT**は TDプライベートメモリの**アドレス変換**を行い、**PAMT**は**所有者や タイプの一貫性を管理**するものであるため、**互いにまるで別物**である

- SGXで例えるならば前者はOSページテーブル、後者はEPCM相当 であり、如何に違う役割をそれぞれ持つかがよく分かる

SEAMOPS

SEAMOPS

- TDX Module含むSEAM VMX rootの主体は、ある動作を実現する上でCPUの提供する機能に依存する必要がある場合に**SEAMOPS命令**を実行する事がある

- SEAMOPS命令はSEAM VMX rootによってのみ呼び出し可能であり、TDCALLやSEAMCALLと同様にリーフ関数を受け取る[20]

- リーフ関数にはAttestation編の資料で説明するものの他にも、いくつか存在する事がソースコードや公式文書から読み取れる

SEAMOPS

- SEAMOPSの特定可能な限りのリーフ関数の一覧は以下の通り[21]：
  - SEAMREPORT、SEAMDB_REPORT、SEAMVERIFYREPORT以外のリーフ関数名は[21]内の記述からの推測である

リーフ関数	説明
CAPABILITIES	その環境が対応しているSEAMOPSリーフ関数を列挙する。ソースコードより推測
SEAMREPORT	TD Reportの構成要素の一部をなすSEAMREPORT構造体を生成する。古いリーフ関数のようで、現行では既に使われていないようである
SEAMDB_REPORT	TD Reportの構成要素の一部をなすSEAMREPORT構造体を生成する。その環境がTD-Preserving機能に対応している場合に、SEAMREPORTではなくこちらが使用される。つまり現行では基本的にこちらが使われる
SEAMDB_CLEAR	TD-Preservingで使用されるSEAMDB（TD-PreservingにおけるSEAMコンテンツの世代管理DB？）というDBのエントリのクリアを行うと推測される。P-SEAM Loaderにより実行される。ソースコードより推測
SEAMDB_INSERT	SEAMDBへのエントリの挿入を行うと推測される。P-SEAM Loaderにより実行される。ソースコードより推測

SEAMOPS

- 前ページの続き：

リーフ関数	説明
SEAMDB_GETREF	現時点でのSEAMDBの最新のインデックス、nonce、サイズを取得するものであると推測される。TDX ModuleとP-SEAM Loaderの双方から呼び出される。ソースコードより推測
SEAMVERIFYREPORT	TD ReportをLocal Attestation的に検証するためのリーフ関数[5]。
GET_KEY	TDKEYREQUEST（TDXにおけるシーリング機能）の中核をなすSEAMOPSリーフ関数。TDCALL[TDG.MR.KEY.GET]の内部で呼び出される。
SEAMDB_NONCE	不明
SEAMDB_LOAD	不明
SEAMDB_STORE	不明
SEAMFEATURES	不明

まとめ

- TDX Moduleの詳細について説明した

- TDXにおけるメタデータや、TDXのコンテンツが載る専用のメモリ領域についての詳細についても説明した

参考文献

[1] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. ACM Comput. Surv. 56, 9, Article 238 (September 2024), 33 pages. https://doi.org/10.1145/3652597

[2] “Trusted Execution Technology”, 2026/5/14閲覧, https://en.wikipedia.org/wiki/Trusted_Execution_Technology

[3] “GETSEC[ENTERACCS] — Execute Authenticated Chipset Code”, 2026/5/14閲覧, https://www.felixcloutier.com/x86/enteraccs

[4] “Intel® Trust Domain Extensions - SEAM Loader (SEAMLDR) Interface Specification”, Intel, https://cdrdv2-public.intel.com/739045/intel-tdx-seamldr-interface-specification.pdf

[5] “Intel® Trust Domain Extensions (Intel® TDX) Module Base Architecture Specification”, Intel, https://cdrdv2.intel.com/v1/dl/getContent/733575

[6] “Intel Trust Domain Extensions (TDX) Security Review”, Erdem Aktas et al., https://services.google.com/fh/files/misc/intel_tdx_-_full_report_041423.pdf

参考文献

[7] “XuCode: An Innovative Technology for Implementing Complex Instruction Flows”, Intel, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/xucode-implementing-complex-instruction-flows.html

[8] “Intel® Trust Domain Extension Research and Assurance”, Intel, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/tdx-security-research-and-assurance.html

[9] “Architecture Specification: Intel® Trust Domain Extensions (Intel® TDX) Module”, Intel, https://cdrdv2-public.intel.com/733568/tdx-module-1.0-public-spec-344425004.pdf

[10] “Intel® TDX Connect Architecture Specification”, Intel, https://cdrdv2.intel.com/v1/dl/getContent/773614

[11] “Understanding Page Tables in xv6 on RISC-V”, 2026/5/19閲覧, https://cs326-s25.cs.usfca.edu/guides/page-tables

[12] “TDXdown: Single-Stepping and Instruction Counting Attacks against Intel TDX”, Luca Wilke et al., https://uzl-its.github.io/tdxdown/tdxdown_preprint.pdf

[13] “TDXploit: Novel Techniques for Single-Stepping and Cache Attacks on Intel TDX”, Fabian Rauscher et al., https://www.usenix.org/system/files/usenixsecurity25-rauscher.pdf

参考文献

[14] “TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition”, Jalen Chuang et al., https://tee.fail/files/paper.pdf

[15] “Speculative Execution Side Channel Mitigations”, Intel, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-execution-side-channel-mitigations.html

[16] “Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks”, Enrico Barberis et al., https://www.usenix.org/system/files/sec22-barberis.pdf

[17] ”LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection”, Jo Van Bulck et al., https://lviattack.eu/lvi.pdf

[18] “keyhole_manager.c”, GitHub, https://github.com/intel/confidential-computing.tdx.tdx-module/blob/TDX_MODULE_2.0.14/src/common/memory_handlers/keyhole_manager.c

[19] "keyhole_manager.h", GitHub, https://github.com/intel/confidential-computing.tdx.tdx-module/blob/TDX_MODULE_2.0.14/src/common/memory_handlers/keyhole_manager.h

[20] “Intel® Trust Domain CPU Architectural Extensions”, Intel, https://cdrdv2-public.intel.com/733582/intel-tdx-cpu-architectural-specification.pdf

参考文献

[21] “vmcs_defs.h”, Intel, https://github.com/intel/confidential-computing.tdx.tdx-module/blob/TDX_MODULE_2.0.18/src/common/x86_defs/vmcs_defs.h#L402-L422