7. Attestationの基礎とLocal Attestation

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

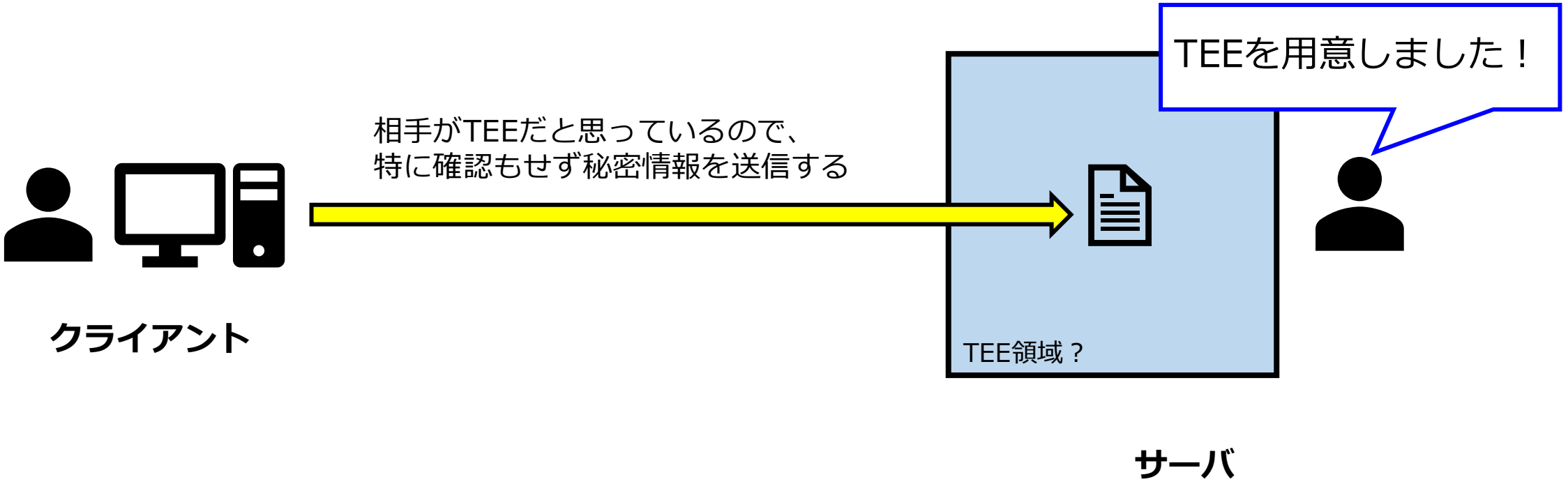
- TEEにおいて極めて重要な概念であるAttestationの必要性とその概念について理解する

- TDXにおけるAttestationの重要な一部である、Local Attestationについて理解する

Attestationの基礎

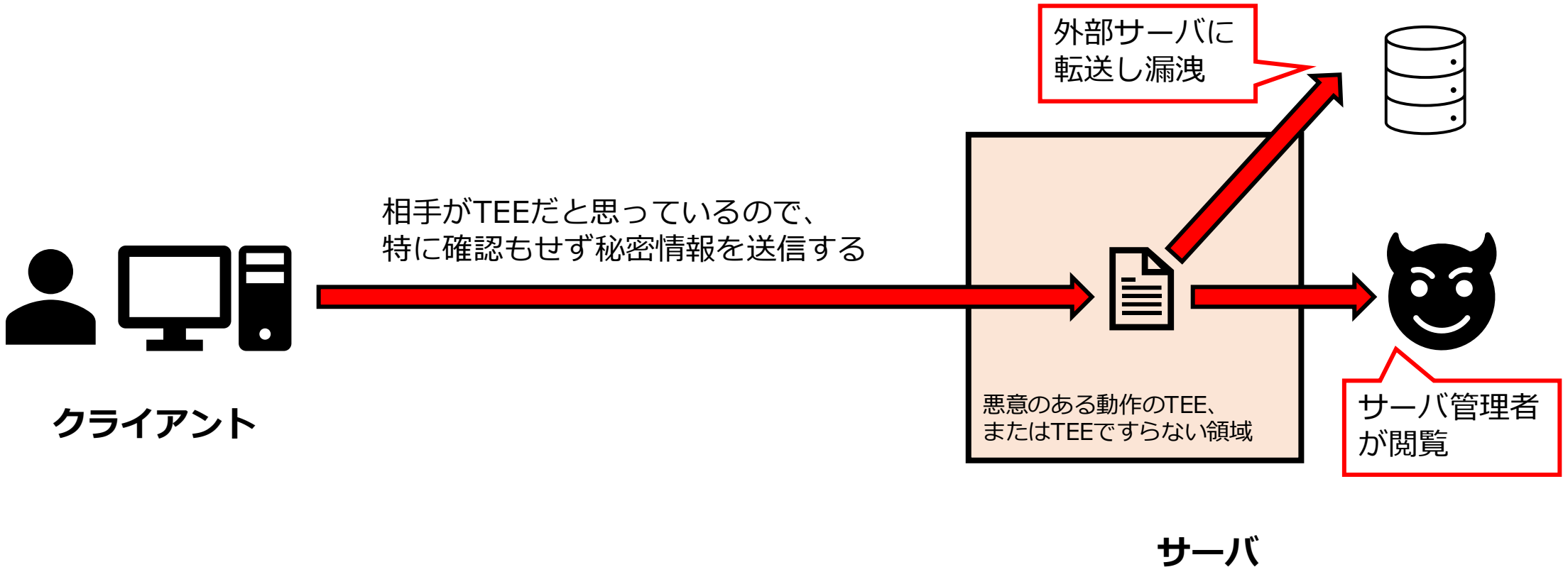
TEEは無条件には信頼できない

- リモートから使用しようとしているTEEは、TEEであれば無条件に信頼して秘密情報を送っても良いか？



TEEは無条件には信頼できない

- 答えはNOで、相手のTEEの真正性を確かめずに秘密情報を送信する事は、極めて危険な行為である



TEEは無条件には信頼できない

- 真正性を確認しないと、相手が以下のような環境を用意している可能性を否定できない
  - 受信したデータを復号しそのまま外部に流出させる、**悪意のあるTEE**を用意して待ち構えている
  - 悪意が無くても、**重大な脆弱性を抱えたマシンやTEE**を使用しており、その脆弱性を突くとTEE内の情報が漏洩してしまう可能性がある
  - そもそも**TEEすら用意しておらず**、平文環境に秘密情報を送信させようとしている

TEEは無条件には信頼できない

- このようなTEEの偽装は、悪意のある攻撃者（例：悪性のクラウド事業者、第三者攻撃者）からすればユーザの秘密情報を窃取するのに使えるため、多分にモチベーションが高くなる

- もう1つ面白い例として、クラウド事業者がリソースの肥大化を嫌い、TEEを謳いながら実はTEEを用意しないモチベーションがある、というものがある[1]

TEEは無条件には信頼できない

- 例えば通常のVMでは、KSM（Kernel SamePage Merging）という機能がメモリを走査し、VM間で重複があればそれをマージしてメモリ使用量を削減する

- しかし、Intel TDXのようなCVM（Confidential VM）では、VMごとに別々の鍵で暗号化されるため、このような機能は使えない

- 結果、メモリ使用量が肥大化し金銭的コストも高くなるので、クラウド事業者がこれを嫌いTEEを有効化せず偽装する、というモチベーションがあるというものである

TEEを信頼する方法

- やり取りしようとしている相手のTEEを信頼するには、
  以下について**暗号的に検証**する必要がある
  - そのTEEが、TEEを提供するHW（ハードウェア）ベンダが製造した、
    真正なマシンを使用している事
  - その真正なマシンの上で、**正しくTEEを動かしている**事
  - そのマシンやTEEには**脆弱性が存在しない**か、存在したとしても
    リモートユーザ側がそれを把握できている事
  - そのTEEの動作定義が、リモートユーザの**期待している通りのものである**事

Remote Attestation（RA）

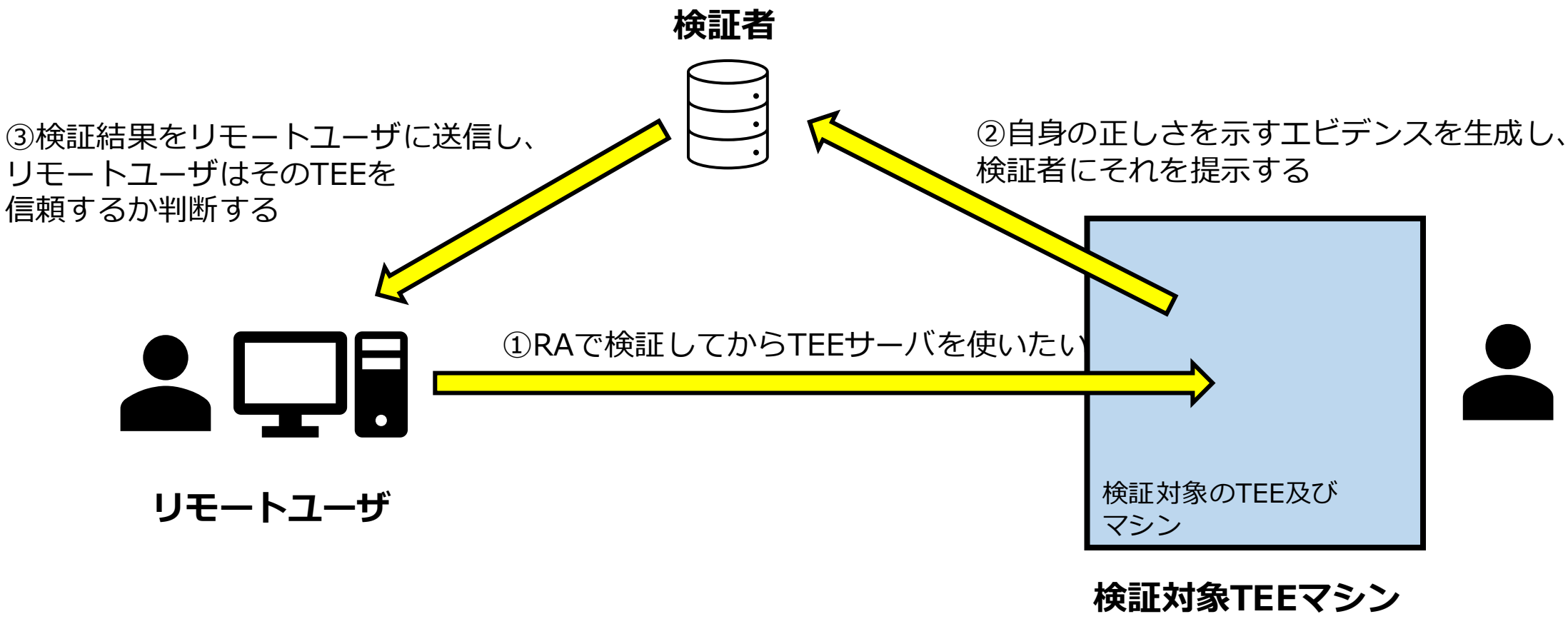
- これを実現するための暗号的なプロトコルを**Remote Attestation**（以下、**RA**）という

- TEEによっては単にAttestationと呼ぶ事もあるが、その場合は基本的にこのRAを指していると考えて良い

- Attestationは、日本語では「**構成証明**」と訳される事が一般的である

RAにおける登場人物

- RAでは、以下のように主に3つの登場人物が関わる事が多い



RAにおける登場人物

- RATSというRAの規格を定めるRFC 9334では、これらの登場人物に以下のように呼び名が与えられている：
  - リモートユーザ→**Relying Party（RP）**
  - 検証対象TEEマシン→**Attester**
  - 検証者→**Verifier**

- Relying Partyは「**依拠当事者**」という意味。TEEサーバに処理をお願いしようとしている当事者、くらいの意味合い

- Attesterは「**証明する者**」という意味。自身、つまりそのTEEやマシンの正しさをRAにより証明しようとしているため

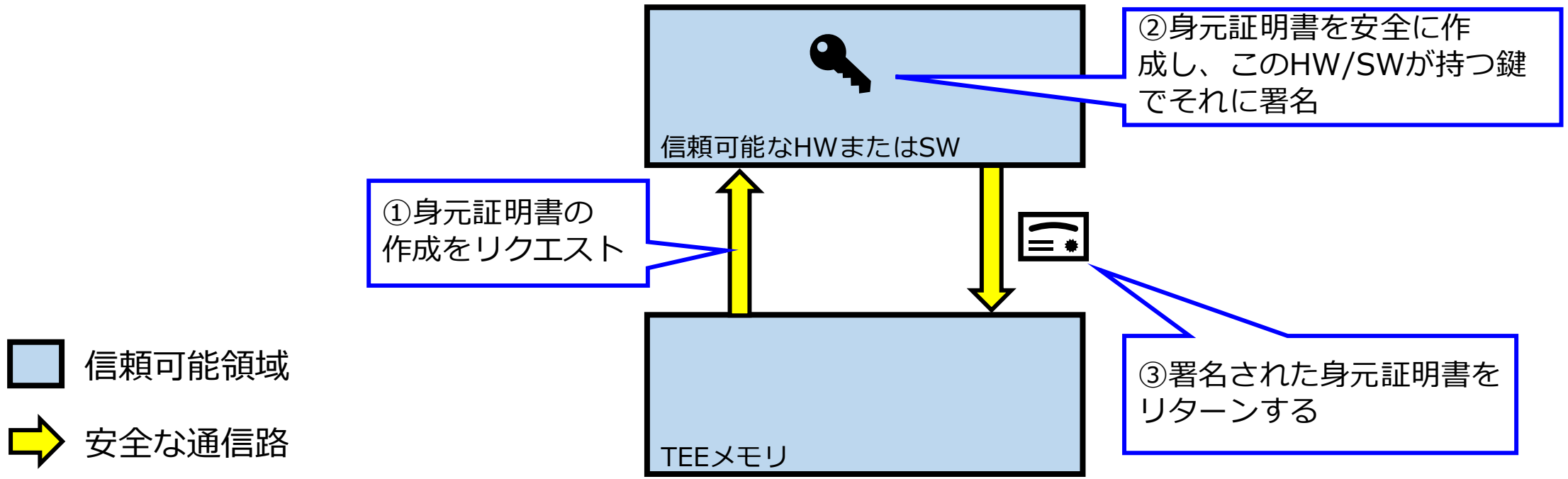
RAにおける登場人物

- TEEのRAの文脈では、RP及びVerifierの環境が侵害されている可能性に関しては考慮しない
  - リモートの（TEEですらない）ユーザの環境の安全性は、TEEが担保すべきスコープからは明らかに外れている

- また、RPがVerifierを兼ねる、つまりはリモートユーザが自前で相手のTEEを検証する事も多い

RAのおおまかな手順 - 身元証明書の生成

- そのTEEにおいて信頼する前提としているようなHWまたはSW、つまりは**狭義のTCB内**でそのTEEの身元証明書に署名する



RAのおおまかな手順 - 身元証明書の生成

- この署名付き身元証明書は、最も一般的な用語としては **Attestation Report（AR）**と呼ばれる
  - TEE次第では、**Quote**や**Evidence**等とも呼ばれ、統一されていない

- 信頼可能なHW/SWは、そのTEEにより何が担当するのかは異なる
  - SGX→CPUマイクロコード・Architectural Enclave（後述）、
    TDX→SGXに加えてTDX Module、SEV-SNP→AMD-SPコプロセッサ、
    OP-TEE RA（TrustZone）→OP-TEE Coreの専用インタフェース

RAのおおまかな手順 - 身元証明書の生成

- 信頼可能なHW/SWが持つ鍵は、そのHW/SWの製造者以外（場合によっては製造者も）知る事はできない
  - これはつまり**RoT**内の**RoT鍵**である事になる

- 実際の署名には、このRoT鍵にCPUやTEEのセキュリティバージョン番号（SVN）等を混ぜて導出した鍵を使う事が多い
  - 導出した鍵をそのまま使うケースも、あるいはまた別に生成したAR署名用の鍵の認証に使うケースも存在する
  - 実際にARへの署名に使われる鍵を、一般に**AK（Attestation Key）**と呼ぶ

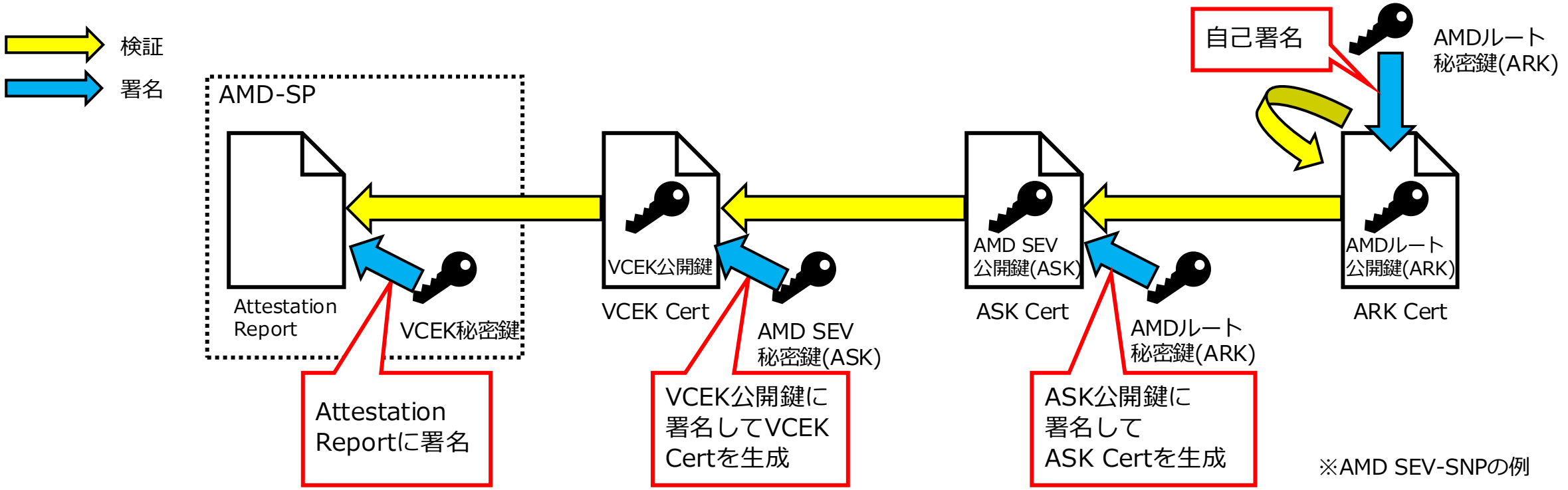
RAのおおまかな手順 - 身元証明書の生成

- 身元証明書には、例えば以下のような情報が格納されている：
  - そのマシンやTEEのセキュリティバージョン番号（SVN）
  - そのTEEの動作定義に対する測定値（ハッシュ値）
  - そのTEEの開発者に関する情報（開発者による署名等）
  - AR生成時に、ユーザが任意に入力しARに同梱できるデータ

- 上記4つ目のデータは**Report Data**と呼ばれ、RPとAttesterのTEEとの暗号通信路の安全な確立に利用する等が可能
  - 例：お互いにキーペアを生成してそれぞれの公開鍵を送り合うと共通鍵を生成できるディフィー・ヘルマン鍵交換（DHKE）において、互いのDH公開鍵の連結に対するハッシュを格納し、中間者攻撃を防ぐ

RAのおおまかな手順 - 身元証明書（AR）の検証

- ARの署名検証には、そのTEEベンダのルートCA鍵をトラストアンカーとした、RoT鍵に対する証明書チェーンを用いて行う事が多い
  - ベンダルートCA証明書は、Verifierが都度フェッチするか、あるいはVerifierのOSに予めインストールしておく等の取り扱いが考えられる



RAのおおまかな手順 - 身元証明書（AR）の検証

- そのTEEのTCBが有するRoT鍵によりARの署名が検証され、RoT鍵に対する証明書チェーンによる検証も全てクリアすると、そのARは**ベンダによるお墨付きを得られた状態**になる

- 後は、ARの中身を見て、RPが期待する通りのTEEであるかの検証を行えば良い

RAのおおまかな手順 - 身元証明書（AR）の検証

- 例えば、以下のような内容について確かめる：
  - そのTEEのセキュリティバージョンは十分か？
  - 特定の脆弱性・攻撃に対する防御機能が有効化されているか？
  - そのTEEの動作定義は意図している通りか？

- 例えば動作定義の一致確認等は、RPまたはVerifierが予め期待する測定値を控えておき、それと比較する事で行う
  - このように、見本となる（期待する）測定値の事を**リファレンス測定値**またはゴールデン測定値と呼ぶ
    - Reference Measurement, Golden Measurement

RAのおおまかな手順 - 身元証明書（AR）の検証

- この時点で、相手のTEEが安全かつ自分の期待するとおりである事を確認できた状態にある

- あとは、TEEとの暗号通信路を確立し、それを通して秘密情報を送りTEEに計算してもらう等してリモートのTEEを利用する
  - 暗号通信路の確立には、Report Data内のDH公開鍵ペアに対するハッシュ値と、RPの手元のDH公開鍵ペアとが整合するかを確認した上で、DHKEを最後まで実施し互いに共通鍵を有した状態になれば良い
  - 他にも、TLS証明書をReport Dataに紐づけるRA-TLS等の方式もあるが、ここでは説明を割愛する

RAが対策できる攻撃

- RAを行う事により、RPの期待に反する、**悪性の動作をするTEE**に秘密を送信してしまう事を未然に防げる

- また、そもそもTEEですらない環境がTEEを**なりすまし**をしている事も、RAによって検知できる

- 脆弱性を抱えたTEEについても、RAで可視化できる事が多いため、間接的には**各種脆弱性の予防**にもなる

RAが破綻するケース

- 脆弱性を突いた攻撃等の何らかの理由により、ARの署名に使われる**RoT鍵（AK）が漏洩**してしまうと、**そのAKが使われ続ける限りRAは破綻**する
  - 漏洩したAKでARに**偽造署名**できてしまうため

- AKは、その材料にSVNが使われる事がほとんどであるため、CPUマイクロコードをアップデートする等により**SVNを更新**する事で、**脆弱性対策と危殆化したAKの廃止**を同時に行える
  - この処理は、特にIntel製TEEの文脈では「**TCB Recovery**」と呼ばれる

RAが破綻するケース

- 古いAMD SEV/SEV-ESのRAでは、AKが全TCBバージョンに対して同一であったため、**全てのマシンにおけるRA自体が永久に破綻**するという致命的な結末を迎えている

- Intel製のTEEでは、AKを生成するために使われるCPU固有鍵が漏洩したケースは、恣意的に異常な状況以外では確認されていない
  - 2017年という大昔に発見・対策された脆弱性を、未対策のマシンで実行した場合にのみ漏洩が確認されている[1]

Local Attestation

ARに誰が署名する？

- ARはそのTEEを信頼するための根拠となるため、間違っても
  その内容が検出不可能な形で侵害されてはならない

- これはつまり、ARはそのTEEにおける**狭義のTCB内で署名される必要がある**事を意味する

- では、TEEにおいて最も安全であろう**CPU内のマイクロコード等で署名**してしまうので良いか？

ARに誰が署名する？

- それができれば理想的だが、Attestationに関連するような高度な暗号処理は、**CPU内に実装するには高度過ぎる**[2]

- また、万一脆弱性があった場合に最も侵害のリスクが甚大になるであろう**HW-TCBが肥大化してしまう**という観点もある

- では、TDXでは次に特権的であろう**TDX ModuleでARに署名**させれば良いか？
  - 丁度TDのメタデータはTDX ModuleがTDCSにて管理している

ARに誰が署名する？

- これも微妙で、TDX内でかなり強い特権であるSEAM VMX rootで動作するTDX Moduleを肥大化させるのも好ましいとは言えない

- AMD SEV-SNPでは、このような処理はプロセッサ内に同居する**コプロセッサであるAMD-SPに全て押し付けており**、この問題を上手く凌いでいる
  - HW-TCBは大きくなるが、プロセッサではなくコプロセッサに押し付けるので、折衷案として非常に美しい

- 結局、TDXでは**ユーザレベルの特権で動く別コンポネント**で**署名するしかない**
  - 例：VMX non-root、SEAM VMX non-root、ユーザ権限プロセス

TDXにおけるQuoteへの署名

- TDXでは、SGX時代の資産を流用する形で、Intel署名付きのSGX Enclaveである**Quoting Enclave（QE）**がARへの署名を実施する
  - Quoting Enclaveのように、主にIntelにより署名されておりIntelから頒布されている、SW-TCBの一部として振る舞うEnclaveを**Architectural Enclave（AE）**と呼ぶ
  - 特に、TDX用のQuoting Enclaveを**TD Quoting Enclave（TDQE）**と呼ぶ

- TDXやSGXでは、ARの事を**Quote**と呼ぶ
  - TDXのQuoteは特に**TD Quote**と呼ぶ

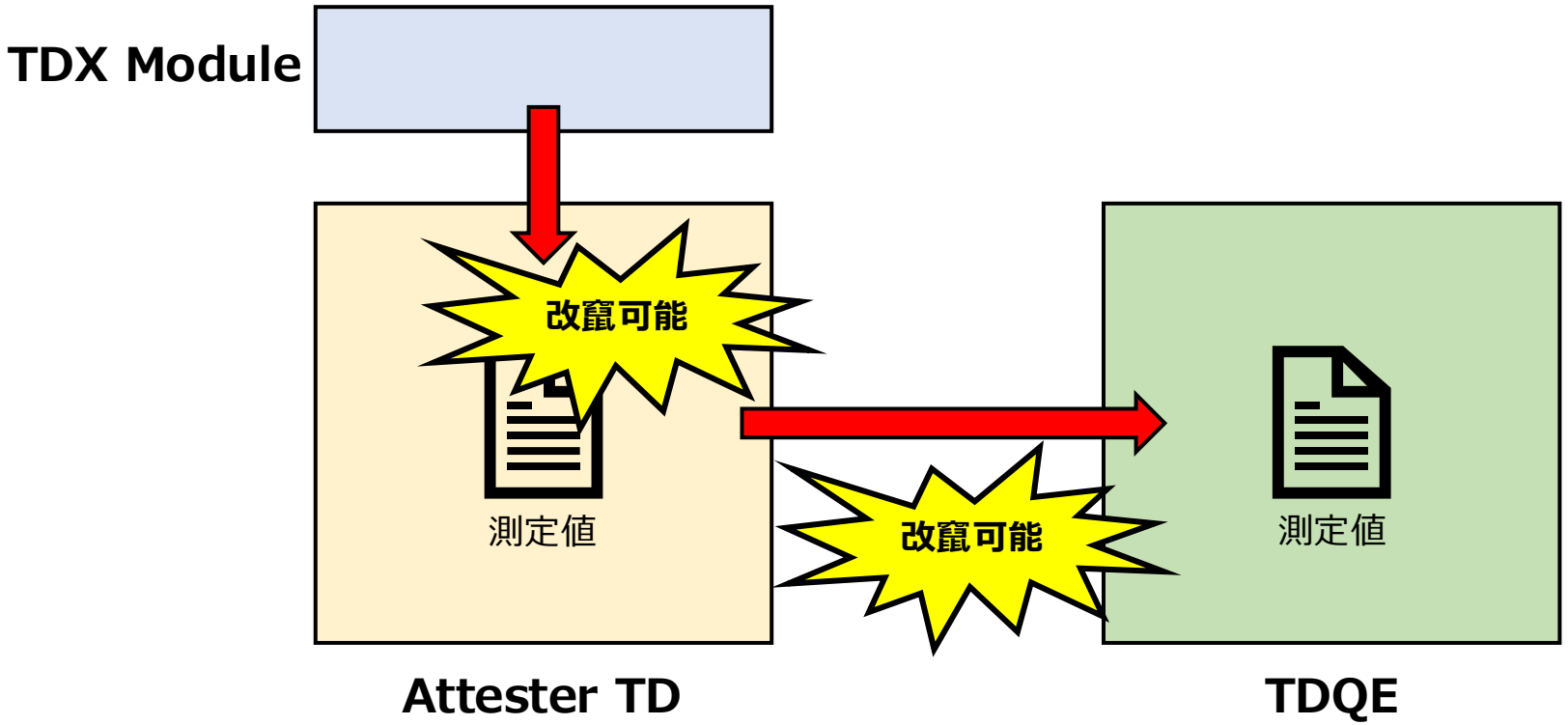
TDXにおけるQuoteへの署名

- しかし、QEは検証対象（Attester）TDとは無関係なユーザレベルプロセスであり、当然**TDX Moduleが管理**する**そのTDの測定値に直接アクセスする事はできない**

- 代わりに、その**Attester TD自体が自身の測定値やメタデータを取得**し、それを**QEに渡して署名させる**と良いかも知れない
  - **測定値（Measurement）**：そのTEEの動作定義（コード）等、そのTEEの性質を表す要素に対するハッシュ値
  - 専用のCPU命令を発行し、そのTD自身の測定値が確実に取得されるようにする

TDXにおけるQuoteへの署名

- ただ、これは以下の2つの新たな問題を引き起こす：
  - 検証前に**Attester TDがその測定値を改竄**できてしまうと本末転倒である
  - QEへの配送には**Untrusted領域に出る必要があり、そこで改竄されてしまう**可能性がある



Local Attestationの概要

- この問題を解決し、Attester TDが真にそのマシンにて正しい測定値を取得しQEに渡した事を保証するための機能が**Local Attestation（LA）**である

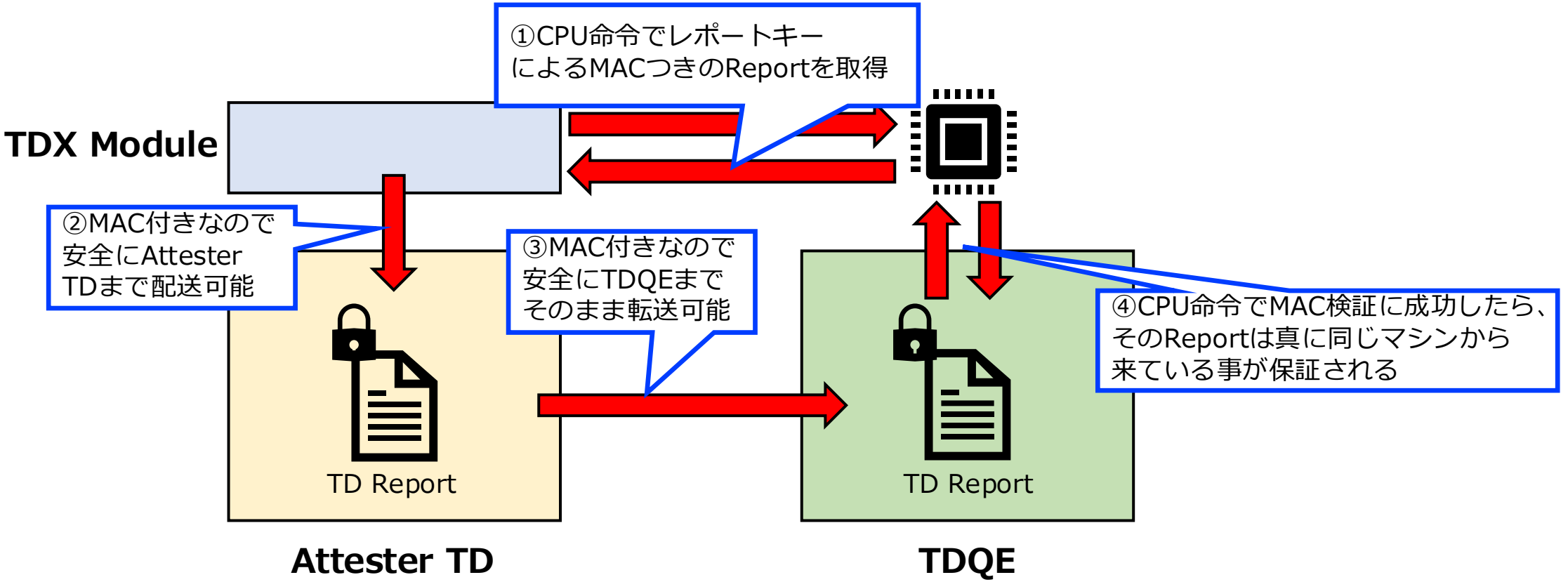
- 別の言い方をすると、LAは「**相手のTEEが真に自身と同じマシン上に存在しているか**」を**暗号学的に確かめる手続き**である

Local Attestationの概要

- 実は、Attester TDがCPU命令経由で取得できる測定値（の集合）には、そのマシンのみで生成可能なHW鍵である**レポートキー**（Report Key）でMACが打たれている
  - MAC：メッセージ認証符号
- このCPU内でレポートキーによりMACを打たれた測定値の集合を**Report**（特にTDXのものは**TD Report**）と言う
- そのレポートキーは、当然にして**そのマシン以外で再び導出する事は絶対にできない**

Local Attestationの概要

- つまりは、QEがレポートキーでの検証を行うCPU命令を呼び出してこのReportのMACを検証すれば、Attester TDが**真にそのマシンの正しい測定値集合を渡してきたかを確かめる事ができる**
  - もし**違うマシンのReport**なら、**命令内でのレポートキーでの検証に失敗**する



Local Attestationの概要

- QEは、そのマシンのRoT鍵に紐づく**AKでARに署名する**事を**唯一にして最大の任務**としている

- もしQEが他のTEEのReportに署名してしまっては、極論**危殆化したTEEに自身でお墨付きを与えてしまう**事になる
  - こうすると危殆化したTEEのAKが例え失効していても、RPは一生それを検知できない。何故ならばそもそも別のAKで署名されているため

- こうなるとRAの信頼性が根底から覆ってしまうため、このLAは**セキュリティ上非常に大きな意義を持つ**

Local Attestationの概要

- ちなみに、この説明はかなりRAを前提とした説明をしているため、より一般的には「**同一マシン上のコンポネント**に**そのAttesterの真正性を証明**する事」と説明できる
  - MACを検証後、Report内の各種メタデータや測定値を確かめ、相手が期待通りの動きや要件のAttesterであるかをチェックできる

- が、SGX時代からEnclave間のLAの応用方法は比較的限定的で、TDXに至ってはRAでしか使われないため、このような説明をした

- ちなみに、LAの概念は**TPM時代から存在**しているものである
  - TPMにおけるLAも上記の一般的なLAの内容のそれに等しい

Local Attestationの概要

- さらに余談として、GPU TEEのNVIDIA Confidential Computing（NCC）でもLAやRAが登場するが、これは**VerifierがローカルCVMとリモートサーバのどちらで動くかで区別**している
  - つまりはTDXやSGXとは**全然違う言葉の使い方**である

- よって、NCCの方は一部でNVIDIA公式も使っているように、それぞれLocal GPU AttestationやRemote GPU Attestationと呼ぶのが良い

楕円曲線ディフィー・ヘルマン鍵共有

楕円曲線ディフィー・ヘルマン鍵共有（EC-DHKE）

- 唐突に思うかも知れないが、Attestation時には**同時並行で安全な暗号通信路を張る**事が多い
  - QEとAttester TDとのLA時には張らないが
- Attested-TLS等、高度な方式も多々あるが、その中でも最も原始的なものが**楕円曲線ディフィー・ヘルマン鍵共有**である
  - EC-DHKEと略す
- 知っておいて損はないので、この機会に説明を行う

楕円曲線ディフィー・ヘルマン鍵共有（EC-DHKE）

- **楕円曲線暗号**という**公開鍵暗号**を用いる事で、**二者間**で**安全に共通鍵（共有秘密）を生成**するためのプロトコル

- **相手から受け取った公開鍵**と**自身の秘密鍵**を**かけ合わせる**と、**二者ともに全く同じ値が導出される**（＝**共通鍵になる**）という特性を利用するものである

楕円曲線ディフィー・ヘルマン鍵共有（EC-DHKE）

- 次の通りパラメータを定義:

G: 使用する楕円曲線

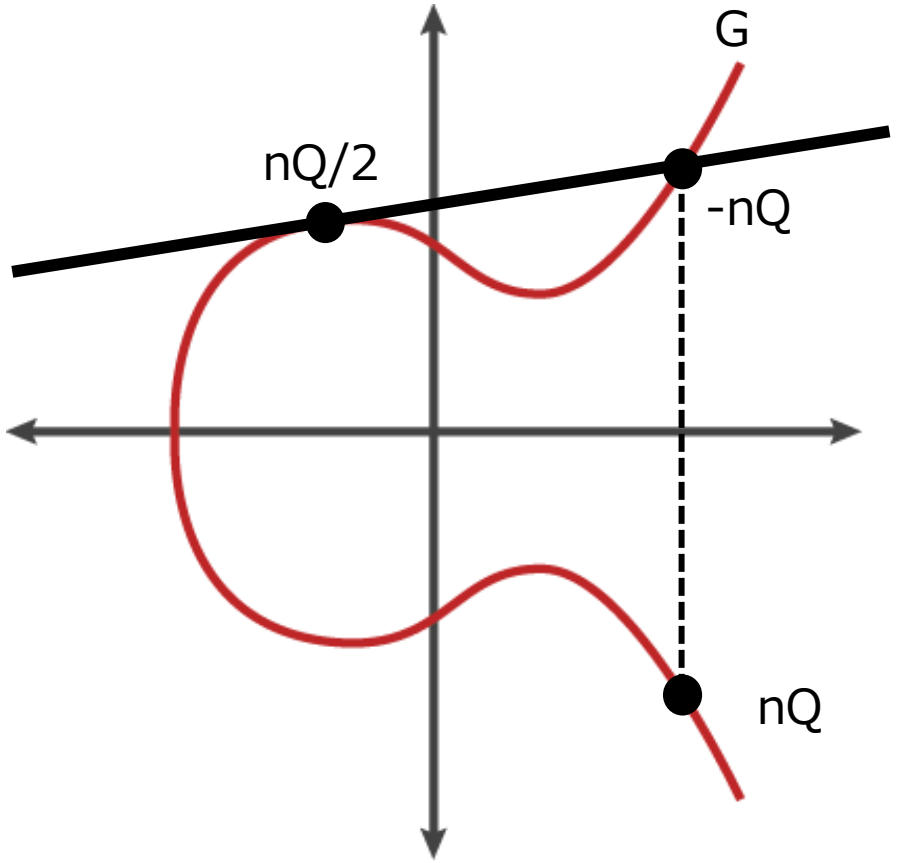
Q: ベースポイント (楕円曲線上の加算を適用する基準点)

n: Qに楕円曲線上の加算を適用する回数

P: nQ

楕円曲線パラメータ

- この例では…



- 楕円曲線: G
  ベースポイント: Q
  加算回数: n
  P: nQ

- $\boldsymbol{P}$ を n, Q, G から導出するのは**容易**
- $\boldsymbol{n}$ を P, Q, G から導出するのは**非常に困難**

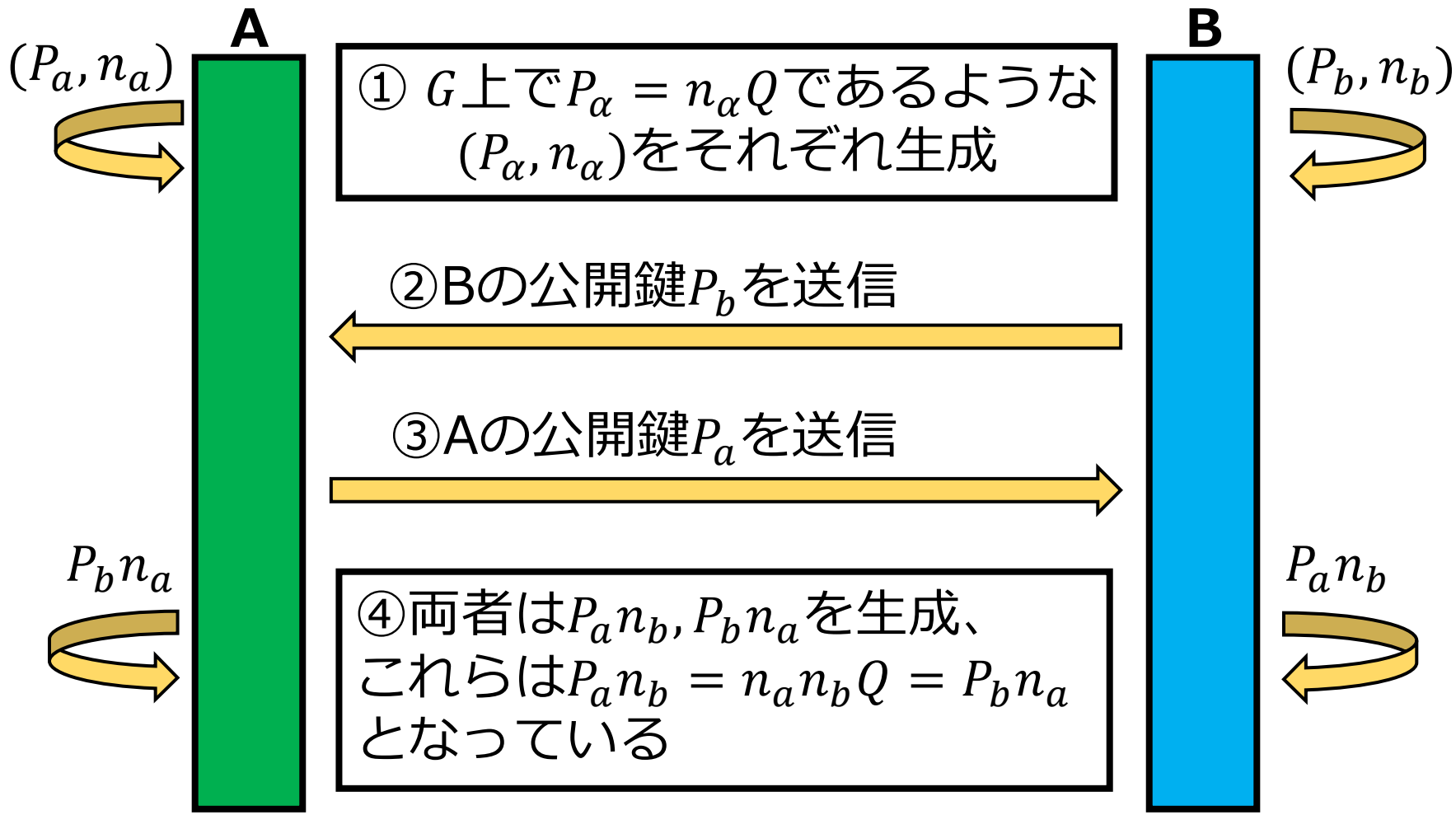
楕円曲線ディフィー・ヘルマン鍵共有（EC-DHKE）

- EC-DHKEは**EC-DLP**を安全性の根拠としている

- 前提条件: G, Q (楕円曲線, ベースポイント)
- 公開鍵: P $(=nQ)$
- 秘密鍵: n (加算回数)

楕円曲線ディフィー・ヘルマン鍵共有（EC-DHKE）

- 前提条件: 楕円曲線G, ベースポイントQ



LAの詳細

LAの一般的な説明

- あるEnclaveが、**同一のマシン**（=**ローカル**）**で動作している他のEnclaveまたはTD**について、本当に**同一マシン上で動作しているか**を**検証**するプロトコル
  - LA自体はSGX時代から存在し、それを拡張してTDXで用いている
  - **測定値の検証**については、**LAの必須要件ではない**
- TDXにおいては、原則的に**EnclaveがTDを検証する**という**一方向のみ**となっている
  - 検証にはSGX命令が必要だが、以前のスライドの通りTD内でSGX命令を実行する事はできない
- Quoting TDというもの等を用いる場合には相互LAが可能だが、この辺りについては次回以降のスライドで解説予定
  - Quoting TDは2026/6**現在計画上の存在でしかない**点には注意

LAの一般的な説明

- 相手のTDが**意図している動作定義を持つか**の確認を行う場合は、**Report構造体**内の**MRTD**を**RTMR**を検証する事によって行う
  - MRTDやRTMRについては次回以降のスライドで説明

- 相手のTDが自身と**同一のマシンで動作している事の確認**は、**同じマシン**であれば**必ず同一**となる**レポートキー**を根拠とする

- 上記についての詳細な解説はいずれも後述

TDXにおけるLAのフロー

- TDXにおいては原則としてEnclaveがTDを検証する事しかできないため、フローとしてはかなりシンプルになる：

TD Reportの生成

- TD Reportを生成するには、**TDCALL[TDG.MR.REPORT]**を
そのTD内にて呼び出す事で取得できる
  - さらに、この内部では**SEAMOPS[SEAMREPORT]**というコア中のコアの
命令が呼び出される

- TD Reportは、CPU内で**256bitのReport Key**によってMACが
付与されるため、Attester TDや第三者が**改竄する事は不可能**

TD Reportの生成

- また、ReportにはReport要求時に引数として渡す事のできる、ユーザ指定の任意の64Bのデータを同梱させる事ができる

- この64Bのデータの事を**Report Data**と呼ぶ

- Report Dataは、例えばnonceを同梱する、あるいはRAにおける暗号通信路に紐づく情報を入れて真にAttestationしたTEEとやり取りしている事を保証する、といった様々な使い方ができる
  - Report DataはQuoteにもそのまま安全な方法で同梱される

TD Reportの検証

- 検証側Enclave（QE等）は、Attester TDのTD Reportを受け取ったら、**ENCLU[EVERIFYREPORT2]**命令を呼び出す事でReport KeyによるMAC検証を実施できる

- ちなみに、SGXでは**Report発行のCPU命令**（EREPORT）**の度**に**Report Keyが変わった**が、TDXではどうなっているか**公式文書の範囲では不明**

- SGXと異なり全てがHW-TCBであるCPU内でReport Keyの取り扱いが完結するため、Report Keyの呼び出し毎の更新はTDXでは**冗長であるとして廃止されている可能性もある**
  - SGXでは検証時Report Keyを直接EnclaveにCPUから読み込んでいたため、Enclaveへのサイドチャネル攻撃等で漏れる可能性があった

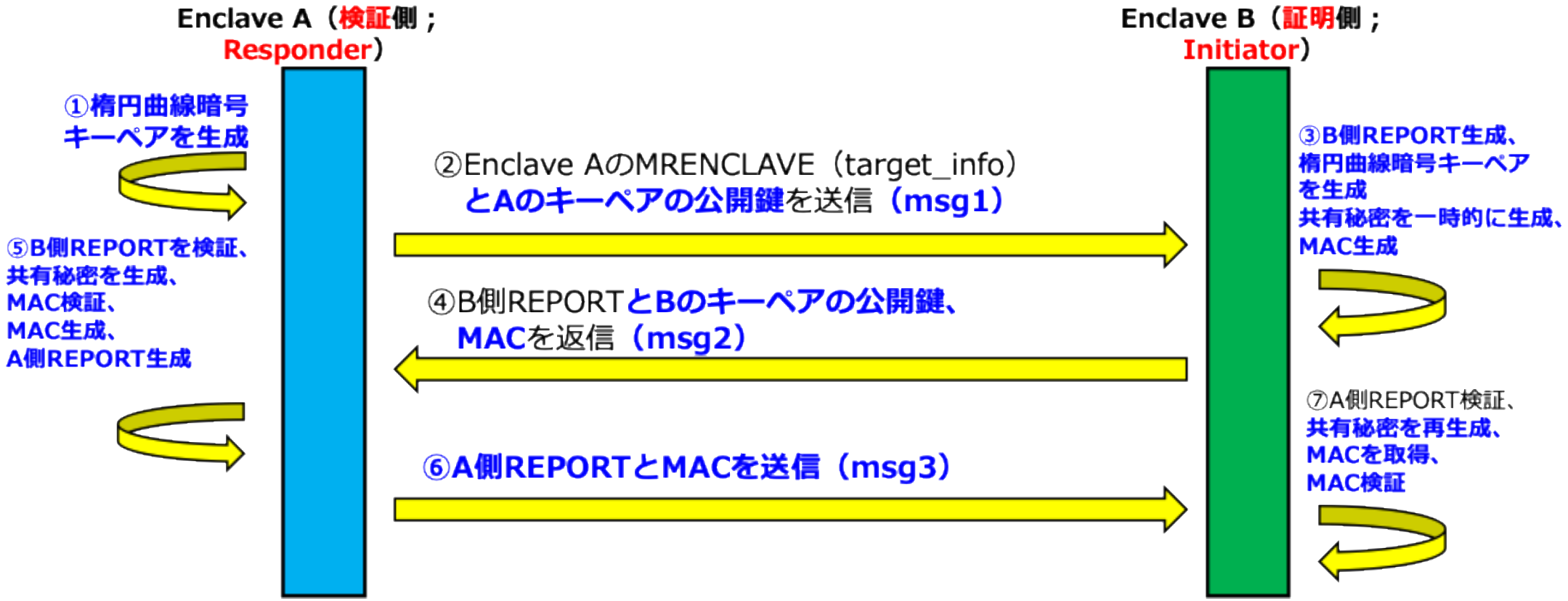
TD Reportの検証

- MAC検証によりそのReportが改竄されていなく、かつ真に同一マシン上のTDのものであると判明したら、追加でその中身を見て期待する測定値やメタデータとのチェックを行う事もできる

- が、前述の通りTDからEnclaveは検証できないため、一方的にEnclaveがTDの測定値を見たいというかなり限定的なシナリオでの議論となる
  - ただし、TDCALL[TDG.MR.VERIFYREPORT]を用いる事でTD内でTD ReportのLA的な検証を行う機能の提供が想定されているようで[3]、これが実現した場合はTD同士での連携への活用も期待される

(参考)SGXにおけるLA

- SGXでは互いに互いのEnclaveを検証できるので、TDXにおけるよりも複雑な事がLAで可能であった
  - TDXでも前述のQuoting TDが用いるような命令を用いれば同様の事が可能



コラム：Target Infoについて

- SGXのLAを知っている人が読むと、TDXのLAにはTarget Infoが存在しない事に気づくかも知れない
  - Target Info：LAにおける検証側の動作定義（コード）に対するハッシュ値がメインとなる構造。SGXであればMRENCLAVEという測定値がメインとなる

- 結論から言えばTDXにTarget Infoは存在しない

- これは、SGXにおいてはReport KeyをEGETKEY命令で直接取得できた一方で、TDXではそれが不可能となっている事に起因する

コラム：Target Infoについて

- 例えば、SGXにおいてTarget Infoが存在しない場合、以下のような攻撃が可能となってしまう：
  1. 攻撃者は、EREPORTを発行し、ある正しいReport Aを作る。
  2. 攻撃者はEGETKEYでReport Keyを取得し、デタラメな測定値を寄せ集めてそれに署名し偽造Report Bを作る。
  3. EREPORTを再発行しない限りReport Keyは不変なため、Aと同じReport Keyが使われる。
  4. 偽造Report Bを相手に渡す。
  5. 相手はEGETKEYを発行し、Report KeyでBを検証し信じ込む。

コラム：Target Infoについて

- 実際のEGETKEY命令（SGX関連HW鍵の取得命令）は、発行者EnclaveのMRENCLAVEを導出材料とする

- この事実に基づき、EREPORTによるREPORT発行時にTarget Infoとして検証側のMRENCLAVEを指定しておく

- そうすれば検証側Enclave以外のEnclaveがEGETKEYを発行しても、同じReport Keyを導出できず検証時にMAC不一致で検知できる
  - 逆に、Target Infoという仕組みが無ければいくらでも同じReport Keyを再導出し偽造署名できてしまう事になる

コラム：Target Infoについて

- 一方で、TDXにおいては検証の際にEGETKEYを用いてReport Keyを導出するのではなく、EVERIFYREPORT2というENCLU命令を用いて直接Reportを検証するようになった

- これは鍵がCPU内を出る事が無いため、EGETKEYで偽造署名という事がそもそも不可能であり、結果としてTDXではTarget Infoは必要ない、というからくりとなっている

まとめ

- Attestationとその必要性についてできるだけ平易に説明した
- TDXにおけるRAを実現する上で不可欠なLAについて詳細に説明した

参考文献

[1]”Insecure Until Proven Updated: Analyzing AMD SEV‘s Remote Attestation”, Robert Buhren et al., https://dl.acm.org/doi/10.1145/3319535.3354216

[2]”Intel SGX Explained”, Victor Costan and Srinivas Devadas, https://eprint.iacr.org/2016/086.pdf

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