5. Local Attestation

Ao Sakurai

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

本セクションの目標

- SGXの登竜門でもある、2通りのアテステーション（Attestation）の内、Local Attestation（LA）について説明する
- Attestationという概念自体についても簡単に説明を行う
- 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

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**）

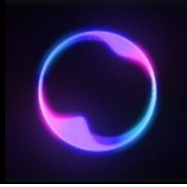
楕円曲線ディフィー・ヘルマン鍵共有

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:

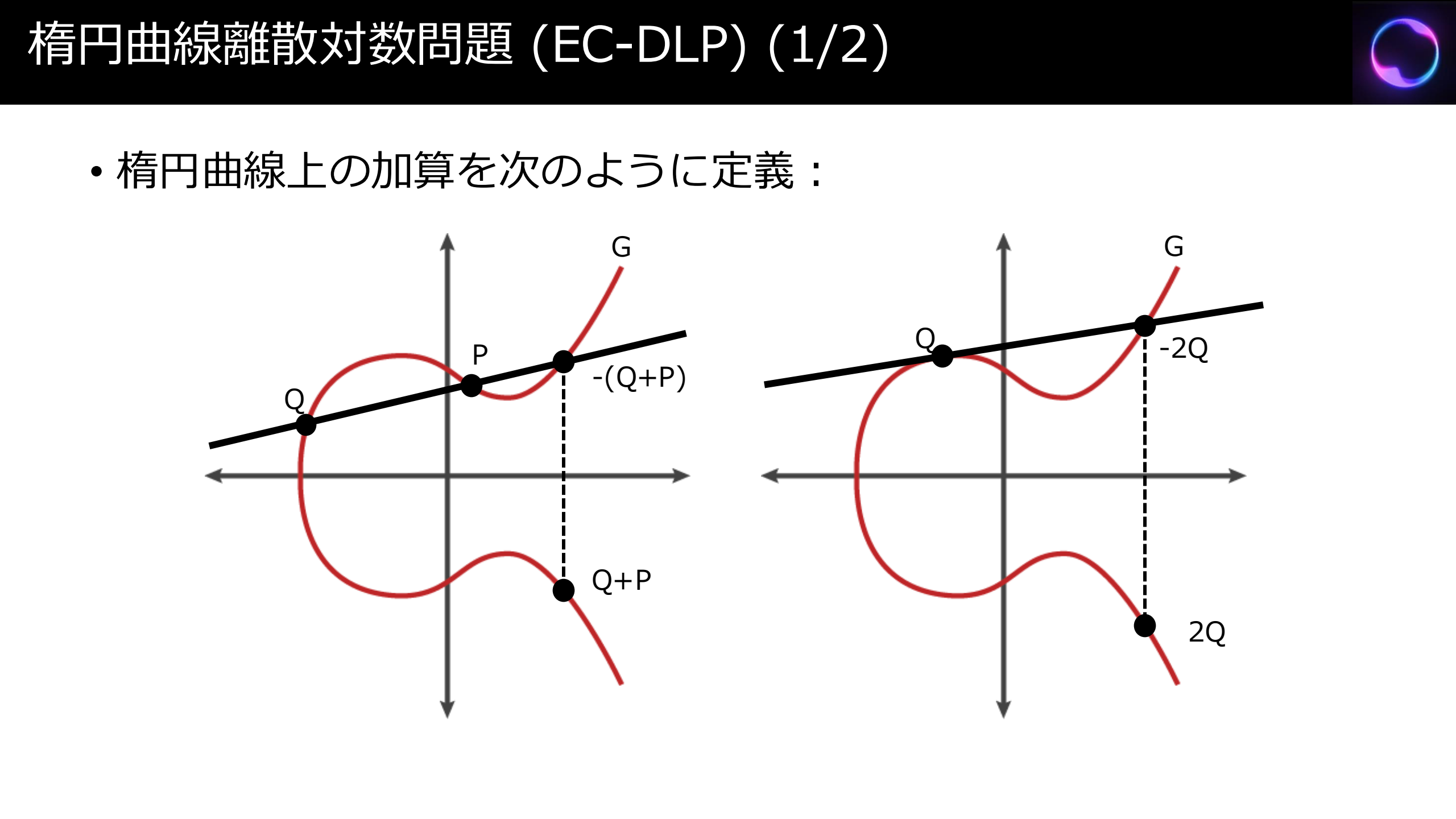
G: 使用する楕円曲線

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

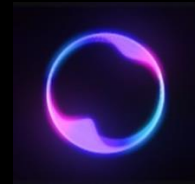
P: nQ

楕円曲線離散対数問題 (EC-DLP) (1/2)

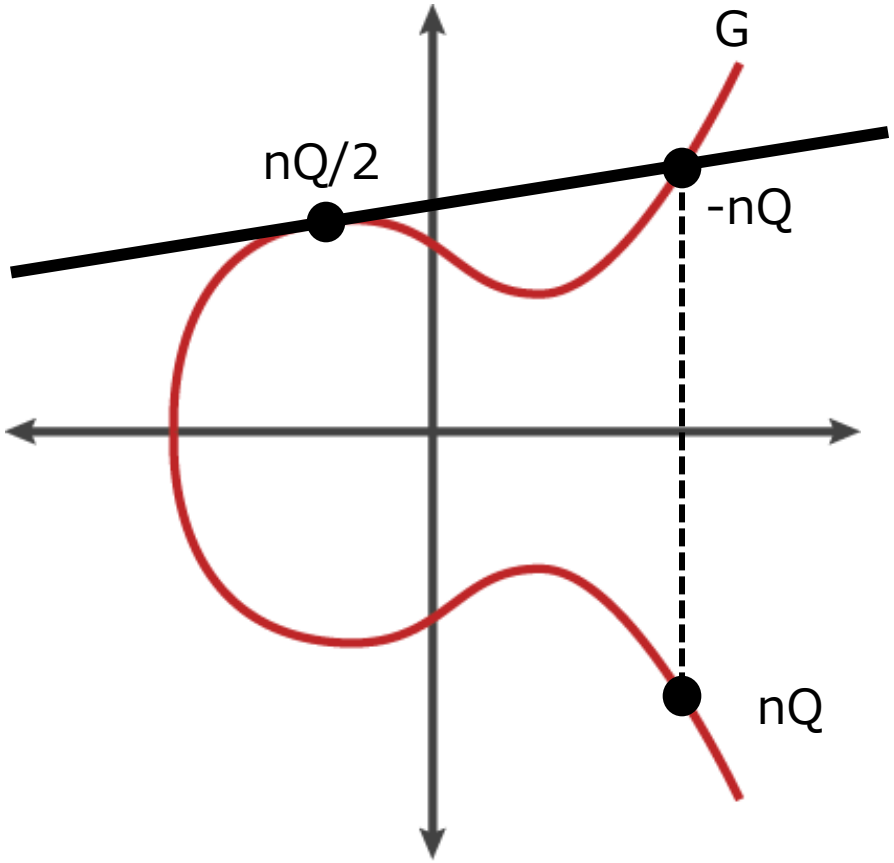


- 楕円曲線上の加算を次のように定義：

楕円曲線離散対数問題 (EC-DLP) (2/2)



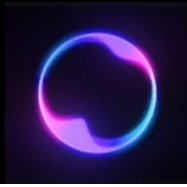
- この例では…



- 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)



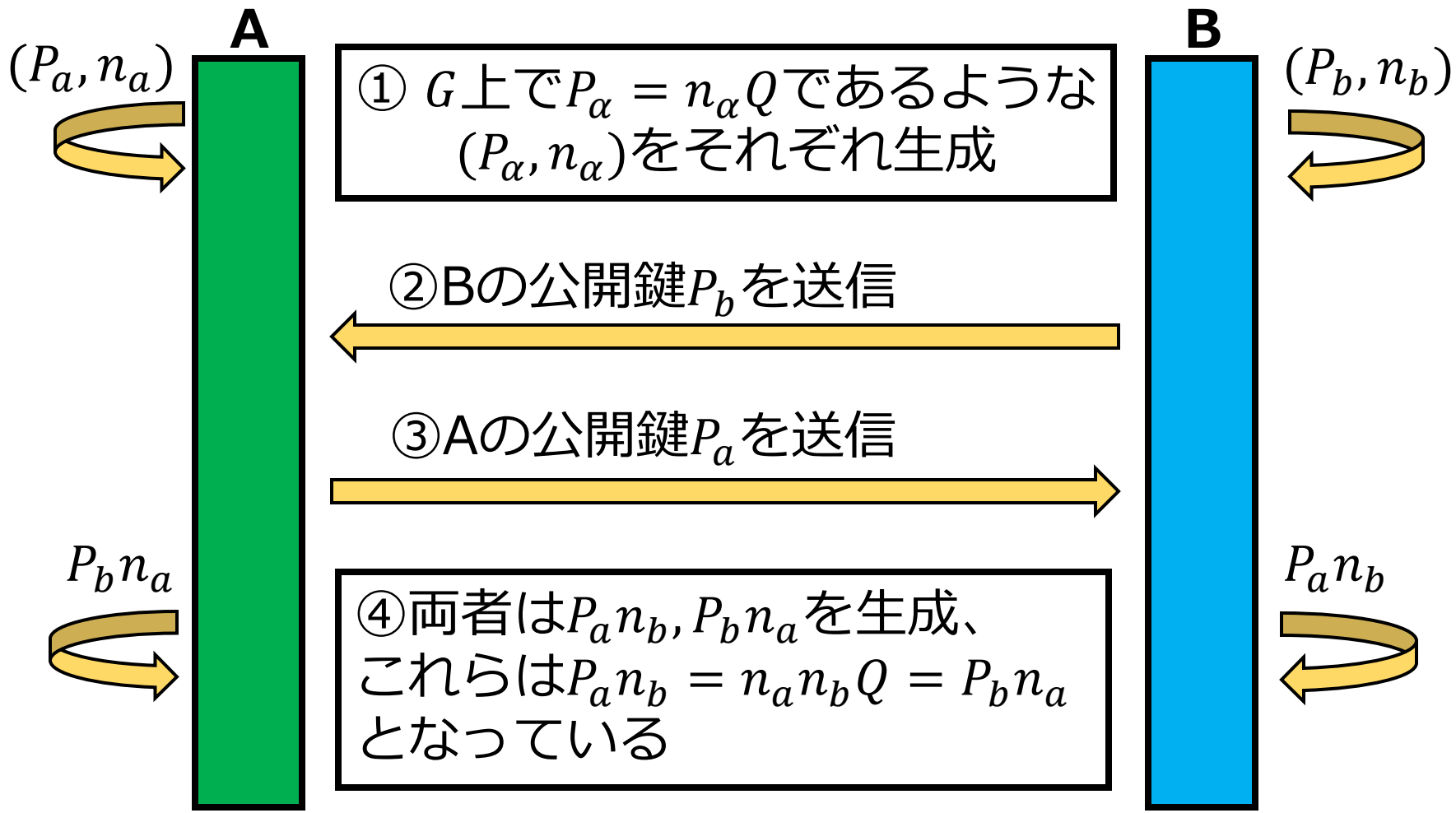
- EC-DHKEは**EC-DLP**を安全性の根拠としている

- 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G_α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2章で述べた通り、SGXコンポネント一式に対するSVN。**TCB Recovery**により変化する
  - **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、基本的には**正体不明**。文献[2]によれば、RPKを主要な導出源として使用し、機密かつ複雑な生成ロジックで導出するらしい

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。KSS（Key Separation and Sharing）機能有効化時に使用可能[7]
sgx_attributes_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。KSS機能有効化時に使用可能[7]

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。KSS機能有効化時に使用可能[7]
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。KSS機能有効化時に使用可能[7]
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

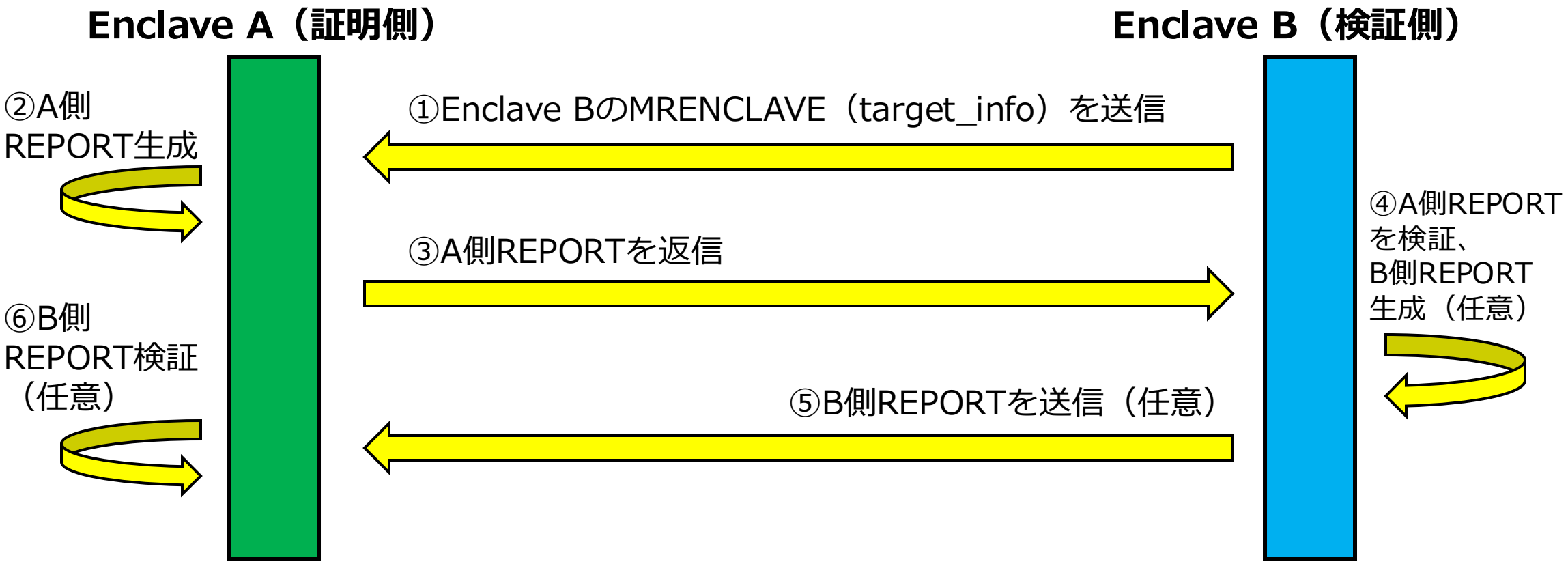
REPORT構造体の実装上の詳細（6/6）

- 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

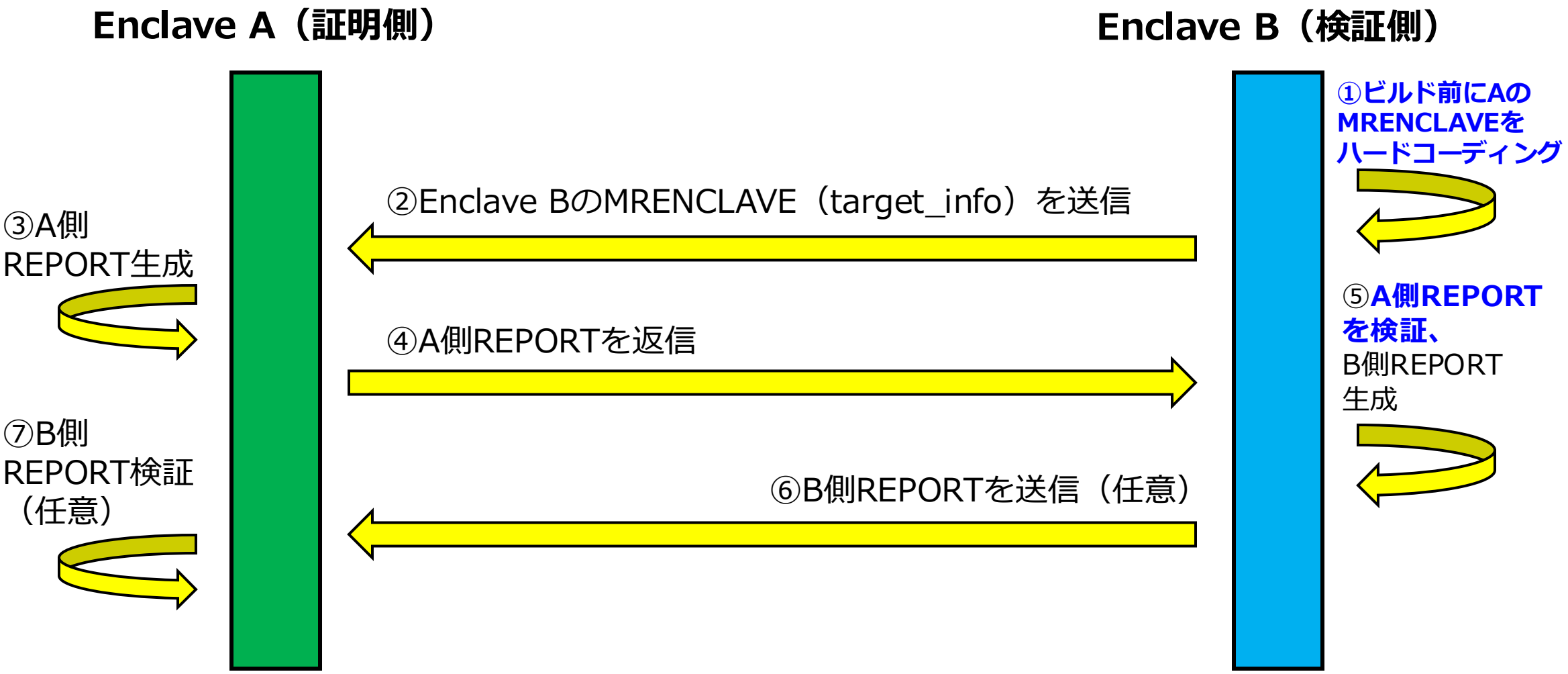
LAの基本形のフロー

- 最低限かつ最も基本的なLAは、以下のフローで進行する



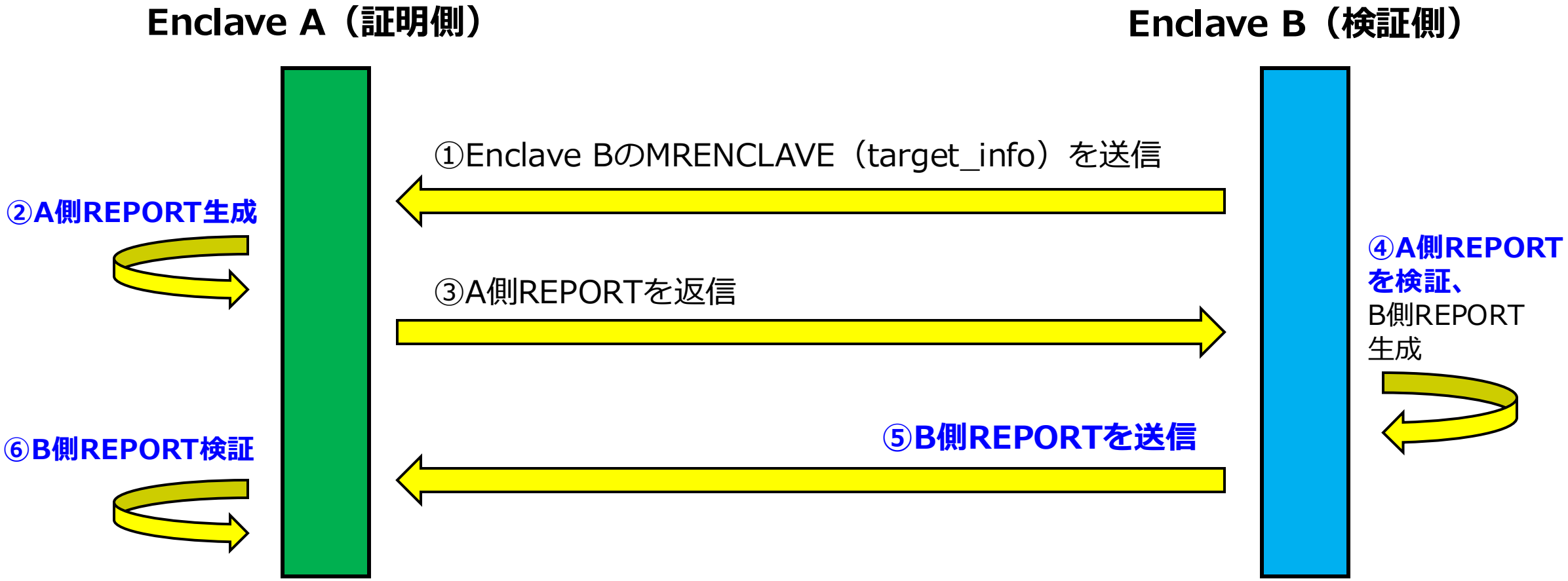
Enclave同一性検証を行うLAのフロー

- 検証側が**証明側のMRENCLAVEを検証**する場合は以下の通り



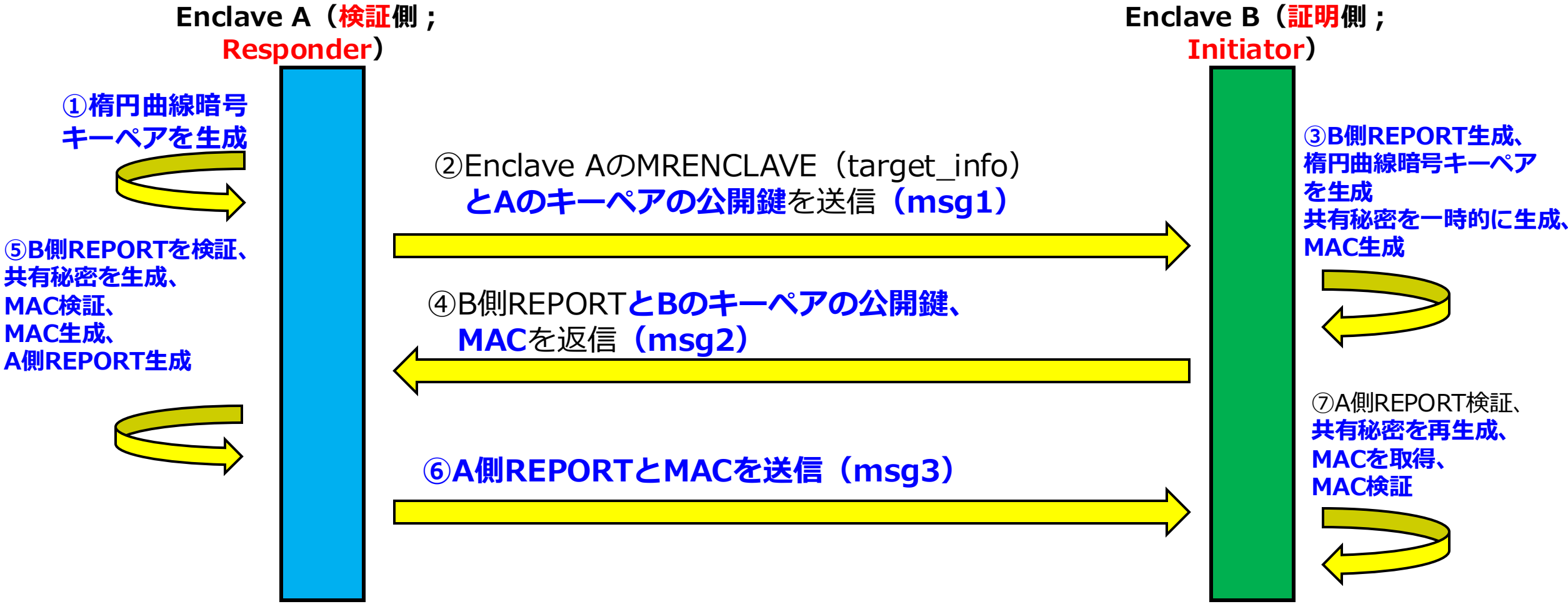
Enclave同一性検証を相互に行うLAのフロー

- 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（1/3）

- 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（2/3）

- 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

証明側 - 検証側REPORTと共有秘密の検証（3/3）

- あるいは、**KSS機能有効化時**に使える**ISVファミリID**や**ISV拡張プロダクトID**は**MRENCLAVEに影響しない**ため、ここに**相手側MRENCLAVE**を入れれば**相互検証できる可能性はある**
  - この2つのフィールドは合わせて32バイトであるため、1つのMRENCLAVEがピッタリハマる

- これらの値は**Enclaveイメージ生成時**に**SIGSTRUCT**構造体の**ボディ**に格納されるため、**Enclave署名鍵で署名**してしまえば**改竄もできない**
  - ただし、その**Enclaveを自身で署名できる利用モデル**の場合に限る

- 講師の**独自考察**な上に**未検証**であるため、**確実な方法ではない**

ちなみに

- これらのKSS値であるが、特にIntel側から用途はほぼ指定されておらず、**使い方の自由度がかなり高い**

- 講師も参画した、同僚主導のIntelとの共同論文では、FaaSのFunctionに対応するEnclave（Worker）の鮮度維持のために、KSSのConfig IDにそのWorkerに一意のIDを含めている[8]
  - Enclave起動時に一度だけセットできる値が必要であったが、その要件を満たすのはこのConfig IDだけであった
  - Gramineを使っているため、講師はGramineにKSSを導入（実装）した[9]

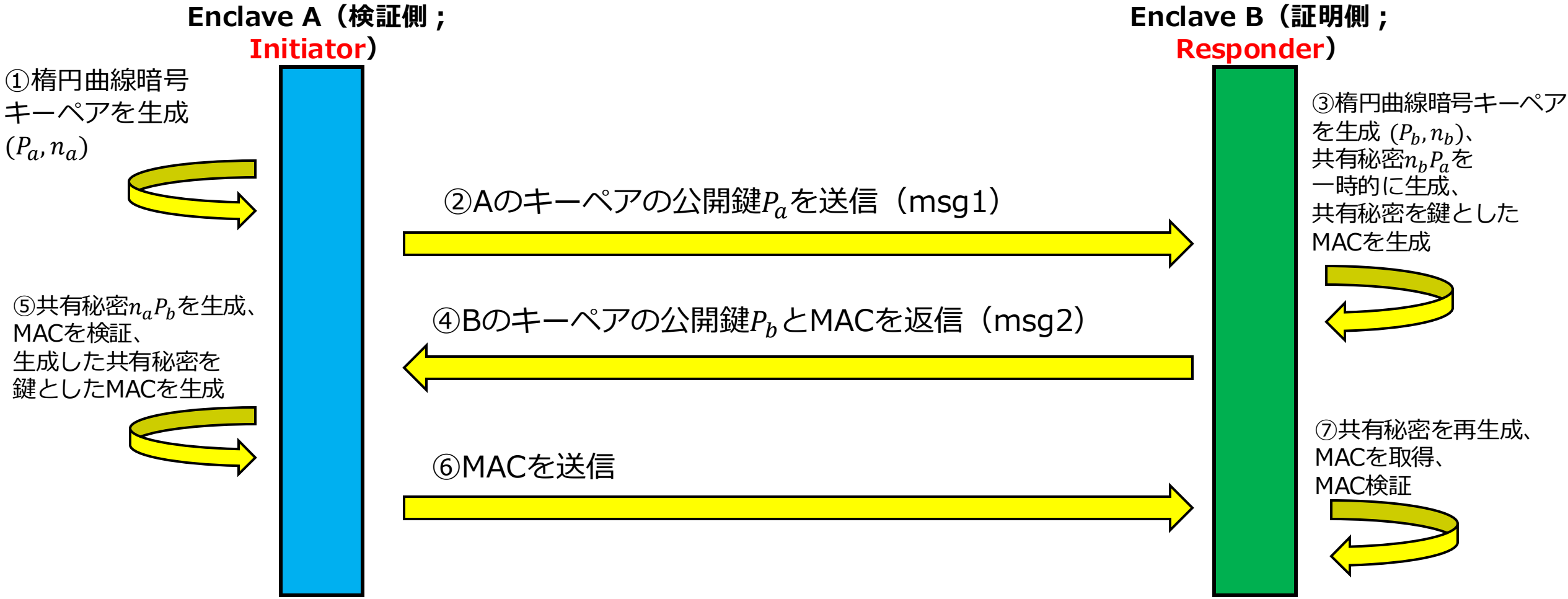
中間者攻撃（1/4）

- 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

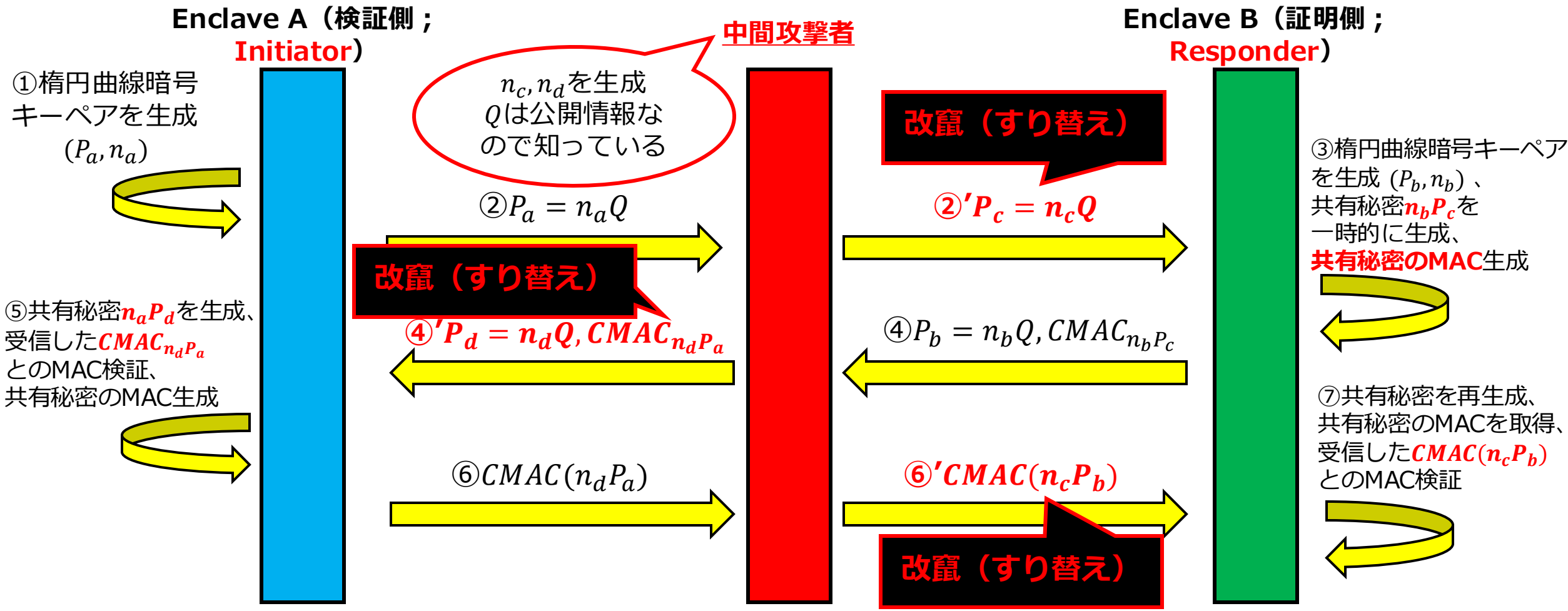
中間者攻撃（2/4）

- 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がAから渡された情報を漏洩させる可能性もある
  - 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

人道的LAフレームワーク

- LAのサンプルコードはLinux-SGXに同梱される形で**Intelによっても提供されている**が、やたら**複雑**であり**解読難易度が高い**
  - https://github.com/intel/linux-sgx/tree/main/SampleCode/LocalAttestation

- よって、より人道的な簡単さでLAを理解しベースとして使用できるフレームワークを、**Humane-LAFW**（**人道的LAフレームワーク**）として講師が開発し**OSSで公開**している
  - https://github.com/acompany-develop/Humane-LAFW
  - LAを行いたいのであれば絶対にこちらの方が手頃である

本セクションのまとめ

- SGXにおいても特に重要な機能の1つであるAttestation機能の内、Local Attestationとそれに付随する技術について説明した

- LAをシステムに組み込むために使用できる人道的なフレームワークとして、Humane-LAFWを紹介した

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

[7]”Introduction to SGX”, Gramine Project, https://gramine.readthedocs.io/en/stable/sgx-intro.html

参考文献

[8]”Securing Nested Attestation of Confidential Serverless Computing without Intra-Enclave Isolation”, Atsuki Momose, Kailun Qin, Ao Sakurai and Mona Vij, https://eprint.iacr.org/2025/727.pdf

[9]”Added KSS features support including Config ID/SVN”, GitHub, https://github.com/gramineproject/gramine/pull/2101/commits