モダンな機能を搭載した標準プロトコル(IPsec / IKEv2)ベースのユーザ空間で動作するVPNソフトウェアの設計と実装

Rockhopperプロジェクト
花田哲治(rhpenguine@gmail.com)

Rockhopperプロジェクト

開発の経緯（１）

- ２００４年頃（？）～
  - TUN/TAPドライバを利用した「ユーザ空間VPNソフトウェア」デザインの人気
    - Pipsec, VTun, OpenVPN, OpenSSH　など
  - 新しいIPsec仕様(IKEv2,ESPv3)のIETF標準化完了
    - IPsec v2(IKEv1+ESPv2)への長年の不満
- ～２００７年～
  - オープンソフトウェアなどによる仕様検証・参照実装
    - Racoon2, strongSwan, OpenIKEv2, IKEv2 Project
  - RFC4718 “IKEv2 Clarifications and Implementation Guidelines”の発行
  - IKEがようやく「理解しやすい」「ツカエル」仕様に！

開発の経緯（2）

- 2008年～
  - IKEv2 & ESPv3ベースのLinux 2.6上で動作する新しいVPNソフトウェアの実装を趣味で始める。
    - 「Rockhopper(仮称)」 -- オープンソース（Lesser GPL v2.1）
    - TUN/TAPドライバを利用。全てユーザ空間で動作する実装。
    - モダンなVPN機能を標準プロトコル上にインポート。
    - 新しいIKEv2仕様を「てこ」にしたよりシンプルなIPsec実装を目指して。
- 2009年夏
  - SOURCEFORGE.JPにプロジェクト登録
  - 現在、Version 1のリリースへ向け開発作業中。

特徴（１）

- 標準プロトコルベース（IKEv2 & ESPv3）
- 全てをユーザ空間アプリケーションとして実装。
  - ESPプロトコル処理も含めて。
  - TUN/TAPデバイスドライバを利用。
  - シンプルなソフトウェア構成。
  - OSの保護下で全ての処理を安全に実行可能。
  - 全てOSの管理下で動作（スケジューリング、リソース管理など）。
  - デバッグが楽。開発ツールが豊富（Eclipse 3.4 + CDT + 各種プラグイン を利用）
- 処理に応じてプロセスを安全に分離
  - セキュリティ的に重要な処理は、ネットワーク処理などのプロセスから隔離。それぞれ別ユーザ権限で実行。
  - IKEv2 / Web認証、キーストア(X.509,PSK)、ネットワーク管理

特徴(2)

- モダンなハードウェアを活用しやすい設計
  - 暗号オフロードハードウェア:ESP処理をプラグイン化
  - マルチコアCPU:分散ルールベースのマルチスレッドプール
- カスタマイズのしやすい構造
  - ベンダ依存度また運用システム依存度の高い機能をSPI (Service Provider Interface)を通じてプラグイン可能な構造。
    - X.509証明書SPI
    - 暗号処理 SPI
    - ESPプロトコル SPI
    - EAP機能 SPI
    - デバッグトレース / ログ機能 SPI
  - ワーカースレッドの使い方を後からカスタマイズしやすい設計。

特徴（3）

- Role / IDベースのVPNポリシー管理・設定
- AJAX(Comet)ベースのWebユーザインタフェース
- モダンなVPN機能をいいとこどり。
  - 仮想イーサネット機能
  - ルーティングベース
  - P2P型通信の支援機能（予定） など
- モバイルネットワークおよびマルチホーミング環境の標準サポート（MOBIKE）
- プラガブルIKEv2拡張機能の追加。

など

Rockhopperプロジェクト：開発の背景

暗号プロトコルによるVPN実装の動向

- ここではエンドユーザがPCから接続できるスタイルのVPN実装にフォーカス。
- 様々な発想で設計されたセキュアプロトコルを利用したVPNソフトウェアの人気
  - オープンソース / フリーソフトウェア
    - OpenVPN, VTun, CIPE, ...
  - プロプライエタリ
    - SSL-VPN, ...

==> **独自プロトコルまたは非公開のプロトコル**

  - TCPカプセリング： PPP over SSH ==> TCP over TCP問題
  - ユーザ空間IPsec実装
    - pipsec ==> IKEサポートなし（？）。現在の開発状況は（？）

IPsec

- セキュアVPNを実現するための標準暗号プロトコル
- 様々な接続形態をサポートできる柔軟なプロトコル
  - Site-to-Site型（拠点間）VPN
  - Remote Access型VPN
- End-to-Endホスト間の安全な通信の標準プロトコル
- １５年以上の歴史あるプロトコル
- 数多くのネットワーク機器やソフトウェア、OSが広くサポート
- IPv6では標準搭載される機能

IPsec(IKEv1+ESPv2)のイメージ（１）

- 設定が難しそう。。。？
- 仕様が複雑でややこしそう。。。？
- ネットワークレイヤーのための技術？
- ユーザ空間アプリケーション開発には直接関係ない？

IPsec(IKEv1+ESPv2)のイメージ（2）

- 暗号の大家：ブルース・シュナイヤーさん
  - ”[…],we do not believe that it(IPsec) will ever result in a secure operational system.

    It is far too complex, and the complexity has lead to a large number of ambiguities, contradictions, inefficiencies, and weaknesses. ”
  - “We strongly discourage the use of IPsec in its current form for protection of any kind of valuable information, and hope that future iterations of the design will be improved.”

出典：「A Cryptographic Evaluation of Ipsec」
(Niels Ferguson and Bruce Schneier)

IPsec(IKEv1+ESPv2)のイメージ（3）

- Site-to-Site型（拠点間）VPN
  - 「ポリシーベース」モデルの不人気（？）
    - 難解（？）なRFC仕様、制約の多い運用モデル
      - トラフィック条件（セレクタ）を設計・設定する必要。
    - SPDモデルとファイアウォールモデルとの重複（？）
      - IPsec機能がファイアウォール機能と一緒に提供される場合も多い。
        - でも、ファイアウォールの方が機能が豊富な場合も。
          - 例：ステートフルインスペクション
      - 通常はアクセス制御はファイアウォールで。

IPsec(IKEv1+ESPv2)のイメージ（４）

- 一方で「ルーティングベース」モデルの広まり。
  - IPsec接続を仮想トンネルインタフェースを利用したルーティング経路として表現。
  - ルーティング設定の考え方でVPNを運用可能。
    - ダイナミックルーティングもOK。
  - しかし、標準ベースの運用モデルや実装モデルは？
  - ベンダ毎に異なる実装、相互接続性に難点。
    - ブロードキャスト・マルチキャストへの考え方。。。
    - ESPトンネルモード + IP (そのままカプセリング)
    - ESPトランスポートモード + GRE + …
    - ESPトランスポートモード + L2TP + …
    - ESPトランスポートモード + EtherIP + Ethernet + …

IPsec(IKEv1+ESPv2)のイメージ（５）

- リモートアクセスVPNの標準仕様なし
  - デファクトのMode-config, XAUTHはExpiredなドラフト扱い。
  - クライアントソフトウェアのベンダーロックイン的な状況。異なるベンダ製品間の相互接続性はサポートに難点。

標準仕様としてのIPsecの利点が生かされない状況。

IKEv2の登場！（1）-シンプルな仕様！

- IKEv1
  - 汎用的なプロトコル(DOI)
  - 交換手順（フェーズ1）
    - Mainモード
    - Aggressiveモード
  - VPN接続交換メッセージ数
    - Mainモード　　：9
    - Aggressiveモード：6
  - 認証方式：
    - 事前共有鍵(PSK)方式
    - デジタル署名方式
    - 公開鍵暗号化方式
    - 改良型公開鍵暗号化方式

- **IKEv2**
  - **DOI廃止！IPsec用**
  - **単一の交換手順のみ**
  - **モードなし: 4**
  - **2方式のみ**
    - 事前共有鍵(PSK)方式
    - デジタル署名方式

    （非対称型の認証方式が可）

IKEv2の登場！（2）

- 実際に必要な数々の仕様を「きちんと」規定。
  - より柔軟な設定モデル：
    - 柔軟なトラフィックパターンを単一Child SA(IPsec SA)に収容可能
      （トラフィックセレクタのリスト化やNarrowingが可能）
      - IKEv1ではポート番号を指定するとそのポートの接続毎に別々のIPsec SAを接続する必要あり。設定・管理工数的に。。。
    - SAのライフタイムの交渉が不要に。
    - マルチホスティング（同一IP/Portで複数管理ドメインを収容可能）
  - 暗号鍵の更新（Rekey）手順をちゃんと規定。
  - NATトラバーサルを内蔵
  - Dead Peer Detection（ピアの死活監視）

→ Keep-alive機能を内蔵

etc...

IKEv2の登場！（3）

- 非標準だったリモートアクセス仕様を規定
  - ユーザ認証（XAUTH）
    → EAPのトランスポートとして動作可能
  - リモート端末の自動設定（Mode-Config）
    → Configurationペイロードの交換を規定
- MOBIKE（IKEv2 Mobility and Multihoming Protocol）
  - ダイナミックネットワークやモバイルネットワーク環境からの利用をサポート
    - SA接続後、ピアのアドレスが変わっても良い
  - マルチホーミング（VPN経路の冗長化）のサポート

IKEv2の登場！（4）

- より安全なプロトコル
  - 非対称型の認証が可能に（例：イニシエータ：PSK、レスポンダ：RSA-SIG）
  - DoS防御機能を内蔵(Cookies)
  - リプレイ攻撃防御が可能に
  - IKEv1プロトコルの複数の脆弱性を解決
- 「きちんと」信頼性と互換性を規定されたプロトコル
  - 「デフォルトで」きちんとIPsec通信(Child SA)の開始タイミングをピア間で同期できる。
  - きちんとピアのSAを消せる（Deleteペイロードの応答/再送）
  - メッセージ再送の手順や送達確認がきちんと決められた。
  - 両ピアから同時発生したネゴが衝突してもその処理手順がきちんと決められた。

IKEv2の登場！（5）

- まだまだ発展する便利な拡張仕様
  (IETF ipsecme-WG)
  - 「Redirect Mechanism for IKEv2」
    - ピアノードからの接続を他ノードへリダイレクトするしくみ。
      (1) ノードシャットダウン時に現在の接続を他ノードへ誘導。
      (2) IPsec接続を複数ノードで負荷分散する。
      などで利用できる。
  - 「IKEv2 Session Resumption」
    - 最初のIPsec接続完了時にそれを証明するチケットをピアへ発行。
    - ピアはVPNをサスペンド（PCをハイバネーション）しても、その後、接続をレジュームする際、そのチケットを使って簡略化、ゲートウェイの負荷やクライアントを軽減。

Rockhopper : 設計方針

（1）スクラッチから開発

- 既存の他のIKEv2オープンソース実装とは異なるデザインを採用。
- 「リファレンス実装」は目指さない。
  - 利用シーンが実際に想定できる機能にフォーカスして実装する（ただし、互換性を失わない範囲で）。
- IKEv2 = “VPN管理プロトコル”として広義に扱う。
  - 「暗号鍵交換」以外の用途でも、おもしろそうなものやメリットが考えられるなら積極的に利用するという実利的な立場（ただし、互換性は意識して）。

（2）TCPカプセリングは。。。

- 実装例： SSL-VPN , PPP over SSH , etc...
- 現状はサポートする計画なし：保守的な立場
  - 「TCP over TCP」問題
    - 再送制御やフロー制御などが同一のTCPパケットにカプセリングの前と後で重複して実行される副作用。また、リアルタイム系アプリケーション通信に悪影響の可能性。
    - 単一コネクションにVPN上の複数セッションの通信がシリアライズされてしまう、または複数コネクションに割り振られる場合でも大量に消費される可能性。
  - 近年、続々とTCPプロトコルスタックへの脆弱性（少なくともDoS攻撃は受けてしまう）が報告されている。
  - Web/SSH通信に見せかけてファイアウォールを透過できる点は本当に良いこと？？？

（3）柔軟”すぎない”設計：「複雑性は安全の敵」

- IPsec v2(IKEv1+ESPv2)
  - 柔軟で細かいセキュリティ・パラメータの設定が可能。

  = “利用者が細部の設定を理解・意識しなければならない”

  - 設定の複雑さに起因するとも言える脆弱性レポートも存在。
    - 2005年5月

    「NISCC Vulnerability Advisory IPSEC – 004033」
    - メッセージ認証なし、暗号化（CBC-mode）のみのESP設定を行った場合にパケット復号処理後に情報漏えいを引き起こさせる攻撃手法。

      →設定方法を間違えると「意図しない」脆弱性が発生！

（3）柔軟”すぎない”設計：「複雑性は安全の敵」

- （参考）SSL / TLS
  - クライアント側は基本的には設定レス。
  - 暗号/ハッシュアルゴリズム、認証方式をスイートとしてあらかじめビルトイン。
  - セキュリティ強度、後方互換、処理負荷コスト的に最もリーズナブルなものをピア間で交渉する。

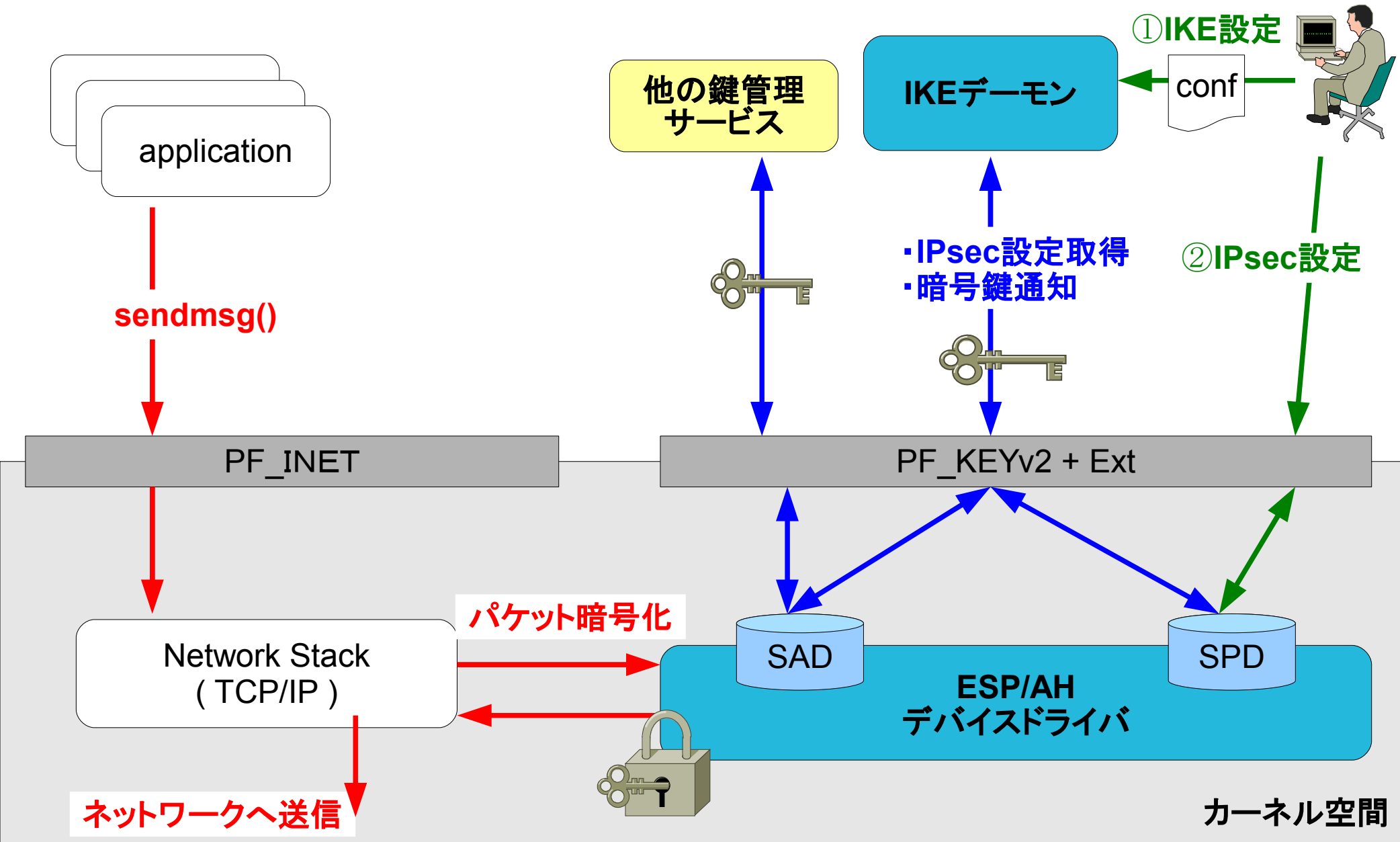
- せっかくIKEv2はシンプルな仕様になった。
  - その設計哲学を忘れずに！
  - 柔軟過ぎる設定の可能性は逆に危険。

Rockhopper : 実装

IPsecの伝統的な実装モデル（1）

- 鍵交換サービス(IKE)と暗号プロトコルスタック(ESP)を相互に独立して実装。
  - IKE：ユーザ空間サービス/デーモン
  - ESP/AH：デバイスドライバ
- プロトコル設計上はIKEv1はIPsec専用のサービスではない。
  - ESP/AH以外の暗号プロトコルにも適用可能。DOIの導入。
  - ESP/AHは他の鍵管理機能から暗号鍵を受取ることも可。
- 主なIPsecオープンソース実装はこのモデルを採用。

IPsecの伝統的な実装モデル(2)



IPsecの伝統的な実装モデル（3）

- メリット
  - 非常に汎用的な設計
  - 暗号化／復号はカーネル空間で実行。効率的かつ高速。
  - IPレイヤやレイヤー2の再実装をせずにシームレスに動作。
- デメリット
  - ESP/AHデバイスドライバを実装する必要。
    - 安全なシステムのためにはできるだけOSとの干渉を避けるべき。
    - デバッグのしやすさ、開発環境の充実度に難あり（？）
    - 特権モード動作の難しさ。他を邪魔しないリソース利用のフェアネスの確保。
    - OS依存の実装
  - IKEデーモンとESP/AHドライバ間の複雑なAPIインタフェース (PF_KEYv2 + Ext)

本ソフトウエアの実装モデル（1）

- 全てをユーザ空間で実装
  - OSに標準搭載のTUN/TAPドライバを利用
    - 暗号プロトコル(ESP)もIKEv2機能も同一ユーザ空間アプリへ実装。
    - レイヤー2のエミュレーションを行う機能。

      よって上位レイヤーであるIPレイヤからは透過的であり、IPレイヤの再実装は必要なし。
    - レイヤー2機能も再実装する必要なし
      - 仮想インタフェース管理
      - ブリッジ
  - 仮想イーサネット機能を実現するために仮想ハブ機能はかかえこむ必要
    - ただし、下位レイヤ（L2）動作のため動作や実装は非常にシンプル。

本ソフトウエアの実装モデル（2）

- メリット
  - OSのセキュリティ機構をフルに利用。「最小権限の基本原則」
    - 実行ユーザ権限の委譲
    - 不要なケーパビリティの放棄
    - OSの保護下で安全に実行。今後セキュアOSトレンドの恩恵も(?)。
  - デバッグが容易。開発ツールも豊富。
  - リソース管理を全てOSへ任せられる。
  - 実装する対象が1バイナリ（DOI廃止。IKEv2は事実上IPsec専用。暗号プロトコル処理部を分離する必然性は薄い）。
- デメリット
  - オーバーヘッド（パケットコピー、コンテキストスイッチ）

TUN/TAPドライバ(Linux 2.6.x) (1)

- “Universal TUN/TAP device driver”
  - 仮想トンネルインタフェースとして動作し、アプリケーションが送受信するパケットをキャプチャしてユーザ空間へ転送するドライバ。
  - linux/drivers/net/tun.c
  - linux/Documentation/networking/tuntap.txt
  - パケットは/dev/tunファイルから読み出し、書き出し。
  - ２つの動作モード
    - Tunnelモード：IPパケットカプセリング
      - IPのみ, Point-to-Point型接続, No-ARP
    - TAPモード：Ethernetフレームカプセリング
      - Ethernetエミュレーション, Multipoint型接続

TUN/TAPドライバ(Linux 2.6.x) (2)

```
$ sudo /sbin/ifconfig tun0 201.10.0.1/24
$ sudo /sbin/route add -net 201.10.1.0/24 gw 201.10.0.10

$ /sbin/ifconfig tun0
tun0  Link encap:イーサネット  ハードウェアアドレス 52:b0:68:2f:a2:a6   ←ランダム割り当て
      inetアドレス:201.10.0.1  ブロードキャスト:201.10.0.255  マスク:255.255.255.0
      inet6アドレス: fe80::50b0:68ff:fe2f:a2a6/64 範囲:リンク
      UP BROADCAST RUNNING MULTICAST  MTU:1500  メトリック:1
      RXパケット:0 エラー:0 損失:0 オーバラン:0 フレーム:0
      TXパケット:0 エラー:0 損失:24 オーバラン:0 キャリア:0
       衝突(Collisions):0 TXキュー長:500
      RXバイト:0 (0.0 B)  TXバイト:0 (0.0 B)

$ /sbin/route -n
カーネルIP経路テーブル
受信先サイト    ゲートウェイ     ネットマスク      フラグ  Metric  Ref  使用数  インタフェース
192.168.1.0     0.0.0.0         255.255.255.0   U       1       0    0       eth0
201.10.0.0      0.0.0.0         255.255.255.0   U       0       0    0       tun0
201.10.1.0      201.10.0.10     255.255.255.0   UG      0       0    0       tun0
169.254.0.0     0.0.0.0         255.255.0.0     U       1000    0    0       eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG      0       0    0       eth0
```

TUN/TAPドライバ(Linux 2.6.x) (2)

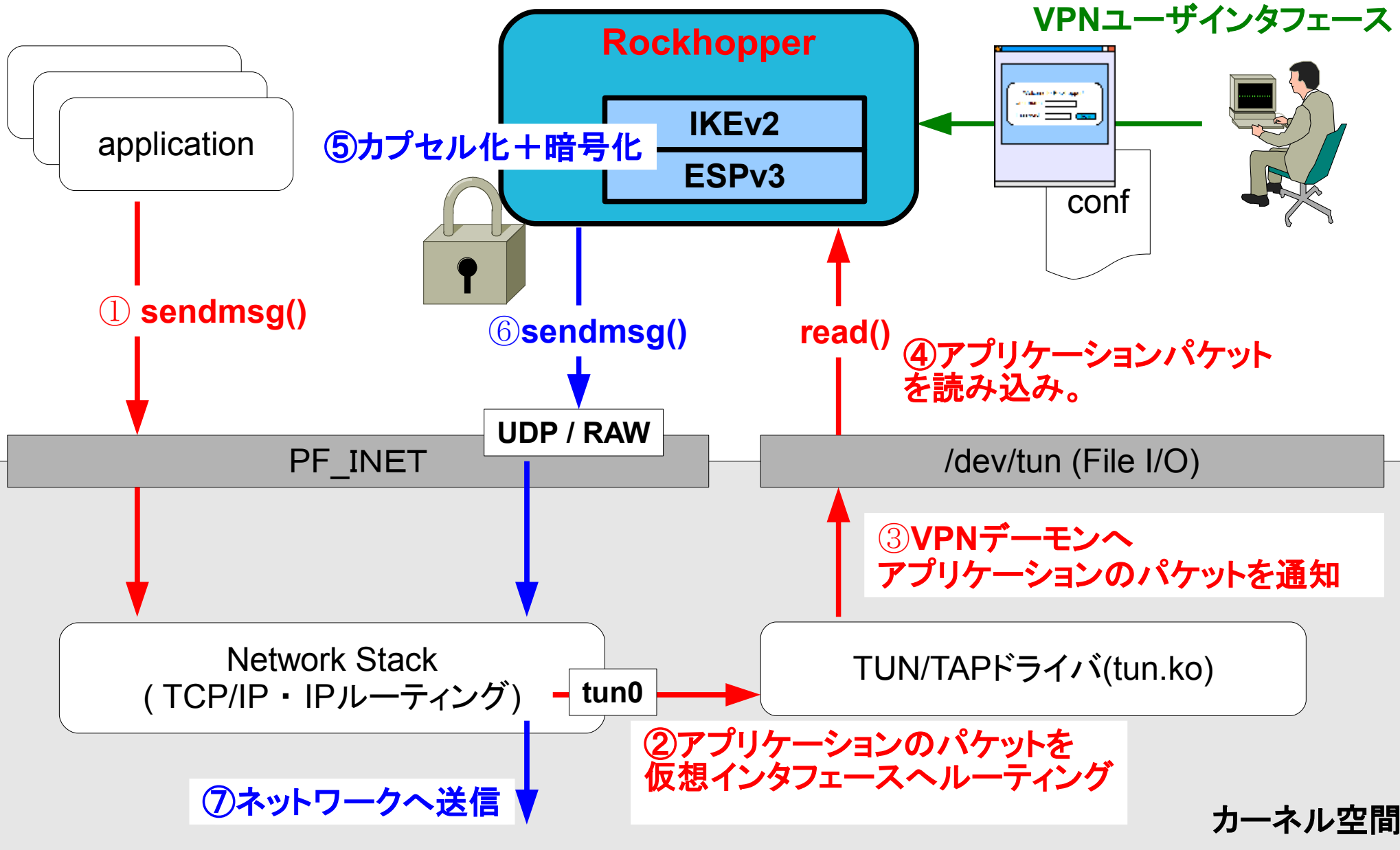
$ ping 201.10.0.10 → routeコマンドで指定したtun0ネットワーク上のネクストホップ

$ arp -vn

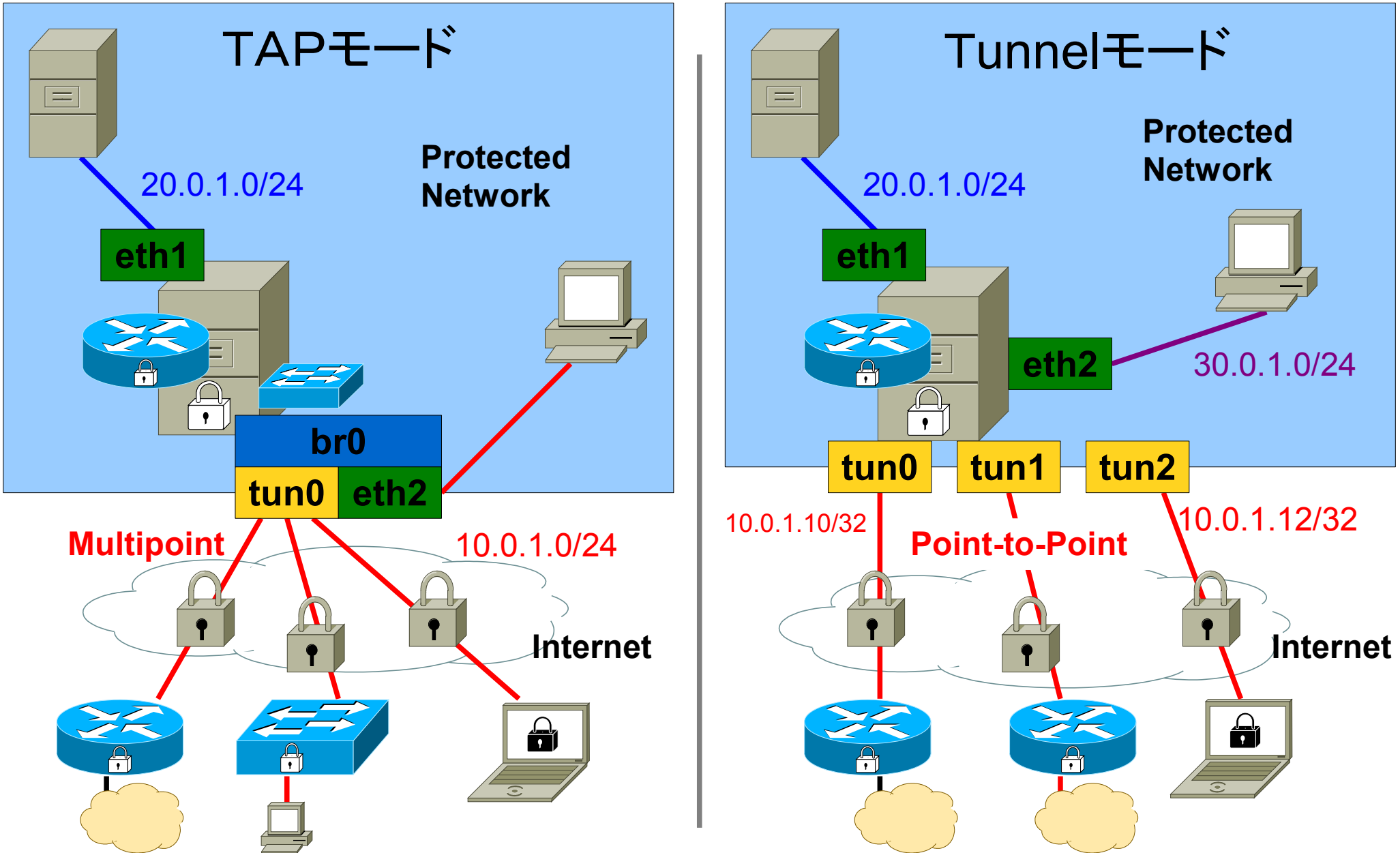
アドレス	HWタイプ	HWアドレス	フラグ	マスク	インタフェース
192.168.1.22	ether	00:xx:zz:xx:xx:aa	C		eth2
192.168.1.23	ether	00:xx:aa:xx:xx:bb	C		eth2
192.168.1.1	ether	00:xx:bb:xx:xx:cc	C		eth2
201.10.0.10		(不完全)			tun0

エントリ: 4　スキップ: 0　発見: 4

TUN/TAP + Rockhopper



TAP vs Tunnel



（問）Tunnelか？TAPか？　または両方か？

- (解)（現状は。。。）TAPモードのみ採用。
  - 仮想イーサネット機能　←本ソフトウエアの大きな目的の一つ
    - Ethernetエミュレーション
    - IPパケット以外の処理も必要。
    - ブリッジ動作が可能。Bridge-utilsを利用できる。
  - 接続先ピア毎に仮想インタフェースを作成する必要なし。
    - マルチポイント型の接続
    - 一つの仮想インタフェースで全ピアを収容可能。
  - ルーター構成とブリッジ構成の両方が可能
  - 両モードのサポートは開発・サポート工数の増大へ

脱線（2）TUN/TAPドライバのMACアドレス生成方法

– linux/drivers/net/tun.c –：かなりシンプル。。。

・～Linux 2.6.26：32bitsのランダム値

```
/* Generate random Ethernet address. */
u8 addr[6];
*(__be16 *)addr = htons(0x00FF);
get_random_bytes(addr + sizeof(u16), 4);
```

・Linux 2.6.27～：46bitsのランダム値

```
static inline void random_ether_addr(u8 *addr)
 {
    get_random_bytes (addr, ETH_ALEN); /*ETH_ALEN:6bytes*/
    addr [0] &= 0xfe;       /* clear multicast bit */
    addr [0] |= 0x02;       /* set local assignment bit (IEEE802) */
 }
```

脱線（2）TUN/TAPドライバのMACアドレス生成方法

- アドレス衝突がちょっと心配かも。。。
  - （単純ですが。。。）バースデーパラドックスに従って、衝突する確率が50%を超えるノード数（アドレス数）を計算。
    - ただし、乱数生成器が十分な性能を持っていると仮定。

# 割り当てbit数	ノード数（アドレス数）	確率	
# 8	20	[0.533167]	
# 16	302	[0.500722]	
# 24	4823	[0.500009]	
# 28	19291	[0.500002]	
# 32	**77164**	[0.500009]	～ Linux 2.6.26
# 36	308652	[0.500002]	
# 40	1234605	[0.500000]	
# 42	2469209	[0.500000]	
# 46	**9876832**	[0.500000]	Linux 2.6.27～

- ちなみにノード数（アドレス数）が10,000の場合の衝突確率は以下の通り。
  - ～ Linux 2.6.26 (32bits) : 0.011573
  - Linux 2.6.27～ (46bits) : 0.000001

仮想イーサネット機能

- IPsec上にレイヤー2VPNを構築。
  - IPレイヤー上に仮想的なイーサネットをオーバーレイ
    - 社内と社外を安全に「ブリッジ」接続
    - 小中規模VPN：ルーティング設計なしでVPNを運用できて便利。
  - 仮想Hub機能
    - ピアを端末ノード、IPsecトンネルを擬似的なワイヤとして仮想的に収容。
    - MACアドレス学習：ピアのMACとVPN接続のマッピング
    - フラッディング：MAC未学習時、ブロードキャスト、マルチキャスト
    - ARPキャッシュ / プロキシ（代理応答）機能
  - カプセリング方式：「EtherIP」 シンプル！ (参考) L2TPv3
    - ただし、以下で機能を補強。
      - **MTU値の交渉：暗号/ハッシュアルゴリズム次第で可変なためIKEのネゴ中に決定。**
      - **パケットシーケンス制御：ESPヘッダのリプレイ攻撃防御用のシーケンス番号を利用。順序が入れ替わったパケットをドロップ可能。**

カプセリング方式

- 2つのカプセリング方式で対応
  - Ethernetフレームカプセリング： EtherIP(RFC3378)
    - IPsecトランスポートモード
    - IP以外の任意のプロトコルを転送可能

  - IPカプセリング： IP over IP
    - つまり、IPsecトンネルモード

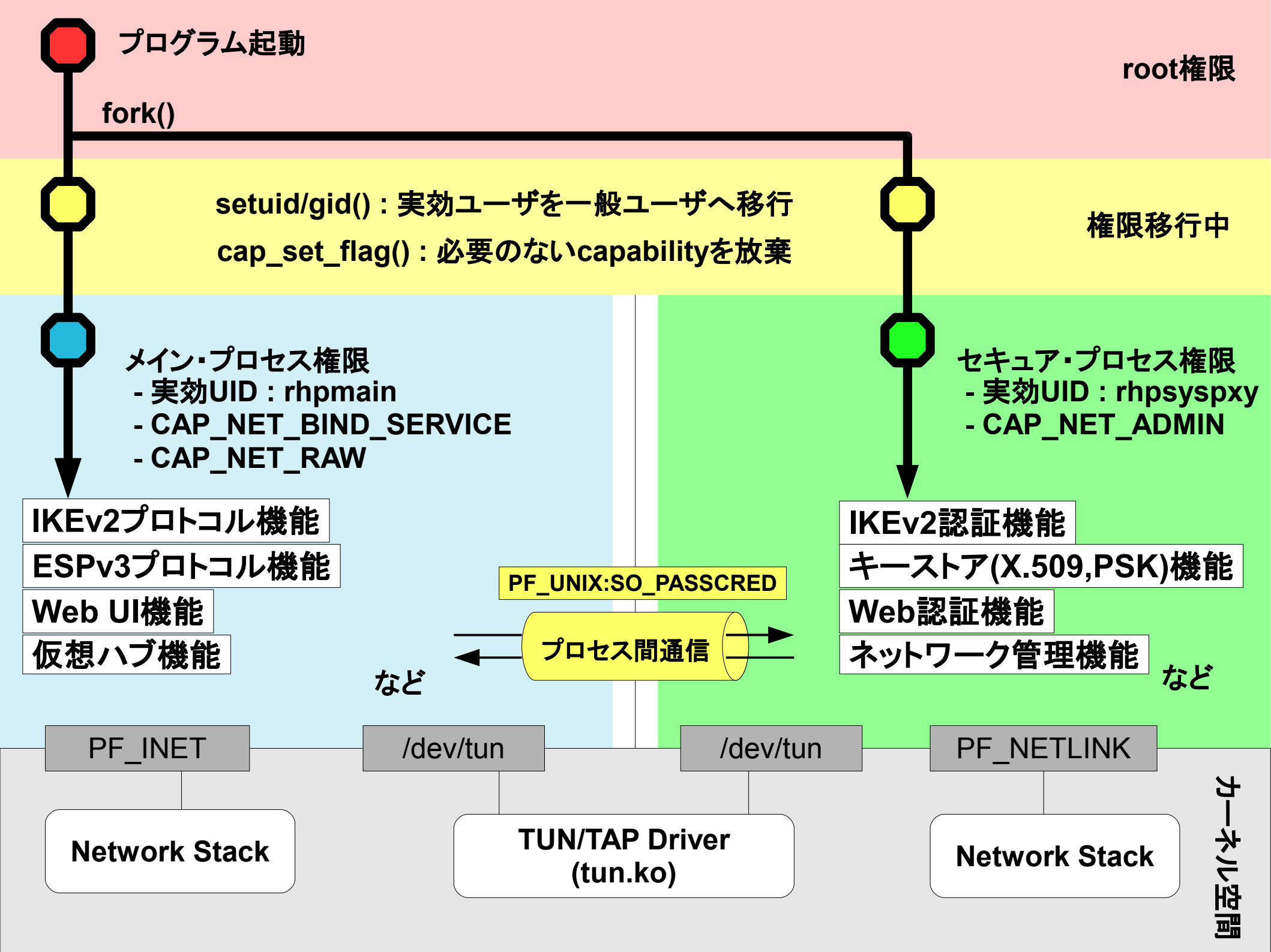
  - NATトラバーサルの場合：UDPカプセリングも。

IPカプセリング時のTAP転送

- TAPモードではEthernetエミュレーション。
  - IPカプセリング時：パケット復号後にEthernetヘッダがない。
- 擬似Ethernetヘッダを付加してTAPへ転送。
  - 送信元MAC：
    - IPsecトンネル接続時にピア毎に擬似アドレスを生成。
    - 仮想HubのARPプロキシが代理応答。
  - 送信先MAC：
    - 転送先ピアのアドレスは仮想Hubがキャッシュまたは ARP解決。
- ピュアなIPsec実装との相互接続性を確保するしかけ
  - 通常、他の実装はEtherIPをサポートしていない。
  - ただし、もちろんIPプロトコルのみ可能。

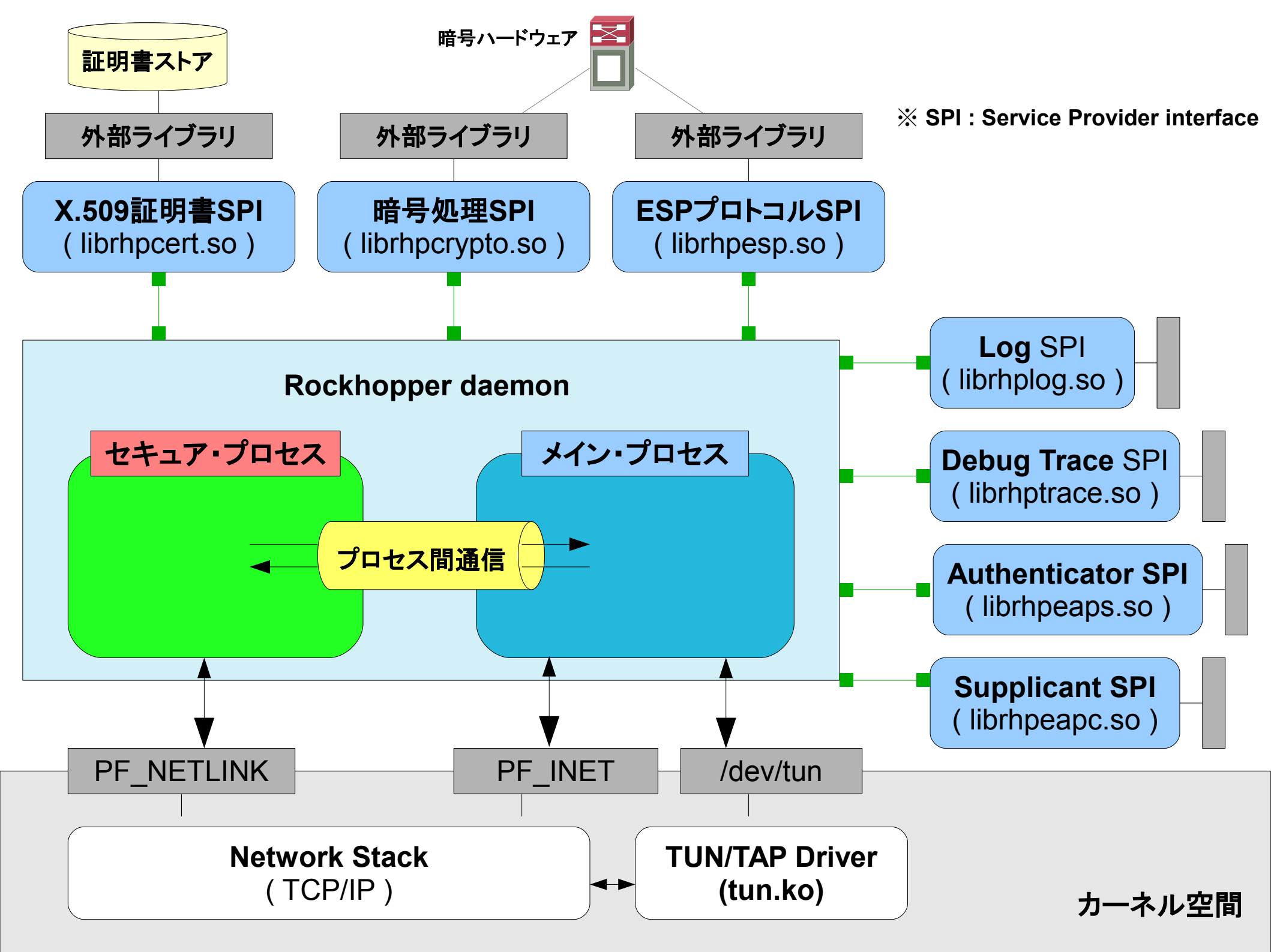
プロセスの分離によるセキュアな設計

- 「最小権限の基本原則」
- OSのセキュリティ機能を最大限活用
  - 伝統的なベストプラクティスに素直に従う。
    - 実効ユーザを起動後すぐに一般ユーザへ移行。
    - 不要なケーパビリティは起動後すぐに放棄。
    - セキュリティ的に重要な機能を別プロセス、別ユーザ権限で実行。
    - 外部（ネットワーク）へアクセスする機能を持つプロセスから隔離。
    - 必要な機能間コミュニケーションはプロセス間通信で実行。
  - 利用機能
    - setuid/gid()
    - capabilities
    - SO_PASSCREDソケットオプション(PF_UNIX)　　など


プログラム起動
root権限
fork()
setuid/gid() : 実効ユーザを一般ユーザへ移行
cap_set_flag() : 必要のないcapabilityを放棄
権限移行中
メイン・プロセス権限
- 実効UID : rhpmain
- CAP_NET_BIND_SERVICE
- CAP_NET_RAW
セキュア・プロセス権限
- 実効UID : rhpsyspxy
- CAP_NET_ADMIN
IKEv2プロトコル機能
ESPv3プロトコル機能
Web UI機能
仮想ハブ機能
など
PF_UNIX:SO_PASSCRED
プロセス間通信
IKEv2認証機能
キーストア(X.509,PSK)機能
Web認証機能
ネットワーク管理機能
など
PF_INET
/dev/tun
/dev/tun
PF_NETLINK
Network Stack
TUN/TAP Driver
(tun.ko)
Network Stack
カーネル空間

カスタマイズしやすい設計

- ベンダ依存度また運用システム依存度の高い機能をSPI(Service Provider Interface)を通じてプラグイン可能な構造。
  - X.509証明書SPI
  - 暗号処理 SPI
  - ESPプロトコル SPI
  - EAP機能 SPI
  - デバッグトレース / ログ機能 SPI


証明書ストア
暗号ハードウェア
※ SPI : Service Provider interface
外部ライブラリ
外部ライブラリ
外部ライブラリ
X.509証明書SPI
(librhpcert.so)
暗号処理SPI
(librhpcrypto.so)
ESPプロトコルSPI
(librhpesp.so)
Rockhopper daemon
セキュア・プロセス
メイン・プロセス
プロセス間通信
Log SPI
(librhplog.so)
Debug Trace SPI
(librhptrace.so)
Authenticator SPI
(librhpeaps.so)
Supplicant SPI
(librhpeapc.so)
PF_NETLINK
PF_INET
/dev/tun
Network Stack
(TCP/IP)
TUN/TAP Driver
(tun.ko)
カーネル空間

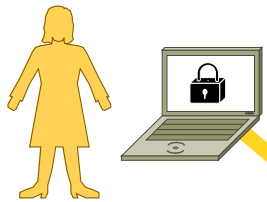
Role / IDベースのVPNポリシー管理

- セキュリティポリシーを共有するノードを”VPNレルム”へロール化。ロール毎に設定を行うモデルを採用。
  - ピア毎に設定を記述する運用モデルはメンドウ。
  - マルチホスティングを実現
    - 同一IP/Port番号上で複数のロールを定義可能。
  - ロールはIKE_AUTHで交換されるID値への部分マッチフィルタで識別。
    - 設定可能な部分マッチ条件
      - E-Mailアドレスのサブドメイン部 (*@sales.hoge.com)
      - FQDNのサブドメイン部 (*.sales.hoge.com)
      - X.509証明書SubjectのRDNの組み合わせ (O=hoge,OU=sales)
      - X.509証明書SubjectAltName: E-Mail / FQDN (上述と同様)

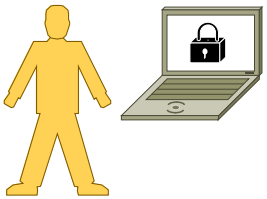
（例）*@sales.hoge.com を含むID ==>「営業部レルム」へマップ

Role/IDベースのVPNポリシー管理

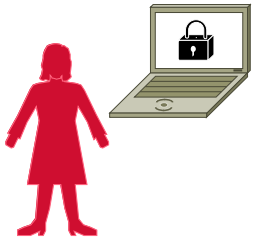
*@sales.hoge.com	営業部	VPN設定1	キーストア1
*@dev.hoge.com	開発部	VPN設定2	キーストア2
...	...	...	...



hanako@sales.hoge.com

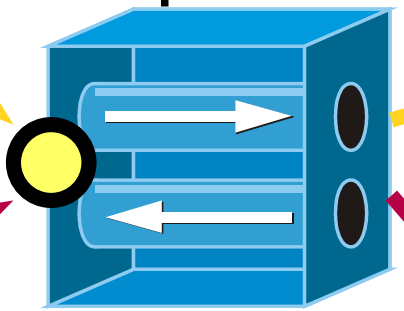


taro@sales.hoge.com

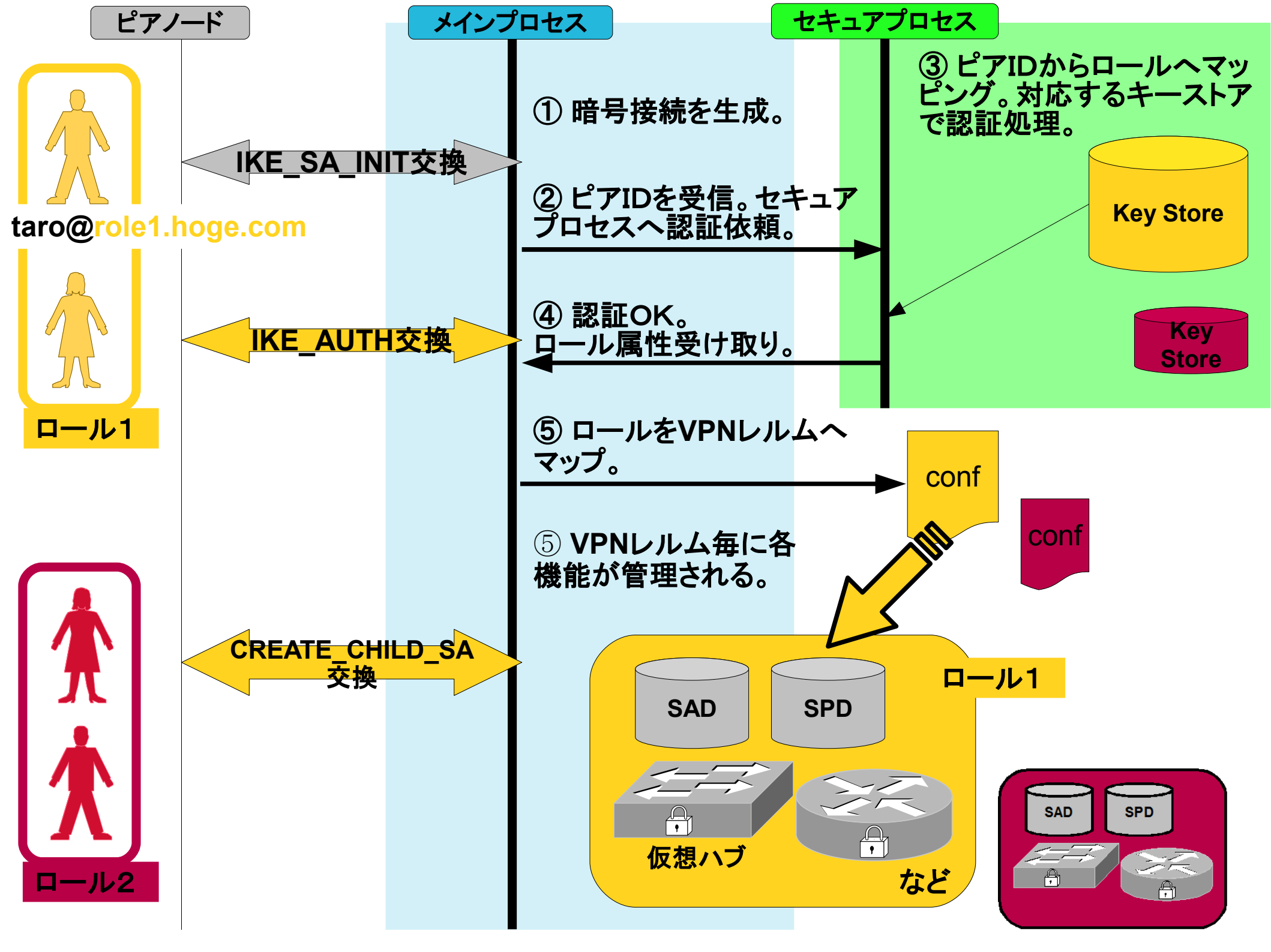


keiko@dev.hoge.com

manabu@dev.hoge.com

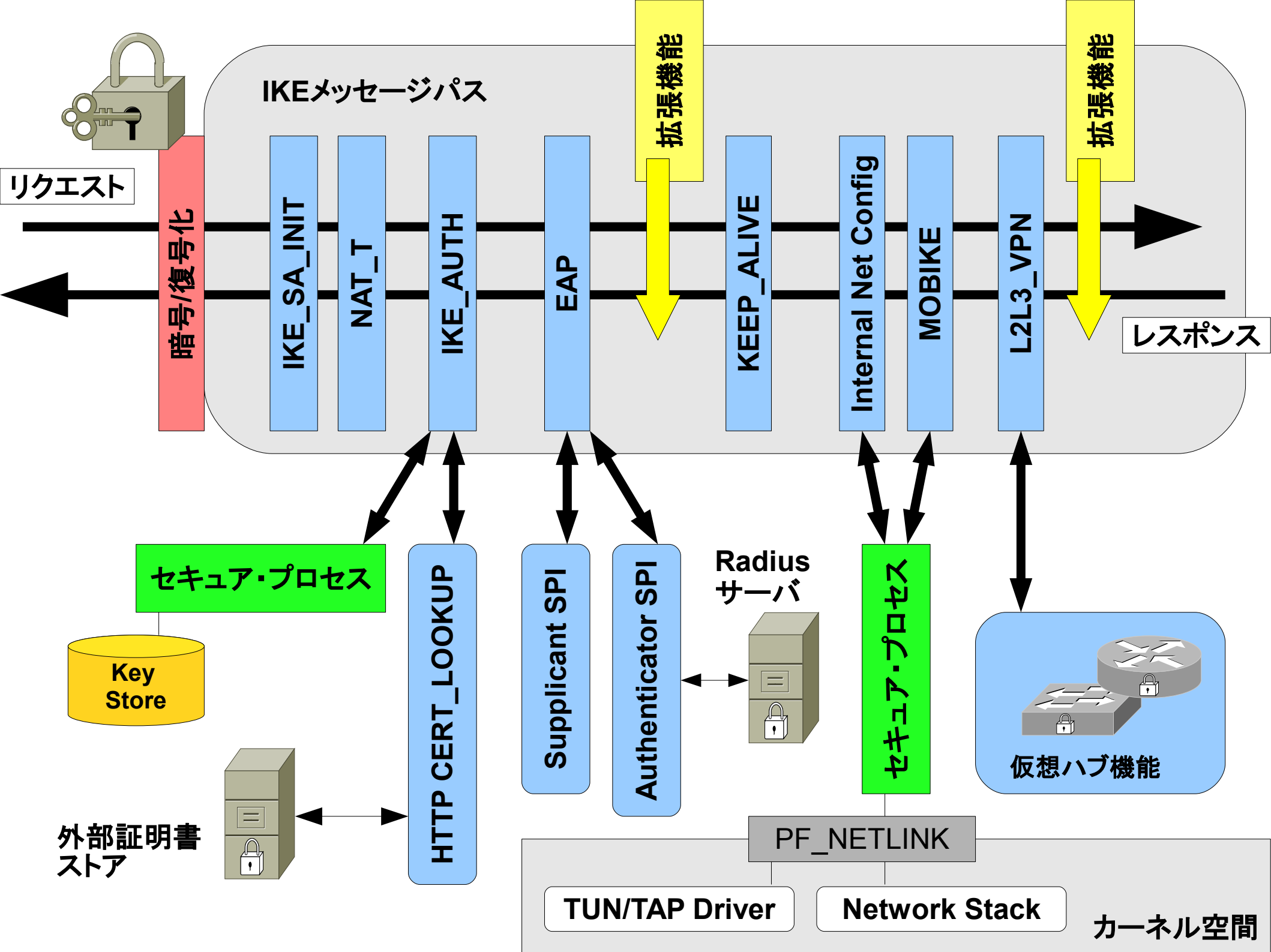


同一IPアドレス/ポート番号
でマルチホーミング


ピアノード
メインプロセス
セキュアプロセス
taro@role1.hoge.com
ロール1
ロール2
IKE_SA_INIT交換
IKE_AUTH交換
CREATE_CHILD_SA
交換
① 暗号接続を生成。
② ピアIDを受信。セキュア
プロセスへ認証依頼。
③ ピアIDからロールへマッ
ピング。対応するキーストア
で認証処理。
Key Store
Key
Store
④ 認証OK。
ロール属性受け取り。
⑤ ロールをVPNレルムへ
マップ。
conf
conf
⑤ VPNレルム毎に各
機能が管理される。
SAD
SPD
ロール1
仮想ハブ
など
SAD
SPD

拡張機能の追加を容易に

- IPsec仕様はまだまだ進化し続けている。
  - IETF ipsecmeワーキングループが「ツカエル」拡張を議論中。
  - やりたいことはまだたくさんある。
- Linux / Netfilterのようなメッセージパスとメッセージハンドラによるプラグイン設計をIKEv2処理部へ導入。
- 実際、IKEv2のそれぞれの機能はこのしくみを利用して実装されている（次図の青色部分）。


IKEメッセージパス
拡張機能
拡張機能
リクエスト
暗号/復号化
IKE_SA_INIT
NAT_T
IKE_AUTH
EAP
KEEP_ALIVE
Internal Net Config
MOBIKE
L2L3_VPN
レスポンス
セキュア・プロセス
Key Store
HTTP CERT_LOOKUP
Supplicant SPI
Authenticator SPI
Radius サーバ
セキュア・プロセス
仮想ハブ機能
外部証明書 ストア
PF_NETLINK
TUN/TAP Driver
Network Stack
カーネル空間

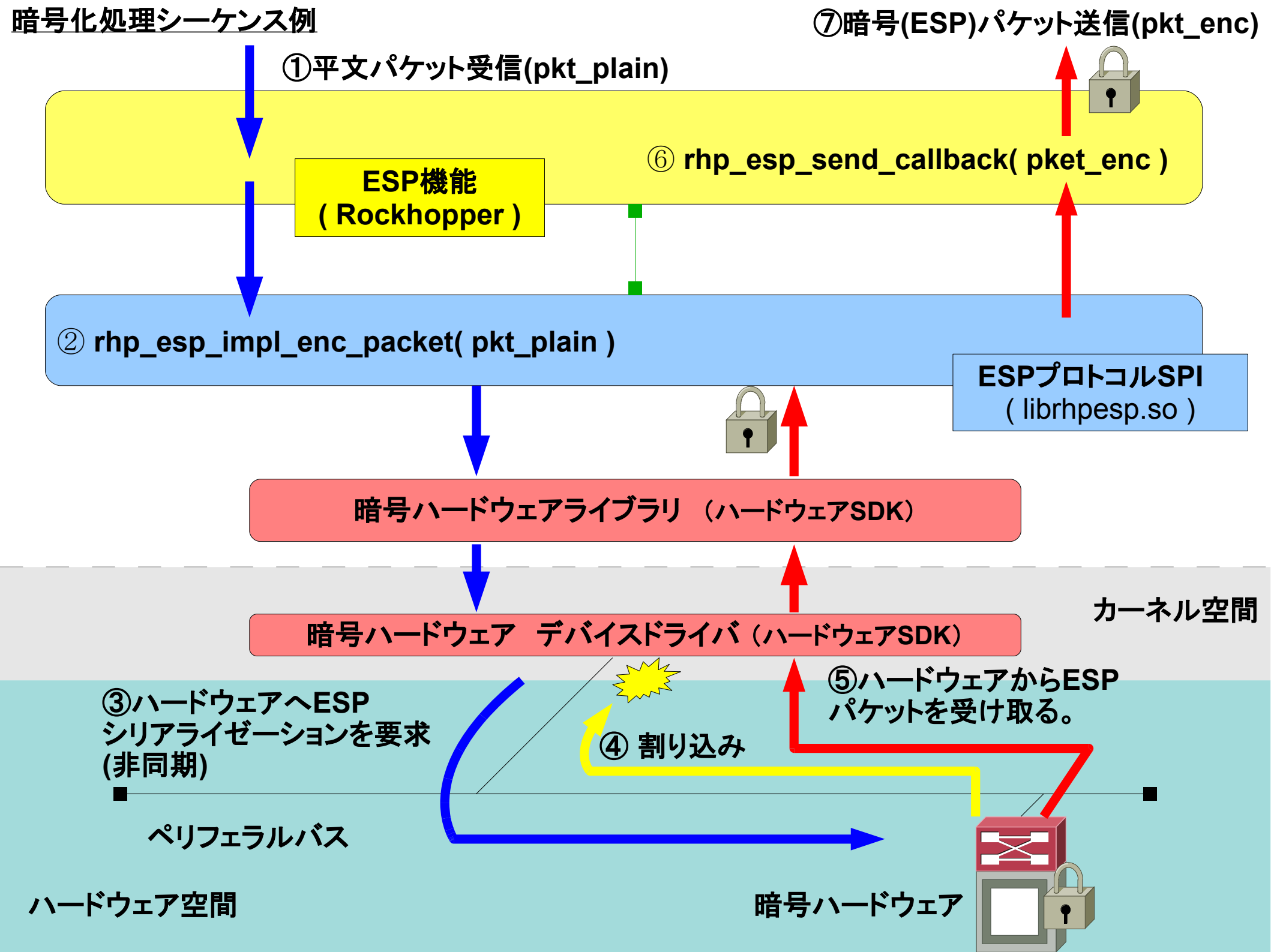
暗号オフロードハードウェアとの連携

- IPsec処理はCPUへ負担をかける。
  - 暗号鍵の生成（Diffie-Hellman）、パケットの暗号化/復号、ハッシュ計算
- 専用ハードウェアへオフロードするアプローチ

  (1) アルゴリズム（暗号・ハッシュ・鍵生成）処理をオフロード。

  (2) (1)に加え、ESP/AHパケットのシリアライゼーション・デシリアライゼーションもオフロード。

  (3) (2)の機能をNIC上でインライン高速処理。
- しかし、通常はハードウェアベンダ毎に異なるSDK（ライブラリ）を利用する必要あり。
- SPIを実装したプラグインの組み込みで柔軟に対応
  - 暗号プラグインSPI
  - ESPプラグインSPI


暗号化処理シーケンス例
⑦暗号(ESP)パケット送信(pkt_enc)
①平文パケット受信(pkt_plain)
⑥ rhp_esp_send_callback(pket_enc)
ESP機能
(Rockhopper)
② rhp_esp_impl_enc_packet(pkt_plain)
ESPプロトコルSPI
(librhpesp.so)
暗号ハードウェアライブラリ （ハードウェアSDK）
カーネル空間
暗号ハードウェア デバイスドライバ（ハードウェアSDK）
③ハードウェアへESP
シリアライゼーションを要求
(非同期)
④ 割り込み
⑤ハードウェアからESP
パケットを受け取る。
ペリフェラルバス
ハードウェア空間
暗号ハードウェア

AJAX(Comet)ベースのWeb管理インタフェース

- UIのデザインと実装は難しい、特にGUIは。。。

  （例）

  - Site-to-Site型（拠点間）VPN用の管理者用のビュー
  - リモートアクセス型VPNのクライアント用Viewはゲートウェイ管理者用のビューとは違ってシンプルで良い。
  - 特定の運用環境用のカスタマイズが必要かもしれない。
    - 社内管理システムへの統合など
  - 個人利用ではビューのパーソナライズの要望
  - ローカライズの問題

AJAX(Comet)ベースのWeb管理インタフェース

- 利用者に委ねてしまおう。
  - Webベースなら開発やカスタマイズもしやすい。

- ユーザの操作例（将来検討も含めて）
  - VPNの接続・切断などの操作
  - ユーザ認証インタフェース
  - ソフトウェアの設定
  - ログやモニタリングの閲覧
  - ピアのロケーション検索操作（予定）

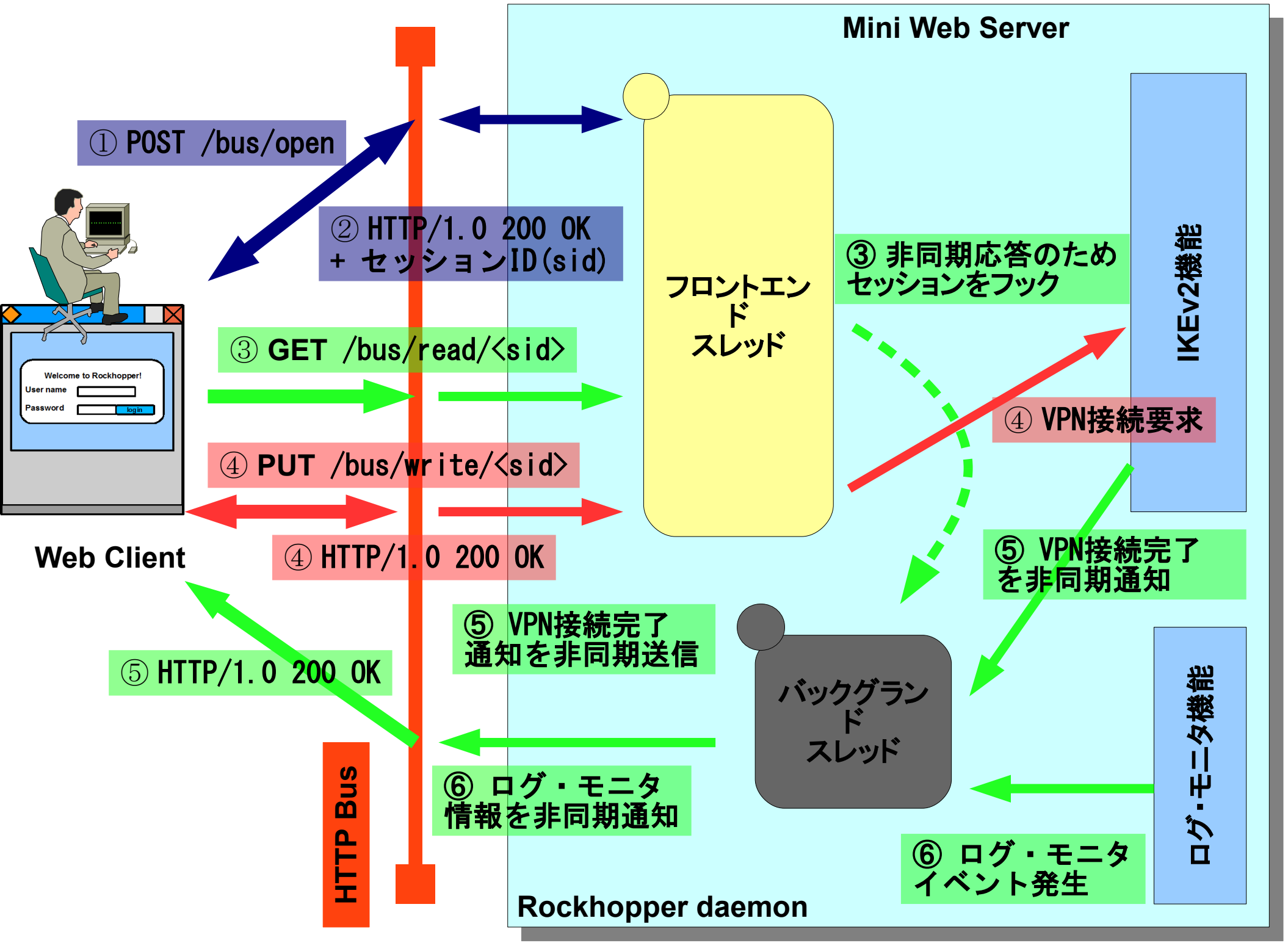
AJAX(Comet)ベースのWeb管理インタフェース

- AJAXの採用
  - データ交換、内部情報形式にXMLを採用
  - デーモン側で発生した非同期イベントを扱う必要あり。

    “Comet”を採用。
    - HTTPサーバがクライアントからのリクエストをイベント発生までフックし応答することで擬似的に非同期通信を実現。
- 専用のライトウェイトなHTTPサーバ機能を内蔵
  - Cometライク動作を想定：
    - 受信メッセージを逐次処理するスレッドとフックされるセッションを監視するタイマースレッドの2スレッドのみで構成。
  - Web認証：現状はBasic認証方式のみ
  - 接続元の制限機能（デフォルトでは127.0.0.1からのみ許可）

AJAX(Comet)ベースのWeb管理インタフェース

- RHP Bus Protocol :
  - HTTP 1.0上で動作、XMLを使った独自メッセージ仕様
  - Bus I/O メタファー
    - “Open” : POST /bus/open
      - Web認証後、サーバとBusセッションをオープン。
    - “Write” : PUT /bus/write/<SesssionID> + <XML Mesg>
      - メッセージをBusへ書き込む。
    - “Read” : GET /bus/read/<SessionID> + <XML Mesg>
      - メッセージをBusから読み込む。
    - “Close” : DELETE /bus/close/<SessionID>
      - Busセッションをクローズする。


Mini Web Server
① POST /bus/open
② HTTP/1.0 200 OK
+ セッションID(sid)
フロントエンド
スレッド
③ 非同期応答のため
セッションをフック
IKEv2機能
③ GET /bus/read/<sid>
④ VPN接続要求
④ PUT /bus/write/<sid>
Welcome to Rockhopper!
User name
Password
login
Web Client
④ HTTP/1.0 200 OK
⑤ VPN接続完了
を非同期通知
⑤ VPN接続完了
通知を非同期送信
⑤ HTTP/1.0 200 OK
バックグランド
スレッド
ログ・モニタ機能
HTTP Bus
⑥ ログ・モニタ
情報を非同期通知
⑥ ログ・モニタ
イベント発生
Rockhopper daemon

マルチコアCPU環境への対応

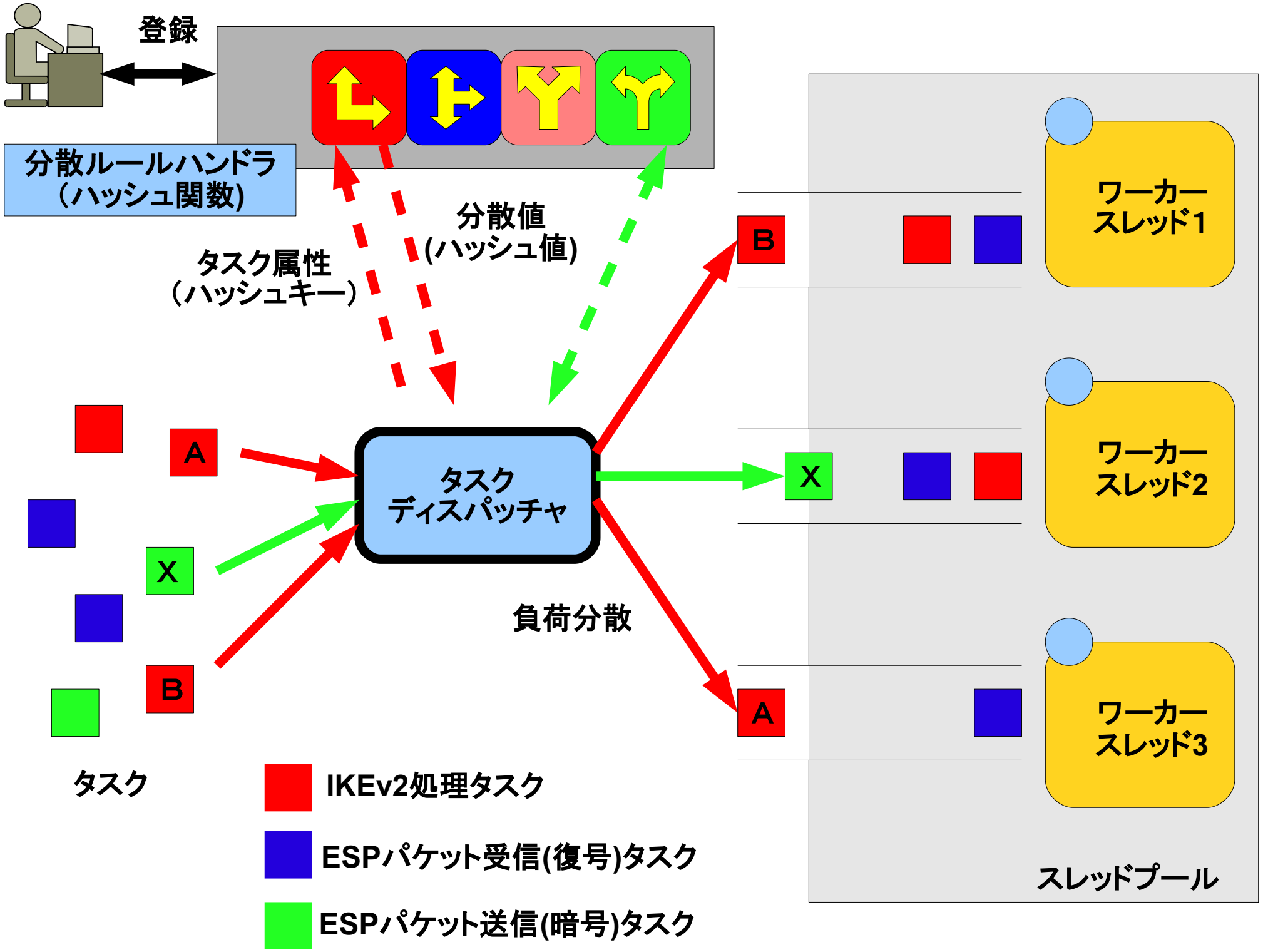
- 重い暗号処理の負荷分散に主眼
- ワーカースレッドプールを利用
  - なるべくシンプル構造で！
  - 粒度のなるべく粗い形での処理分散
- ルールベースのタスク負荷分散
  - 負荷分散の方法は機能毎に、実行環境毎、運用環境毎に異なる可能性。
    - どのようにそれぞれのワーカスレッドに分散するを機能毎、状況毎、条件毎に柔軟に。
  - 一度分散を開始したらそのコンテキストに属する処理の分散先の一意性を確保できること。

ルールベースのタスク負荷分散機構

- タスクディスパッチャが各タスクをプール内のワーカースレッドへ割り当て
- 処理の分散をしたい各機能がやること
  - 自身のタスクのための負荷分散アルゴリズムや一意性保証処理を記述したルールハンドラ関数を実装し、タスクディスパッチャへ登録。
  - コンテキスト値とともに負荷分散を要求。
  - タスクディスパッチャがルールハンドラをコールバック。それが返す値により分散先ワーカースレッドを決定、タスクをキューする。
    - ルールハンドラは一種のハッシュ関数。
  - ワーカスレッドが順次キューされたタスクを実行。

ルールベースのタスク負荷分散の例

- ESP処理部（ESPプロトコルSPIの実装部分）
  - 暗号化タスク:
    - 平文パケットの送信先IPアドレス値により分散（ハッシュ値）、一意性保証(リプレイ攻撃防御のため)
  - 復号タスク
    - ESPヘッダのSPI値により分散（ハッシュ値）、一意性保証（パケット送信の順序入れ替えをなるべく防ぐ）
- 他に考慮できるカスタマイズ方法
  - VPNレルム毎に特定のスレッドへ分散
    - VPNレルム毎の性能をある程度保証するため。
  - 暗号オフロードハードウェアのAPI
    - 同期API：ワーカースレッド数を多めに。
    - 非同期API：ワーカースレッド数は少なくても可（？）。


登録
分散ルールハンドラ
（ハッシュ関数）
分散値
(ハッシュ値)
タスク属性
（ハッシュキー）
A
X
B
タスク
ディスパッチャ
負荷分散
B
X
A
ワーカー
スレッド1
ワーカー
スレッド2
ワーカー
スレッド3
スレッドプール
タスク
IKEv2処理タスク
ESPパケット受信(復号)タスク
ESPパケット送信(暗号)タスク

ルールベースのタスク負荷分散

- メリット
  - 機能間依存性をなるべく排除したシンプルな構造。
    - 粒度の粗いレベルでの処理分散
  - 実効環境毎にパフォーマンスチューニングが必要になっても変更が可能。
    - その環境に適した分散アルゴリズムや一意性保証処理をルールハンドラ関数へ実装、入れ替えれば良い。
    - ワーカースレッド数を調整。

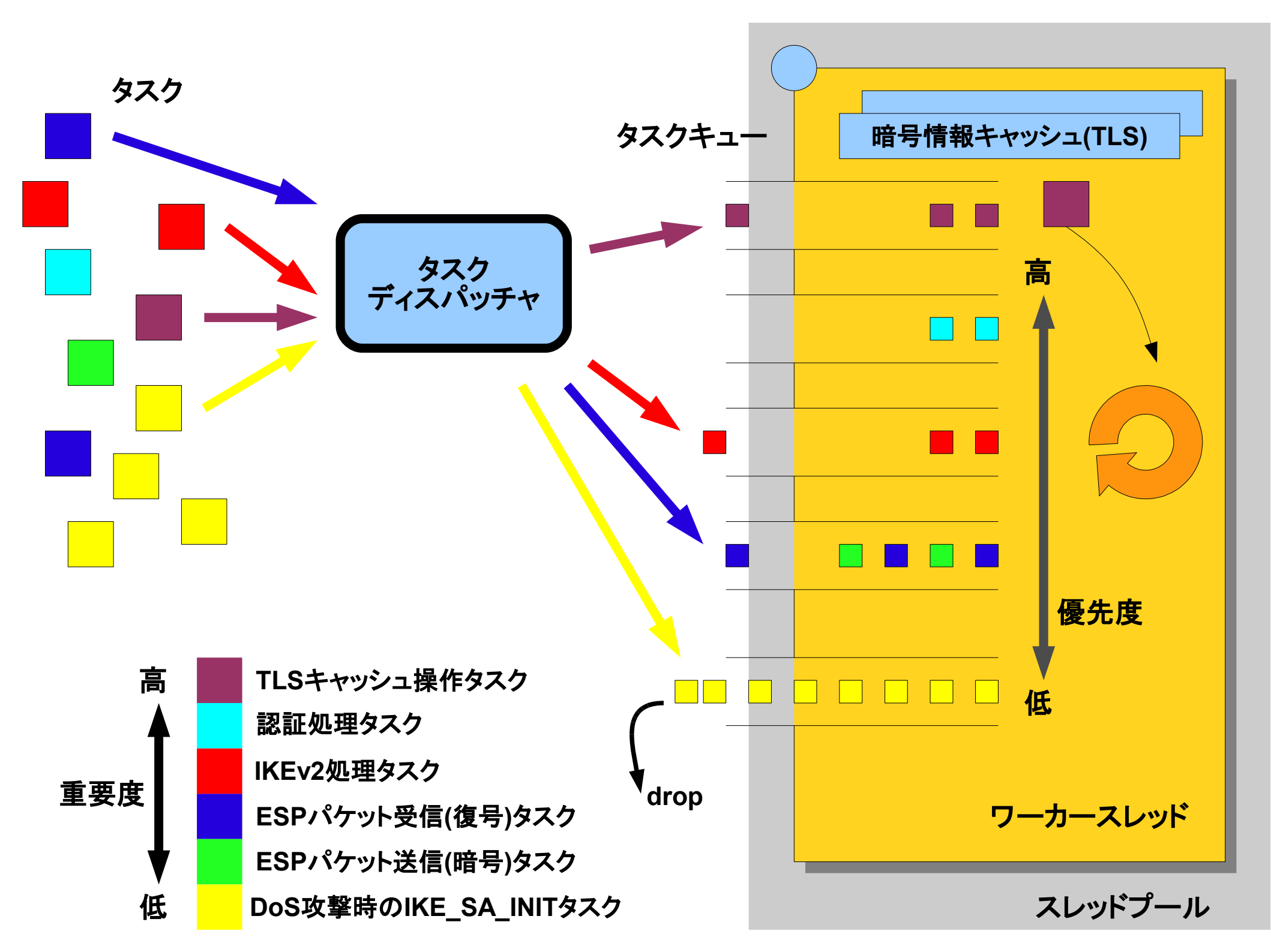
→ある程度自由度の高いカスタマイズを行えるような構造。

タスク実行の優先制御

- （例）大量の暗号トラフィック(ESP)が発生。
  - リソースの割り当てが追いつかず。。。
  - IKEv2のメッセージ処理がなかなか実行されない。。。
  - Linuxでも発生する場合あり。
    - LinuxではNAPIの導入などである程度軽減されてきが、ESP処理はソフト割り込みコンテキストで負荷の重い処理（暗号化、復号）を行うため？？？
  - 管理プロトコルであるIKEv2やダイナミックルーティングプロトコルなどは本来優先的に処理したい。

タスク実行の優先制御

- スレッドプール内の各ワーカースレッドは実行優先度毎にタスクキューを持つ。
- タスクディスパッチャは利用機能に指示された優先度のキューへタスクをつなげる。


タスク
タスク
ディスパッチャ
タスクキュー
暗号情報キャッシュ(TLS)
高
優先度
低
drop
ワーカースレッド
スレッドプール
高
重要度
低
TLSキャッシュ操作タスク
認証処理タスク
IKEv2処理タスク
ESPパケット受信(復号)タスク
ESPパケット送信(暗号)タスク
DoS攻撃時のIKE_SA_INITタスク

タスク実行の優先制御の利用例

- IKEのセッション開始時にハーフオープンの中途半端なネゴシエーション状態をレスポンダ側へ残す類のDoS攻撃。
  - TCP SYN Flood攻撃と同じ発想。
- IKE_SA_INIT交換時にステートレスクッキーをレスポンダがイニシエータへ返すことで攻撃を緩和する。
  - TCP SYN Cookiesと同じ発想。
- ハーフオープンのセッション数が閾値を超えるとこの機能が動作開始。
  - 処理は専用のスレッド内で、かつ最低実行優先度で。
  - 確立済みの他のVPN処理への影響をできるだけ緩和。

Thread Local Storage(TLS)を利用する試み

- 現在、ESP処理部では暗号化、復号で別々のワーカースレッドへ分散する。
  - 双方向通信のパフォーマンスの確保
- 暗号鍵、復号鍵の管理する内部テーブルへの競合が発生！
  - 同時に発生した場合、片方のワーカースレッドはロック(Mutex)の開放待ちに。
- 鍵情報は各ワーカースレッド毎のTLS領域にキャッシュ。その後はロックなしで処理を実行。
  - TLSにキャッシュされた情報を更新、クリアする操作タスクは前述の最優先キューで実行、遅延をなるべく防ぐ。

その他（詳細は論文をご覧ください）

- モバイル環境を前提とした設計(MOBIKE)
- VPN接続の冗長化（マルチホーミング）
- ユーザ端末のリモート・コンフィグレーション
  - Configuration Payload交換
  - 仮想イーサネット上でDHCPを運用。
- ブロードバンド・インターネット対応
  - 暗号/ハッシュ方式毎に(P)MTUが可変。
  - Path MTU Discovery
  - TCP MSSの自動書き換え

その他

- EAPモジュールとのSPI経由での連携
  - Supplicant SPI Plugin
  - Authenticator SPI Plugin
- Peer-to-Peer(P2P)型通信の支援機能(予定)
  - ロケーションサービス（ピアアドレスのダイナミックな発見）
  - NATトラバーサル機能
    - strongSwanでは実はすでに一部実装されている。

将来検討中（詳細は論文をご覧ください）

- 将来検討中
  - Web GUIのサンプル実装の提供
    - Dojo Toolkitを利用した開発を開始したばかり。
  - 多様な暗号アルゴリズムへの対応
  - トンネルレス型VPN

    マルチキャスト用IPsec仕様を利用した軽いVPN
  - ipsecme-wgの拡張仕様への対応
    - リモートアクセスVPNゲートウェイの負荷分散
    - セッション・レジューム機能
  - CPU/プロセッサ アフィニティ

などなど

（将来）: End-to-End Security

- End-to-End(Peer-to-Peer)の安全な通信
  - IPsecの元来の目的？
  - RFC4306 “Internet Key Exchange (IKEv2)Protocol”

    “Although this scenario may not be fully applicable to the IPv4 Internet, it has been deployed successfully in specific scenarios within intranets using IKEv1. It should be more broadly enabled during the transition to IPv6 and with the adoption of IKEv2.”

実際には普及しているとは言えない。。。

（将来）: End-to-End Security

- ”End-to-End Security and Transparency”の哲学をサポートする実装
  - ”The complexity of the Internet belongs at the edges, and the IP layer of the Internet should remain as simple as possible.” - RFC3439 -
  - セキュアなEnd-to-End間接続をもっと容易に実現する暗号トランスポートの実験実装を提案

    ー最終のゴールー

（将来）: End-to-End Security

- これまでのIPsec仕様のデザイン方向
  - ネットワークレイヤーのアプリにフォーカス（？）
    - ビジネス的にも。
    - Site-to-Site型（拠点間）VPN
    - リモートアクセス型VPN
    ］に主眼。
    - 標準化プロセスの歴史的な経緯からベンダーロックインも少なからず発生？？？
  - エンドホスト（PC，サーバ）のIPsec機能
    - スタティックな設定モデル
    - ネットワークレベル(L3)の粗い設定モデル
    - 制約の多い、柔軟性に欠ける実装モデル

（将来）: End-to-End Security

- エンドホストで利用するための実装モデル
  - (1) スタティックな設定モデル（現在）
    - あらかじめトラフィック条件が明らかな場合。
  - (2) Programmableモデル（チャレンジ？）
    - IPsecをアプリケーション開発に組み込るように。
      - （参考）SSL

（将来）: End-to-End Security

- 現状、以下のような場合にIPsecは有効とは言えない。。。

  (例)

  - 「このプロセス」のトラフィックを保護。
  - 「このユーザ属性」のトラフィックを保護。
  - アプリケーション毎に別々のX.509証明書（例：異なるCAから発行された証明書）を利用
  - 同一のアプリ間でダイナミックに変化するポート番号を利用する。（参考） SIP + RTP
  - 通常は軽い平文で通信。重要な情報のやりとりを行う場合だけ暗号で通信。

  - 多くの場合、アプリ毎のセキュリティレイヤーの実装
  - SSL(TLS)/DTLSを利用したアプリ開発

（将来）: End-to-End Security

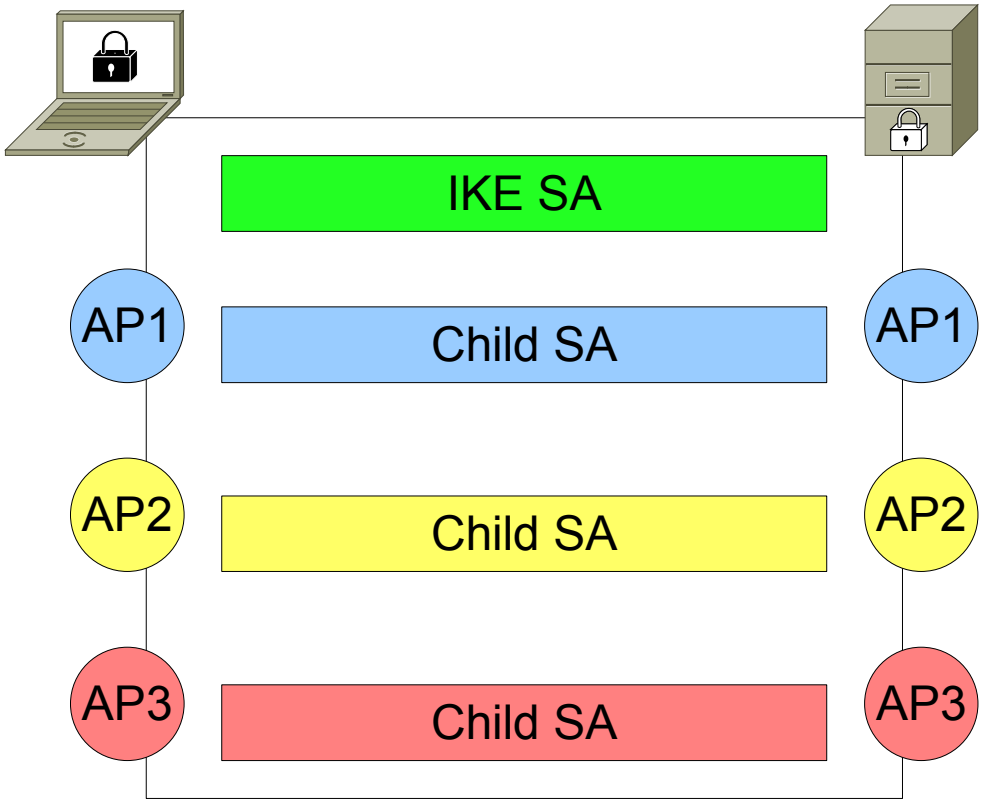
- IPsecは次世代IPv6では標準搭載。
  - 多くのOSで利用可能。
- 干渉性の低い、低レイヤーでサービスを提供可能。
  - 近年、続々と報告されるTCP仕様または実装の脆弱性
    - トランスポートレイヤの保護も大事な機能。
  - オーバーレイネットワーク
    - アクセスネットワークを移動しても接続中のTCPコネクションを切断する必要なし。（マルチホーミング）
  - アプリケーション毎に独自の暗号化のしくみを設計、実装する工数を減らせる？

アプリケーション開発者に貢献するために、IPsecにできることがまだまだあるのではないか？

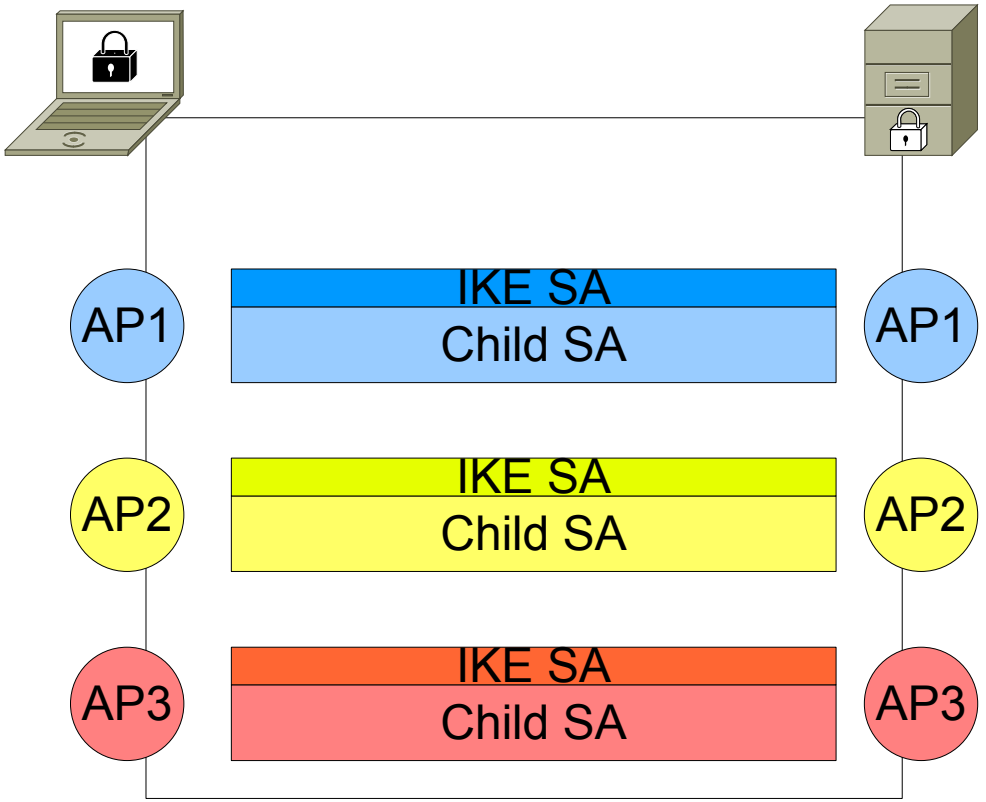
（将来）: End-to-End Security

- いくつかのアイデア（1）
  - 認証の粒度をアプリケーション毎でも可能に。
    - IKEv2のマルチホスティング機能(ID)をアプリ毎に割り当て

AP：アプリケーション



ネットワークレイヤー中心（ノード間認証）



アプリケーションレイヤー中心（アプリ間認証）

（将来）: End-to-End Security

- いくつかのアイデア（2）
  - ランタイム時に決定するセキュリティポリシーをアプリケーションから取得可能に。
    - (例)ダイナミックに決定されるトラフィックセレクタ
      - プロトコル(TCP,UDP,SCTP)
      - ポート番号
    - Child SAのトラフィックセレクタのダイナミックアップデート方法の検討。
      - リキー動作を利用したChild SAの移行
      - セレクタのみを更新可能な新しいネゴ手順（？）
  - アプリケーションが接続のタイミングを管理できるように。
  - IKE/IPsec機能とアプリケーション間のAPIを定義する必要（？）。

Q&A

-
-
-
-

ご清聴ありがとうございました。