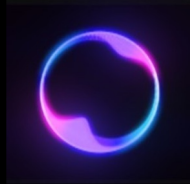
6. Attestation

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



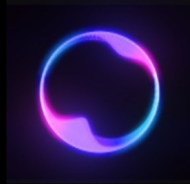
- SGXの登竜門でもある、2通りのアテステーション（Attestation）について解説する

- Attestationに伴い実施される事の多い、安全な通信路の確立のための楕円曲線ディフィー・ヘルマン鍵交換（EC-DHKE）について簡単に解説する

- SGX-VaultにRemote Attestationを組み込む事により、パスワード管理機能を遠隔で使用できるように改造する

Attestation（アテステーション）

Attestation（1/2）

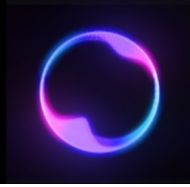


- 特にSGXにおいては、ある**Enclave**が**検証者**（他のEnclaveやリモートユーザ等）**の意図している通り**のものであり、かつ**信頼可能なマシン上で動作**している事を**検証**するプロトコルの事

- 日本語訳する場合、「**構成証明**」という表現が使われる事が多い

- Attestation自体は**SGXやTEE特有の概念ではなく**、比較的**以前より使用されている**概念である（例えば参考文献[4]は2006年）

Attestation（2/2）

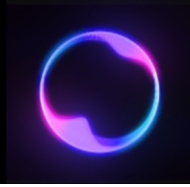


- Attestationにおいては、証明の確立後に安全にデータをやり取りをするための**鍵交換処理**を**同時に行う**場合が多い

- SGXにおいては、Enclaveやマシンの**正当性・完全性を証明する相手によって**、以下2通りのAttestationが使い分けられる
  - **ローカル・アテステーション**（Local Attestation; 以降**LA**）
  - **リモート・アテステーション**（Remote Attestation; 以降**RA**）

楕円曲線ディフィー・ヘルマン鍵共有

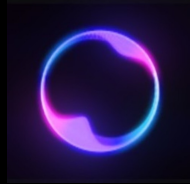
楕円曲線ディフィー・ヘルマン鍵共有（ECDHKE）



- **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

楕円曲線パラメータ



- 次の通りパラメータを定義:

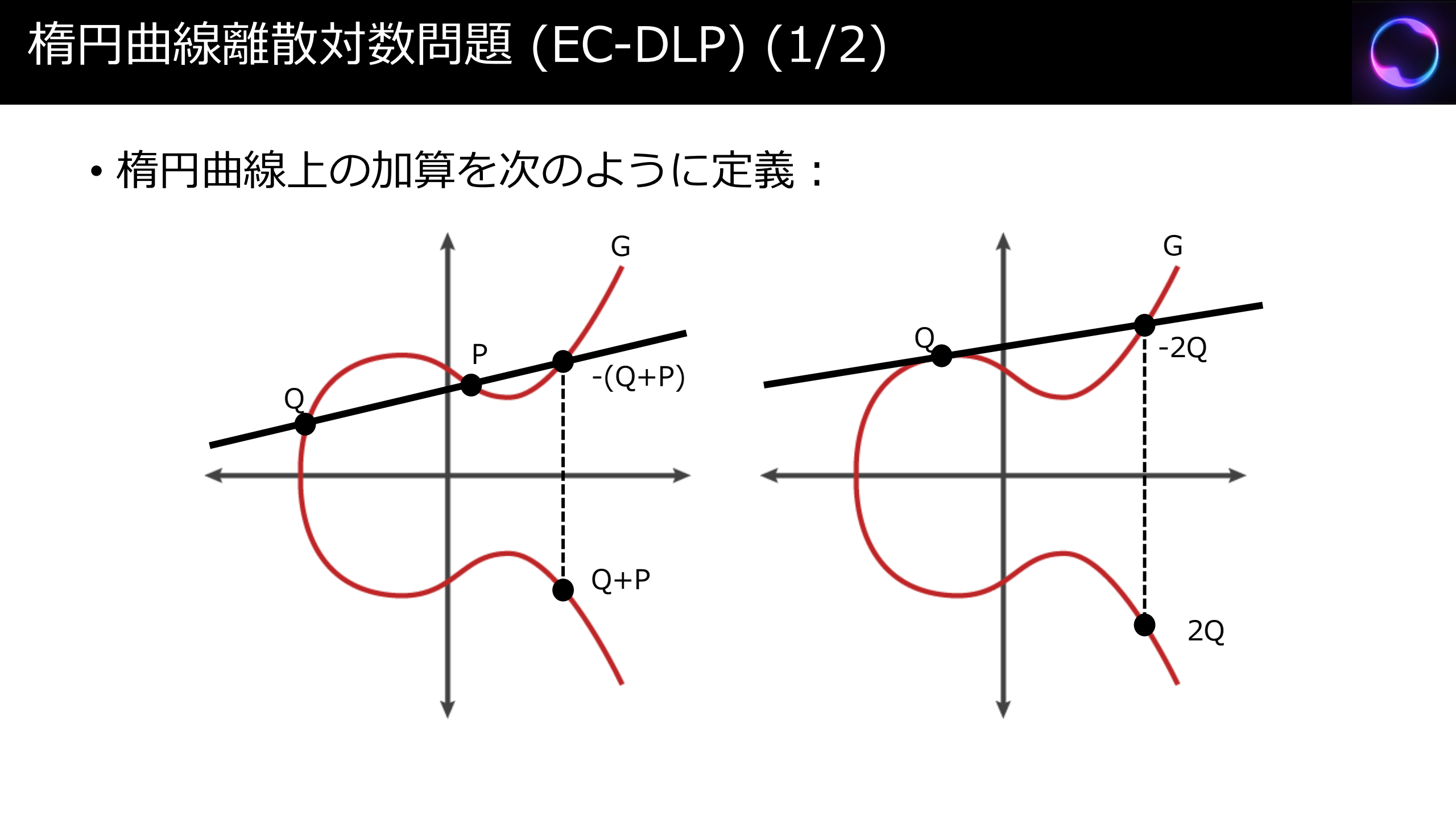
G: 使用する楕円曲線

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

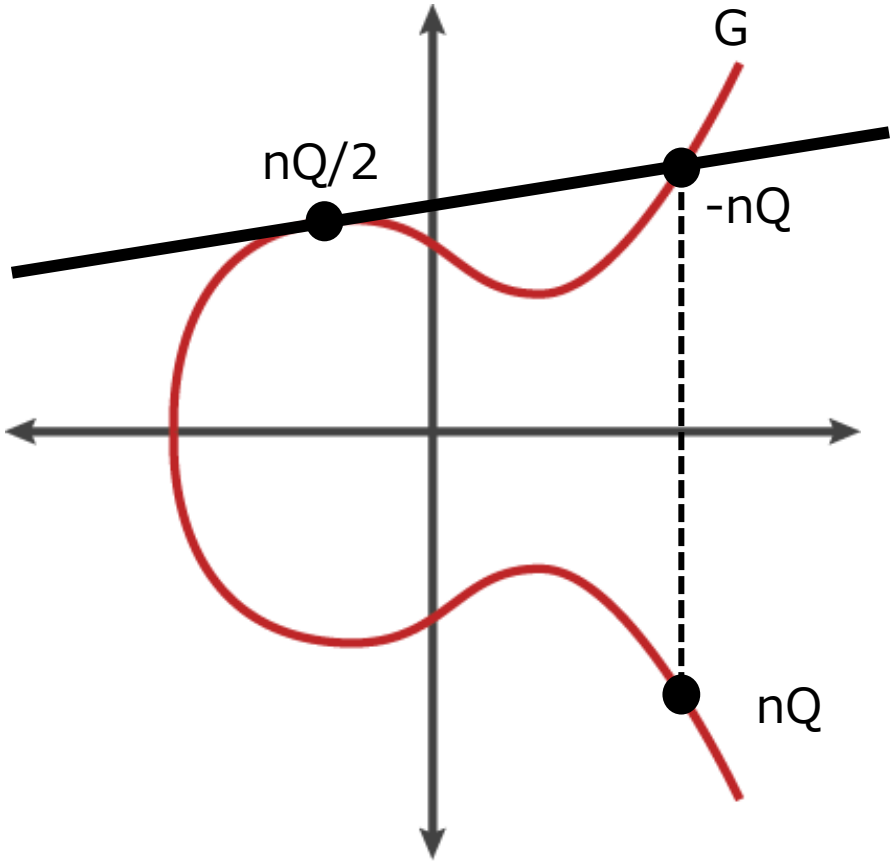
楕円曲線離散対数問題 (EC-DLP) (1/2)



- 楕円曲線上の加算を次のように定義：

楕円曲線離散対数問題 (EC-DLP) (2/2)

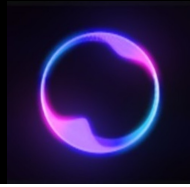
- この例では…



- 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**

EC-DHKE (1/2)



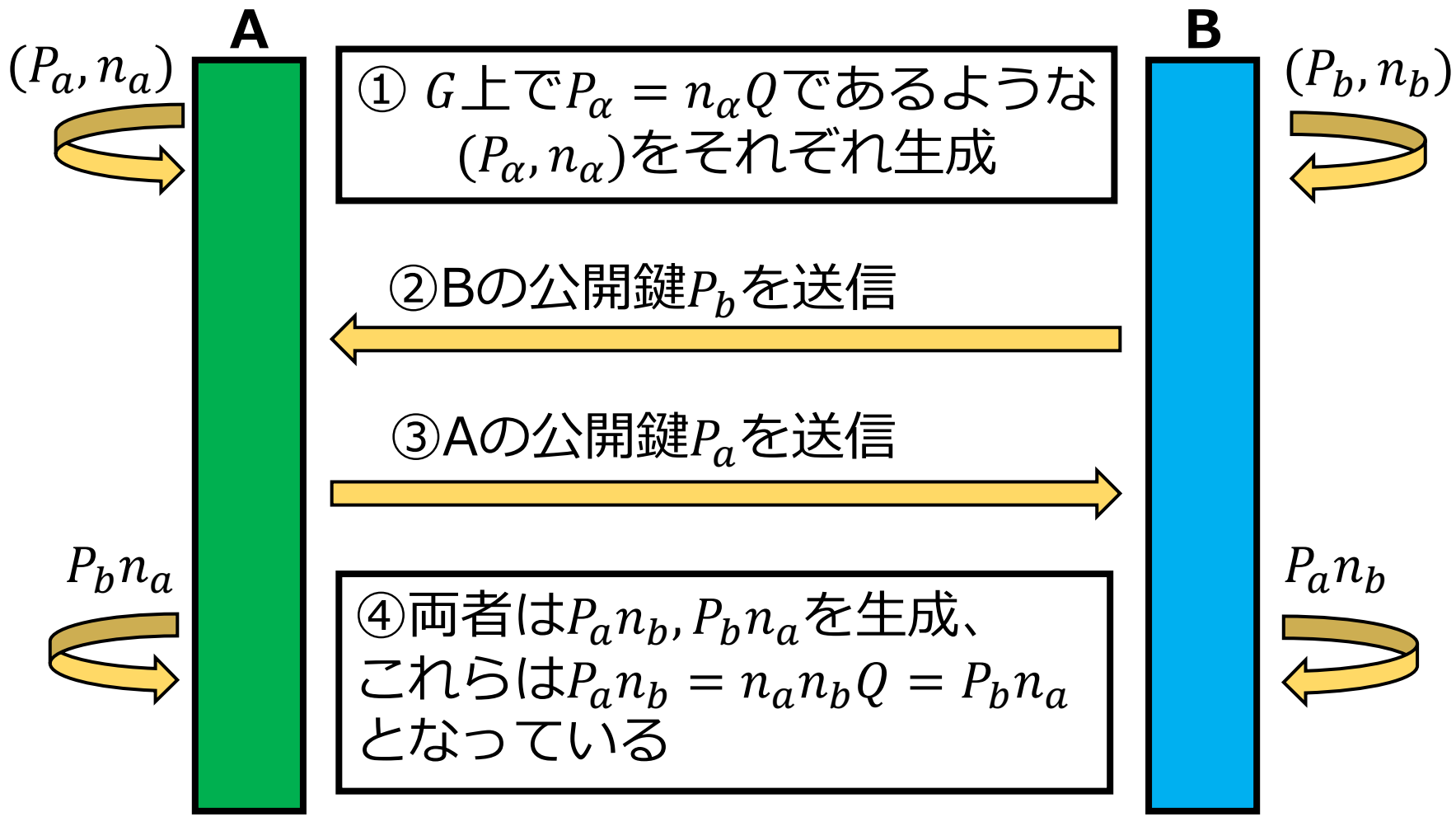
- EC-DHKEは**EC-DLP**を安全性の根拠としている

- 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)

- SGXSDKの構造体では、**公開鍵**を$\boldsymbol{G_\alpha}$という形で表現している事がほとんどであるため注意（上記定義の楕円曲線と紛らわしい）

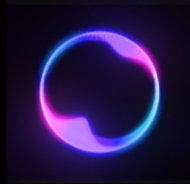
EC-DHKE (2/2)

- 前提条件: 楕円曲線G, ベースポイントQ



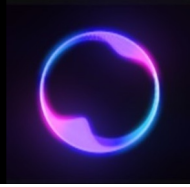
Local Attestation

Local Attestationの概要（1/2）



- あるEnclaveが、**同一のマシン**（=**ローカル**）**で動作している他のEnclave**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
  - 自身の動作しているマシン（**自身が動いているくらいだから安全**）で相手のEnclaveが動作しているなら、少なくとも相手Enclaveの**マシンの安全性は保証される**であろうというロジック
  - **同一性の検証**（MRENCLAVE等のチェック）については、**LAの必須要件ではない**。実施する場合、後述の性質上MRENCLAVEの検証については**原則として一方向的**になる
- **同一マシン上に存在するかの確認**は、原則的に**単一のLA実行**で**相互に行う事が出来る**

Local Attestationの概要（2/2）

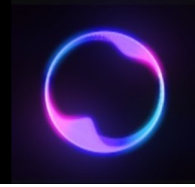


- 相手のEnclaveが**意図している同一性を持つか**の確認を行う場合は、**REPORT構造体**内の**MRENCLAVE**を検証する事によって行う

- 相手のEnclave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

EREPORT命令とREPORT構造体（1/3）

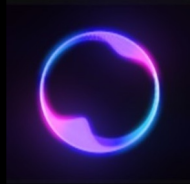


- **EREPORT**：**自身の同一性を証明**する**REPORT構造体**を生成する**ENCLU命令**（のリーフ関数）

- REPORT構造体には**MRENCLAVE**等の**同一性に関する情報**が含まれるが、これは**SECS**から**直接取得**されるため、不正なMRENCLAVE等で**改竄したりは出来ない**
  - SECSは通常のEPC上のコードからすらアクセスできない、**特別に隔離されている領域**となっている

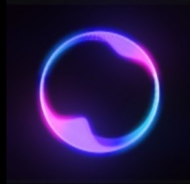
- さらに、Enclave内で**レポートキー**によってその**REPORTのMAC**を計算し添付するため、**REPORT構造体を改竄するのは不可能**

EREPORT命令とREPORT構造体（2/3）



- レポートキーは、**RSK**と共に、**CPUSVN**や**KeyID**、そして**報告先のEnclaveの同一性情報**などと共に、**SGXマスター導出鍵**を用いた**CMAC**という形で導出される
  - **CPUSVN**はCPUのセキュリティ上のバージョンで、**TCB Recovery**に伴い**変化**する
  - **KeyID**は**EREPORT発行ごと**にランダムに決定される**乱数**である
  - **報告先のEnclaveの同一性情報**のメインは**MRENCLAVE**である。この報告先Enclaveの同一性情報を**Target Info**という
  - **SGXマスター導出鍵**については、文献[2]によると**正体不明**。機密かつ複雑な生成ロジックになっているらしい

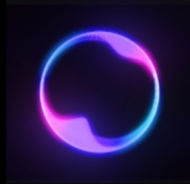
EREPORT命令とREPORT構造体（3/3）



- レポートキーは、**同一マシン上**であれば**再度EREPORTを発行しない限り**は**必ず同一のものが導出される**
  - CPUSVNは**TCB Recoveryしない限りは同一**
  - KeyIDは**再度EREPORT命令を呼ばなければ同一**
  - Target Infoは**検証側Enclave自身のMRENCLAVE**なので自明
  - **SGXマスター導出鍵も変わらない**

- この性質により、**報告対象のEnclave**は、自身のMRENCLAVE等（=**Target Info**）を携えて**EGETKEY命令**を発行しレポートキーを導出する事で、受け取った**REPORT**が**改竄されていないか・同一マシン上のEnclaveから来ているか**を**検証できる**

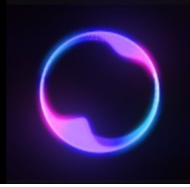
REPORT構造体の実装上の詳細（1/6）



- REPORT構造体に直接関連するSGXSDK上の型として、**sgx_report_t**, **sgx_report_body_t**, **sgx_report_data_t**が存在する

- **sgx_report_t**は、REPORT本体である**sgx_report_body_t**と、**KeyID**と**REPORT**に対するレポートキーを用いたMACで構成されている

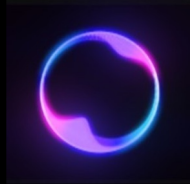
REPORT構造体の実装上の詳細（2/6）



- **sgx_report_body_t**は言わば**REPORTの本体**で、言わばそのEnclaveの**同一性情報の報告書**である。**sgx_report_data_t**を含む

- **sgx_report_data_t**は、ユーザが**任意に何らかのデータ**を**同梱する**ための構造体である。
  ここに同梱したデータは、**レポートキーによるMAC**による**極めて強力な改竄防止機能の恩恵に預かれる**

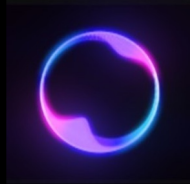
REPORT構造体の実装上の詳細（3/6）



- sgx_report_tの構造

メンバ	説明
sgx_report_body_t body	REPORTの本体。詳細は次ページで解説。
sgx_key_id_t key_id	EREPORT発行時にレポートキーの導出に 使用された乱数。
sgx_mac_t mac	レポートキーにより生成された、REPORT本体に 対するMAC値。

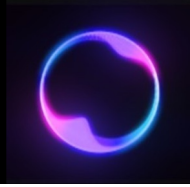
REPORT構造体の実装上の詳細（4/6）



- sgx_report_body_tの構造（1/2）

メンバ	説明
sgx_cpu_svn_t cpu_svn	CPUのセキュリティバージョン番号。TCB Recoveryで更新
sgx_misc_select_t misc_select	将来実装されるかも知れない機能のための予約領域
uint8_t reverved1[12]	将来のための予約領域。現時点ではオールゼロとする
sgx_isvext_prod_id_t isv_ext_prod_id	ISVに割り当てられた拡張Prod ID。普通は意識しない
sgx_attribute_t attributes	Enclaveの権限や属性に関する設定をする値
sgx_measurement_t mr_enclave	REPORTを発行したEnclaveのSECSから取得したMRENCLAVE
uint8_t reserved2[32]	将来のための予約領域。現時点ではオールゼロとする
sgx_measurement_t mr_signer	REPORTを発行したEnclaveのSECSから取得したMRSIGNER
uint8_t reserved3[32]	将来のための予約領域。現時点ではオールゼロとする
sgx_config_id_t config_id	Enclave設定のID。普通は意識しない

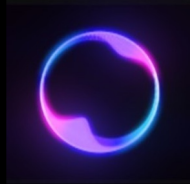
REPORT構造体の実装上の詳細（5/6）



- sgx_report_body_tの構造（2/2）

メンバ	説明
sgx_prod_id_t isv_prod_id	Enclave設定XMLで設定するISVのProd ID値
sgx_isv_svn_t isv_svn	Enclave設定XMLで設定するISVのセキュリティ上の番号
sgx_config_svn_t config_svn	Enclave設定のSVN。普通は意識しない
uint8_t reserved4[42]	将来のための予約領域。現時点ではオールゼロとする
sgx_isvfamily_id_t isv_family_id	ISVファミリのID。普通は意識しない
sgx_report_data_t report_data	ユーザがREPORTに任意のデータを組み込むための領域

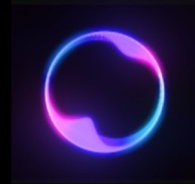
REPORT構造体の実装上の詳細（6/6）



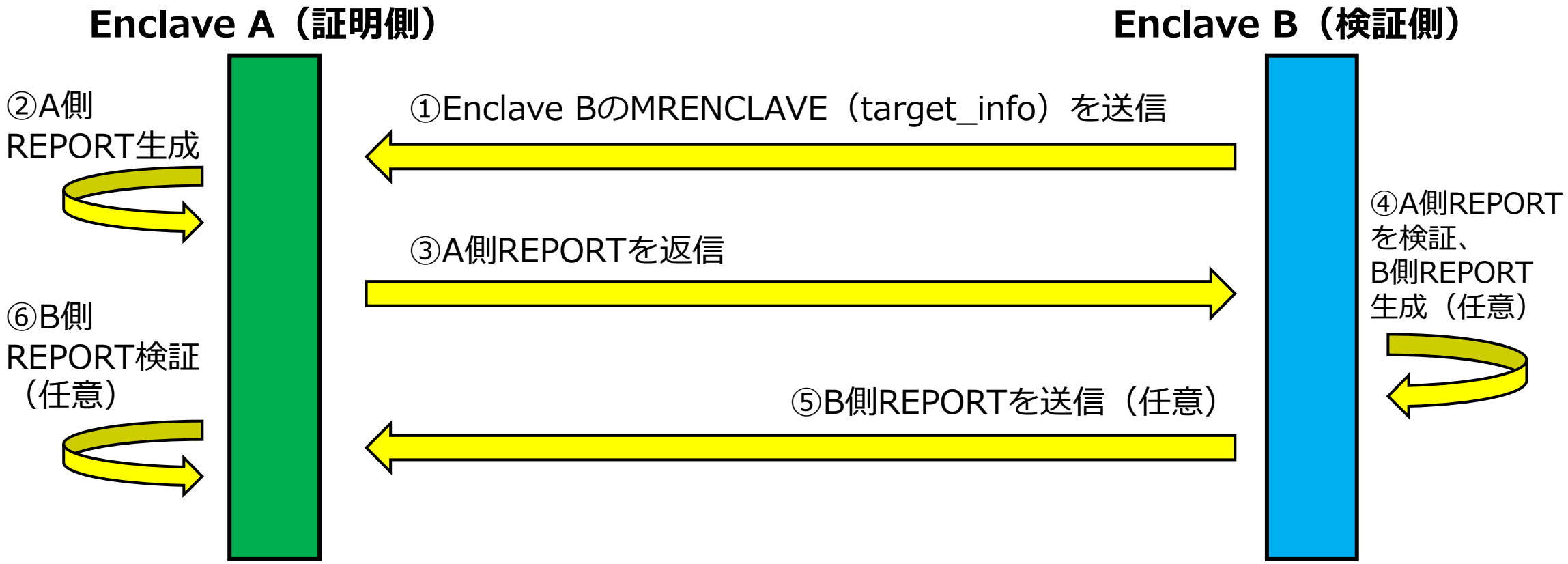
- sgx_report_data_tの構造

メンバ	説明
uint8_t d[64]	ユーザが任意のデータを格納できる領域。ここにデータを格納する事で、レポートキーによる改竄防止を行う事が出来る

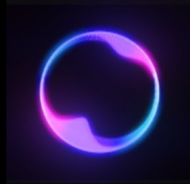
LAの基本形のフロー



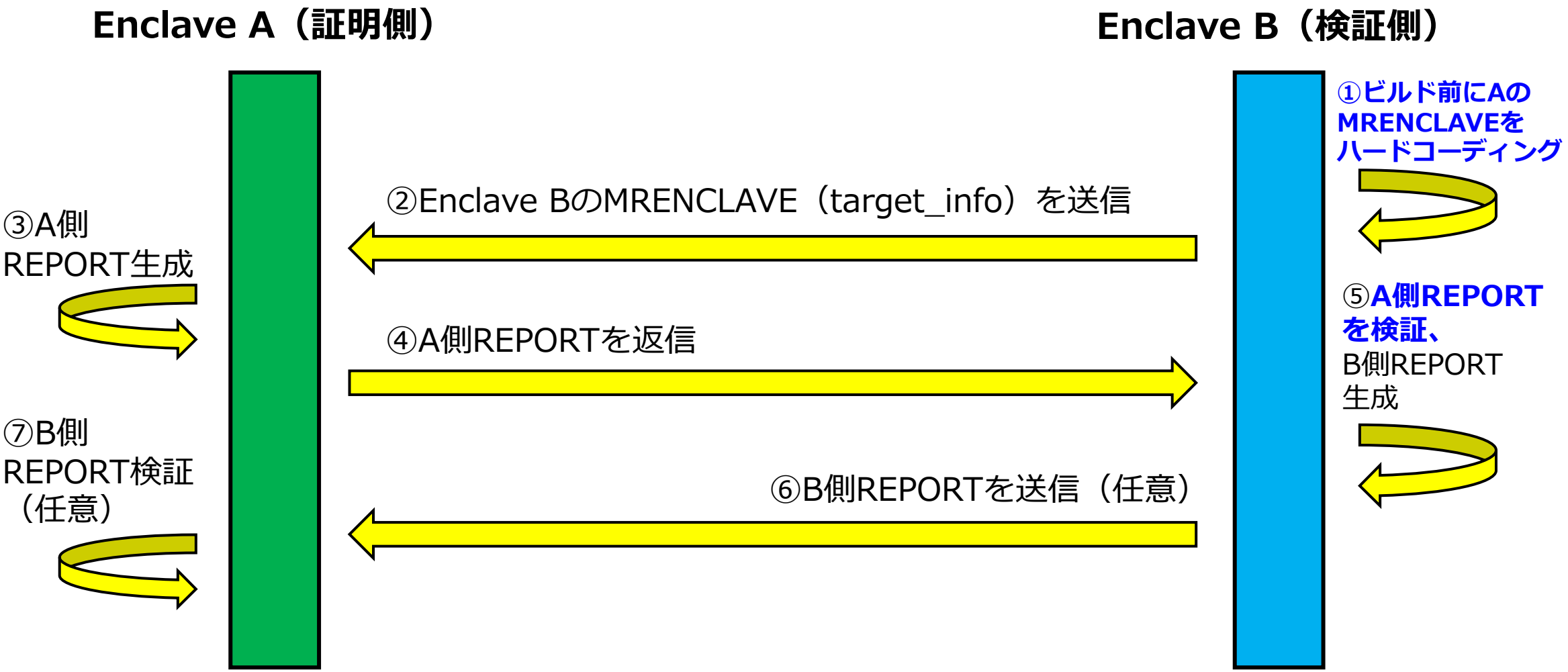
- 最低限かつ最も基本的なLAは、以下のフローで進行する



Enclave同一性検証を行うLAのフロー

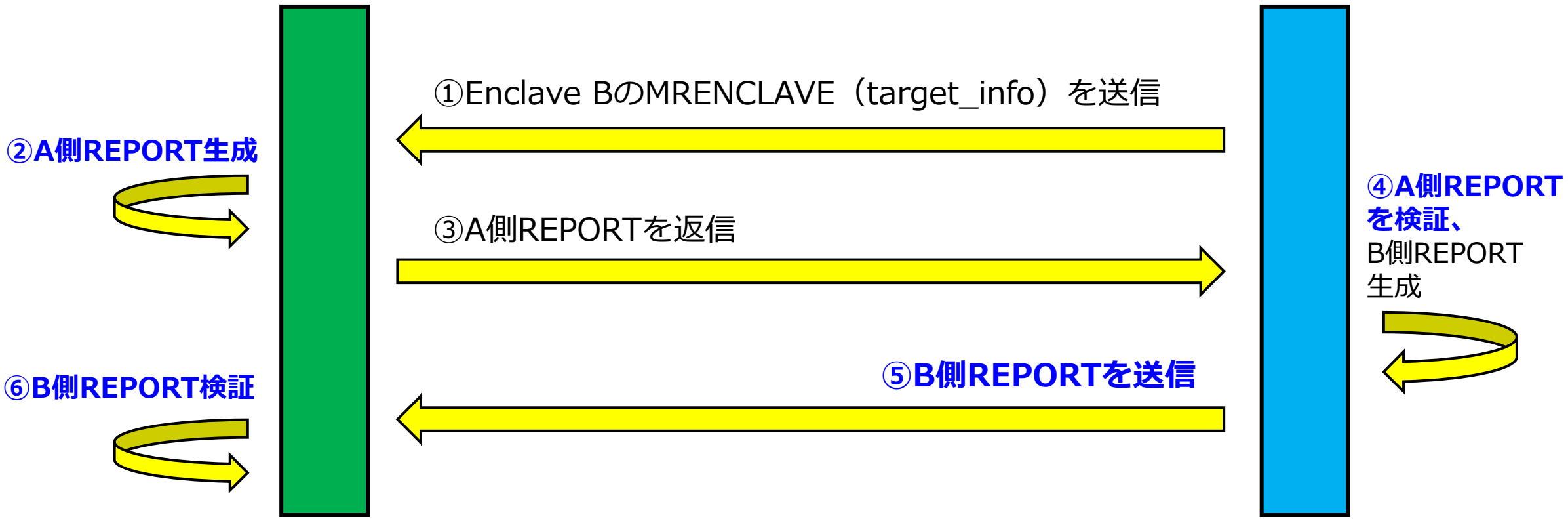


- 検証側が**証明側のMRENCLAVEを検証**する場合は以下の通り



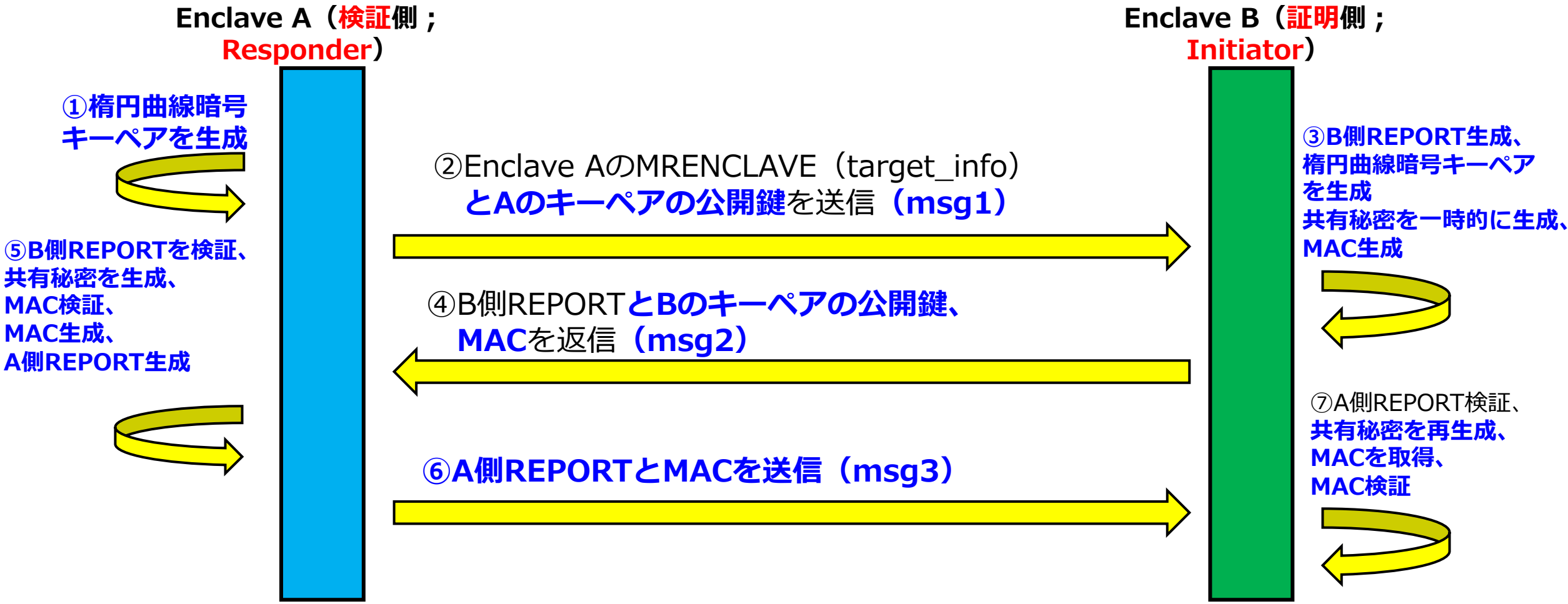
Enclave同一性検証を相互に行うLAのフロー

- 特定の条件下で**相互にMRENCLAVEを検証**する場合は以下の通り
  - 具体的には、AとBが**同じEnclaveイメージから生成**されており、従って**MRENCLAVEが同一**であるような場合

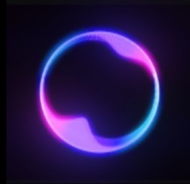


鍵交換も行う場合のLAのフロー

- LA後にEnclave間で安全な通信路を確立するために、**ECDHKE**も同時進行させる場合の例は以下の通り：



LAフローの詳説

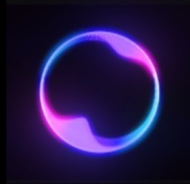


- ここでは、より実用性の高い**鍵交換込みの基本形LA（1ページ前の図のもの）**をベースに説明していく

- 適宜出現する関数名は、実際に**SGXSDKで提供されているAPI**を用いて実装する場合に使用可能なものである

- どのフローでも共通の事項として、**LA成立までは**Enclave間の**通信路は暗号化されていない**

ECDHセッションの初期化

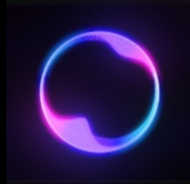


- 双方のEnclaveは、**sgx_dh_init_session**を呼び出す事で、以降の相互のやり取りのための**セッションを初期化**する

- 検証側は、**Responder**ロールとして初期化する

- 証明側は、**Initiator**ロールとして初期化する

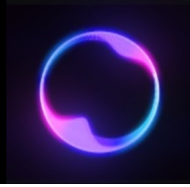
検証側 - Target Infoと公開鍵の送信



- 検証側Enclave（A）は、**sgx_dh_responder_gen_msg1**を呼び出し、自身の**MRENCLAVE**を含む**Target Info**を取得する
  - 証明側（B)と検証側が**同一のレポートキーを導出**できるようにするための処理。
    互いに**同一マシン**で**同一のTarget Info**を使ってレポートキーを生成すれば、それらは**必ず同一**になる
  - 仮にレポートキー生成に**証明側のMRENCLAVE**を用いると、**検証側でその正当性を検証できない**（**偽装可能**であるため）

- この際**楕円曲線暗号（ECC）のキーペア**も生成される為、その内の**公開鍵**を**Target Info**と共に**証明側Enclaveに送信**する（msg1）

証明側 - REPORTと共有秘密の生成



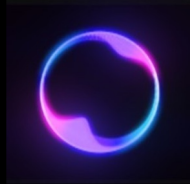
- 証明側Enclaveは、**sgx_dh_initiator_proc_msg1**を呼び出す事で、受信した検証側の**Target Info**を用いて**EREPORT**を**発行**し、**Target Infoに基づくレポートキー**で**MAC**を取った**REPORT構造体**を生成する

- 同時に、ECCキーペアも生成され、受信した検証側公開鍵とかけ合わせて**共有秘密**（共通鍵の素）が**一時的**に生成される

- さらに、**Bの公開鍵、上記REPORT等に対するMAC**も計算される
  - CMACの鍵には**共有秘密から導出**したものを使用

- この証明側REPORT、公開鍵、MACを**検証側に送信する**（msg2）

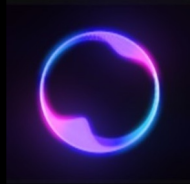
検証側 - REPORTの検証と共通鍵の生成（1/3）



- 検証側Enclaveは、**sgx_dh_responder_proc_msg2**を呼び出す事で、**受信した証明側のREPORT構造体**を**検証**する
  - 前述の通り、証明側が検証側のTarget Infoを用いて同一マシン上でEREPORTを発行していれば**レポートキーは同一**になる
  - よって、ここで検証側Target Infoと共にEGETKEY命令で得たレポートキーで検証すれば、**相手が正常**であれば**検証に成功**（REPORTのMACが一致）するはずである

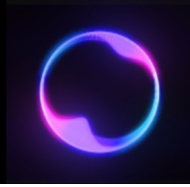
- また、受信した証明側公開鍵を用いて共有秘密が生成され、さらに受信したMACと比較して改竄が無いかが検証される

検証側 - REPORTの検証と共通鍵の生成（2/3）



- **証明側の同一性**に関しても検証する場合は、予め証明側の**同一性情報**を**検証側Enclave**に**ハードコーディング**する
  - **OS含むEnclave外**は**信頼不可能**であるため、Enclave外で**ファイルや標準入力から読み込む**のは**改竄の危険**がある
  - 同一性情報としては、MRENCLAVE、MRSIGNER等がある（RA編で詳説）
- **ハードコーディング**するとそのEnclaveの**MRENCLAVEが変わる**ため、**互いに互いのMRENCLAVEをハードコーディング**する事は**出来ない**
  - MRENCLAVEの**循環参照**が発生してしまうため
  - よって、**互いのEnclaveが完全に同一でない場合**は、原則としてLAによるMRENCLAVEの検証は**一方向的**になる

検証側 - REPORTの検証と共通鍵の生成（3/3）

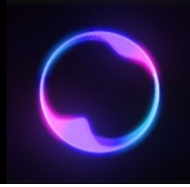


- 証明側も**検証側が同一マシン上に存在するか**を**確認**出来るように、受信した**証明側EnclaveのREPORT**から**MRENCLAVE等を抽出**して生成した**証明側Target Info**を用いて**EREPORT**が発行され、**検証側のREPORTが生成**される

- また、共有秘密から**共通鍵**（AEK）がEnclave内で導出される
  - LA成立後に証明側との暗号通信に使用できる

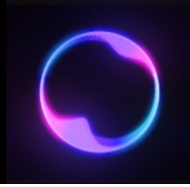
- 検証側REPORTと、REPORT及び任意の付加情報に対するMACを**証明側**に**送信**する（msg3）
  - CMAC生成時の鍵は証明側同様**共有秘密から導出**

証明側 - 検証側REPORTと共有秘密の検証（1/2）



- 証明側Enclaveは**sgx_dh_initiator_proc_msg3**を呼び出す事で、受信した検証側EnclaveのREPORTを検証する
- また、受信した共有秘密のMACを用いて、改竄が発生していないかが検証される
- また、共有秘密から**共通鍵**（AEK）がEnclave内で導出される
- 無事LAが完了したので、送信時は**Enclave内でAEKで暗号化**し、受信時は**Enclave内でAEKで復号**する事で、**Enclave間の暗号通信が実現**する

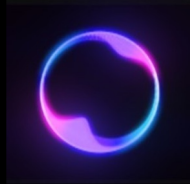
証明側 - 検証側REPORTと共有秘密の検証（2/2）



- 前述の通り、**MRENCLAVE検証は一方向的**であるため、**証明側**が**検証側**のMRENCLAVEを**直接検証する事は出来ない**

- 代わりに、各**Enclaveに署名する鍵**を**相異なるもの**にした上で**厳密に管理**し、**MRSIGNERを検証する**ようにする事で、人的な手間は挟まるが**ある程度同一性検証の確度**を上げられる
  - 署名自体も実行環境とは別の場所で行い、署名済みイメージのみを実行環境にデプロイするなど、様々な運用上の注意が必要となる

中間者攻撃（1/4）

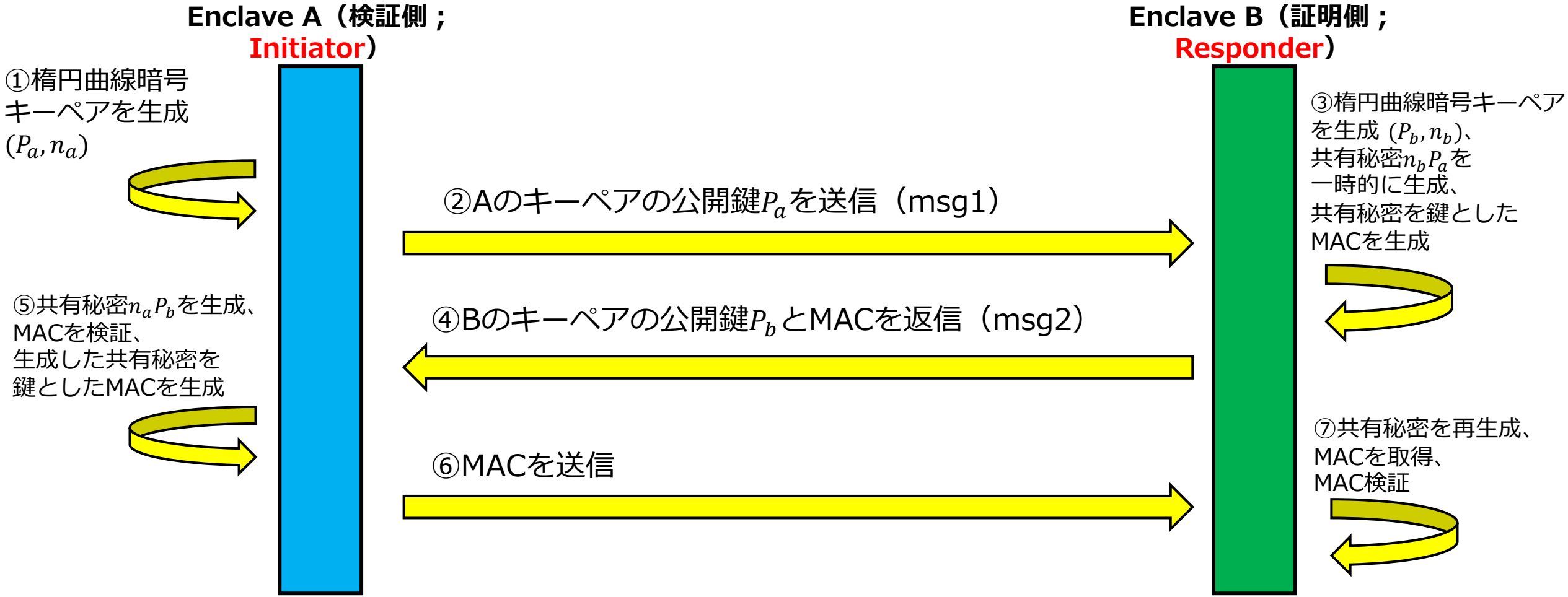


- ところで、LAに伴う鍵交換に使われるディフィーヘルマン鍵共有プロトコルは、**中間者攻撃**（Man-In-The-Middle Attack; MITM）に弱い事で有名

- 実際には**LA**は**MITMに対して耐性を持つ**が、耐性を付与する部分を省いた仮の形でMITMの仕組みを説明

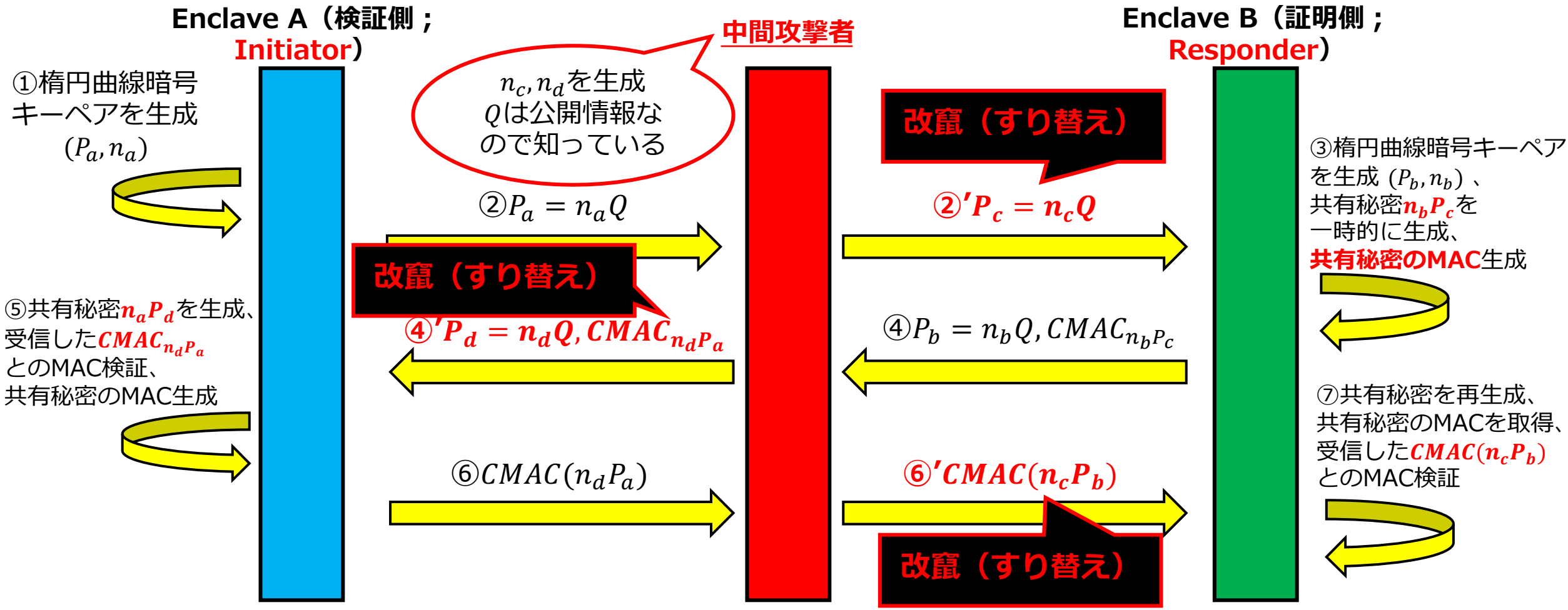
中間者攻撃（2/4）

- 以下の図は、LAのうち鍵交換に関連する部分のみを抽出し描写したものである：



中間者攻撃（3/4）

- しかし、これでは以下のような攻撃者による**攻撃**が行われてしまう



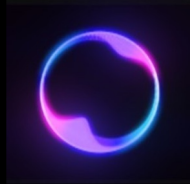
中間者攻撃（4/4）

- Enclave A・Bは、それぞれ相手が元々送っていた公開鍵P_bやP_aの代わりに、中間攻撃者によってP_dやP_cに**すり替えられている**と**知る由もない**

- $n_d P_a = n_a P_d, n_c P_b = n_b P_c$であるため、**CMAC値すら偽造できてしまう**

- **中間攻撃者**は、Aの暗号データは$\boldsymbol{n_d P_a}$、Bの暗号データは$\boldsymbol{n_c P_b}$を用いて**復号**する事で、容易にその**平文を読めてしまう**

中間者攻撃に対するLAの対策（1/2）

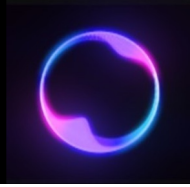


- LAでは、実はREPORT内のreport_dataに、**Enclave A・B双方の公開鍵**に対する**SHA256ハッシュ値**が同梱されている

- このreport_dataはREPORT構造体全体ごと**レポートキーで署名**されているため、**レポートキーを知る術のない**中間攻撃者が改竄した公開鍵を同梱したREPORTへのMACを**偽造する事は不可能**

- よって、公開鍵の改竄を行った時点でREPORTの**検証に失敗**するため、**LA**では**中間者攻撃を行う事が出来ない**

中間者攻撃に対するLAの対策（2/2）

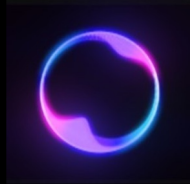


- 折角無事に鍵交換しても、Bが**自ら復号**し**OCALLで外にデータをバラす**ような挙動を取るのでは話にならない

- より回りくどい方法として、**B側が中間攻撃者と結託**（あるいはB側が**中間攻撃者を用意**）し、B側でSGXAPIを迂回し⑦**での検証をスキップ**後、B側の情報を漏洩させてしまう可能性もある
  - 一方、検証（A）側が自らの秘密情報をわざわざ漏らす動機づけは余りない

- よって、検証側は証明側Enclaveのコードを**予め確認**しておき、それに対する**正しいMRENCLAVE**を渡してきているかをハードコーディングとの比較等で**検証した方が良い**
  - 後述のRAでは、この役割は**リモートユーザ**の仕事となる

Local Attestation v2（1/2）

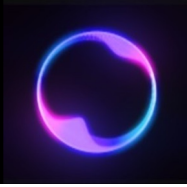


- report_dataに**両者の公開鍵**P_aとP_bのハッシュを同梱する代わりに、証明側（Enclave B）のREPORTを特定の形で簡略化した内部的な構造体（**Proto Spec**）と、処理する**Enclave自身の公開鍵**（P_aかP_b）を連結したものに対するハッシュ値を同梱する方式
  - msg2とmsg3の内容がLAv1と若干異なってくる

- 最終的に互いにP_aやP_bを同梱したreport_dataを送り合う事になるので、**MITMに対して脆弱になる事はない**

- REPORT自体に対しても改竄検知が出来るようになるので、**わずかに安心感が向上**する、といった所か

Local Attestation v2（2/2）

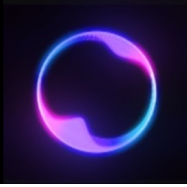


- LAv2を使用したい場合、sgx_dh.hをインクルードした上で、#define SGX_USE_LAv2_INITIATORのマクロを宣言する

- Proto Specの詳細はかなり解読難易度が極悪であるため、深入りはあまりオススメしない

Remote Attestation

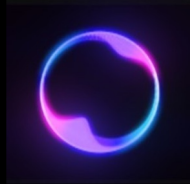
Remote Attestationの概要（1/7）



- SGXマシン上のEnclaveの機能を使用したい**リモートのユーザ**が、本当にその**マシン**や**Enclave**が**信頼可能であるか**を**検証**するためのプロトコル

- SGXプログラミングにおいては**特に難易度が苛烈**であり、これさえ実装できればSGXSDKで提供されている機能を利用してのSGX関連の実装で**他に恐れるものは無くなる**レベル

Remote Attestationの概要（2/7）

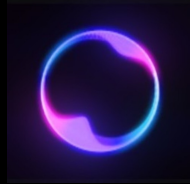


- RAにおいて、**SGXマシン側**の事を**ISV**（Independent Software Vendor）と呼ぶ事が多い

- 一方、ISVのSGX機能利用する**リモートユーザ（非SGX側）**は**SP**（Service Provider）と呼ぶ事が多い

- 何故このような命名であるかは、後のセクションで解説する

Remote Attestationの概要（3/7）

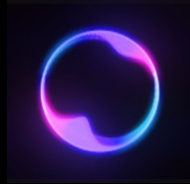


- RAは実装は非常に面倒臭いが、その根源的な目的自体は単純である：

[ISV・SP] RA後にTLS用の**セッション鍵を交換する**（LA同様）

[SP] ISVのCPUとEnclaveの**正当性や同一性をリモートから検証する**

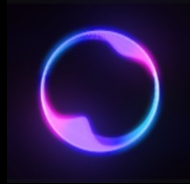
Remote Attestationの概要（4/7）



- LAでは、相手のマシンの信頼性の根拠として、「**自身と同じマシン上で動作している**」という事をレポートキーの同一性を通して使用していたが、RAでは**リモート**なのでこれは不可能

- 代わりに、**Quoting Enclave**（QE）がそのEnclaveと**LA**を行った上で、**プロビジョニング**で配備された**Attestationキー**でそのEnclaveの**REPORTに署名**し、**QUOTE構造体**を生成する

Remote Attestationの概要（5/7）

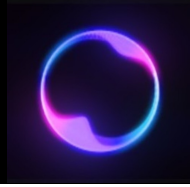


- その後、SPはISVから受け取った**QUOTE**を、**第三者検証機関**である**Intel Attestation Service**（**IAS**）に送信する

- **Quote署名**に対する、IASの持つ**EPIDグループ公開鍵**による検証や、QUOTE内のREPORT内に存在する**CPUSVN**等から、そのマシンが**信頼可能であるかをIASに判定してもらう**

- SPはその判定結果である**アテステーション応答**（**RA Report**）から、リモートのEnclaveが**信頼可能であるかを判断**し、その後**やり取りを続けるかを決定**する
  - Enclave同一性の検証はQUOTE内のREPORTを参照して実施する

Remote Attestationの概要（6/7）

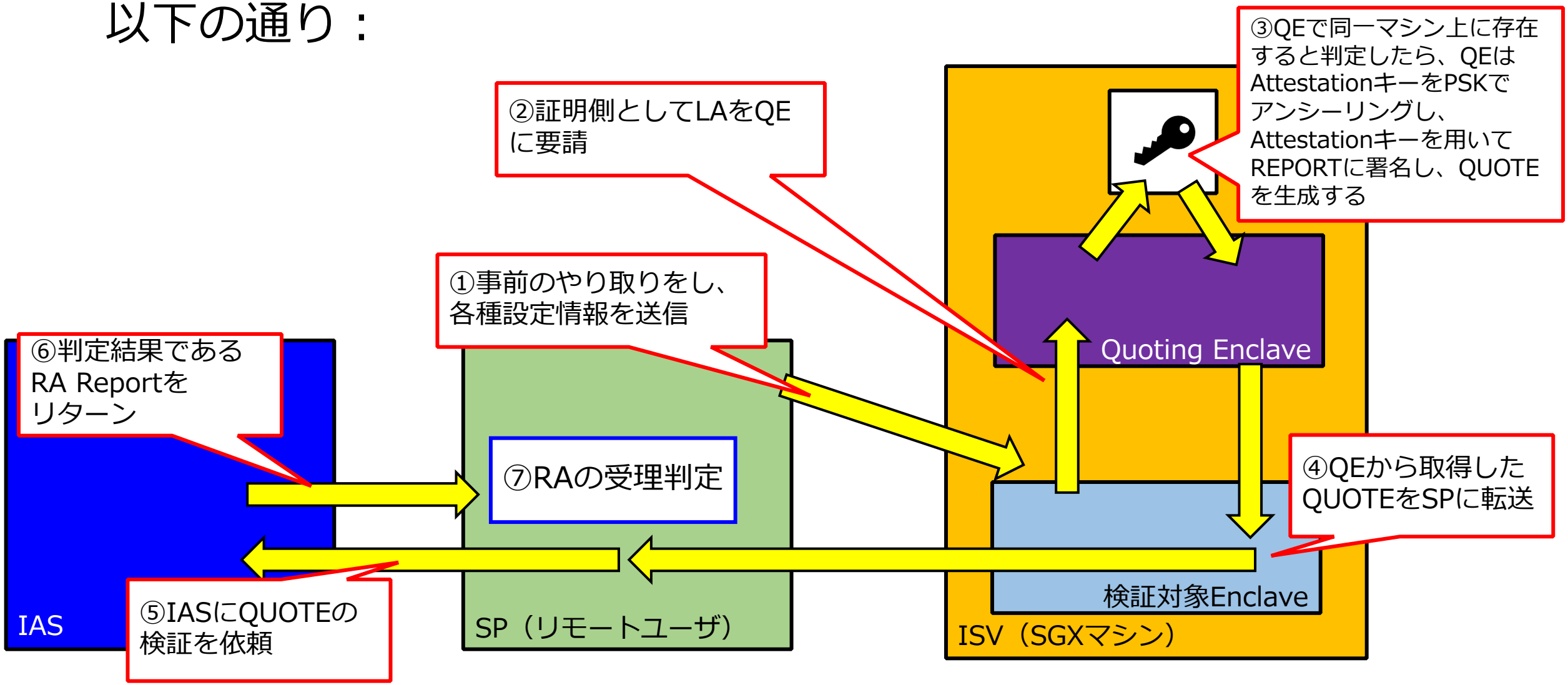


- LA同様、RAを進める上でRA受理後の**通信に用いる共通鍵**の交換を**EC-DHKE**を用いて実施する

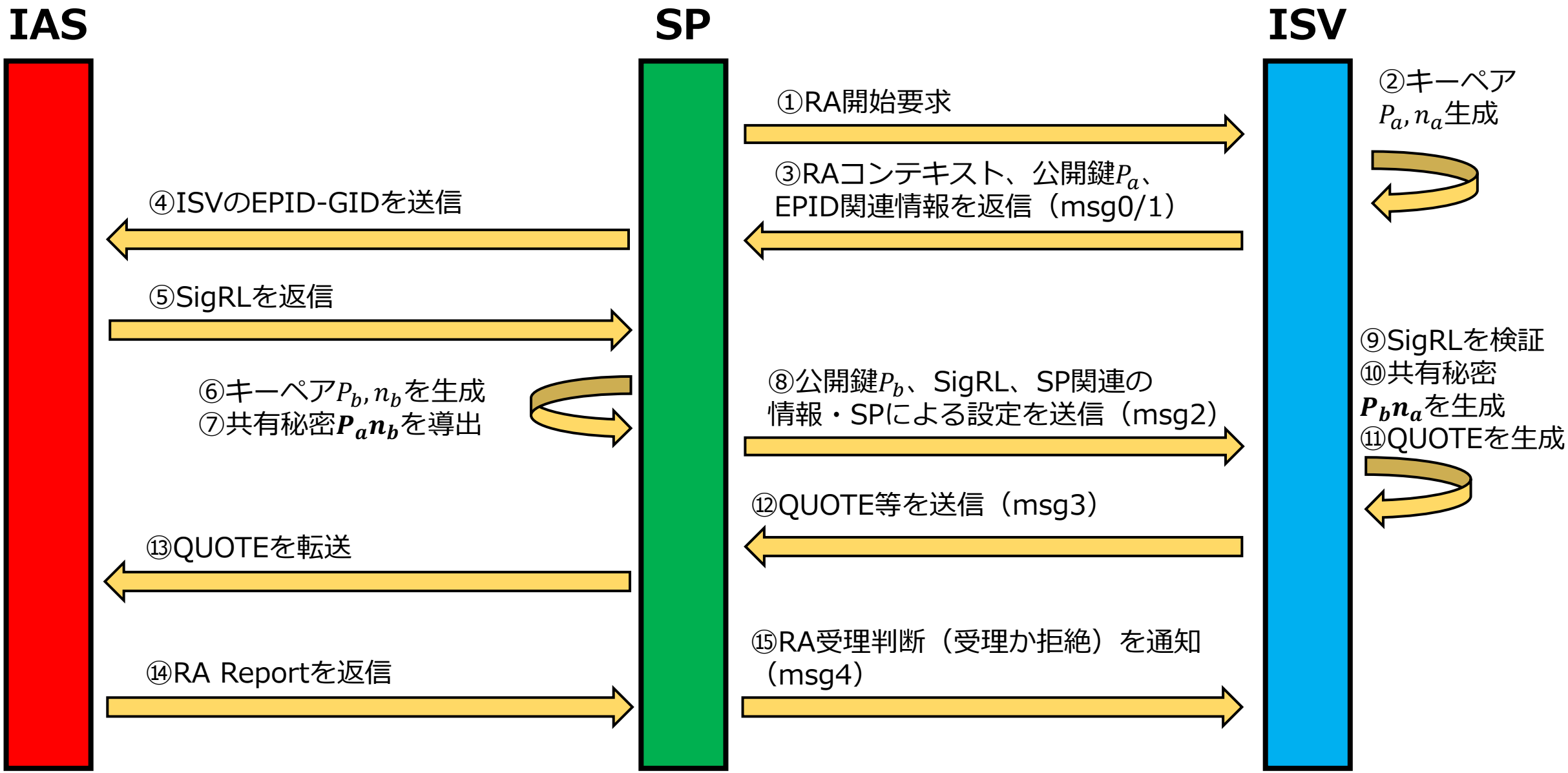
- 相手のEnclaveが本当に意図する通りのEnclaveであるのかの**同一性検証**は、**予め取得した**ISVのEnclaveの**MRENCLAVE**を、**QUOTE内のREPORT内**に含まれる**MRENCLAVE**と比較する事で**SPが行う**
  - RAでは、**SP側の環境**は**完全に安全**であるという前提で**脅威モデルが設定される**事に注意

Remote Attestationの概要（7/7）

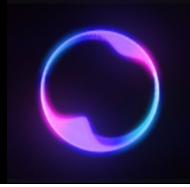
- RAについて、LAからの拡張に着目した場合の概要図は以下の通り：



Remote Attestationの全体フロー



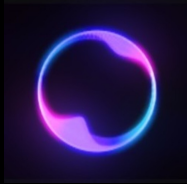
RAを行うにあたって（1/3）



- RAを実行するには、Intelの**EPID Attestation**の**サービスに登録**する必要がある

- 事前学習で実施予定であるが、Intelのアカウントを作成し、以下のリンク先で**SPのサブスクライブ**（SPが指定する各種情報を設定し、そのSP固有のID等を取得する）を行う必要がある
  https://api.portal.trustedservices.intel.com/EPID-attestation

RAを行うにあたって（2/3）



EXPLORE EPID ATTESTATION TO ENHANCE ENCLAVE SECURITY

Intel® SGX Attestation Service Utilizing Enhanced Privacy ID (EPID)

The Intel SGX attestation service is a public web service operated by Intel for client-based privacy focused usages on PCs or workstations. The primary responsibility of the Intel SGX attestation service is to verify attestation evidence submitted by relying parties. The Intel SGX attestation service utilizes Enhanced Privacy ID (EPID) provisioning, in which an Intel processor is given a unique signing key belonging to an EPID group. During attestation, the quote containing the processor's provisioned EPID signature is validated, establishing that it was signed by a member of a valid EPID group. A commercial use license is required for any SGX application running in production mode accessing the Intel SGX attestation service.

Enroll in Intel SGX Attestation Service

One of the key decisions when subscribing to the Intel SGX attestation service is the mode chosen for the EPID signature, Random Base Mode or Name Base Mode. To get more info on EPID signature modes as well as provisioning and attestation services, click here to download a white paper.

Linkable Quotes (Name Base Mode): A name is picked for the base to be used for a signature, making signatures linkable. Verifying two signatures enables you to tell whether they were generated from the same or different signers. Name Base Mode is preferred to protect against compromise.

Unlinkable Quotes (Random Base Mode): Every signature gets a different random base, making the signatures unlinkable. Verifying two signatures does not enable you to tell whether they were generated by the same or different signers.

The Intel® SGX Services and Intel® TDX Services Terms of Use govern your use of these services except where we expressly state that separate terms (and not these) apply. By using our services, you are agreeing to these terms. Make sure you read them carefully.

API Documentation

Attestation Report Root CA Certificate: DER PEM

Development Access

Subscribe now for immediate access to the development environment where non-production Intel SGX enabled applications can test attestation functionality in debug mode prior to releasing to production.

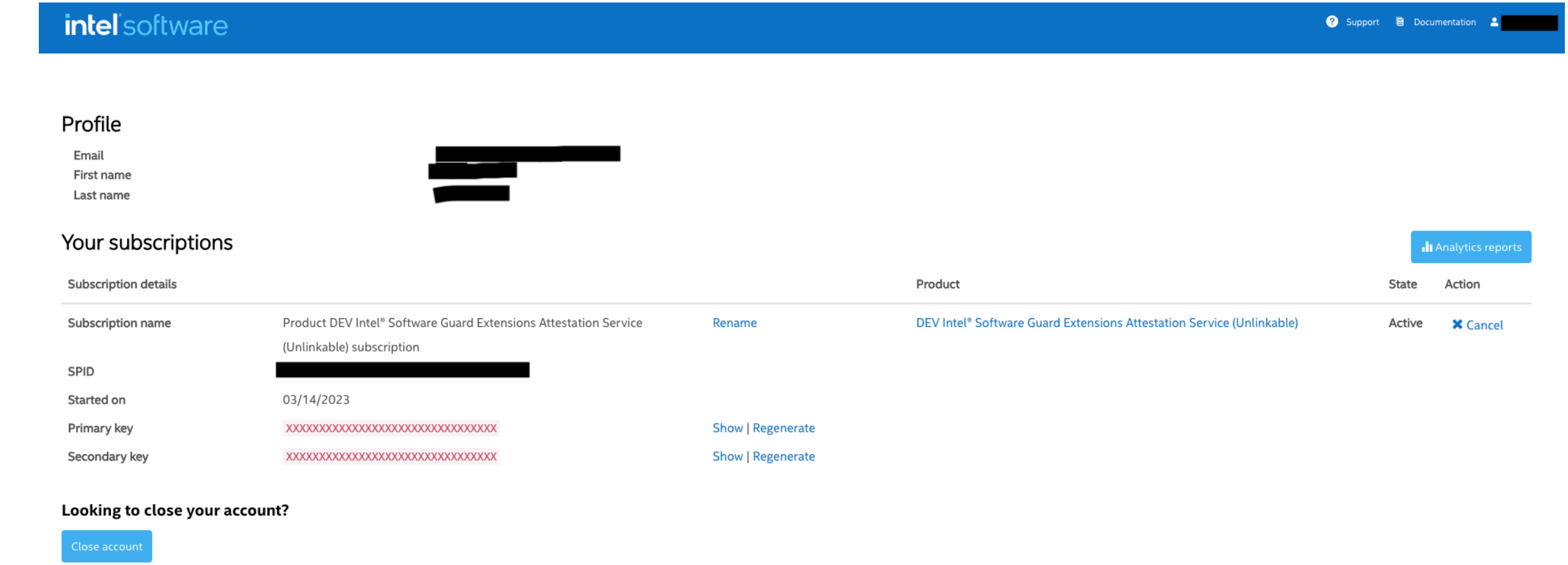
Subscribe (linkable)

Subscribe (unlinkable)

Production Access

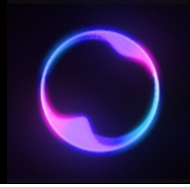
Once a commercial use license has been executed and your application/solution has been added to the Launch Policy List (if applicable), you just need to Subscribe for production access. Once your subscription is activated you will be able to utilize the production version of the Intel SGX attestation service. For more information on these required steps, refer to our Commercial License Request page on Intel Developer Zone.

RAを行うにあたって（3/3）



RA詳説

RAフロー詳説

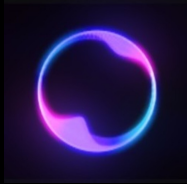


- ここからは、具体的にどのようにRAの処理を進行させていくかを詳説していく

- **RAが成立する**（SPがRAを受理し、msg4をISVが受け取る）までは**通信路は保護されない**点もLAと同様

- 実際に実装する際は、暗号鍵の**エンディアン**の**SP-ISV間での不整合**などの、**非常に細かくかつ厄介な問題**が**数多くつきまとう**が、それらについては**説明は割愛する**

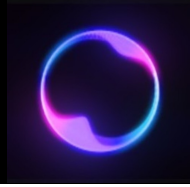
RA開始要求・msg0の送信（1/2）



- SPはISVに**RA開始要求**（チャレンジ）を送信する
  - 特定のポートへのアクセスなど、やり方やフォーマットは自由

- ISVは**RAを初期化**し、**RAコンテキスト**と**拡張EPID-GID**を生成し、それらを**msg0**として**SPに返信する**

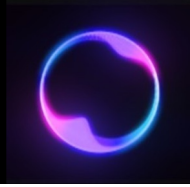
RA開始要求・msg0の送信（2/2）



- **RAコンテキスト**は今**どのRAについて処理しているか**の**識別**をEnclaveが行うために使用する値
  - RA中SPからISVにデータを送信する際は必ずこれも含まれる
  - RA完了後の**セッション鍵取得**や、ひいては**SPの識別**にも使えるので**非常に重要**

- 拡張EPIDグループID（**拡張EPID-GID**）は、RAの**第三者検証機関**に対するIDである[6]
  - EPID Attestationの場合、**第三者検証機関**として**IAS以外を利用する実例は存在しない**ため、この値は必ず**IASのIDである0になる**

msg0の受信・処理、msg1リクエスト

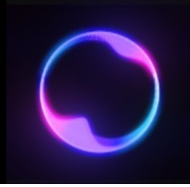


- SPはmsg0を受信したら、後続の処理のために**RAコンテキスト値**を抽出し保持しておく

- 拡張EPID-GIDに関しても検証し、もし**0でない場合**にはこの時点でRAを拒絶する
  - EPID Attestationで**IAS以外の第三者検証機関**を**使う事はありえない**ため

- その後、ISVに**msg1**（後述）を**送信するように要求**する
  - この手間を省くために、最初からmsg0とmsg1を**同時に送信させても良い**

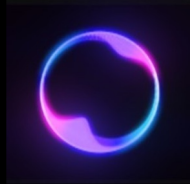
msg1の生成・送信（1/3）



- ISVは、**sgx_ra_get_msg1**関数を呼び出す

- この関数により、**256bitの楕円曲線暗号**の**キーペア**が**生成**される
  - 使用される楕円曲線は**NISTのP-256の要件**を満たしている必要がある
  - SGXではこの楕円曲線コンテキストとして**NID_X9_62_prime256v1**が使用される
  - 公開鍵には**それぞれ256ビット（32バイト）**の**x成分**と**y成分**があり、秘密鍵は同じく**256ビット**の**単一の値（ベースポイントに対する係数）**である

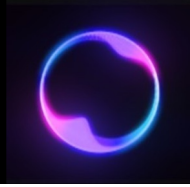
msg1の生成・送信（2/3）



- また、ISVのCPUが含まれている**EPIDのグループID**（**EPID-GID**）もこの関数で取得される
  - 先程の**拡張EPID-GIDとは別物**なので注意。[6]のフォーラムで回答者が一瞬混同して勘違いしていたレベルには紛らわしい
  - このEPID-GIDは、プロビジョニングのセクションで説明した通り、**同一のCPUの種類**と**CPUSVN**に対して割り当てられた（単一の）**EPIDグループについてのID**である

- 上記キーペアの**公開鍵G_a**（x, y成分であるG_{a_x}とG_{a_y}の連結である512ビットの値）と、**EPID-GID**により構成される構造体が**msg1**としてリターンされる

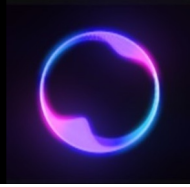
msg1の生成・送信（3/3）



- ちなみに、このsgx_ra_get_msg1の呼び出しにより、裏で**QEのTarget Infoを取得する処理**（**LAの最初の処理**）が行われている

- リターンされた**msg1をSPに返信**する

SigRLの取得

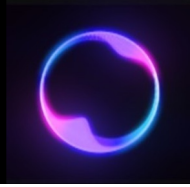


- SPはmsg1を受信後、その中からISVの**EPID-GID**を抽出する

- その後、IASに**EPID-GIDを転送**し、そのEPIDグループの**SigRL（署名失効リスト）**を**IASから受信**する
  - https://api.trustedservices.intel.com/sgx/dev/attestation/v5/sigrl/{gid}にGETリクエストを送信する事で、対応するSigRLを取得できる（{gid}の部分はISVのEPID-GIDと置き換える）

- SigRLについての**詳細は後述**

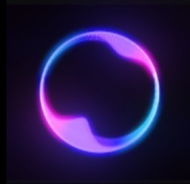
msg2の作成（1/5）



- SPは、ISV同様**256bitの楕円曲線暗号キーペア**を生成する
  - **SP側**では、これを含む**暗号関連の操作**は基本的に**OpenSSLのライブラリ**を用いる事になる

- ISVの**キーペア公開鍵**G_aと生成した**SPのキーペア秘密鍵**を用いて**共有秘密**G_{ab}を生成し、そのx成分であるG_{ab_x}から**鍵導出鍵（KDK）**を生成する
  - リトルエンディアン化したG_{ab_x}に対し、オールゼロな16バイトのバイト列を鍵として128bit AES-CMACを取る
  - KDKはKey Derivation Keyの略

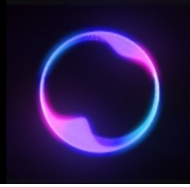
msg2の作成（2/5）



- 続いて、**KDK**を用いて**SMK**（Session MAC Key）を生成する
  - バイト列「¥x01SMK¥x00¥x80¥x00」に対してKDKを鍵とした128bit AES-CMACを取得し、それをSMKとする

- KDKやSMKは**共有秘密をベースとしたMAC値**であるため、**ISV側**もEnclave内で**同一のそれらを導出**する事が出来る
  - 逆に言えば、この当事者であるSPとISV以外は、これらの鍵は知り得ない

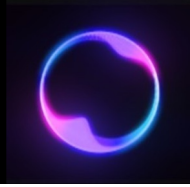
msg2の作成（3/5）



- 続いて、 SPの**公開鍵G_b** とISVの**公開鍵G_a**を**結合したバイト列**（文字通りG_bの後にG_aを繋げた128バイトのバイト列）に対し、SPの**署名用ECC秘密鍵**で署名する

- この**署名用ECC秘密鍵**は、これまでのセッションで生成したキーペア（G_bなど）とは**全くの別物である**事に注意
  - **セッションキーペアの方の公開鍵G_aとG_b**が**改竄されていない**事を**検証**するために使用される**署名検証用の鍵**である
  - この**署名検証用のキーペア**（**セッションキーペアではない**）は**RA前に予め生成**しておき、**検証用の公開鍵**の方は改竄されないように**ISVのEnclaveコードにハードコーディングしておく**

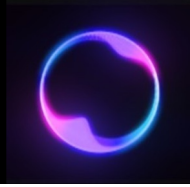
msg2の作成（4/5）



- この署名用秘密鍵により、G_bとG_aの連結に対して署名して生成された**電子署名**を**SigSP**とする

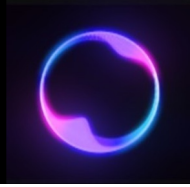
- SigSPは後述の通り**msg2に同梱**され、ISV側での**msg2処理時**に**Enclave内で検証**される
  - 中間者攻撃等によって改竄が発生した場合、ここでまず気付く事が出来る
  - 後述するが、msg3におけるreport dataの検証でも検知可能

msg2の作成（5/5）



- SPのセッション公開鍵G_b、SPID、Quoteタイプ、KDF-ID、SigSP（これらをまとめて**A**とする）をmsg2に入れる
  - **SPID**：EPID Attestationのサブスクリプション後、管理画面で確認できる、その**SPに割り当てられたID**。
    後述する、EPID署名対象の1つである**Basename**のもとになる
  - **Quoteタイプ**：**同一Attestationキー**による**複数のQUOTE構造体**が存在する場合、それらが同一Attestationキーによるものかを**識別できるか否か**を設定する項目。詳細は次ページ
  - **KDF-ID**：鍵導出関数ID。現在のSGXSDKの実装では、この値として**1以外は受け付けられない**ようになっている

Quoteタイプ（1/2）

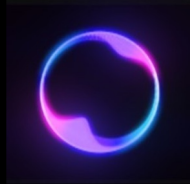


- Quoteタイプには、**Unlinkable**モード（旧称：Random Base Mode）と**Linkable**モード（旧称：Name Base Mode）が存在する

- Unlinkableモードでは、**同一Attestationキー**により**生成**された**QUOTE**が**複数存在する**時、それらが同一のAttestationキーにより生成されたと**紐付ける事が出来ない**
  - QEの実装を見ると、SPIDに乱数を結合してBasenameを生成している[9]
  - **乱数**を使用しているため、旧称が**Random** Base Modeであったのであると推測できる

```
// 最初の&basenameには既にSPIDが格納されている状態である
uint8_t *p = (uint8_t *)&basename + sizeof(*p_spid);
se_ret = sgx_read_rand(p, sizeof(basename) - sizeof(*p_spid));
```

Quoteタイプ（2/2）



- 一方、Linkableモードでは、同一Attestationキーにより生成されたQUOTEが存在する場合、**同一Attestationキーにより生成された**ものであると**特定できる**
  - QEの実装を見ると、BasenameとしてSPIDをそのまま代入している
  - 旧称の**Name** Base Modeは、SPID（**SPのName**）をそのままBasenameにしている所から来ていると思われる

- **どちらのモードを選ぶべき**かは**トレードオフ**である。とりあえずデバッグ版Enclaveの開発に用いるなら**Unlinkableでも問題ない**が、この考察は**後のセクションで行う**

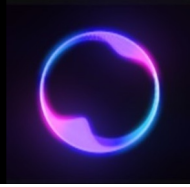
msg2の作成・送信

- そして、SPは先程の**A**に対する、**SMK**を鍵とした**128bit AES-CMAC**を生成し、その**MAC値**（$CMAC_{SMK}(A)$）も**msg2に入れる**

- 最後に、**SigRL**と**SigRLの文字列長**をmsg2に入れ、**msg2を完成**させる

- msg2を完成させたら、SPはその**msg2をISVに送信**する

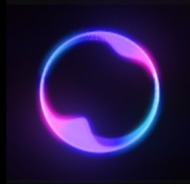
msg2の構造



- sgx_ra_msg2_tの構造は以下の通り：

msg2のメンバ	説明
sgx_ec256_public_t g_b	SPのセッション公開鍵G_b
sgx_spid_t spid	SPID
uint16_t quote_type	Quoteタイプ
uint16_t kdf_id	鍵導出関数ID
sgx_ec256_signature_t sign_gb_ga	SigSP
sgx_mac_t mac	上記5つ（A）に対する、SMKを鍵とした$CMAC_{SMK}(A)$
uint32_t sig_rl_size	SigRLの文字列長
uint8_t sig_rl[]	SigRL本体

msg2の処理・msg3の生成



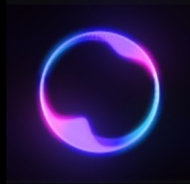
- ISVはSPからmsg2を受信したら、**sgx_ra_proc_msg2関数**にmsg2を渡し、**QUOTEを内包**している**msg3**を取得する

- この関数内で**SigSPの検証**や、QEによるISVのRA対象Enclaveと**LAの完了処理**（QEが対象EnclaveのREPORTを検証）、**msg2**内のAに対する**MACの検証**も行われる

- **共有秘密G_{ab}**や**SMK等の鍵の生成**もこの関数内で行われる

- **LA**によって**ISVのEnclaveが正当**であると**QEに見なされた**後は、QEにより**Attestationキー**を用いてそのISV Enclaveの**REPORT等**に対し**EPID署名**が行われ、その結果として**QUOTE**が生成される

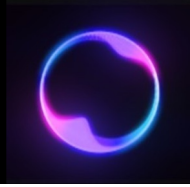
QUOTEの構造



- sgx_quote_tの構造は以下の通り：

QUOTEのメンバ	説明
uint16_t version	QUOTE構造体のバージョン
uint16_t sign_type	EPID署名のタイプ（恐らくQuoteタイプ）
sgx_epid_group_id_t epid_group_id	ISVのマシンのEPID-GID
sgx_isv_svn qe_svn	QEのセキュリティバージョン番号
sgx_isv_svn pce_svn	PCE（Provisioning Certificate Enclave; DCAP Attestationの文脈で登場するAE）のセキュリティバージョン番号
uint32_t xeid	拡張EPID-GID
sgx_basename_t basename	QUOTEの生成に使用されたBasename
sgx_report_body_t report_body	REPORTのボディ（本体）
uint32_t signature_len	EPID署名の長さ
uint8_t signature[]	EPID署名本体。QEにハードコーディングされたIASの秘密鍵により暗号化されている

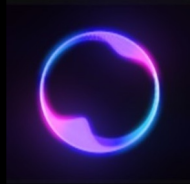
EPIDにおける失効リスト（1/2）



- EPIDには、3つの**失効リスト**（その要素を持っている主体は侵害されているので無効であると見なすための**ブラックリスト**）が存在する
  - GroupRL、**SigRL**、PrivRLの3つ
  - RLはRevocation Listの略

■GroupRL
グループ失効リスト。このリストに載っている**EPID-GIDのマシン**は全て**問答無用で拒絶**されるため、**最も影響力の大きい失効リスト**

EPIDにおける失効リスト（2/2）



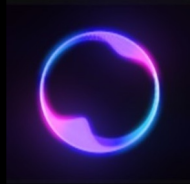
■ **SigRL**

署名失効リスト。Attestationキーは漏れていないが、**そのマシンが侵害されている可能性がある**場合に、そのマシン（Attestationキー）による**署名をブラックリスト化**するもの

■ **PrivRL**

秘密鍵失効リスト。**Attestationキー**（＝EPIDメンバ秘密鍵）の**漏洩**が発覚し、実際にその鍵がIntelによっても実体が確認された場合に、そういった**Attestationキー自体をブラックリスト化**したもの

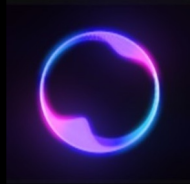
SigRL（1/3）



- EPID署名では、**署名対象のコンテンツ**（SGXでは**REPORT**等）に対する署名の他に、**Basenameに対する署名**が別個行われ、その結果も電子署名に含まれる

- SigRLは、侵害されたマシンについての**Basename**と**それに対する電子署名**の**ペア**を記録している**ブラックリスト**である

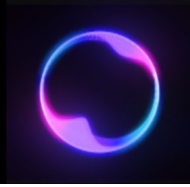
SigRL（2/3）



- このBasenameは**巡回群の元B**と見なす事ができ、この元Bに対する、**Attestationキー由来**の**値f**で**べき乗した元**がBasenameに対する**署名B^f**となる[1][8]

- Unlinkableの場合、**元B**（=Basename）は**ランダム**であるが、**このRAにおける元Bへの署名**に、**SigRLに登録されている署名**を**生成したAttestationキー**が使われているかは、**ゼロ知識証明**を用いて**判別する事が出来る**
  - 失効していると判定された場合、IASによるRA Reportにて**SigRLに基づき失効している**ので**信頼ならない**、との返答がSPに渡される

SigRL（3/3）

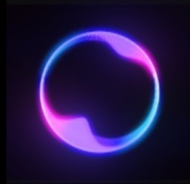


- **SigRLのエントリに対する署名**も**QUOTE**における**署名対象**に含まれるため、もし**SigRLを意図的に無視**すると、IASはQUOTE中のEPID-GIDから本来のSigRL有無を判別し、**検証を迂回された事を検知**できる
  - あるEPID-GIDに対する**SigRLが空**の場合は、**署名対象の構造が変わるため**（[9]の323行目）、**IAS側による検証時**にSigRL関連の有無で**一発でバレる**

- **無関係のSigRL**を使用してQUOTEを生成しても、IASによる**ゼロ知識証明に失敗**するので**偽造は不可能**

- よって、その**EPID-GID**に紐づく**正しいSigRL**を用いて**QUOTEを生成**しないと**IASにバレる**仕組みになっている

msg3の取得・送信

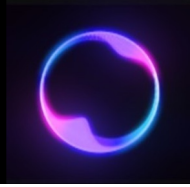


- sgx_ra_proc_msg2を実行し**正常にQUOTEが生成**されると、以下のような構造である**msg3**（sgx_ra_msg3_t）が返却される

msg3のメンバ	**説明**
sgx_mac_t mac	以下3つの連結に対する、SMKを鍵とした 128bit AES/CMAC値（Enclave内で計算される）
sgx_ec256_public_t g_a	ISVのセッション公開鍵G_a
sgx_ps_sec_prop_desc_t ps_sec_prop	PSE（Linuxでは使用不可になっている例のAE）を 使用する際の追加情報。 PSE不使用時は0で埋めておく
uint8_t quote[]	QUOTE構造体（sgx_quote_t）

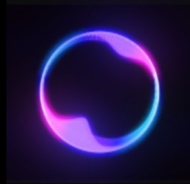
- ISVは、この**msg3**を**SPに返信**する

msg3の処理



- msg3の受信後、SPは以下の確認を行う：
  - **msg1**中のG_aと**EPID-GID**が、**msg3**中のG_aと**EPID-GID**と**一致しているかを確認**する。msg1では**改竄可能**であったが、msg3内のこれらは**CMAC付き**で**Enclave内**で**取得・同梱**されるため、ここで**改竄検知が可能**である
  - SMKを用いて**msg3のMAC**を検証する
  - QUOTE内のREPORTのさらにその中にある**report dataの先頭32バイト**が、**G_a, G_b, VKを連結したバイト列**（$G_a||G_b||VK$）の**SHA256ハッシュ**と**一致しているかを確認**する。
    VKはバイト列「¥x01VK¥x00¥x80¥x00」の、KDKを鍵として生成した128bit AES/CMAC値

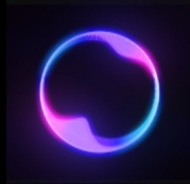
IASによるQuoteの検証



- 検証に成功したら、SPは**QuoteをIASに送信**し、IASによるQuote検証結果として**アテステーション応答**（**RA Report**）を受信する
  - https://api.trustedservices.intel.com/sgx/dev/attestation/v5/report に対してPOSTリクエストで送信する
  - この「**RA Report**」は**REPORT構造体とは別物**なので注意

- 予め取得しておいたIASのルートCA証明書を用いて、RA Reportに対するIASによる署名を検証する

RAの受理/拒絶判断（1/4）

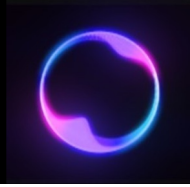


- RA Report（JSON形式）のisvEnclaveQuoteStatusフィールドに、QUOTE（つまり**ISV自体**）が**信頼可能であるか**の**IASによる判定**が同梱されている

- このフィールドの値が”**OK**”であれば少なくとも**ISVのマシンは信頼可能**であり、後続の処理に進んで良い

- その他の場合は**何らかの問題がある**ので、本来は**RAを拒絶すべき**である。しかし、**現実問題として”OK”になるケース**は**多くなく**、判断に**一定の困難なトレードオフ**が発生する
  - 後のセクションで解説

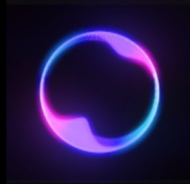
RAの受理/拒絶判断（2/4）



- IASによるQuoteの検証結果がOKでない場合、どのような状況であれば妥協して受理するか判断する上で、RA Reportに同梱されている様々なフィールドを参照する事が出来る

- 例えば、isvEnclaveQuoteStatusの示すステータス自体の他、**advisoryIDsフィールド**を参照すると、**ISVのマシンが抱えている脆弱性のIntelアドバイザリのIDの一覧**を取得できる
  - 例えば、本ゼミの攻撃パートで説明する**Load Value Injection**に対して脆弱である場合、それに対するアドバイザリのIDである INTEL-SA-00334がこのフィールドに格納されている

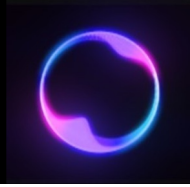
RAの受理/拒絶判断（3/4）



- RA ReportにおけるQUOTEステータスの検証によりISVマシンを信頼すると決定したら、今度はmsg3内のQUOTE内の**REPORT構造体**を取り出し、ISVの**RA対象のEnclave**の**同一性を検証**する

- 予め控えておいた**対象Enclave**の**MRENCLAVE**と**MRSIGNER**、**ISV ProdID**と**一致**するか、また**ISVSVN**が**要求値以上**であるかを検証する
  - MRENCLAVEとMRSIGNERは、sgx_signツールでEnclaveイメージからメタデータをダンプし取得できる
  - ISV ProdIDやISVSVNは、Enclave設定XMLで設定する値である

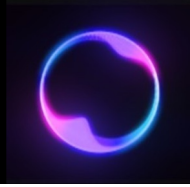
RAの受理/拒絶判断（4/4）



- 以上の検証を全て踏まえて最終的に**RAの受理・拒絶を判断**し、SPは**ISVにmsg4を送信**する

- msg4は、最低限**RAの受理判定**さえ同梱されていれば**フォーマットは何でも良い**
  - 追加で、RA Report内に含まれる事のあるPlatform Info Blob（PIB）や、必要に応じて判定理由の詳細等を同梱すると良い

セッション共通鍵の生成

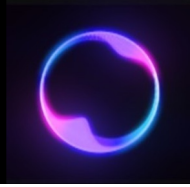


- 無事**RAを受理**したら、SPは**ISVとの暗号通信**（**128bit AES/GCM暗号化**）に使用する、**セッション共通鍵**の**SK**と**MK**を生成する

- SKとMKは、それぞれ以下のバイト列に対しKDKを鍵として生成した128bit AES/CMAC値である：
  - SK：バイト列「¥x01SK¥x00¥x80¥x00」
  - MK：バイト列「¥x01MK¥x00¥x80¥x00」

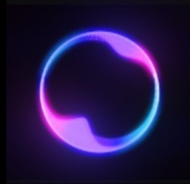
- ISVでは、**sgx_ra_proc_msg2**関数を呼び出し正常に完了した後であれば、Enclave内で**sgx_ra_get_keys**関数を呼び出して**SKとMKを取得する事が出来る**

SPとISV Enclaveとの暗号通信



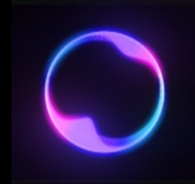
- RA成立後に安全にEnclaveと通信するには、SPは**SKかMK**で**送りたいデータ**を**暗号化**し、**RAコンテキスト**や**暗号など**を**ISVに送信**し、Enclave内に読み込んで**Enclave内で復号**させる
  - デフォルトで提供されているSGXAPIの都合上、暗号方式は**128bit AES/GCM**となる
  - 基本的に**SKやMK**は**Enclave外に出る事はない**。OCALL等で無理矢理出す事も出来るが、そういった**悪性のEnclave**は、**RA前**にMRENCLAVEを得る時点で**目視等で確認**しておく

AES/GCMの特徴



- 平文と暗号文の**バイト長が同じ**である
- 暗号化と復号には**初期化ベクトル**（IV）を用いる
  - **IVは公開情報**であり、普通**12バイト**である事が多い（NISTによる推奨）
- 暗号化すると、暗号文の他に**GCMタグ**（メッセージ認証符号）も出力され、復号時にはこのタグも渡す必要がある
  - タグは**16バイト**であり、**公開情報**である
- 必要に応じて追加認証データ（AAD）を同梱する事も出来る

AES/GCM暗号処理

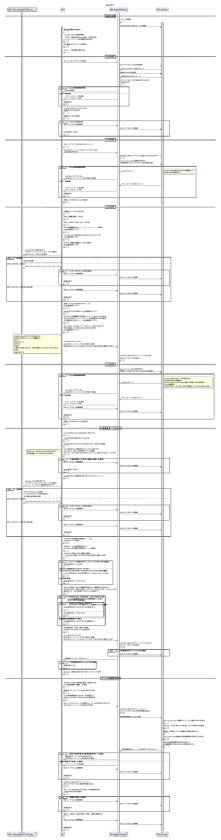


- Enclave内においては、
  AES/GCM暗号化は**sgx_rijndael128GCM_encrypt**関数を、
  復号には**sgx_rijndael128GCM_decrypt**を使用する

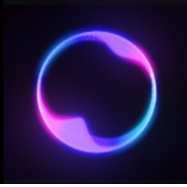
- SPにおいては、**OpenSSL**の適切な関数を用いてAES/GCM暗号化
  及び復号を行う

RAの詳細フロー全体のシーケンス図

- ここまでで説明したRAのフローを厳密に**シーケンス図**に起こすと**以下のようになる**。SVGファイルなので拡大して綺麗に閲覧可



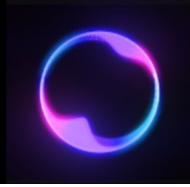
実践・RAの実装



- それでは、**ここまでの内容**を踏まえ、**RAを実行するSPとISVのプログラムを実装**しましょう！

初学者に実装させるには
RAはあまりにも非人道的

Intelのサンプルコードは？

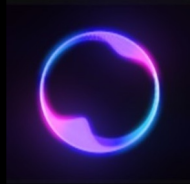


- SGXSDKに同梱されているRAのサンプルコード[10]は、**IASとの通信**やその他様々な部分を**短絡**しているので、**全く役に立たない**

- sgx-ra-sample[11]の方は実際に**完全な形のRA**を**実行可能**である
  - しかし、後のセクションで議論する理由により、**ISV（SGX側）がクライアント**で**SP（非SGX側）がサーバ**である
  - さらに、例えば**通信用関数**（msgio）の**パフォーマンスが極めて悪い**（**数MBの送信に10分かかる**）など、**大幅な改良が必要**
  - しかし、**ありえないコーディング**（**GOTO文を平然と使用**している）、**中途半端なビルド自動化**、**煩雑な構成等**のため、**解読と修正が極めて困難**

- つまり**まともなC++実装のRAのサンプルコード**が**提供されていない**

人道的RAフレームワーク（1/2）

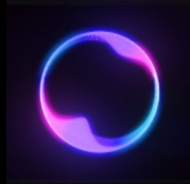


- これでは困るので、講師が自前で**人道的RAフレームワーク（Humane-RAFW）**を実装し提供している

- 基本的に、ISVとSP共に**関数を1つ呼び出す**だけで、**RAが一発で最後まで自動的に進行**する

- 前述の問題を全て解決しており、通信も**比較的モダン**な**httpライブラリ**を用いて**JSON形式**でやり取りするため、**ここまでで説明した地獄を実装する必要は一切ない**

人道的RAフレームワーク（2/2）

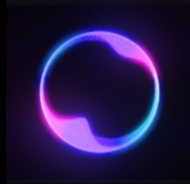


- さらに、RAを行う上で必要な、SPの**署名・検証用キーペア**を生成し、そのまま**ハードコーディング出来る形で出力**する補助ツールも同梱している

- また、ISVのEnclaveイメージから**MRENCLAVE**と**MRSIGNER**を（裏でsgx_signを実行する事によって）取得し表示する補助ツールも用意している

SGX-VaultのSaaS化

SGX-VaultのSaaS化

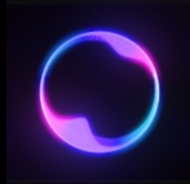


- **Humane-RAFW**をベースとし、これまでに作った**SGX-Vault**を**移植**する事で、SGX-Vaultを**リモートマシンに配置**し**RA後にリモートから利用**できる（SaaSもどき）ように改良する
  - ただし、本ゼミではSPとISVは同一マシン上に存在し、**ローカルホスト通信**で完結する、**擬似的なリモート通信**の形で良い

- Humane-RAFWによるRAに必要な事前準備は全て**リポジトリのREADMEに記載**してあるので参照しながら進める

- httplibやSimpleJSON、Base64エンコードを用いた送受信のコーディング方法は、SP_App/sp_app.cppやISV_App/isv_app.cppが参考になる（はず）

リモート版SGX-Vaultの要件

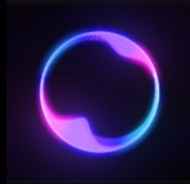


- 基本的な要件はこれまでに実装したSGX-Vaultと全く同じ

- ただし、**ユーザをSP**、**SGXマシンをISV**とし、**リモート**から**各種機能を利用できなければならない**

- 機能の利用に伴う**各通信**は、**セッション共通鍵**である**SK**あるいは**MK**で**保護されていなければならない**

本セクションのまとめ



- SGXが提供する機能の中でも最も難しいものの1つである Attestationについて詳細に解説した

- これを完全に理解及び実装する必要は現時点ではないが、RAを用いてSGXを安全にリモート利用できるようにHumane-RAFWを用いながらSGX-Vaultを改良した

参考文献

[1]”Attestation”, SGX 101, https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation

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

[3]”Code Sample: Intel® Software Guard Extensions Remote Attestation End-to-End Example”, Intel, https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html

[4]”Attestation and Trusted Computing”, J. Christopher Bare, https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/bare.pdf

[5]”SGX Local Attestation 源码分析”, 2023/6/19閲覧, https://ya0guang.com/tech/LocalAttestation/

[6]”What does the "Extended EPID Group ID" mean?”, Intel, https://community.intel.com/t5/Intel-Software-Guard-Extensions/What-does-the-quot-Extended-EPID-Group-ID-quot-mean/td-p/1166244?profile.language=ja

参考文献

[7]”Attestation Service for Intel® Software Guard Extensions (Intel® SGX): API Documentation”, Intel, https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf

[8]”Intel SGX Remote Attestation is not sufficient”, Yogesh Swami, https://eprint.iacr.org/2017/736.pdf

[9]”linux-SGX/psw/ae/qe/quoting_enclave.cpp”, GitHub, https://github.com/intel/linux-sgx/blob/sgx_2.19/psw/ae/qe/quoting_enclave.cpp

[10]”linux-sgx/SampleCode/RemoteAttestation”, GitHub, https://github.com/intel/linux-sgx/tree/master/SampleCode/RemoteAttestation

[11]”Intel® Software Guard Extensions (SGX) Remote Attestation End-to-End Sample for EPID Attestations”, GitHub, https://github.com/intel/sgx-ra-sample