Encrypted Client Hello

栗原頂

兵庫県立大学大学院

はじめに

はじめに

この資料は、ECH(Encrypted Client Hello) についてまとめた資料。この技術が生まれた経緯・目的・技術 (現状の標準化) について説明する。

ECHが考えられた経緯

TLS (Transport Layer Security)

クライアント (Web ブラウザ) とサーバ (Web サーバ) が http 通信を安全に行うためのセキュリティプロトコル

TLS の目的

- Web サイトの認証
- クライアントとサーバ間の http 通信の暗号化 (https の実現)

https を実現するためのセキュリティプロトコルはもともとは Netscape Communications 社で開発された SSL だった

↓

開発の取り組みを IETF へ移管して SSL の改良を重ねたのち TLS が開発された

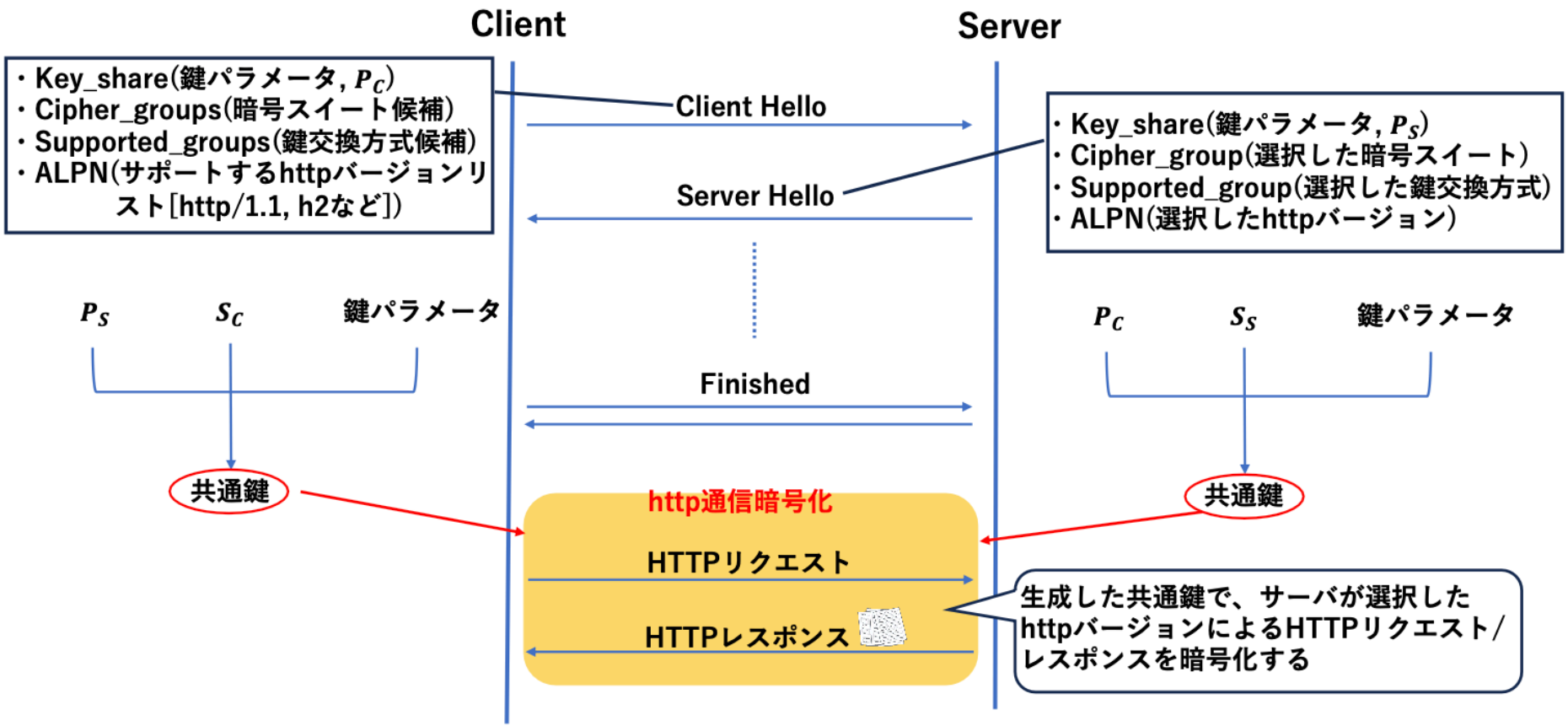
Table: セキュリティプロトコルの歴史

リリース年	プロトコル version	利用状況
-	SSL1.0	非公開
1994	SSL2.0	禁止
1995	SSL3.0	禁止
1999	TLS1.0	無効化予定
2006	TLS1.1 [RFC2246]	無効化予定
2008	TLS1.2 [RFC5246]	◯
2018	TLS1.3 [RFC8446]	◯

TLS1.3

鍵交換方式 (DHE, ECDHE) によりクライアントとサーバ間で共通鍵をそれぞれ生成し、その鍵を用いて http 通信を暗号化する

※鍵交換は Client Hello/Server Hello の時に共通鍵の生成に必要なパラメータを送っている



TLS1.3 以前

- 鍵交換は Client Hello/Server Hello の後に
  Client Key Exchange/Server Key Exchange に鍵パラメータを含んで行われていた
- 脆弱性のある古い暗号アルゴリズムも対応していた (RC4, CBC ブロック暗号など...)

TLS1.3

- TLS1.3 では、Client Hello/Server Hello に key_share 拡張を含んで鍵パラメータを送っている
  →TLS1.3 では暗号化通信開始までのネゴシエーションが一往復減っている
- 脆弱性のある暗号アルゴリズムは廃止し、AEAD が要求されるようになった
- Ephemeral な鍵交換方式に統一 (DHE, ECDHE)

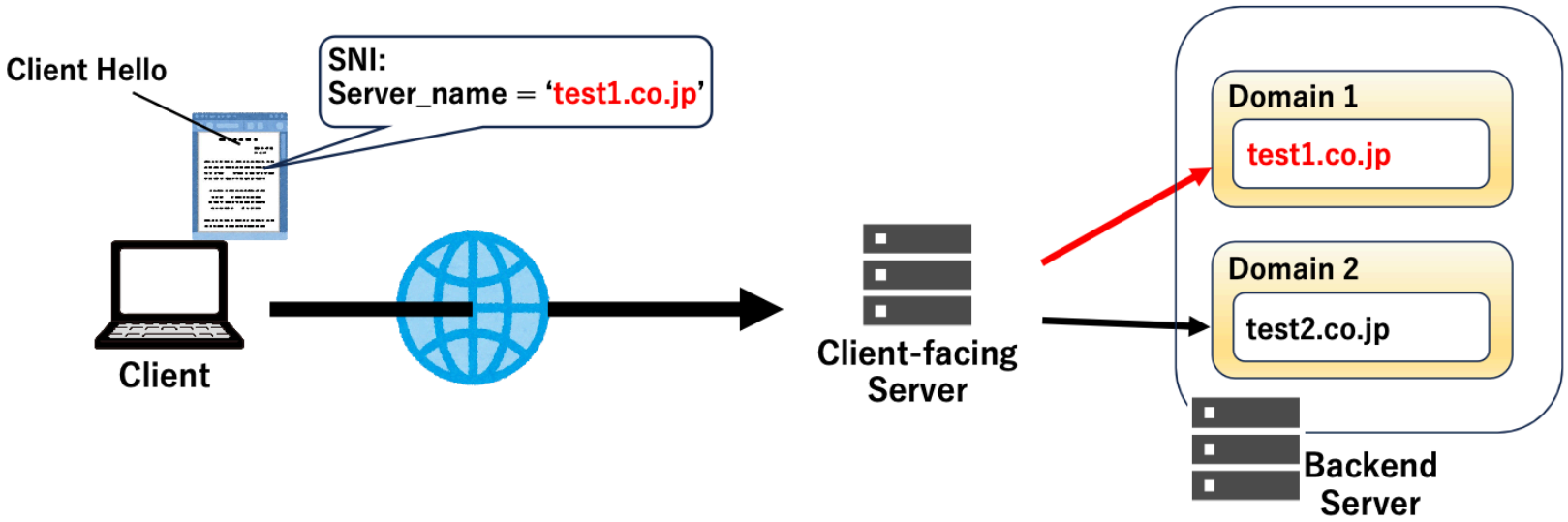
SNI(Server Name Indication)

SNI

ClientHello に含まれる TLS プロトコルの拡張機能。クライアントが接続したいサーバーのホスト名 (ドメイン名) が示されている。

TLS サーバは Client-Facing Server が一つの IP アドレスで複数のドメインを管理していることが多い

SNI により、Client-Facing Server が一つの IP アドレスで複数のドメインを管理していても、正しい Backend Server にクエリを送ることができる



ESNI(Encrypted Server Name Indication)

ESNI

ClientHello の SNI 部分を暗号化する技術。

クライアントとサーバの通信を盗聴する攻撃者は、クライアントがどのバックエンドサーバと通信しようとしてるかを知ることができない。

しかし、TLS 接続が確立された後、どのアプリケーションレイヤープロトコルを使用するかを決定するのに必要な ALPN が暗号化されていない。

⇓

Encrypted Client Hello (ECH)

Encrypted Client Hello (ECH)

ECH の目的

- ClientHello そのものを暗号化する
- TLS1.3 の既存のセキュリティ特性に影響を与えることなく実現

ECH により、ClientHello に含まれているクライアントの全ての機密情報を保護することができる

ECHの技術

HPKE(Hybrid Public Key Encryption)

ClientHello の暗号化には HPKE(Hybrid Public Key Encryption) を使用している

HPKE は RFC9180 で公開された規格。

公開鍵暗号方式と共通鍵暗号方式を組み合わせて任意の平文を暗号化するための、汎用的な枠組みとして標準化されている。

⇓

HPKE として規格化することにより、ハイブリッド暗号化そのものの枠組みを変えることなく、将来的な拡張性を高めることができる

---

https://datatracker.ietf.org/doc/rfc9180/

HPKE の構成

KEM(Key Encapsulation Mechanism)

- 送信者は公開鍵暗号を使って、固定長の shared secret から固定長カプセル化情報 を生成 (暗号化)
- 受信者はカプセル化情報から shared secret を復号する

KDF(Key Derivation Function)

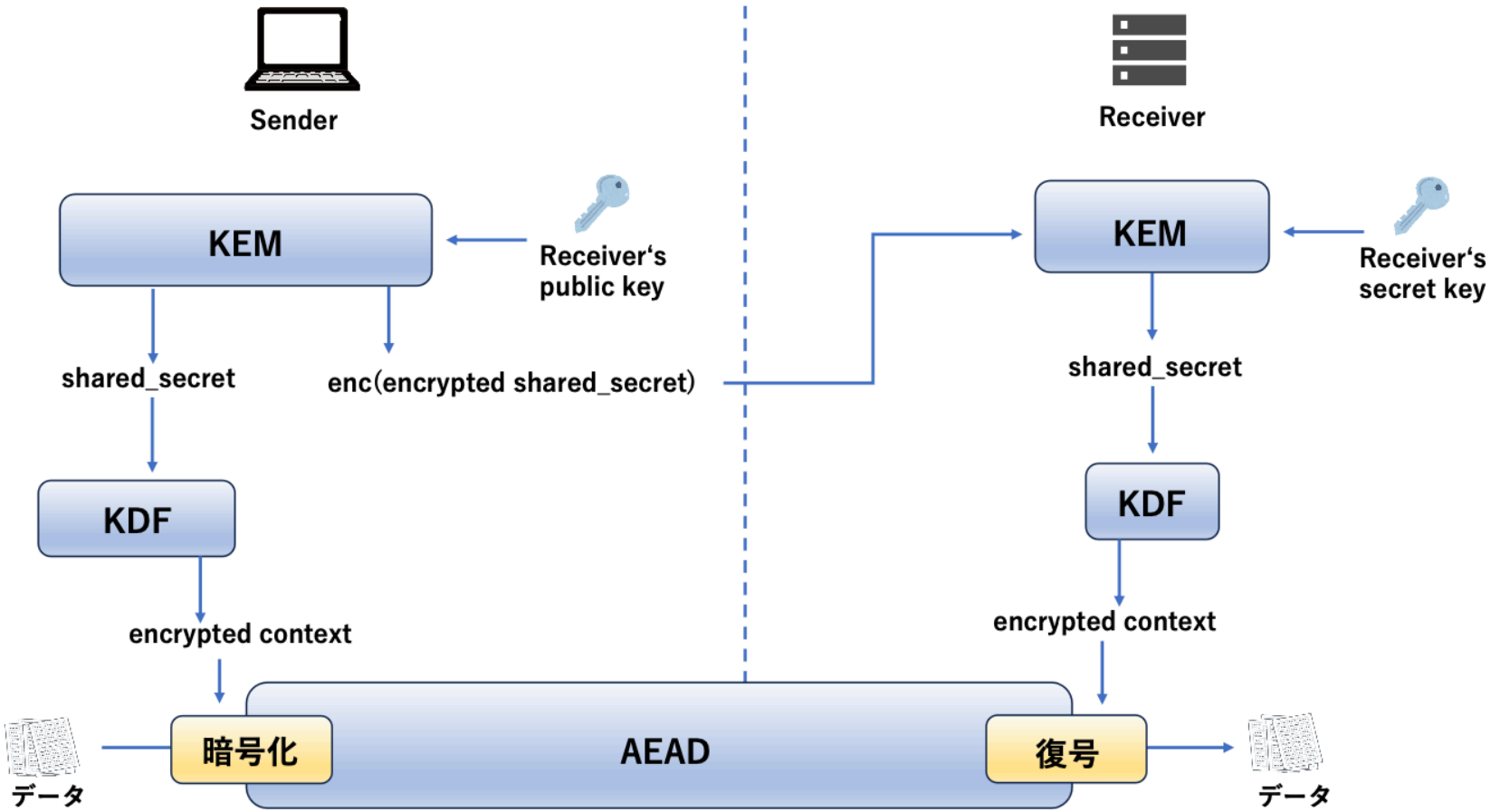
KEM で生成あるいは復号した shared_secret を伸長・攪拌して、AEAD で利用する擬似ランダム共通鍵を生成

AEAD(Authenticated Encryption with Associated Data)

KDF で生成した共通鍵を用いて認証タグ付き共通鍵暗号を行う。HPKE では AES-GCM 等が使われる

HPKE のフロー

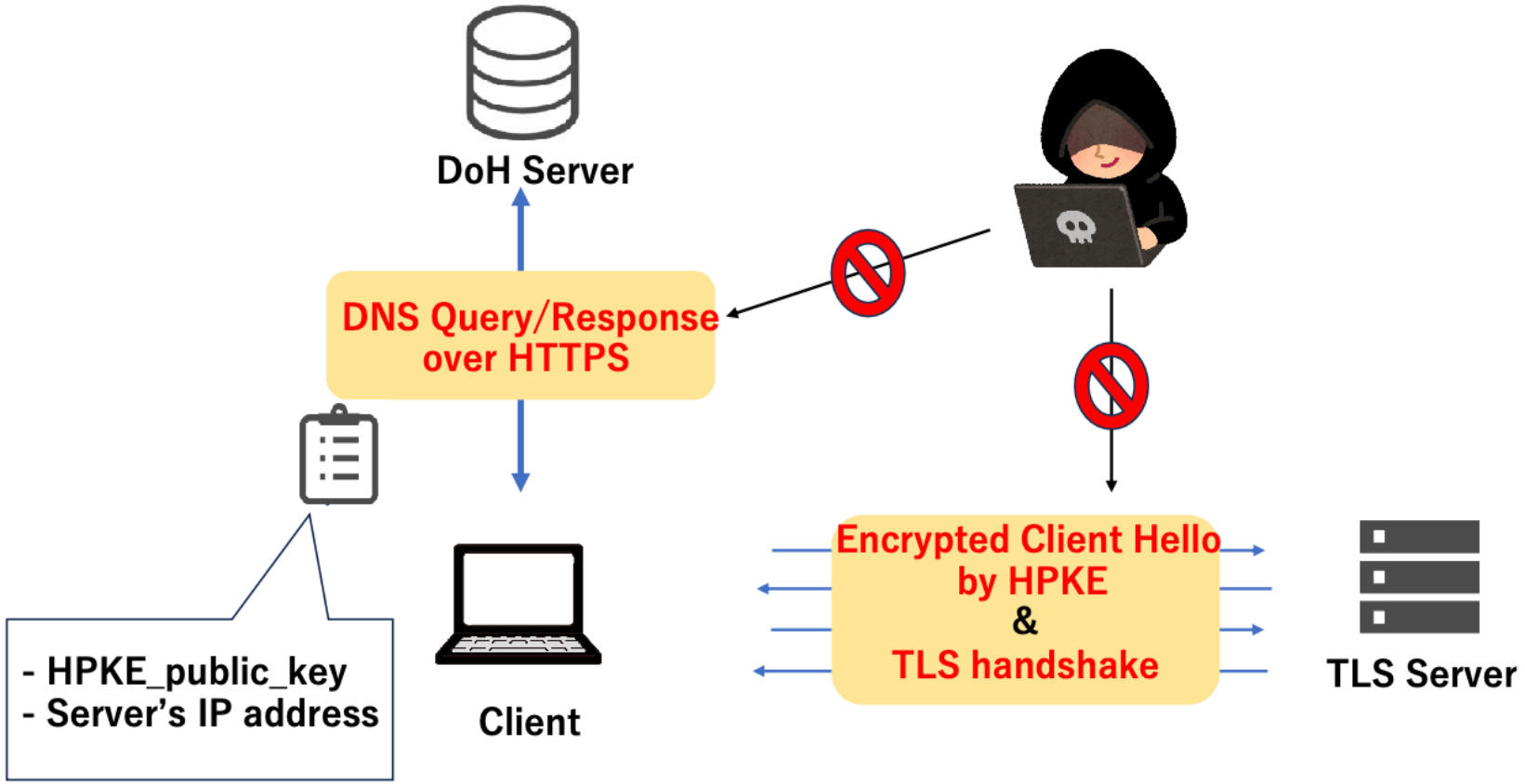
1. 送受信者で’encrypted context’ を生成 (KEM,KDF を実行)
2. ’encrypted context’ に格納されている共通鍵を使ってメッセージの暗号化、復号をする (AEAD の実行)



HPKE の ECH への応用

ECH への利用方法

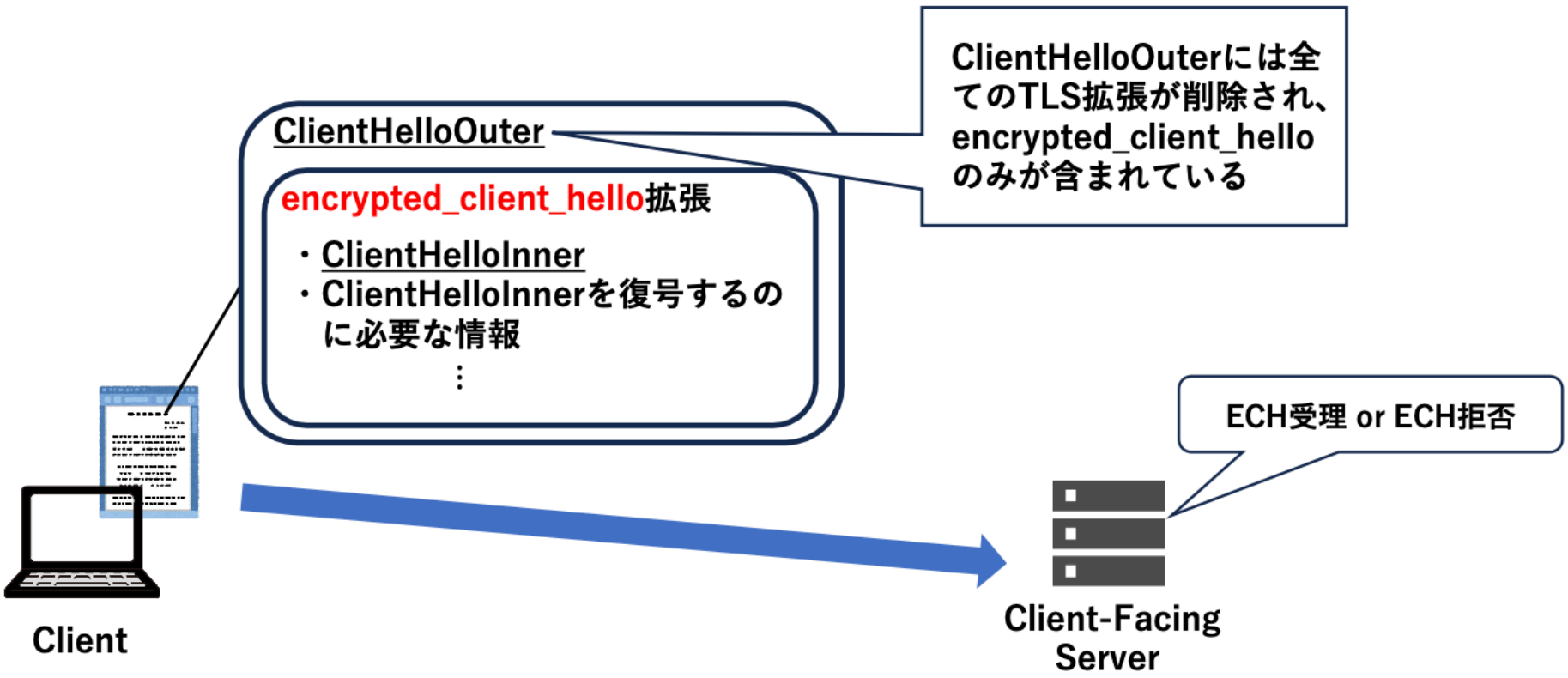
サーバの hpke 公開鍵を DNS の HTTPS Record で提供することで、クライアントは HPKE により ClientHello を暗号化する



ECH 仕様の ClientHello 構成

クライアントは、見かけ上の Client Hello(ClientHelloOuter) に TLS の拡張として、encrypted_client_hello 拡張を含めて送信

※この拡張に暗号化した Client Hello(ClientHelloInner) と ClientHelloInner を復号するのに必要な情報などが格納される



ECH の拒否・受理

ClientHelloOuter を受け取った Client-Facing Server

- ECH をサポートしていない場合、または ClientHelloInner を復号出来ない場合
  →ECH 拒否
- ClientHelloInner の復号に成功した場合、ClientHelloInner をバックエンドサーバに転送し、ハンドシェイクを完了
  →ECH 受理

⇓

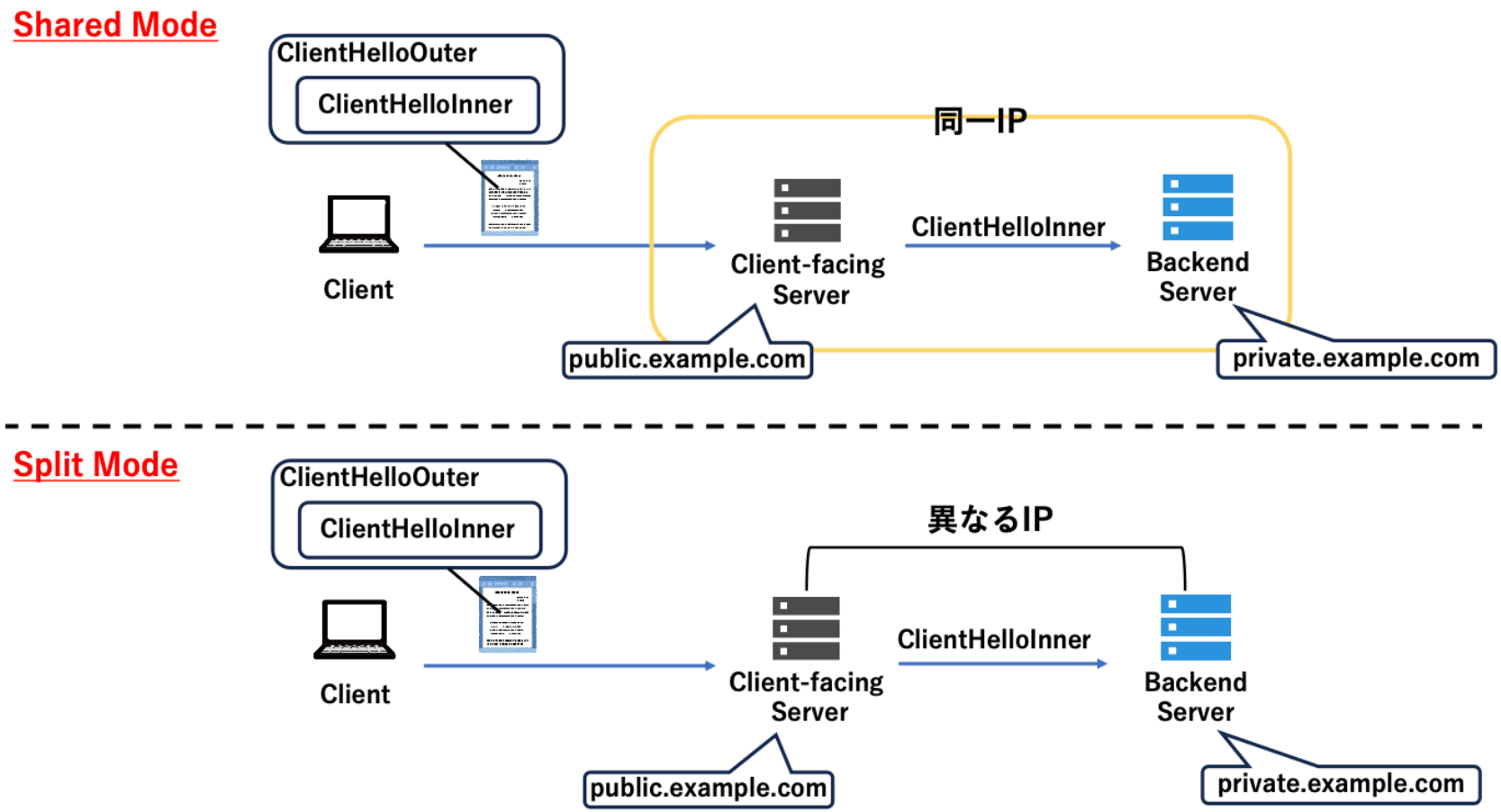
クライアント

サーバの応答を受信すると、ECH が受け入れられたかどうかを判断し、それに応じてハンドシェイクを進める。

※クライアントとサーバの詳しい動作については後に説明

ECH はトポロジとして、2 つのモードを想定している

- Shared mode
- Split Mode(※本資料ではこのモードを想定している)



Split Mode では Client-Facing Server と Backend Server が物理的に分離されており、クライアントは DoH サーバから Client-Facing Server の IP アドレスを得る

ECHConfig 構成の説明

ECHConfig 構成

Client-Facing Server は DoH Server を経由して Client に ECHConfig を公開する (クライアントは ECHConfig に格納された HPKE 用の公開鍵で ClientHello を暗号化する)

```
opaque HpkePublicKey<1..2^16-1>;
       uint16 HpkeKemId;  // Defined in I-D.irtf-cfrg-hpke
       uint16 HpkeKdfId;  // Defined in I-D.irtf-cfrg-hpke
       uint16 HpkeAeadId; // Defined in I-D.irtf-cfrg-hpke

       struct {
           HpkeKdfId kdf_id;
           HpkeAeadId aead_id;
       } HpkeSymmetricCipherSuite;

       struct {
           uint8 config_id;
           HpkeKemId kem_id;
           HpkePublicKey public_key;
           HpkeSymmetricCipherSuite cipher_suites<4..2^16-4>;
       } HpkeKeyConfig;

       struct {
           HpkeKeyConfig key_config;
           uint8 maximum_name_length;
           opaque public_name<1..255>;
           Extension extensions<0..2^16-1>;
       } ECHConfigContents;

       struct {
           uint16 version;
           uint16 length;
           select (ECHConfig.version) {
             case 0xfe0d: ECHConfigContents contents;
           }
       } ECHConfig;
```

ECHConfig 構造体について

```
struct {
    uint16 version;
    uint16 length;
    select (ECHConfig.version) {
      case 0xfe0d: ECHConfigContents contents;
    }
} ECHConfig;
```

version

ECH のバージョン。クライアントは、サポートしていないバージョンの ECHConfig 構造体は無視する必要がある。

contents

ECH のバージョンにより内容が異なる不透明なバイト列。ECHConfigContents 構造体を示している

ECHConfigContents 構造体について

```
struct {
    HpkeKeyConfig key_config;
    uint8 maximum_name_length;
    opaque public_name<1..255>;
    Extension extensions<0..2^16-1>;
} ECHConfigContents;
```

key_config

HPKE 公開鍵に関連する構成情報を保持する HpkeKeyConfig 構造体を指す。

extensions

ClientHello メッセージを生成する際にクライアントが考慮しなければならない拡張のリスト。

HpkekeyConfig 構造体について

```
struct {
    uint8 config_id;
    HpkeKemId kem_id;
    HpkePublicKey public_key;
    HpkeSymmetricCipherSuite cipher_suites<4..2^16-4>;
} HpkeKeyConfig;
```

config_id

指定された HpkeKeyconfig を表す 1 バイトの識別子。クライアントが ClientHello の暗号化に使用する鍵を示すために使用。

kem_id

public_key に対応する HPKE KEM 識別子。クライアントは、自分がサポートしていない KEM を使用した ECHConfig 構造体を無視しなければならない。

public_key

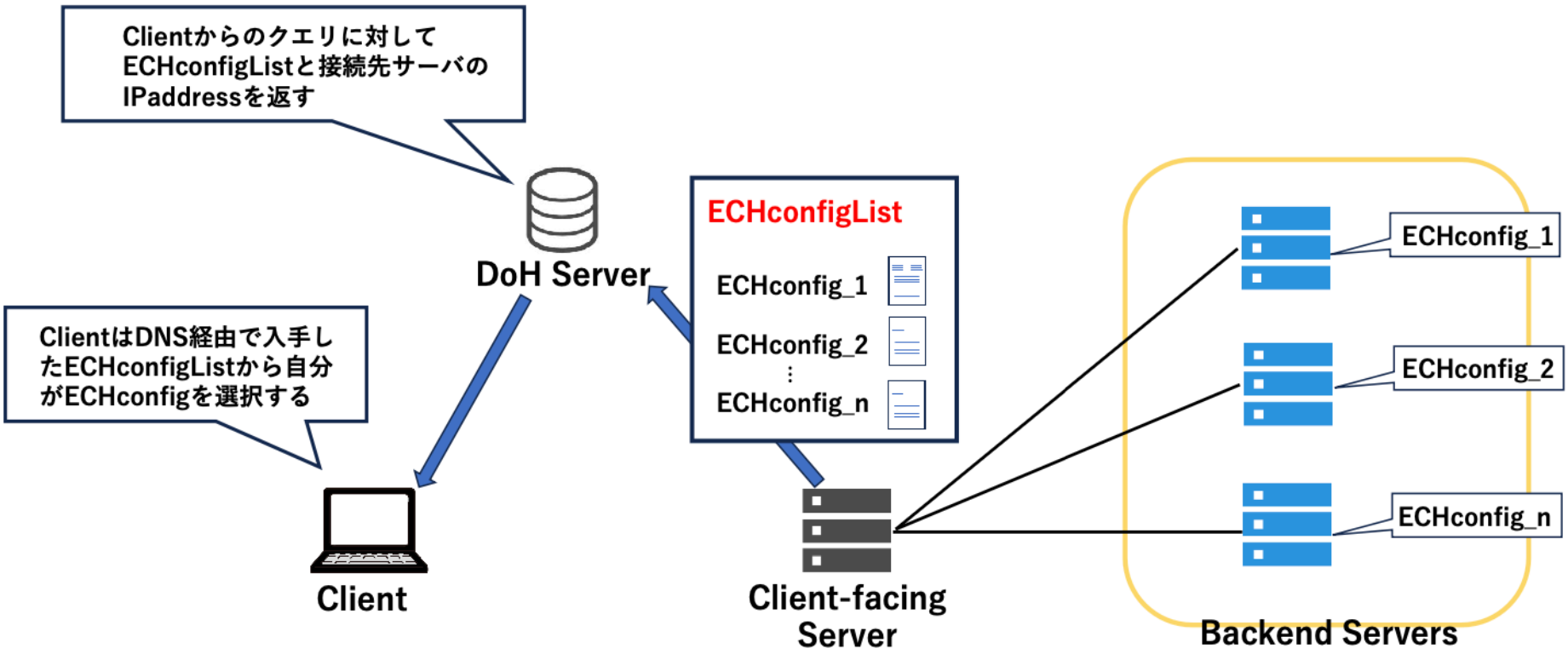
クライアントが ClientHelloInner を暗号化するために使用する HPKE 公開鍵。

cipher_suites

クライアントが ClientHelloInner を暗号化する際に使用する HPKE KDF と AEAD の識別子のペアのリスト。

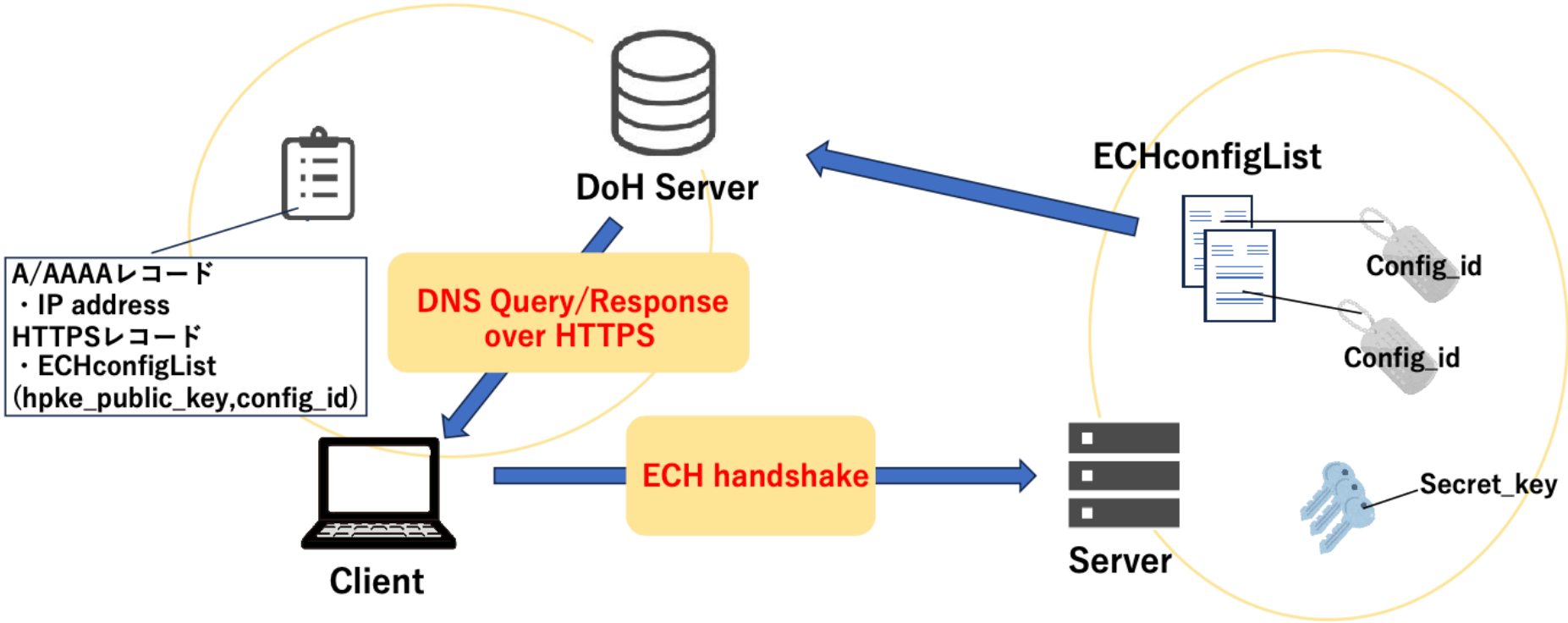
※ Client-Facing Server は各 Backend Server に対して ECHconfig を設定している (複数の Backend Server に同じ ECHconfig を設定しても良い)

Client-Facing Server は、一つ以上の ECHconfig を優先順位の高い順にリスト化してクライアントに公開する



これにより、サーバは複数の ECH のバージョン・パラメータセットをサポートできる

既知の ECHconfig のそれぞれに重複しない config_id が割り当てられており、Client-Facing server はそれぞれに対応する複数の秘密鍵を持っている。



※攻撃者はクライアントが選択した ECHconfig の config_id を知っていても、同じ ECHConfig 構成をもつ Backend Server(ホスト) 間のうち、クライアントがどのホストに接続しようとしてるか見分けることができないようになっている。

'encrypted_client_hello' 拡張について

ECH を提供するために、クライアントは ClientHelloOuter で'encrypted_client_hello' 拡張を送信する

※ ClientHelloInner でもその拡張を送信しなければならない

```
enum { outer(0), inner(1) } ECHClientHelloType;

struct {
   ECHClientHelloType type;
   select (ECHClientHello.type) {
       case outer:
           HpkeSymmetricCipherSuite cipher_suite;
           uint8 config_id;
           opaque enc<0..2^16-1>;
           opaque payload<1..2^16-1>;
       case inner:
           Empty;
   };
} ECHClientHello;
```

- outer extension→ClientHelloOuter に含まれる extension
- inner extension→ClientHelloInner に含まれる extension

outer extension の field

config_id

選択された ECHConfig の ECHConfig.key_config.config_id を示す

cipher_suite

ClientHelloInner を暗号化するのに使用する暗号スイート

enc

サーバが対応する payload field を復号するために使用する HPKE カプセル化 key。

payload

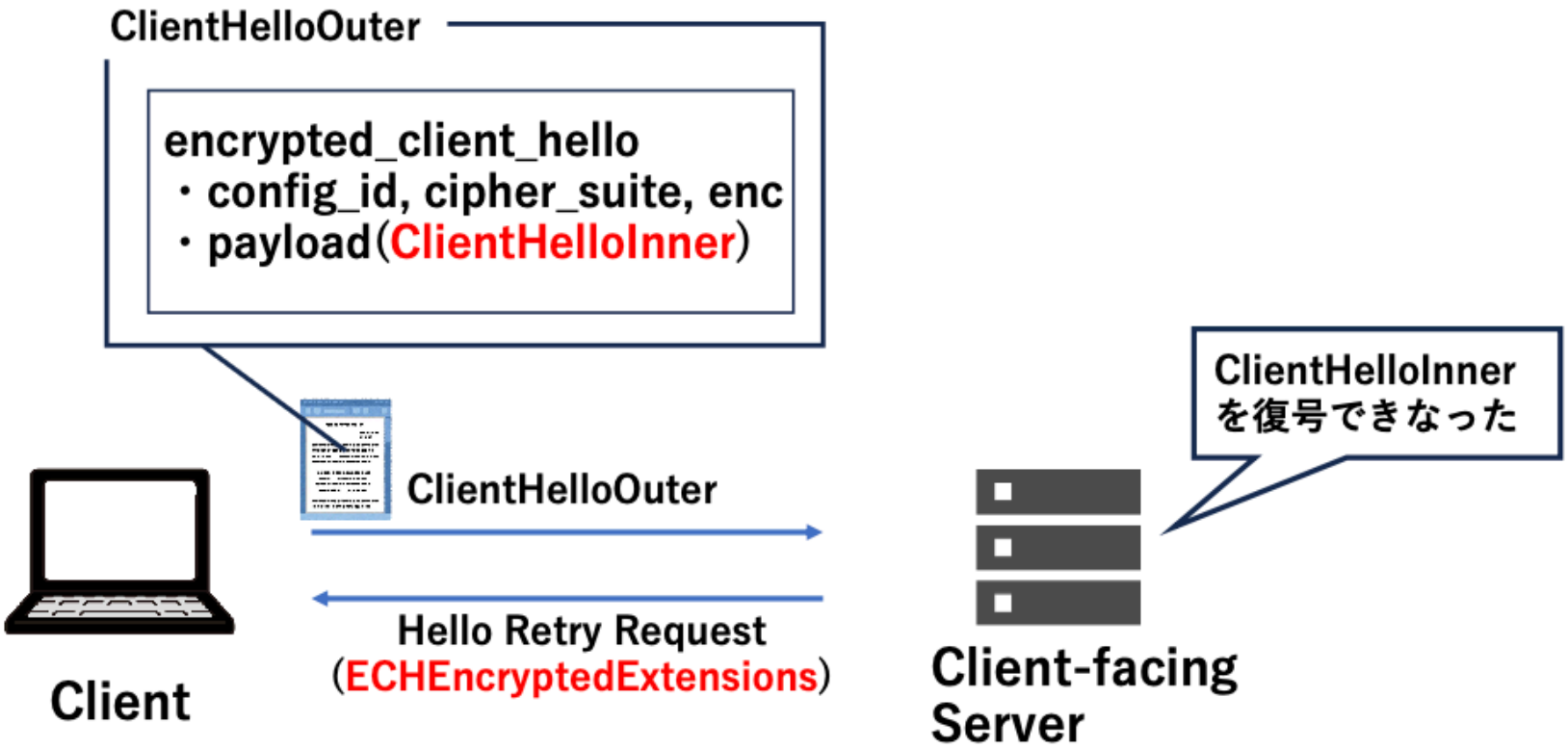
HPKE により暗号化された ClientHelloInner 構造体

Client-Facing Server は encrypted_client_hello の outer extension に含まれる情報を用いて、ClientHelloInner を復号する

Client-Facing Server が ClientHelloInner の復号に失敗した時、サーバは ECHEncryptedExtentions メッセージ (retry_configs を含む) を送り、ECH の再試行をクライアントに要請することができる。

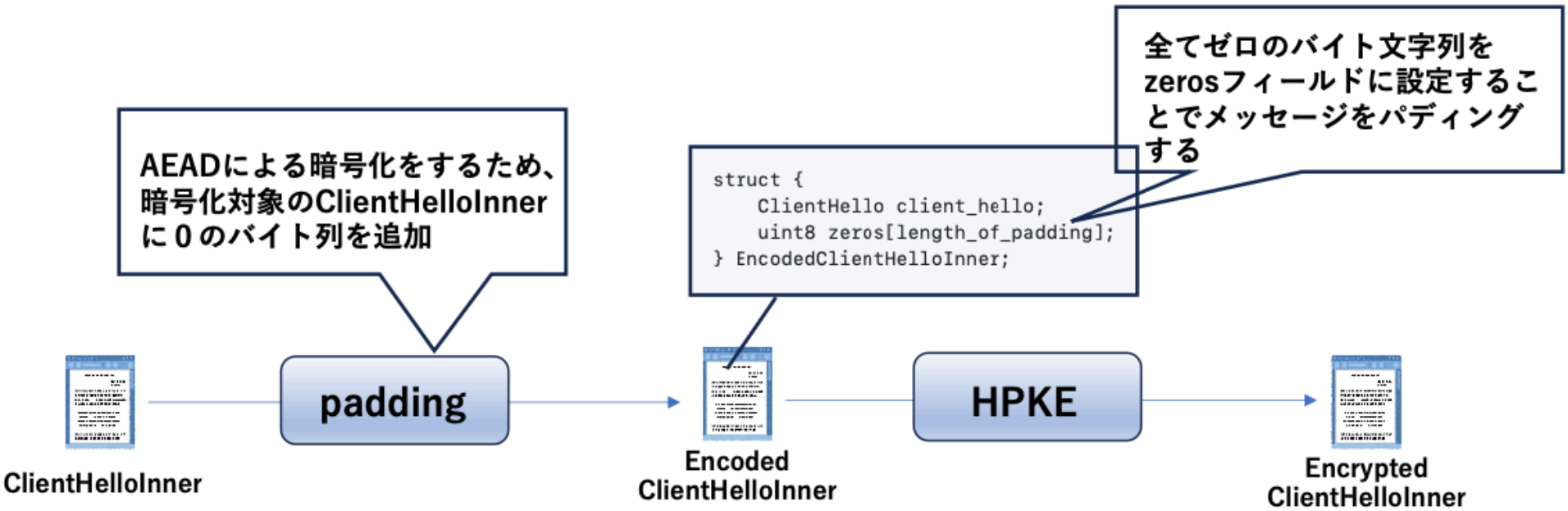
retry_configs

クライアントが使用する 1 つ以上の ECHConfig を、優先順位の高い順に含む ECHConfigList 構造体。



ClientHelloInner のエンコード

ClientHelloInner の暗号化の前に、クライアントは ClientHelloInner をパディングし、圧縮して、EncodedClientHelloInner 構造体にする



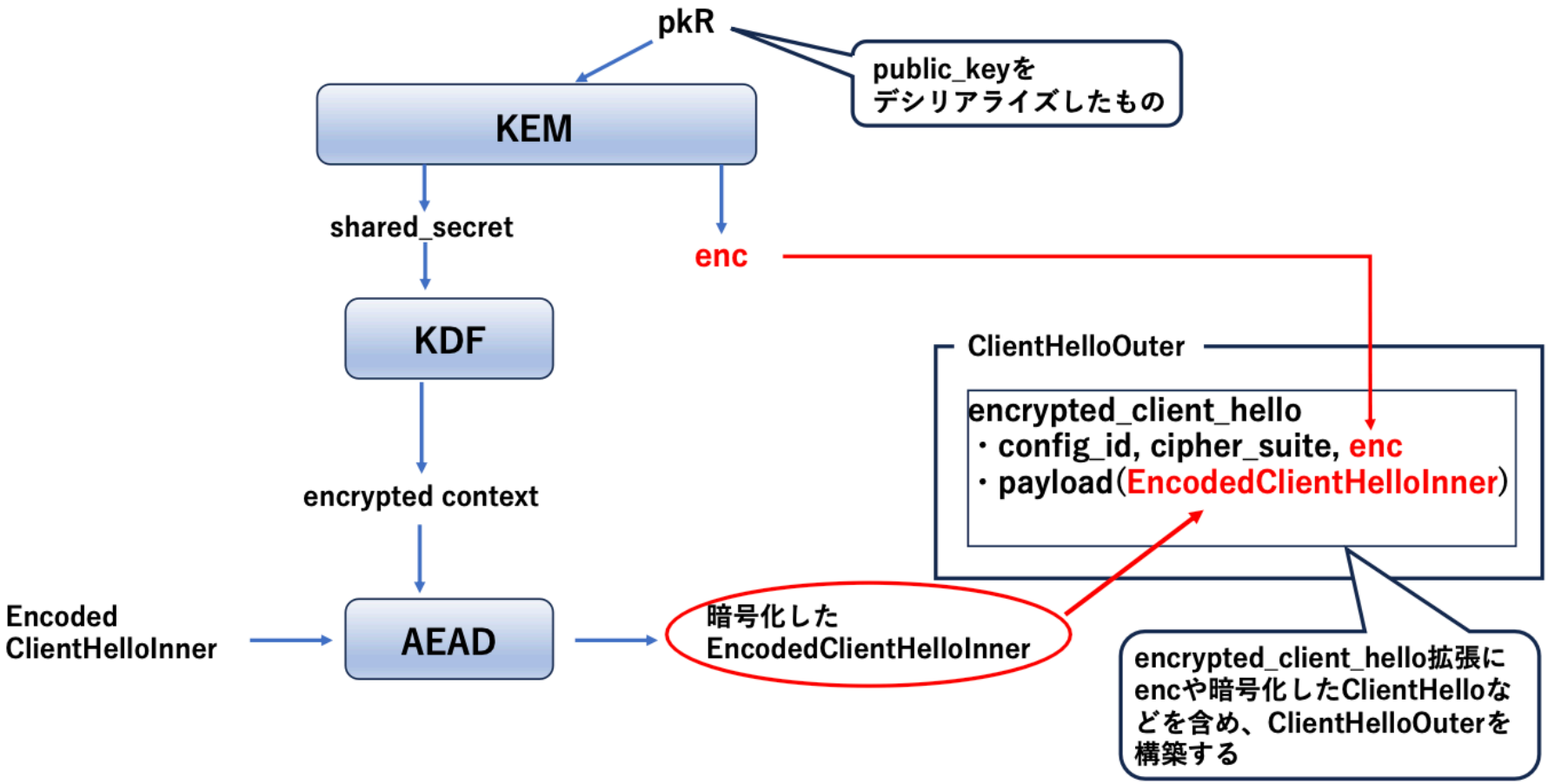
ECH の提供 (クライアントの動作)

クライアントの動作

1. サーバの ECHConfigList から適切な ECHConfig を選択
   ※選択した ECHConfig が適切かどうかを判断するために以下を確認する
   - ECHConfig.version
   - HpkeKeyConfig.cipher_suites
   - HpkeKeyConfig.kem_id
2. 標準の ClientHello と同様に ClientHelloInner を構築
3. ClientHelloInner をパディングして EncodedClientHelloInner を構築
4. HPKE により encrypted context と enc を計算する
5. EncodedClientHelloInner を暗号化して、ClientHelloOuter を構築する

EncodedClientHelloInner の暗号化・ClientHelloOuter の構築

クライアントは HPKE により、enc の生成と EncodedClientHelloInner の暗号化を行い、ClientHelloOuter を構築する



ECH の提供 (サーバの動作)

サーバの動作

各サーバの役割

- Client-Facing Server
  ClientHelloOuter から ClientHelloInner を抽出するために復号処理を進め、復号した ClientHelloInner を Backend Server に転送する
- Backend Server
  ServerHello をクライアントに送信して TLS ハンドシェイクを進める

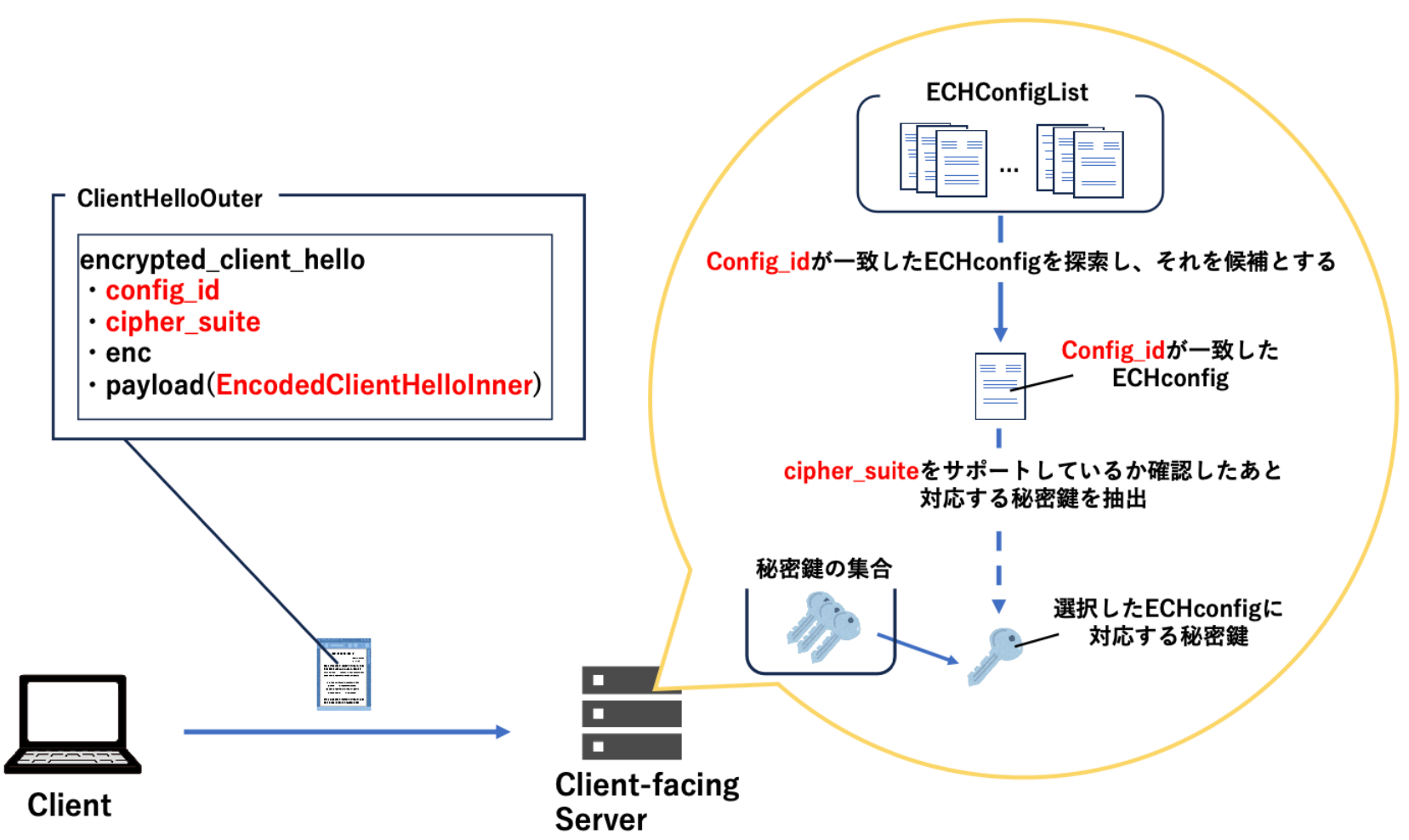
Client-Facing Server の動作

1. クライアントから受信した ClientHelloOuter の”encrypted_client_hello”拡張から enc を得る
2. hpke 用の秘密鍵で enc を復号し、encrypted context を生成 (KEM、KDF)

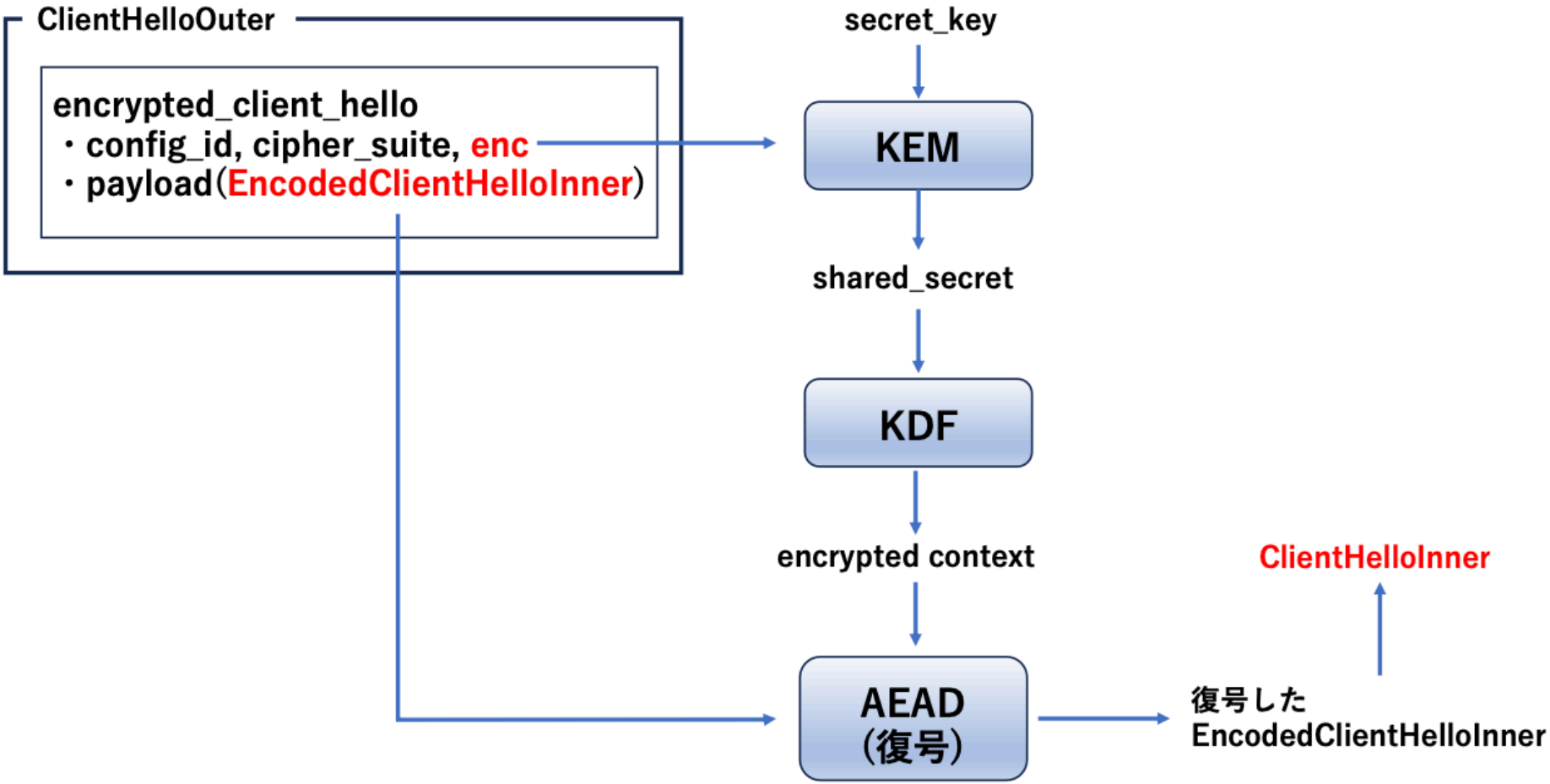
   ※ enc の復号に用いる秘密鍵はクライアントから受信した ClientHelloOuter.encrypted_client_hello.config_id に対応したもの
3. 暗号化された EncodedClientHelloInner を encrypted context を用いて復号 (AEAD)
4. 復号した EncodedClietHelloInner からパディングで追加されたゼロバイト列を取り除き、ClientHelloInner を得る
5. ClientHelloInner にも encrypted_client_hello 拡張を含むこと、TLS1.2 以下を提供していないことを確認する
   ※これを満たさない場合、サーバは”illegal_parameter”アラートを出して中断する
6. Client-Facing Server はバックエンドサーバに ClientHelloInner を送信

enc を復号する秘密鍵の選定方法

- Client-Facing Server は encrypted_client_hello.config_id と一致する ECHconfig を ECHconfigList から探索する
- config_id が一致した ECHconfig が encrypted_client_hello.cipher_suite をサポートしているか確認し、秘密鍵を取り出す



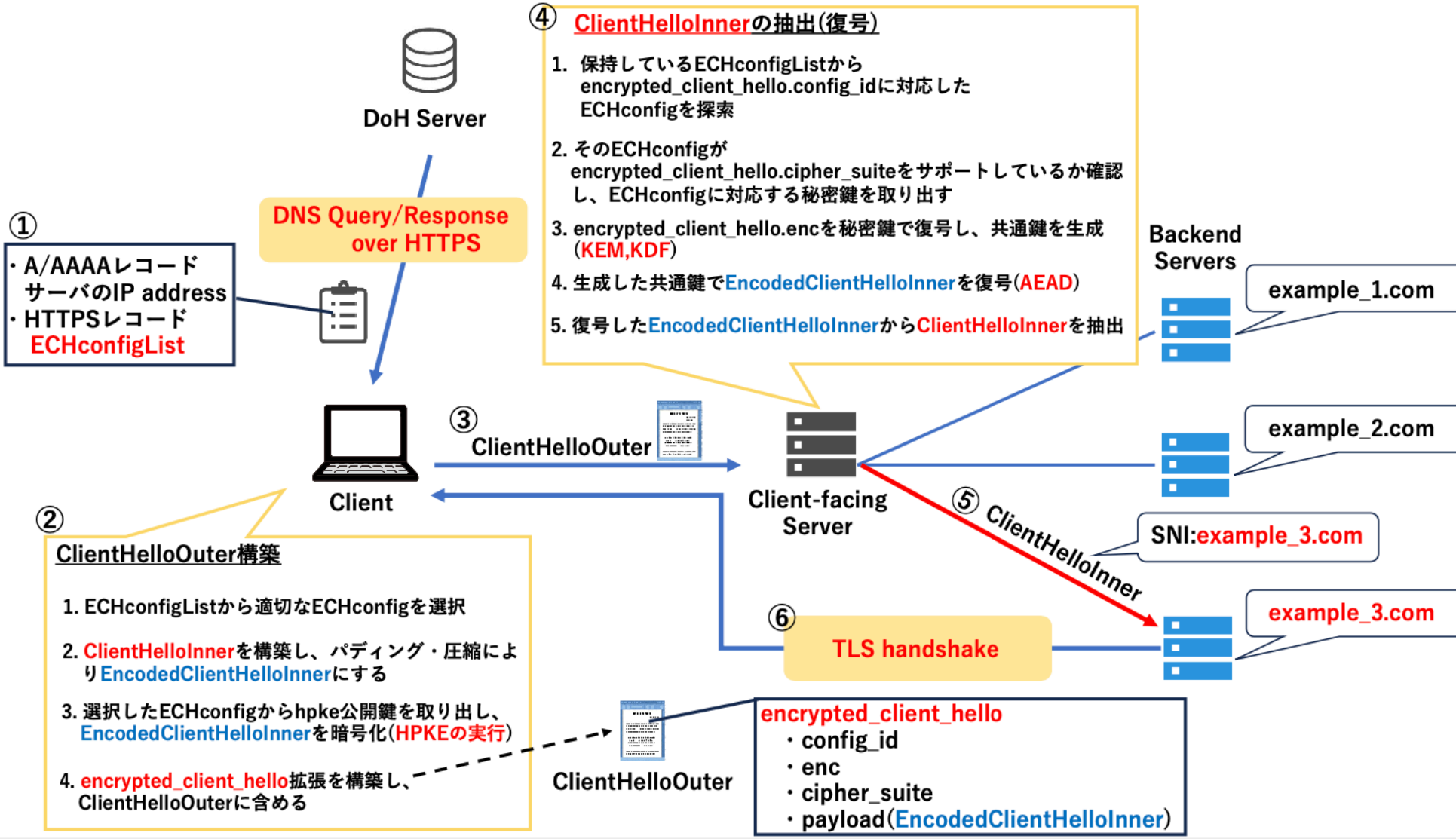
Client-Facing Server による EncodedClientHelloInner の復号



EncodedClientHelloInner を復号できなかったとき、ECH 拒否となり、ECHEncryptedExtensions メッセージをクライアントに送信して ECH の再試行ができる

ECH のフロー

クライアントは example_3.com に接続したいとする



※ EncodedClienthelloInner を復号できなかった場合、ECH を拒否となる

GREASE ECH

クライアントがサーバに接続しようとした際に、利用可能な ECHconfig がない場合、GREASE ”encrypted_client_hello”拡張を設定して送信するべき

GREASE [RFC8701]

Key_share, Supported_Groups, Cipher_Suites などの TLS 拡張にクライアントが入れるべき値が未知のときに設定できる値の集合

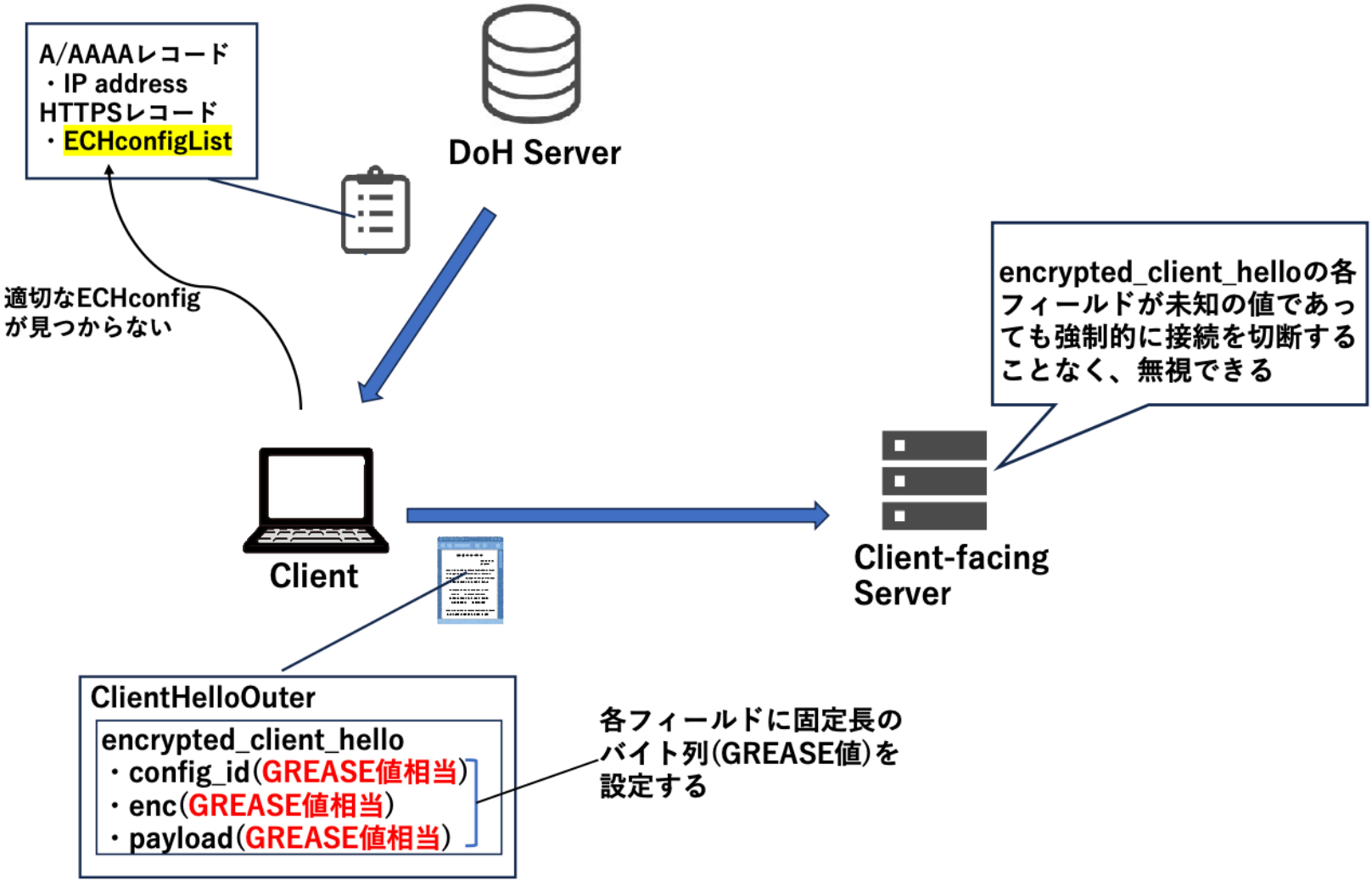
以前は TLS1.3 のハンドシェイクで、サーバがクライアントから受信した TLS 拡張が未知であった場合、サーバはその拡張を無視しなければならないが、正しく処理 (無視) せず TLS セッション自体を切断してしまうことがあった

↓

クライアントは GREASE に定義された値を未知の TLS 拡張に設定することで、サーバに切断されず正しく処理してもらえる

GREASE ECH

encrypted_client_hello 拡張の各フィールドに固定長のランダムなバイト列を設定することで、サーバに切断されずにハンドシェイクを進めてもらえる

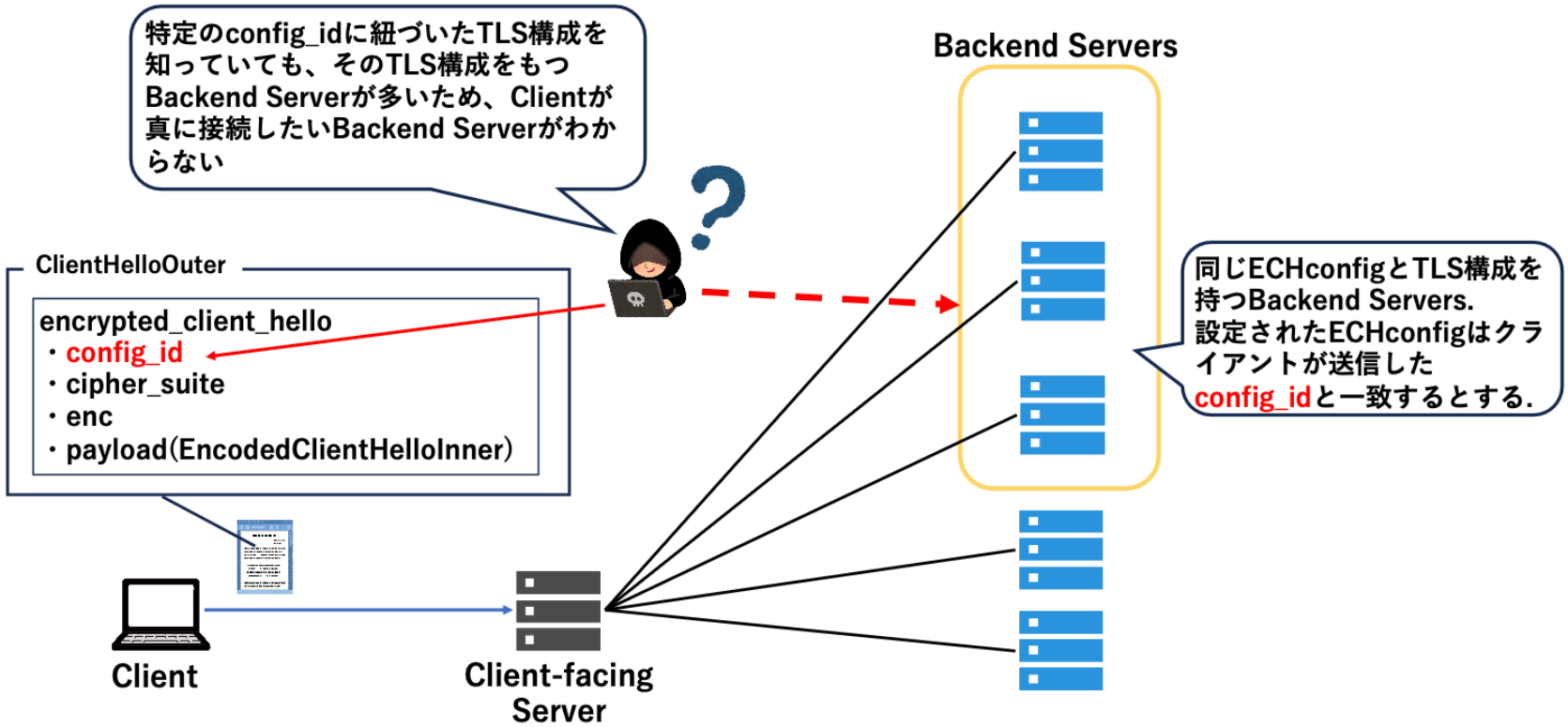


ECH が持つセキュリティ特性

Client-Facing Server は同じ TLS 構成をもつ Backend Server 同士に同じ ECHconfig を設定する

⇓

攻撃者は特定の config_id が特定の TLS 構成を持つ BackendServer に設定されていることがわかっても、encrypted_client_hello.config_id からクライアントが接続したい Backend Server を特定しにくい



ECH のセキュリティにおいて、満たす性質は以下のとおり

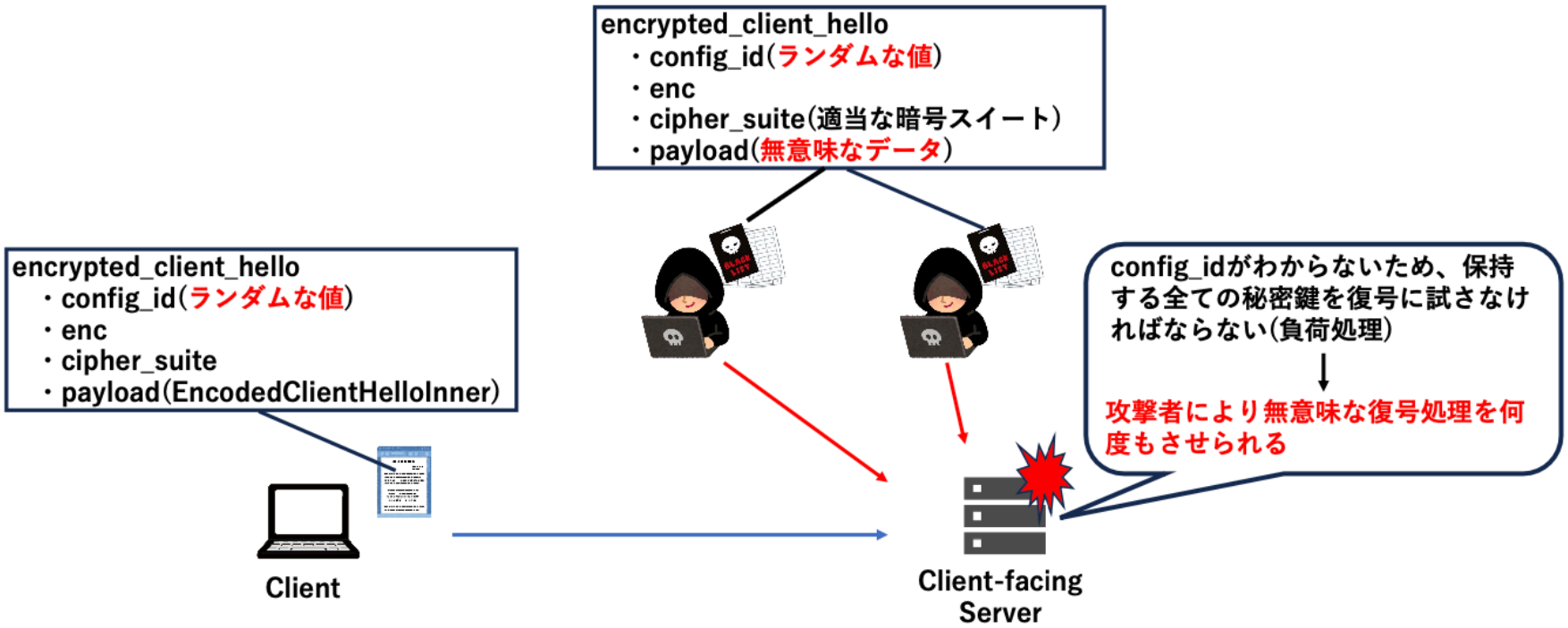
- ECH を使用しても、既存の TLS のセキュリティ性能に影響を与えることはない
- 特定の ECHConfig と TLS 構成を持つホストへの TLS 接続確立は、同じ ECHConfig と TLS 構成を持つ他のホストへの接続と見分けがつかない
- 攻撃者は、ECH を使用しようとする接続に対し、能動的に無効化することはできない

セキュリティ上推奨されない手法

Client-Facing Server は config_id の情報を攻撃者に知られたくない場合、クライアントに encrypted_client_hello.config_id をランダムな値に設定するように要請できる

しかし、encrypted_client_hello.config_id のランダム化を許してしまうと、

- config_id に対応する秘密鍵を特定できないため、全ての秘密鍵を復号に試さなければならない
- 攻撃者が無意味な ClientHelloOuter を送信した場合でも、全ての秘密鍵を試して復号処理せざるを得ない (Dos 攻撃の悪化)



ECH の要件

「TLS Encrypted Client Hello:draft-ietf-tls-esni-17」では、アプリケーションプロファイル規格に規定がない場合、準拠する ECH アプリケーションは以下の HPKE 暗号スイートを実装するように言われている

KEM: DHKEM(X25519, HKDF-SHA256)

KDF: HKDF-SHA256

AEAD: AES-128-GCM

まとめ

まとめ

ClientHello を全て暗号化する Encrypted Client Hello(ECH) という技術についての概略をまとめた。

- ClientHello の暗号化には HPKE を用いている。
- ECH は、ClientHello そのものを暗号化することであり、それ以降の TLS1.3 のハンドシェイクに影響を与えることはない。