6. EPID Remote Attestation

Ao Sakurai

2024年度セキュリティキャンプ全国大会
S3 – TEEビルド&スクラップゼミ

本セクションの目標

- SGXにおいて最も難解であるRemote Attestation（RA）の内、EPID形式と呼ばれるタイプのRAについて解説する

- RAの本処理だけでなく、RAに必要な秘密であるAttestationキーをデプロイする「プロビジョニング」処理についても説明する

Provisioning（プロビジョニング）

用語集

用語	説明
Attestationキー	主にRemote Attestationにおいて非常に重要な役割を果たす鍵で、Provisioningにより生成される。 その実はEPIDメンバ秘密鍵。
Provisioning Enclave (PvE)	Provisioning処理において中核的な役割を果たす。 Architectural Enclaveの一つ。
Quoting Enclave（QE）	PvEが生成しストアしたAttestationキーをロードし、RAにおいてQUOTE構造体を作成する。 Architectural Enclaveの一つ。
Intel Provisioning Service (IPS)	Provisioning処理において、PvEとやり取りを行うIntel側のサービス。
Intel Key Generation Facility (iKGF)	RPK（後述）やEPID関連鍵等、様々な鍵を作成しストアする、Intelの鍵作成管理施設。 インターネットからは接続できない場所に隔離され、強固に保護されている。

Intel EPID（Enhanced Privacy ID）

- ある**（1つの）グループ**に対し**複数のメンバ**を**匿名の状態**で対応させる事が出来るスキーム

- **直接匿名認証**（Direct Anonymous Attestation; **DAA**）の**応用的な実装例**である

- EPIDの**メンバ秘密鍵**で署名すると、EPIDの**グループ公開鍵**を用いて、**署名者を特定する事なく検証**できる
  - 例：あるマシンが特定のCPUグループに属しているかを匿名のまま検証

Provisioningの概要（1/2）

- SGXマシンが**正当なCPUやSWを搭載**しているかを確認した後、ある**EPIDグループのメンバとして加入**させ、**Attestationキーを獲得**させる処理
  - Intel CPUにおけるEPIDグループは、CPUの種類（Core i3, i5, i7）と、セキュリティバージョン番号（SVN）が同一であるような**数百万個のCPUをカバー**している

- Provisioning処理は、SGXマシン側では**Architectural Enclave**（AE）の1つである**Provisioning Enclave（PvE）**が中心となる

- Intel側は**Intel Provisioning Service（IPS）**が中心となる

Provisioningの概要（2/2）

- Provisioning処理は、その**マシンの初使用時**の他、購入後にファームウェア、BIOS、マイクロコード等の**重要なシステムコンポネントが更新**された際にも実行される（**TCB Recovery**）
  - 初使用時とは、厳密には初めてQUOTE構造体の生成を要求した時のようである[17]

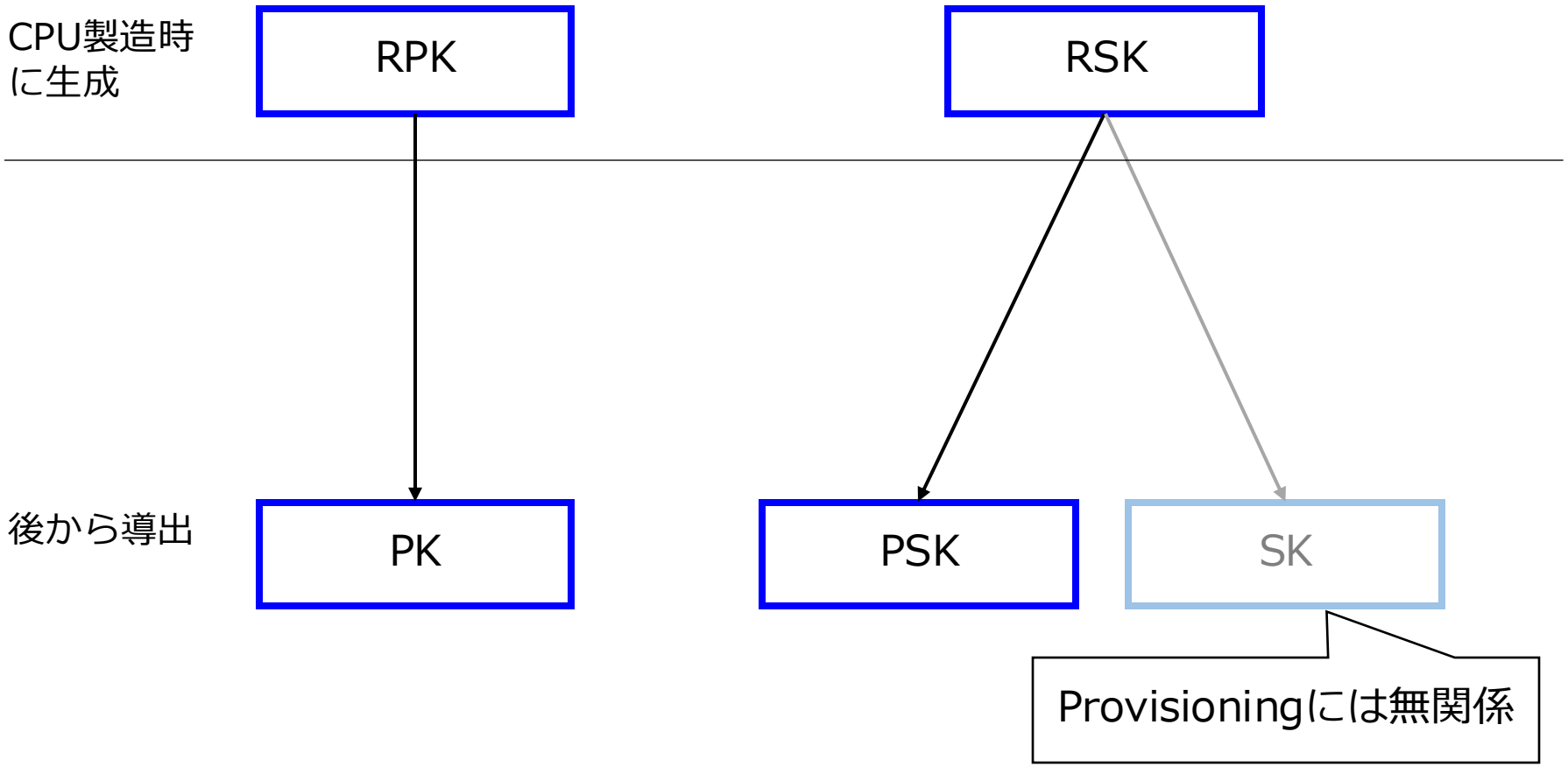
Provisioningに使用する鍵一覧（1/2）

- **Intelによる命名**が**非常に紛らわしい**が、以下のように整理する事が出来る

鍵名	概要
Root Provisioning Key（RPK; またはProvisioning Secret）	CPU製造時に各CPUのe-fuseに焼き付けられる秘密情報。**この値はIntel側もiKGFで管理・保持している**。
Root Seal Key（RSK; またはSeal Secret）	CPU製造時に各CPU内で乱数的に生成され、e-fuseに格納される秘密情報。**この値はIntel側も保持・把握していない**。
Provisioning Key（PK）	RPKから導出されるプロビジョニング鍵。Provisioningの手続きで使用される。
Provisioning Seal Key（PSK）	RSKから導出される、Provisioning手続き上で必要なシーリングを行う為のシーリング鍵。

（参考）Seal Key：通常のシーリングに使用される鍵。これもRSKからポリシ（MRENCLAVE、MRSIGNER）に応じて生成されるが、PSKとは違いOWNEREPOCHという値を有する[2]等の違いがある。

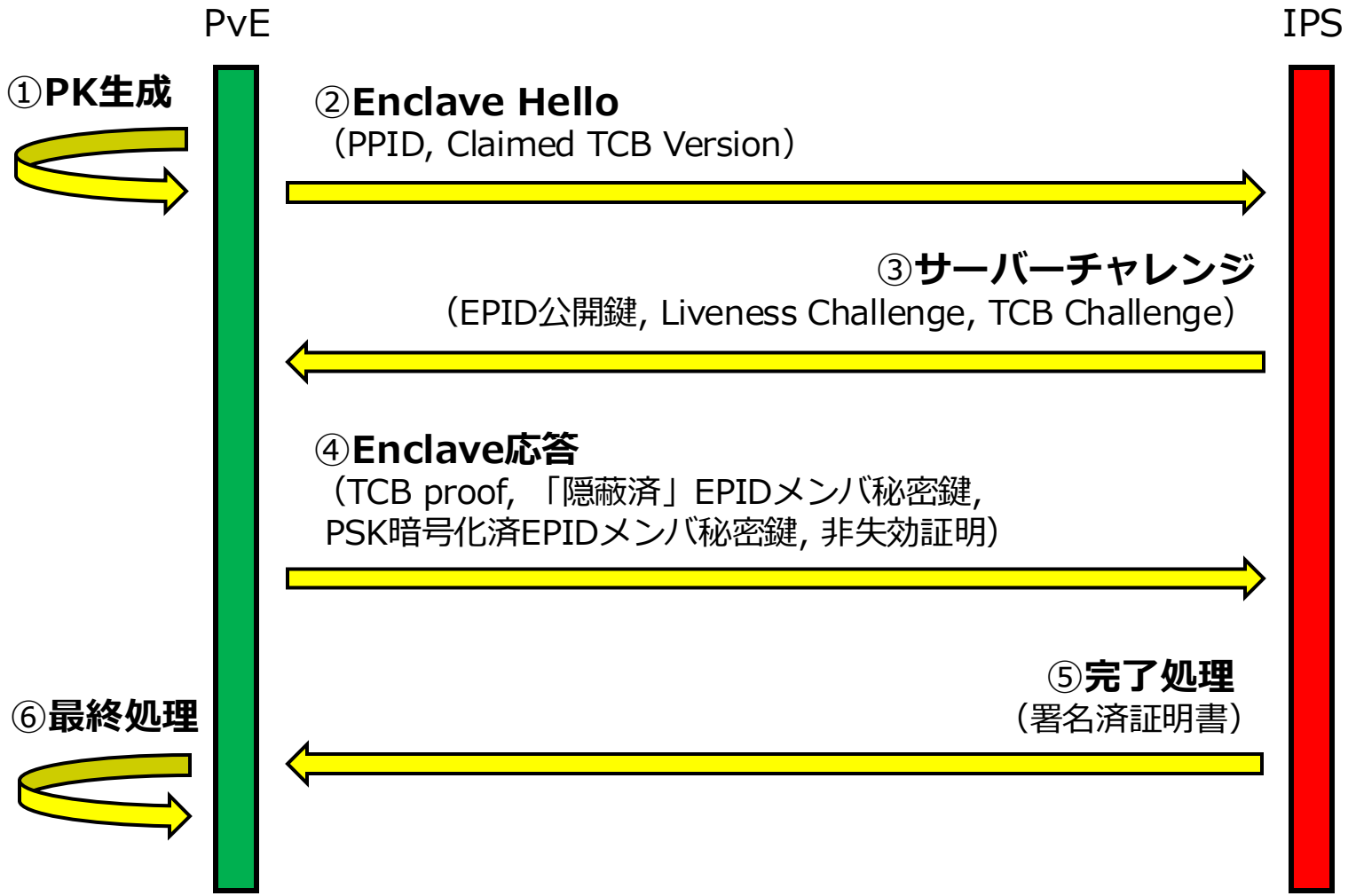
Provisioningに使用する鍵一覧（2/2）



※厳密には、PSKやSKの導出の際、RPKを入力として導出しているらしいSGXマスター導出鍵を使用しているが、この図では省略している。

Provisioningフロー

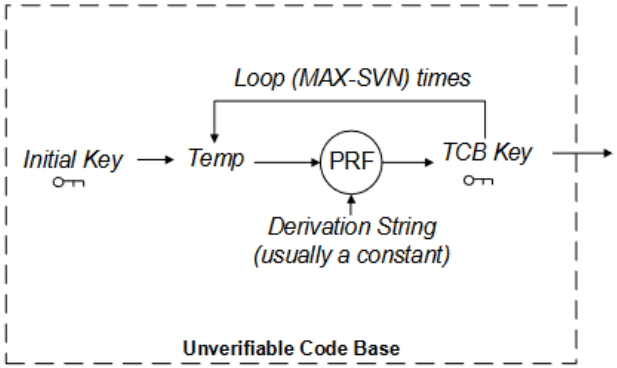
- Provisioningフローを図示すると以下のようになる：



PKの作成

- 簡単に言えば、RPKからHW・SWそれぞれの特定のコンテキストを付与してPKを導出する

RPKのHW-TCBとのバインド：
プロセッサ内で生成されるInitial Key（RPK）をTCBのSVN（Security Version Number）回だけPRF（擬似ランダム関数）に通し、TCB鍵を生成する。恐らくマイクロコードアップデート等に伴いCPU内部で自動実行される。



SW情報の付与：
EGETKEY命令を発行し、上記TCB鍵をベースとして各種SW情報（例：CPUSVN、MRSIGNER）を付与し、これをPKとする

Enclave Hello

- PvEは、以下2つの値を生成する。
  - **PPID**：PKのハッシュ値（128bit AES-CMAC）[18]
  - **Claimed TCB version**：TCBSVNから導出される値

- 上記2つの値を生成後、**IPSの公開鍵**で双方ともに暗号化し、Enclave HelloとしてIPSに送信する。
  - この「**IPSの公開鍵**」が何物なのかは**文献からでは解読不能**だが、気合で探した所**以下のソースコードにハードコーディング**されている事が判明：
    https://github.com/intel/linux-sgx/blob/1efe23c20e37f868498f8287921eedfbcecdc216/psw/ae/data/constants/linux/peksk_pub.hh

サーバーチャレンジ

- IPSは、PvEから送信された**PPIDで検索を実行**し、そのPvEの
  プラットフォームが**以前にProvisioningされた事があるか**を**確認**する

- 確認結果に応じて以下のように**サーバーチャレンジ**をPvEに返送する：

Provisioningが初であった場合
PvEのプラットフォームを**加入させるEPIDグループを決定**後、以下の情報を
PvEに返信する：

- EPIDグループパラメータ（EPID公開鍵）
- Liveness Challenge
- iKGFで予め作成済みのTCBチャレンジ

Provisioning履歴があった場合

上記に加え、以前生成したAttestationキーをPSKで暗号化したものを
TCBチャレンジに追加し返信する

Enclave応答（1/5）

- サーバーチャレンジをIPSから受け取ったら、PvEは自身の**プラットフォームの正当性**を証明するために、**以下の4つの値**を準備する：
  - TCB proof
  - 「数学的に隠蔽」されたEPIDメンバ秘密鍵
  - PSKによって暗号化済のEPIDメンバ秘密鍵
  - 非失効証明

Enclave応答（2/5）- TCB proof

- 以下の2通りの方法のいずれかで、TCBが正当である事を証明する（**TCB proof**の作成）。sgx101のサイトでは後者のみを記載している

方法①
PvEは、iKGFでPKを用いて作成された暗号化済乱数（Liveness Challange）を、自身の持つPKで復号する。
ちなみに、**iKGFが全てのRPKを有している**事から、IPSは**各PKを容易に再現して生成可能**であるらしい（詳細は不明）。

方法②
PvEは、受信したTCBチャレンジをPKで復号し、その復号済TCBチャレンジを鍵として、Liveness ChallengeのCMACを生成する。

Enclave応答（3/5）- EPIDメンバ秘密鍵

- その後、PvEは**EPIDメンバ秘密鍵**（＝**Attestationキー**）を作成し、EPIDのプロトコルに従って「数学的に隠蔽」する
  - 「数学的に隠蔽」する方法の詳細は不明だが、**何らかの群論的な操作**をするのではないかと産総研でのレクチャ時に候補として上がった

- それとは別に、このEPIDメンバ秘密鍵を**PSKで暗号化**する

- このEPIDメンバ秘密鍵は**全てPvE内で生成処理が完結する**ため、**Intel側がこの値を知る事はできない**
  - Intel SGX Explained[5]では**あたかもIPSがこの鍵を送信している**かのような図を載せているが、**これは誤り**

Enclave応答（4/5）- 非失効証明

- 過去にProvisioningを行った事がある（=**再Provisioningである**）場合、そのPvEのプラットフォームが過去に一度も**失効**（Revoke）していない事を**証明**する必要がある
  - この失効は後のセクションで取り扱うRAにおける各種失効とは異なり、永続的にプラットフォーム単位で失効させる処理のようである[19]

- 具体的には、サーバから取得したバックアップ済Attestationキーのコピーを PSKで復号し、それを使ってIntelが選択したメッセージに署名する事で証明する

Enclave応答（5/5）

- PvEは、作成した以下のデータをIPSに送信する。
  - TCB proof
  - 「数学的に隠蔽」されたEPIDメンバ秘密鍵
  - PSKによって暗号化済のEPIDメンバ秘密鍵
  - 非失効証明

完了処理

- IPSはPvEからのEnclave応答中の**TCB proof**を、iKGFから取得した値を用いて**検証**する。
- 検証の結果成功である場合、続いて対象PvEのプラットフォームを**EPIDグループに加入**させる。
- IPSは、応答中の「数学的に隠蔽された」EPIDメンバ秘密鍵と、IPSの持つEPIDグループ発行者鍵を用いて、**署名済証明書**を作成する。
- IPSは、上記で作成した署名済証明書を同梱したメッセージをPvEに返送する（Provisioningの完了）。

最終処理

- 「数学的に隠蔽された」EPIDメンバ秘密鍵と、PSK暗号化済メンバ秘密鍵は、今後の再Provisioningの為IPS側に保存される
  - 非失効証明の作成時に参照されたバックアップ済Attestationキーは、まさに過去に保存されたこれである

- PvEは、AttestationキーをPSKでシーリングし、プラットフォーム上に保存する
  - RA時、Quoting Enclave（QE）はここから持ってくる
  - PSKは**OWNEREPOCH値を持たず**、かつこのシーリングは**MRSIGNERポリシ**である（PSKを使用した場合の仕様）為、**QEでも問題なくアンシーリング**出来る

EPID Remote Attestation（EPID-RA）

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

- RAにはEPID-RA（本章で説明）とDCAP-RA（次章で説明）の2つが存在し、EPID-RAの方が先発（より古い）である
  - 両者の違いについては次章において説明する

- 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

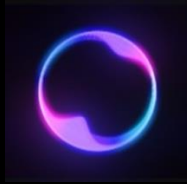
Remote Attestationの概要（2/7）

- 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機能を利用する**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）

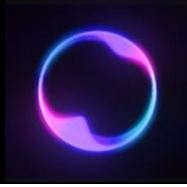


- RAは実装は非常に面倒臭いが、その根源的な目的自体は単純である：

[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**真正性や同一性をリモートから検証する**

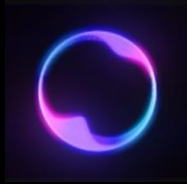
Remote Attestationの概要（4/7）



- 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）

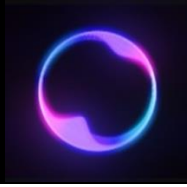


- その後、SPはISVから受け取った**QUOTE**を、**第三者検証機関**である**Intel Attestation Service**（**IAS**）に送信する

- **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）

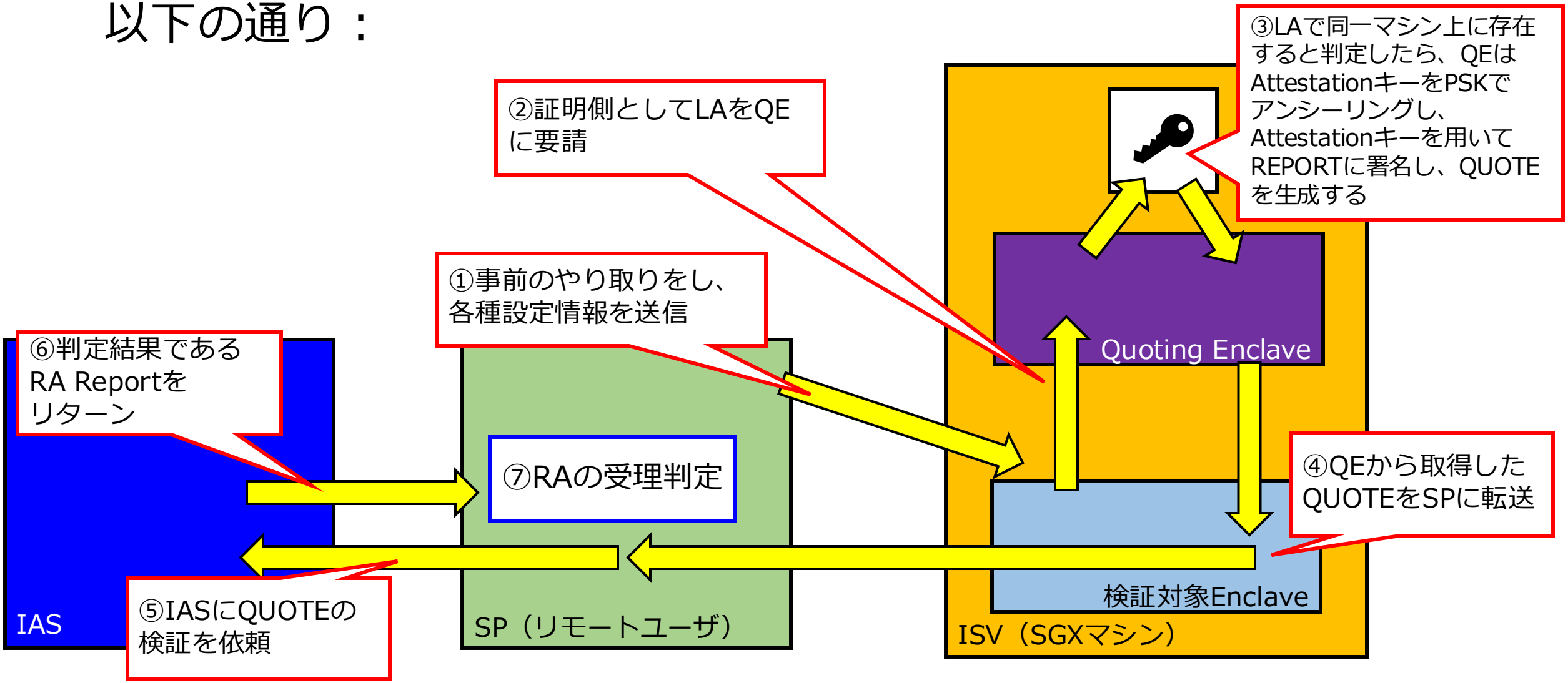


- LA同様、RAを進める上でRA受理後の**通信に用いる共通鍵**の交換を**EC-DHKE**を用いて実施する

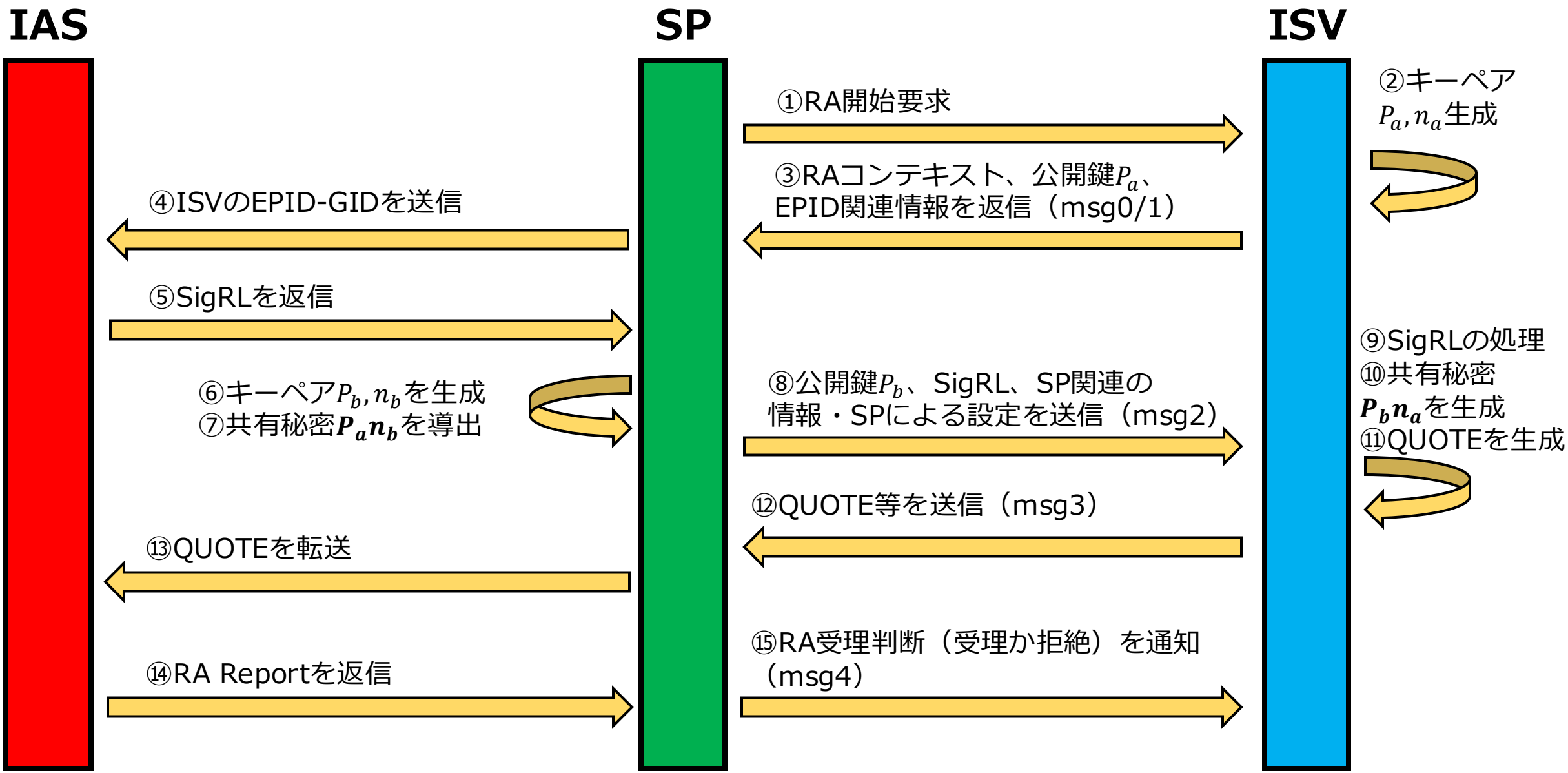
- 相手のEnclaveが本当に意図する通りのEnclaveであるのかの**同一性検証**は、**予め取得した**ISVのEnclaveの**MRENCLAVE**を、**QUOTE内のREPORT内**に含まれる**MRENCLAVE**と比較する事で**SPが行う**
  - RAでは、**SP側の環境**は**完全に安全**であるという前提で**脅威モデルが設定される**事に注意

Remote Attestationの概要（7/7）

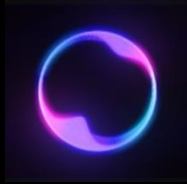
- RAについて、LAからの拡張に着目した場合の概要図は以下の通り：



Remote Attestationの全体フロー



RAを行うにあたって（1/3）

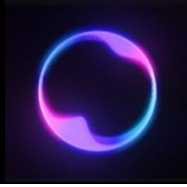


- RAを実行するには、Intelの**EPID Attestation**の**サービスに登録**する必要がある

- 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

- ただし、今年度のゼミでは次章で紹介する**DCAP-RAを使用する**ため、この作業の**必要はない**
  - なぜDCAP-RAに切り替えたのかは次章で種明かしする

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.

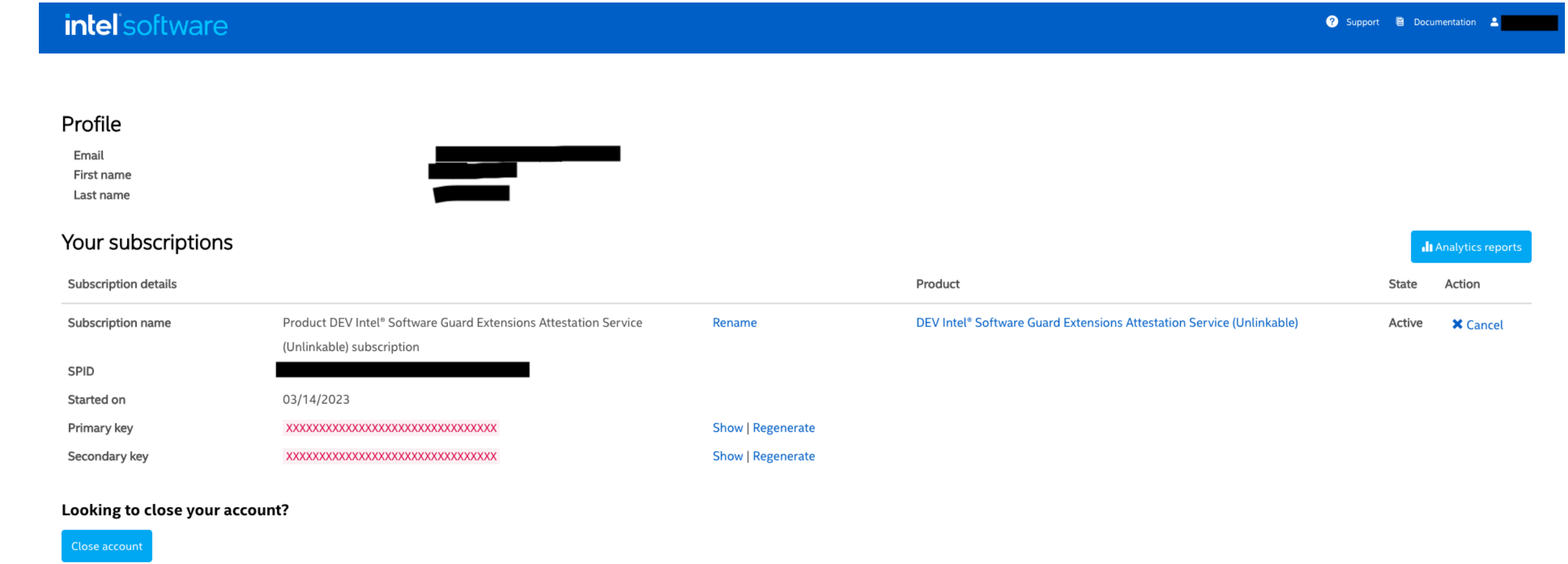
Subscribe (linkable)

Subscribe (unlinkable)

Production Access

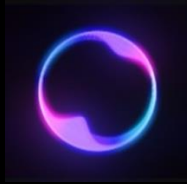
Once a commercial use license has been executed and your application/solution has been added to the Launch Policy List (if applicable), you just need to Subscribe for production access. Once your subscription is activated you will be able to utilize the production version of the Intel SGX attestation 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）



EPID-RA詳説

RAフロー詳説

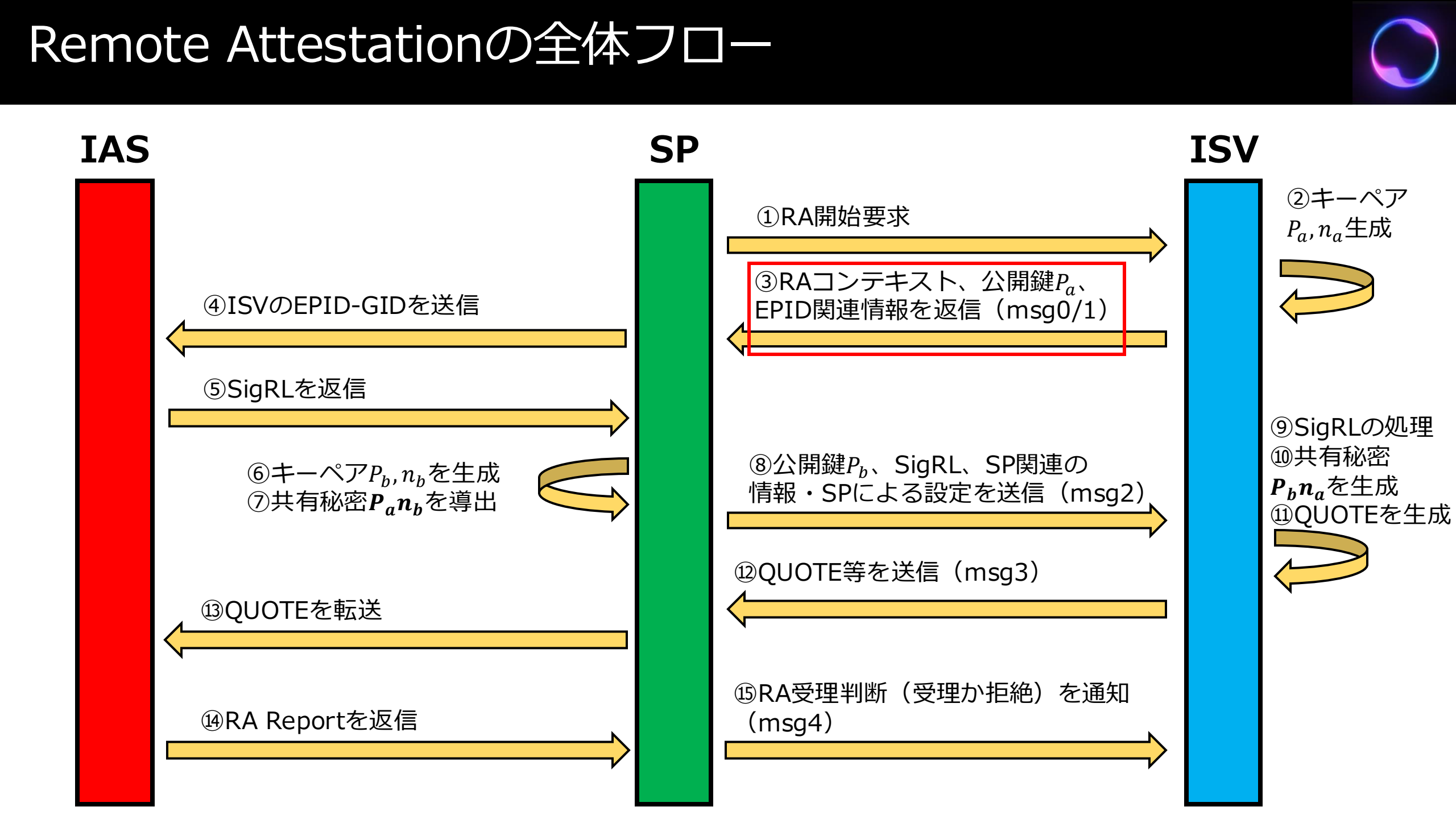


- ここからは、具体的にどのようにRAの処理を進行させていくかを詳説していく

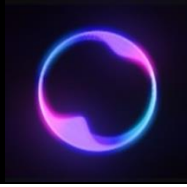
- **RAが成立する**（SPがRAを受理し、msg4をISVが受け取る）までは**通信路は保護されない**点もLAと同様

- 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**

Remote Attestationの全体フロー



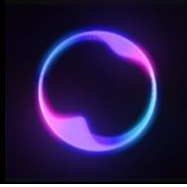
RA開始要求・msg0の送信（1/2）



- 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

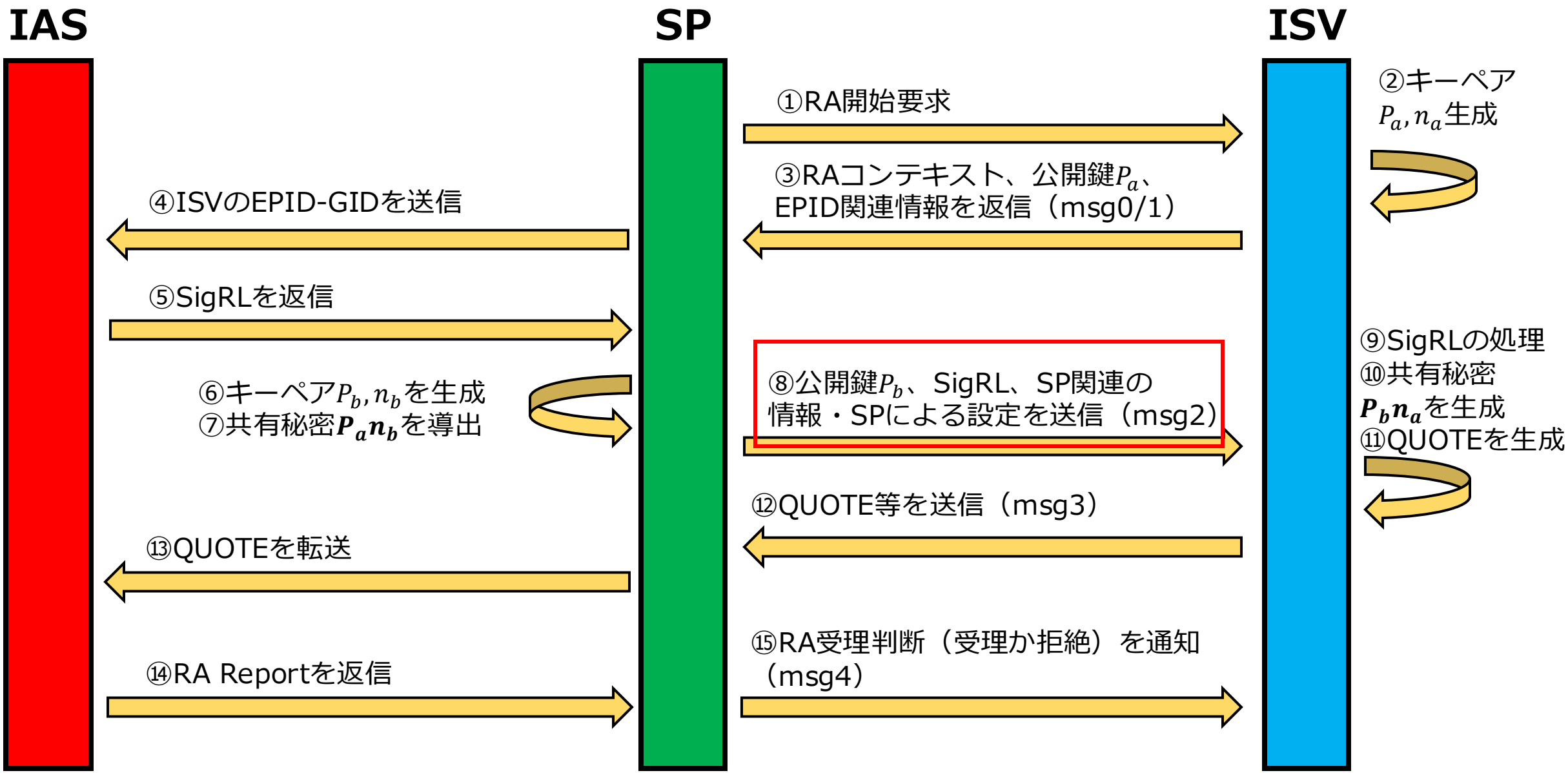
SigRLの取得

- 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**

Remote Attestationの全体フロー



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

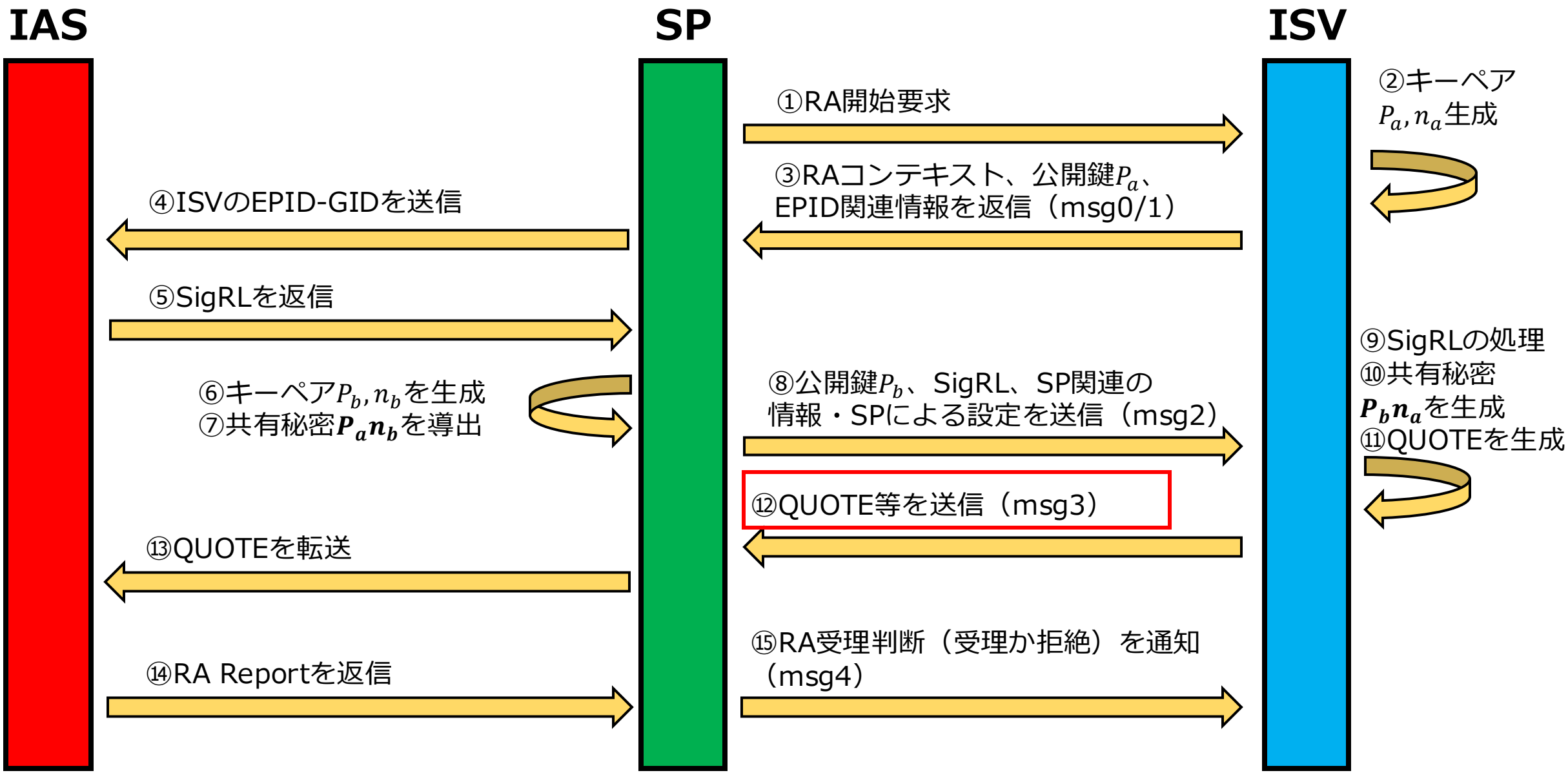
- msg2を完成させたら、SPはその**msg2をISVに送信**する

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

Remote Attestationの全体フロー



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の公開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キーの漏洩は確認できないが、**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

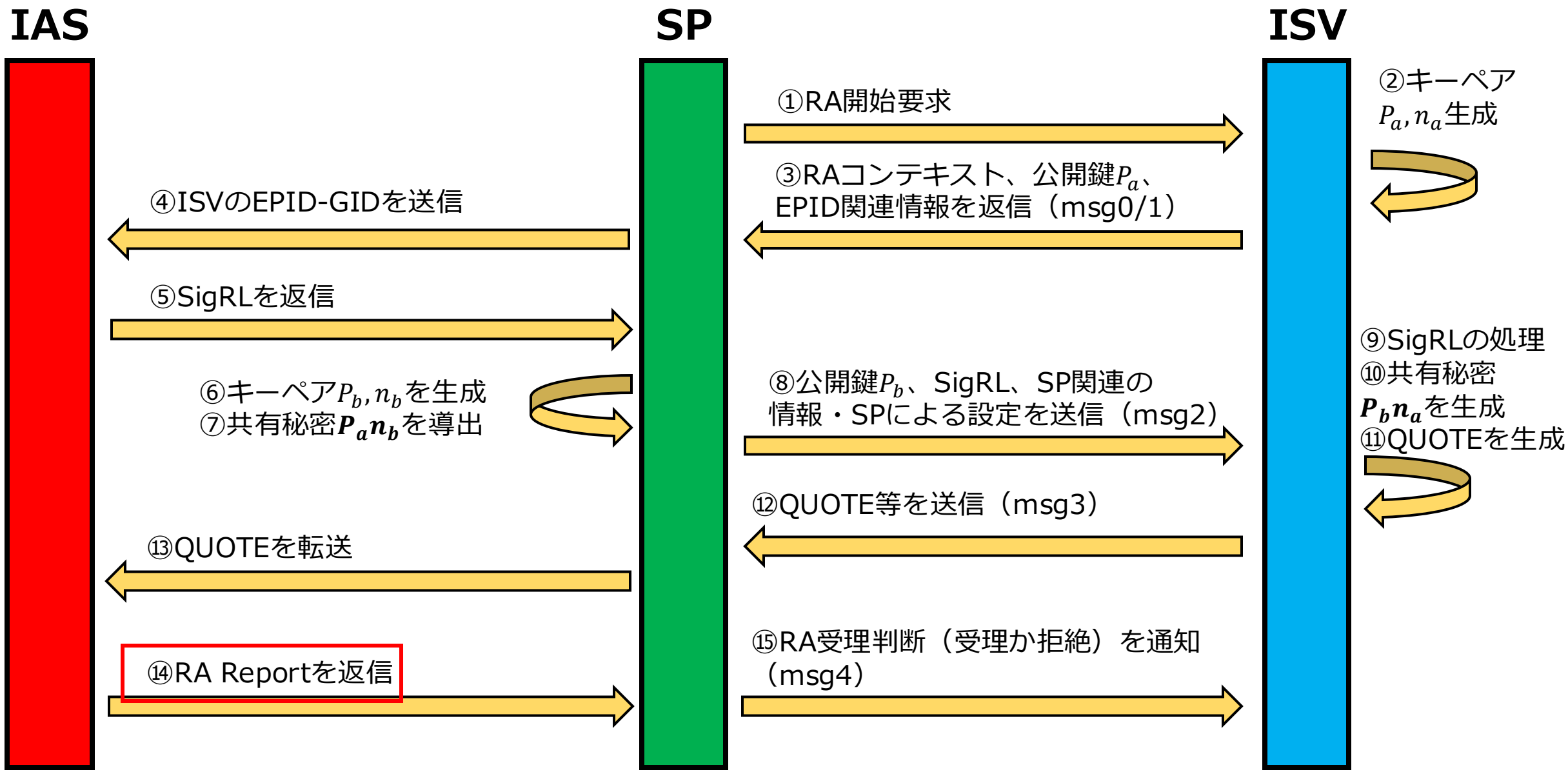
msg3のメンバ	説明
sgx_mac_t mac	以下3つの連結に対する、SMKを鍵とした128bit AES/CMAC値（Enclave内で計算される）
sgx_ec256_public_t g_a	ISVのセッション公開鍵G_a
sgx_ps_sec_prop_desc_t ps_sec_prop	PSE（Linuxでは使用不可になっている例のAE）を使用する際の追加情報。 PSE不使用時は0で埋めておく
uint8_t quote[]	QUOTE構造体（sgx_quote_t）

- 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

Remote Attestationの全体フロー



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

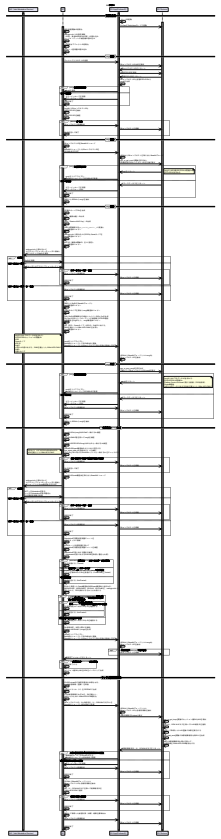
AES/GCM暗号処理

- 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
  https://github.com/acompany-develop/Humane-RAFW

- 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、 EPID-RAを用いてSGXを比較的簡単かつ安全にリモート利用できる ようにHumane-RAFW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

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

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

参考文献

[13] Intel® Software Guard Extensions:
EPID Provisioning and Attestation Services (https://cdrdv2.intel.com/v1/dl/getContent/671370)

[14] Intel® Enhanced Privacy ID (EPID) Security Technology
(https://www.intel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html)

[15] Intel® Software Guard Extensions Trusted Computing Base Recovery
(https://community.intel.com/legacyfs/online/drupal_files/managed/01/7b/Intel-SGX-Trusted-Computing-Base-Recovery.pdf)

[16]”Intel® Software Guard Extensions Trusted Computing Base Recovery”, Intel,
https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/resources/intel-sgx-software-and-tcb-recovery-guidance.html

[17]qe_logic.cpp, GitHub, https://github.com/intel/linux-sgx/blob/sgx_2.23/psw/ae/aesm_service/source/bundles/epid_quote_service_bundle/qe_logic.cpp#L253-L258

参考文献

[18] linux-sgx/psw/ae/pve/helper.cpp, Linux-SGX 2.23, https://github.com/intel/linux-sgx/blob/sgx_2.23/psw/ae/pve/helper.cpp#L104

[19] Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives, Vinnie Scarlata et al., https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf