動的処理解決プロトコル DPRP の改良の検討

鈴木　秀和　渡邊　晃

近年，増加傾向にあるイントラネット内部の犯罪に対するセキュリティ対策が重要視されている．そこでイントラネット内部のセキュリティと運用管理負荷軽減を両立した環境の構築を目指している．この環境では異なるアクセスポリシーを持つ各端末を部署単位や権利者単位でグループ化して閉域通信グループを構築する．閉域通信グループ内の端末間通信は暗号化され，異なる閉域通信グループの端末がアクセスすることや，通信内容を盗聴することが不可能となっている．そこで端末は通信相手が同一の閉域通信グループに帰属しているか確認する必要があり，そのネゴシエーションを行うために，動的処理解決プロトコルが提案されている．動的処理解決プロトコルは端末間の通信経路上に存在する終端暗号装置同士のネゴシエーションにより，動作処理情報を決定して通信経路上の暗号装置に動的に動作処理情報テーブルを生成する．しかし従来の動的処理解決プロトコルには動作処理情報テーブルを不正に生成される懸念があった．

そこで本論文では，より安全に動作処理情報テーブルを生成するために動的処理解決プロトコルの改良を検討する．

Researches on Improvement of DPRP

Hidekazu Suzuki　Akira Watanabe

Recently, crimes inside the intranet are increasing. Therefore security countermeasures for the intranet are important. So, we are aiming at building of the environment where it is coped with both the security and a reduction of responsibility to administer a network system inside the intranet. Each terminal which has a different access policy is made a group in its post unit and the right person. It defines their groups as CCGI (Closed Communication Group for Intranet). Communication between the terminals inside CCGI is enciphered, and the terminal of the different CCGI can't access it. Then, the terminal must confirm whether a partner for communication belongs to the same CCGI. DPRP (Dynamic Process Resolution Protocol) is proposed to negotiate. DPRP authenticate the terminal and generate a PIT (Process Information Table) automatically. But there were concern that it had a PIT generate illegally in the usual DPRP.

In this paper, we propose an improvement of DPRP to generate a PIT safely.

第 1 章 序論

インターネットの普及に伴い, 企業業務においてネットワークの利用が必須となっている今日，ウィルスや不正進入，データの盗聴や改竄といった様々な攻撃へのセキュリティ対策が重要な課題となっている. 一般的な対策方法として, ファイアウォールの設置や通信の暗号化, ディジタル署名やウィルス対策ソフトの導入などがある．これらは外部からの攻撃や第 3 者による悪意のある行為を防ぐ効果があり, 個人ユーザのみならず企業において広く利用されている. しかし企業ネットワークにおけるセキュリティの脅威は, インターネット経由だけではなくイントラネット内部にも存在し, 社員による内部犯罪が多く報告されている[1]．イントラネット内部のセキュリティ対策として相手認証, アクセス管理や社員へのセキュリティに関する研修などがあるが, 外的な対策と比べてあまり行われておらず, 社内のインフラ保護や管理の難しさ, また社員のセキュリティに対する意識の低さが問題視されている[24].

一般的にネットワークセキュリティの基本対策として相手認証と暗号化通信が挙げられる．この機能を併せ持った既存技術として IPsec(Security Architecture for the Internet

Protocol)[9]が考えられる．IPsec とはインターネットの中核的なプロトコル TCP/IP 上において, 個人や企業が汎用的に利用できるセキュリティ・プロトコルとして開発され，ネットワーク層で動作する．1998 年末に IETF(Internet Engineering Task Force)で標準化され，現在では VPN(Virtual Private Network)[12]を構築する手段として広く利用されており，IPv6 では標準サポートされている．IPsec は IPsec ソフトウェアをインストールしたクライアント端末同士で暗号通信を行うトランスポートモードと，セキュリティゲートウェイ(SG ; Security Gateway)によって拠点間あるいは SG-IPsec クライアント間で暗号通信を行うトンネルモードがある．

IPsec は暗号化通信に先立ち，IKE(Internet Key Exchange)プロトコル[10]によって鍵交換を行う必要がある．IKE は暗号・認証に必要なパラメータを動的に生成して安全に情報の交換を行うプロトコルである．IPsec 以外でも利用できる汎用性を持っており, IKE に定められている数々のオプションをクライアント端末およびセキュリティゲートウェイに設定をする必要がある. そのため管理者はセキュリティポリシーを決める作業が必要となり, 設定が複雑になりやすい．そのため IPsec を理解することが難しいため, 高い安全性を実現するには設定や運用に関する高度な専門知識が要求される．また運用上，IPsec はクライアントまたはセキュリティゲートウェイが必ず対で存在していなければならない．このため拠点間や VPN によるリモートログインは利用しやすいが, イントラネット内では部署ごとにアクセスポリシーが異なりセキュリティゲートウェイが階層的に構築されるのが自然なため, ある端末間の通信経路上に 2 台以上のセキュリティゲートウェイが存在する縦列接続構成の場合，通常の設定では通信が行えない. また異なるメーカの IPsec 機器を使用する場合の相互接続性の問題や NAT 技術との相性の悪さなどがあり[29]，IPsec は企業ネットワークにおけるセキュリティ対策の手段としては適当ではないため，ほとんど利用されることはない．IPsec の他に SOCKS[5]や SSL[13]などの既存技術があるが，IPsec と同様に企業ネットワークの特徴に対応でき, セキュリティと運用管理負荷の軽減を共に満たすことは難しい. 現在はイントラネット特有の環境に対応した柔軟なセキュリティ技術が存在しないことが, イントラネット内部の犯罪の増加に結びついていると考えられる．

企業ネットワークではセキュリティ向上を図ることによって, ネットワークシステムの運用や管理が難しくなる傾向がある. そこで既存技術では難しいイントラネット内のセキュリティ対策と運用管理負荷を両立したシステムを実現する FPN（Flexible Private Network）環境を目指している[25]．この環境では業務に応じた部門または個人単位に安全性の確保された閉域通信グループ CCGI(Closed Communication Group for Intranet)[14]が構築されている．CCGI を構成する端末は，IPsec のようにソフトウェアをインストールしたクライアントや SG に類似した装置などで，これらの端末を総称して暗号装置（EE ; Encryption Element）[14]という．CCGI 内の端末間の通信は暗号化によって保護され, 異なる CCGI の端末はアクセスや盗聴が不可能となっている. そこで端末は通信相手が同一の CCGI に帰属しているかを確認することと, どのように通信を暗号化するかを決定しておかなければならない. そのネゴシエーションを行うために動的処理解決プロトコル DPRP（Dynamic Process Resolution Protocol）が提案されている[14]．DPRP は各端末に設定される CCGI 構成定義情報をもとに，EE が通信端末との位置関係を検出しながら動的に動作処理情報を生成する機能を提供する．DPRP を使うことによって，システムの物理的構成に変化があっても, 暗号装置の保持する動作処理情報が動的に再生成されるため管理装置での作業負担が発生しない．このように CCGI と DPRP を利用することで，FPN 環境を実現できる一方，従来の DPRP の方式では，終端に位置する EE 間でネゴシエーションを行うため, 動作処理情報が不正に生成される懸念があった．

本論文では従来の DPRP の動作概要から問題点を調査し, その課題に対応するための改良について検討を行った．検討の結果から DPRP をより安全性の高いものにするため改良を行い, 従来方式と検討案および改良方式の比較評価を行う．その結果，改良方式によってセキュリティを向上することが可能なことを示す．

以下第 2 章に既存 DPRP の動作概要とその課題，第 3 章に改良の検討，第 4 章に DPRP の実装とテストプログラムの概要, 第 5 章に従来方式と検討案および改良方式の比較評価とテストプログラムによる性能評価, 第 6 章にまとめと今後の課題について述べる．

第 2 章 動的処理解決プロトコル DPRP

2.1 FPN と GSCIP

第 1 章で述べたように IPsec はインターネット上に VPN を構築する手法として利用されるが, 設定の煩雑さや既存の社内インフラとの相性等の問題により企業内のセキュリティ対策には向かない. 仮に全ての端末に IPsec ソフトウェアをインストールしてイントラネット内の限られた範囲においてトランスポートモードで通信を行えばセキュリティを向上させることは可能だが, システム構成を変更する際にその都度設定情報の変更が必要となる. そのため企業のように多数の端末を管理していることを考えると, 設定に費やす時間と労力は大変大きい. また組織変更や人事異動, あるいは会議や出張などでユーザおよび端末が移動することで, ネットワーク構成が頻繁に変わるのも企業ネットワークの特徴である. しかしこのような状況に適切に対応できるセキュリティ技術が現状では存在しない.

そこでイントラネット内のセキュリティと運用管理負荷軽減の両立を目指したシステムの構築を目指しており, この環境を FPN といい, 図 1 のような構成になっている. まずユーザは IC カードと指紋を利用した生体認証を行う. ユーザの認証が確認されると鍵管理装置 MS(Management Server) から グループ鍵 GK(Group Key)が配送される. グループ鍵を保持する端末は暗号装置 EE(Encryption Element) と呼び, 同一のグループ鍵を保持する端末をグルーピングしたものを CCGI として構成する. EE にはクライアント端末にソフトウェアをインストールして実現するソフトウェア型暗号装置 EES, ルータに実装させて配下の端末を保護するネットワーク型暗号装置 EEN がある. この EE には同一の CCGI に帰属する端末との暗号通信だけが可能な閉域モード(CL ; Close Mode)と, 全ての端末と通信が可能で, グループ鍵が一致する場合は暗号化通信を, 一致しない場合は平文で通信される開放モード(FR ; Free Mode)の 2 種類の設定がされる. 同一の CCGI に帰属する端末間の通信は暗号化され, 異なる CCGI に帰属する端末がこの通信を盗聴しても内容を知ることはできない. また端末へのアクセスはグループ鍵の一致が認められ

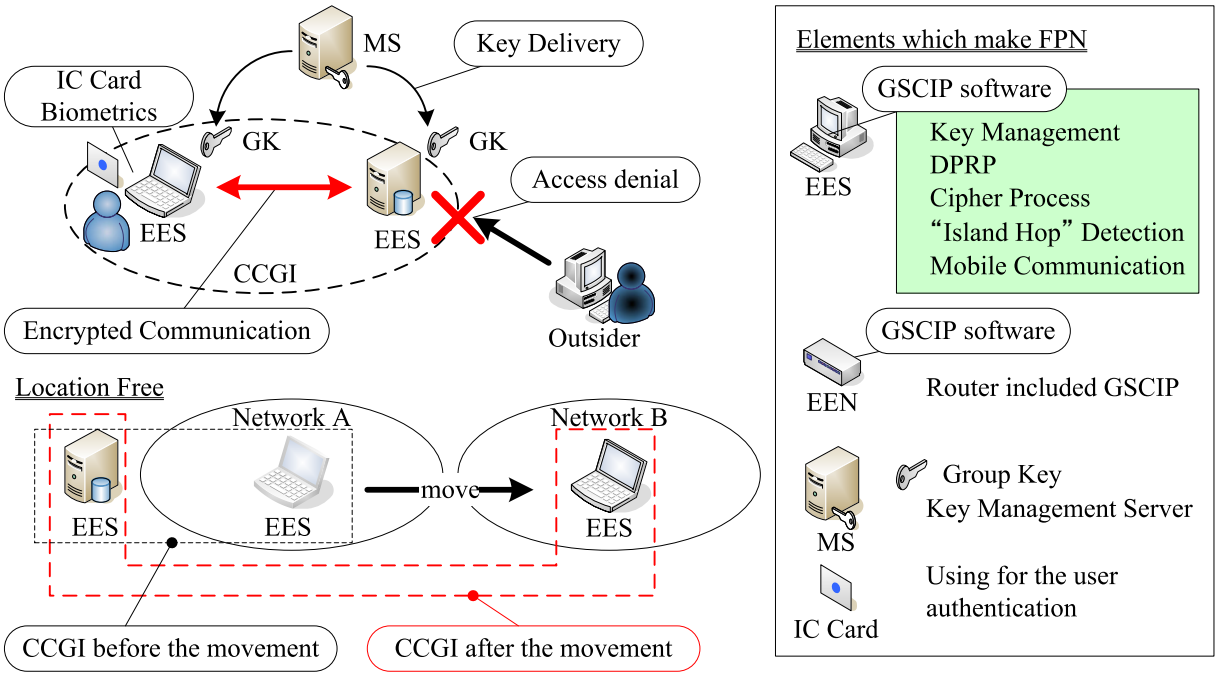


図 1　FPN 環境の概要と構成要素

なければ許可されないため, 第三者による不正なログインは発生しない. ここで端末は通信相手が同一の CCGI に帰属しているかを確認する必要があり, そのネゴシエーションを行うのが DPRP である. DPRP は通信に先立ち行われ, 認証と動作処理情報を動的に決定するため, ネットワーク構成の変更やユーザの移動があっても, 通信をトリガーとして DPRP が実行されるのでシステムに物理的位置透過性があり, これをロケーションフリー（Location Free）と呼んでいる. また企業ネットワークでは階層的にセキュリティドメインが構築されるケースが多いため, ある端末間の通信経路上に 3 台以上の EE が存在する多段構成が自然である. IPsec では 1 対 1 の IPsec 機器間ネゴシエーションしかできなかったが, FPN では通信経路上の EE が相互に情報を交換することで, 多重構成におけるネゴシエーションが可能となっている.

これらの動作を実現するために GSCIP(Grouped Secure Communication for IP ; ジースキップ)[25]という独自のネットワークセキュリティアーキテクチャを検討している. DPRP もこの GSCIP の一機能として組み込まれる. GSCIP は DPRP 以外にも暗号プロトコル PCCOM（Practical Cipher Communication Protocol）[20], IC カードによるセキュア認証プロトコル SPAC（Secure Protocol for Authentication with IC Card）[22], 移動端末が通信中に移動しても通信が保証される移動体通信処理プロトコル Mobile P2P[16]や NATF（NAT Free Protocol）[17]という 5 つのプロトコル群と, 鍵管理処理[21]やユーザの不正アクセスによく利用される多重リモートログインを行う渡り歩きの検知機能[19]が含まれる. この GSCIP によって柔軟な通信グループを構築でき, 高いセキュリティを併せ持ち, かつ管理者に運用管理の負荷を与えないのが FPN の利点である.

2.2 既存 DPRP の概要と課題

企業ネットワークでは部署や役職ごとにアクセスポリシーが異なるため, これに併せて CCGI を設定する必要がある. 部署内部の全端末を部署に設定されるアクセスポリシーを適用させるために, GSCIP ソフトウェアをインストールすることは効率的ではない. そのため, 部署ネットワークのゲートウェイ部分に EEN を設置し閉域モードを設定することで部署内の端末を保護することが可能である. また役職やプロジェクトなど他部門に渡ってアクセス

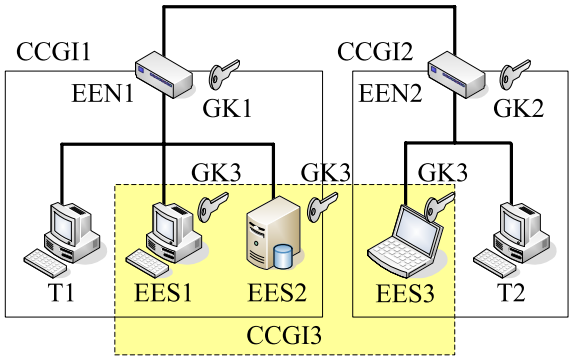


図 2　ネットワーク構成と CCGI 構成

ポリシーを適用する場合, GSCIP ソフトウェアをインストールした EES にする必要がある. ここでは図 2 のようなネットワーク構成と CCGI 構成を考える. CCGI 番号はグループ鍵番号と一致するため, GK3 を持っている端末の集合が CCGI3 となる. 図 2 の場合の CCGI 情報および通信マトリックスは表 1, 表 2 のように表される. CCGI 情報とは暗号装置に設定されている動作モードと帰属している CCGI を示す. 通信マトリックスは端末間の通信の可否と暗号化通信の有無を示す. 例えば EES3-EES2 間の通信は EK3 によって暗号化されることや, T1-EES1 間は平文による通信が可能で, T1-T2 間は通信が拒否されることが読み取れる.

表 1　CCGI 情報

EE	Process Mode	CCGI		
		1	2	3
EEN1	CL	○		
EEN2	CL		○	
EES1	FR			○
EES2	CL			○
EES3	FR		○	○

※CL; Close Mode　FR; Free Mode

表 2　通信マトリックス

	EES1	EES2	EES3	T1	T2
EES1		E3	E3	P	×
EES2	E3		E3	×	×
EES3	E3	E3		×	P
T1	P	×	×		×
T2	×	×	P	×	

※Ex; Encrypted communication by GKx
P; Communication by clear text
×; Impossibility communication

実際の通信において, 表 1 のようなグルーピングの関係をチェックすることや表 2 のよう

な通信を定義するのが[14]において提案されている DPRP である．以後，この提案されているものを既存 DPRP とする．この既存 DPRP では図 3 のように DDE(Detect Destination End EE)，RCN(Report CCGI Number)，DCN(Define CCGI Number)，MCI(Make CCGI process Information)という 4 種類の制御パケットを使用してネゴシエーションを行う．EES3 が EES2 へ通信をする場合，始点 EE である EES3 に作成されている動作処理情報テーブル PIT(Process Information Table)を検索する．EES3-EES2 間に関する情報がない場合は通信パケットを一時的に待避させてから DPRP を開始する．このとき通信経路上で最初に位置する EE を始点 EE，最後に位置する EE を終点 EE，始点 EE と終点 EE の間に位置する EE を中間 EE という．

始点 EE の EES3 は通信の宛先 EES2 に対して DDE を送信して終点 EE を決定する．終点 EE に決定した EES2 は，始点 EE に自分が帰属する CCGI 番号を RCN によって報告する．始点 EE は自分が帰属する CCGI 番号と RCN によって報告された CCGI 番号を比較して，共通の CCGI 番号と動作処理情報を決定する．始点 EE は決定した情報を DCN によって終点 EE へ送信する．終点 EE がこのパケットを認証すると MCI に決定した情報を載せて返信する．この MCI を受信した端末は決定した情報から PIT を自動的に生成する．通信経路上の全 EE に PIT が生成されると DPRP のネゴシエーションは完了して，始点 EE は先ほど待避させておいた通信パケットを暗号化して送信する．

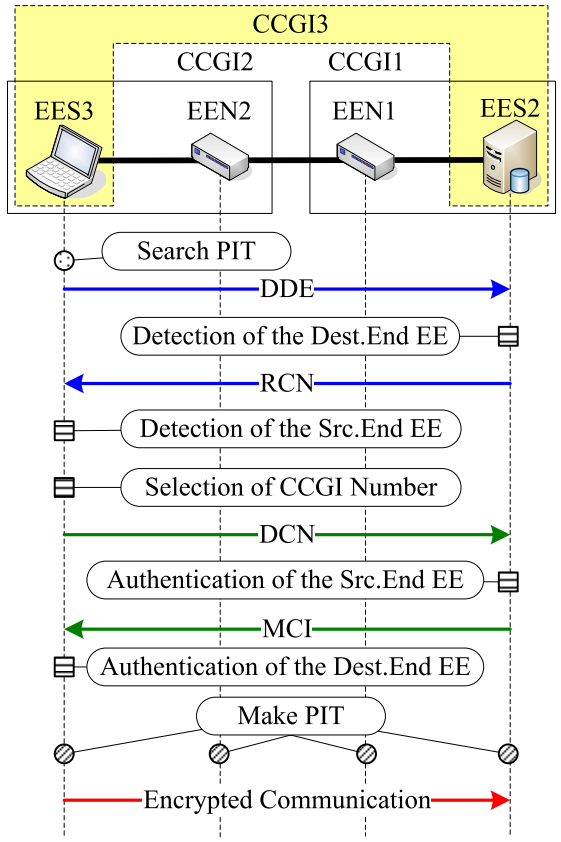


図 3　既存 DPRP シーケンス

ここで EES3-EES2 間の通信経路上に存在する中間 EE の EEN1 に注目する．EEN1 は閉域モードなので EEN1 が通信を許可するのは同一の CCGI に帰属する端末間の通信，つまり CCGI1 の通信のみ可能であるはずが，EES3，EES2 は CCGI1 に帰属していない．そのため本来ならば通信が許可されないはずだが，既存 DPRP では両終端 EE 間で End-End の事前ネゴシエーションを行っているため，中間 EE に該当する EEN1，EEN2 の動作モードに関係なく無条件に中継処理をしている．これでは閉域モードの概念と矛盾が生じ，PIT を作成する制御パケットを無条件で中継させることで EE に不正なテーブルを作成される懸念がある．また無条件で閉域モードの EEN を通過できるため，悪意を持ったユーザによる DoS 攻撃の対象となりうる危険性や不要な DPRP の発生などが考えられる．

第 3 章 DPRP の改良

3.1 DPRP 改良の検討案

第 2 章で述べた既存 DPRP の問題を解決するために，4 通りの検討案を考えた．以下に述べる検討案は図 4 のネットワーク構成を基準として，EES3 から EES2 への通信をトリガーとする DPRP ネゴシエーションについて考える．

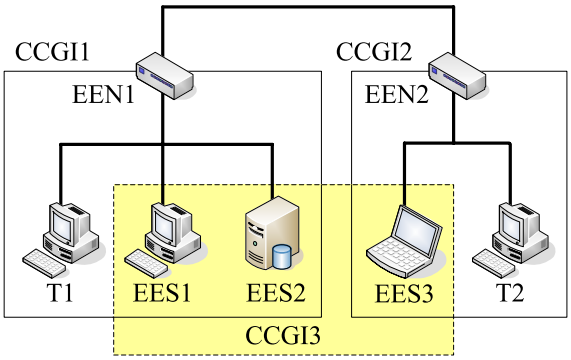


図 4　ネットワーク構成

(1)検討案 1

検討案 1 はグループ鍵の持たせ方を変更した[18][23]. 始点 EE に通信経路上に存在する全ての EE と同一のグループ鍵を持たせることで，両終端 EE 間で行っていた End-End の事前ネゴシエーションの認証処理を通信経路上に存在する全ての EE で行うように変更する．鍵の持たせ方を変更したため，表 3 のような CCGI 情報になる. それに伴い DPRP シーケンスを見直して通信経路の認証と PIT の自動作成を行うために，ACG(Authenticate Communication Groups)，DPI(Define Process Information)，MCI の 3 種類の制御パケットを使用する. 制御パケットの詳細は 3.3 で説明する．

始点 EE となる EES3 に通信経路を認証するために必要な GK1，GK2 を配送させる．既存の DPRP では RCN によって終点 EE の CCGI 番号を取得してグループチェックを行っていたが，検討案 1 では図 5 のように ACG によって始点 EE の CCGI 番号を送信して終点 EE でグループチェックを行うことにした．

表 3　検討案 1 における CCGI 情報

EE	Process Mode	CCGI		
		1	2	3
EEN1	CL	○		
EEN2	CL		○	
EES1	FR			○
EES2	CL			○
EES3	FR	○	○	○

※CL; Close Mode　FR; Free Mode

始点 EE は通信経路上に存在する全 EE を認証するために必要なグループ鍵を保持しているので，今まで中間 EE で無条件に中継させるのではなく，認証を行って正常と確認された後に中継処理を行うことが可能になる．終点 EE が ACG を受信して認証されると，動作処理情報が決定される．この決定した情報を DPI に

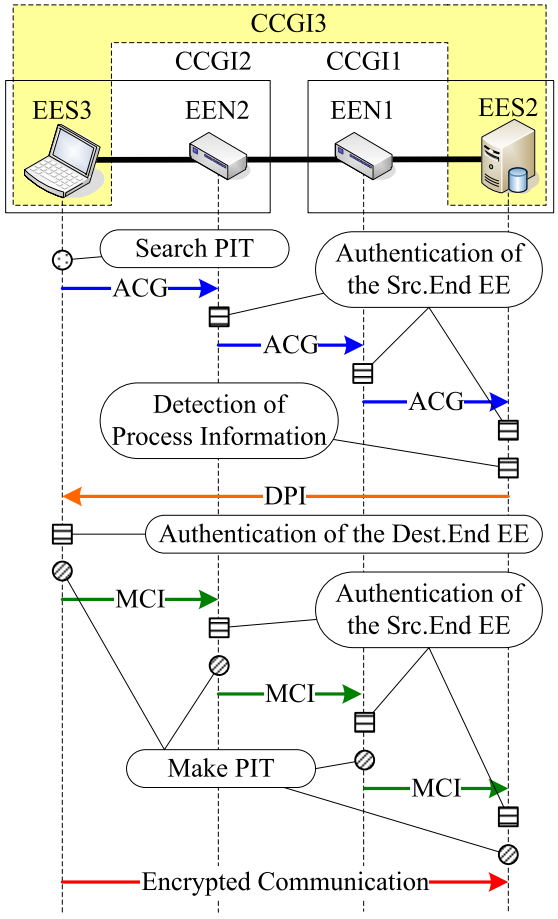


図 5　検討案 1 の DPRP シーケンス

よって始点 EE へ通知する．動作処理情報は PIT を生成する上で非常に重要な情報なので，End-End で決定したグループ鍵で暗号化される．このため中間 EE が暗号化に使用されたグループ鍵を保持していない可能性があるため，中間 EE では DPI を透過中継する．DPI を利用して MCI を始点 EE から送信するのは，始点 EE が通信経路を通過できる鍵を保持しているためである．始点 EE が決定した情報を受信すると PIT を自動的に生成する．生成後は MCI に残りの情報を載せて終点 EE へと送信される．MCI を受信する中間 EE を含む全 EE は ACG と同様に制御パケットを認証した後，PIT を生成する．これにより安全に PIT の作成が行える．

(2)検討案 2

検討案 2 はグループ鍵配送時に行われるディジタル署名検証機能を応用して中間 EE において制御パケットを認証する．End-End は従来方式と同様にグループ鍵による認証を行う．既存 DPRP の鍵配送は[15]で提案されている方式で行われているが，中間 EE の認証を考慮せずに鍵配送パケットを通過させている課題があった．この課題を解決するために新しい鍵配送方式が検討されており[21]，そこでディジタル署名検証が行われる．ディジタル署名とは伝えたい情報の後にディジタル署名を添付することで，他者の「なりすまし」の検出および情報の改竄の検出が可能な認証方式である．

新しい鍵配送方式で行われるディジタル署名は必ず終端に MS が存在する場合でないと検証できない仕組みになっている．そのため DPRP にそのままフィードバックできないので修正して利用する．新しい鍵配送方式において各 EE は表 4 に示す初期情報を持っている．このうち署名生成時に EE の秘密鍵 Prx と暗号化データ Eprs[Pux]を，ディジタル署名検証時に MS の公開鍵 Pus を使用する．始点 EE はまずパケットデータ D に添付するディジタル署名 Eprx[H(D)]を生成する．さらに各 EE が初期情報として保持している暗号化データ Eprs[Pux]を付加してからパケットを送信する．このときディジタル署名付きパケットフォーマットは，D | Eprx[H(D)] | Eprs[Pux]と記述できる．ディジタル署名を使用する場合，通信相手は通信元の公開鍵を保持していなければ検証することができない．DPRP ネゴシエーションの通信相手に該当する終点 EE は通信元である始点 EE の公開鍵を持っていないため，通常の方法ではディジタル署名を検証できない．そのため始点 EE の公開鍵を MS の秘密鍵で暗号化された認証情報 Eprs[Pux]を付加している．各 EE は予め MS の公開鍵を保持しているので，パケットに付加されている認証情報から始点 EE の公開鍵を取得することが可能になり，ディジタル署名を検証できる．

検討案 2 で使われる制御パケットは図 6 のように ACG と MCI の 2 種類である．このパケットはディジタル署名付きとなるので，D | Eprx[H(D)] | Eprs[Pux]の D 部分に ACG と MCI の情報が格納される．始点 EE はディジタル署名付き ACG を生成して通信宛先へ送信する．中間 EE が ACG を受信すると，ディジタル署名を検証する．検証の結果，正当なデータと確認されると ACG は中継処理を行う．終点 EE はディジタル署名を検証した後，ACG データの認証をグループ鍵によって行う．認証が行われると動作処理情報を決定して，ディジタル署名付き MCI を始点 EE へ送信する．中間 EE は ACG と同様に MCI を受信するとディジタル署

表 4　各 EE が保持する初期情報

EES/EEN	IDx：User ID of EE Prx：Private Key of EE Pus：Public key of MS Eprs[Pux]：Authentication Data

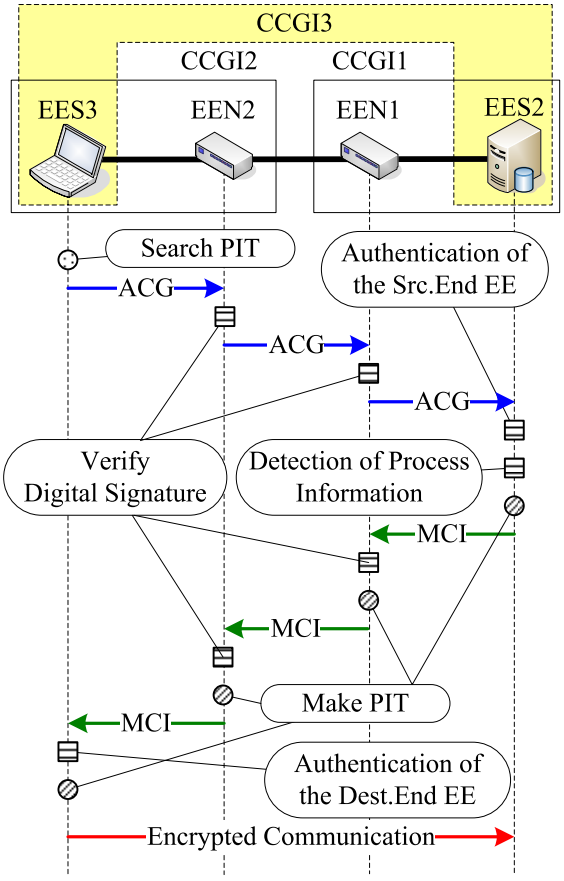


図 6　検討案 2 の DPRP シーケンス

名を検証する．検証の結果，正当性が確認されれば PIT を生成してから MCI を中継する．これにより安全に PIT の作成が行える．

(3) 検討案 3

検討案 3 は全ての EE にグループ鍵とは別の共通の暗号鍵を持たせて，中間 EE を通過する際に制御パケットを認証する方法である．鍵配送時に行われるディジタル署名で使用される MS の公開鍵は，全ての EE が保持しているのでグループ鍵とは別の共通鍵として利用できるが，公開鍵ということで暗号鍵を持たない通常の端末が取得する可能性があり得る．通常の端末が MS の公開鍵を取得してしまうと中間 EE を認証して通過してしまう恐れがあるため，MS の公開鍵を中間 EE 認証のための共通鍵にすることはできない．よって EE 間の秘密鍵となる新たな共通鍵を各 EE に持たせることとなる．この鍵は，鍵配送時にグループ鍵と一緒に配送することで，EE だけに共通鍵を持たせることが可能である．共通鍵は EE を認証するものであり，CCGI の同一性を確認するものではない．よって End-End は従来方式と同様にグループ鍵による認証を行う．

DPRP ネゴシエーションを行う始点 EE と終点 EE が共有秘密鍵を持つため，メッセージ認証コード MAC(Message Authentication Code)を使用する．MAC とは第三者によるメッセージの偽造や改竄を検出し，送信元の認証が行うための技術で，IPsec でも MAC が使用されている．ここでは MAC を生成する方式として MD5 ハッシュ関数[4]を使う HMAC-MD5[6]とする．HMAC-MD5 は ACG で送信されるデータにも使用される技術で，詳細は 3.3 の(2)で説明する．

検討案 3 で使われる制御パケットは図 7 のように ACG と MCI の 2 種類である．このパケットは HMAC-MD5 によって生成された MAC が付加されている．始点 EE は MAC 付き ACG を生成して通信宛先へ送信する．中間 EE が ACG を受信すると，MAC を検証する．検証の結果，正当なデータと確認されると ACG は中継処理を行う．終点 EE は MAC を検証した後，ACG データの認証をグループ鍵によって行う．認証が行われると動作処理情報を決定して，MAC 付き MCI を始点 EE へ送信する．中間 EE は ACG と同様に MCI を受信すると MAC を検証する．検証の結果，正当性が確認されれば PIT を生成してから MCI を中継する．これにより安全に PIT の作成が行える．基本的な考え方は検討案 2 と同じで，使用する認証アルゴリズムと使用する鍵の違いである．

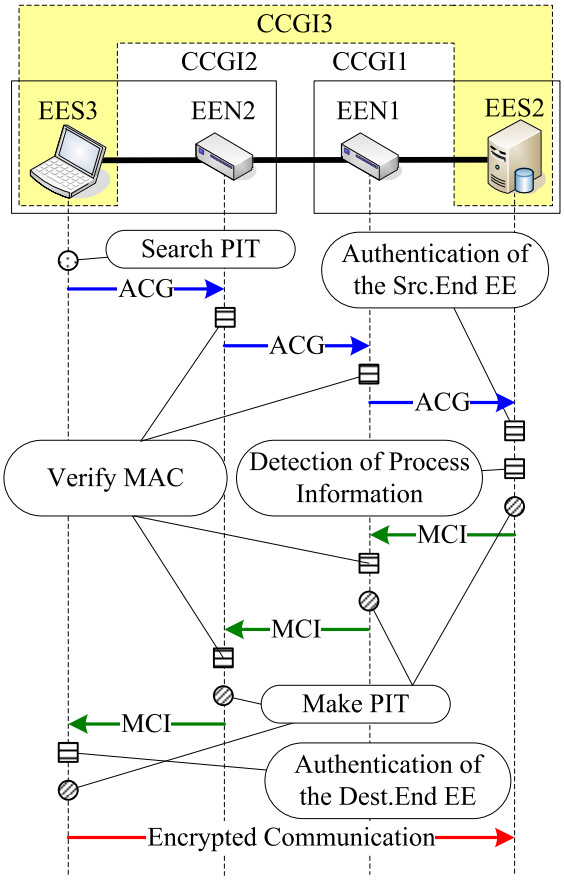


図 7　検討案 3 の DPRP シーケンス

(4)検討案 4

検討案 4 は EEN が配下を保護するということに注目した．図 4 のネットワーク構成で CCGI 情報は従来方式と同じ表 1 のような場合で，EES3-EES2 間と EES3-T1 間の通信を考える．まず EES3 から EES2 へ通信する場合，EES2 は GK3 によって自らを暗号装置化することで安全を確保している．一方，EES3 から T1 へ通信する場合，T1 はグループ鍵を保持していないため CCGI には帰属していないが，EEN1 の配下に存在するため事実上 CCGI1 に帰属しているのと同じ状況にある．この場合 T1 は EEN1 によって安全を確保されていることになる．このように EEN が配下の端末を保護するのは，グループ鍵を保持する EES ではなくグループ鍵を保持しない端末 T のみという考え方ができる．このように考えた場合，EEN は配下に EE があるかを把握して，受信したパケットが配下の EE 宛で CCGI が一致する EE から送信されている場合，宛先 EE は EEN の保護の対象にならないため，パケットを中継しても問題が無いと考えられる．そこでこの考え方を実現するために検討案 4 を改良方式として扱い，以下にその詳細を説明する．

3.2 改良方式の概要

改良方式で使われる制御パケットは REI(Report EE Information)，ACG，MCI を使用する．改良方式は中間 EE において制御パケットを検証するために，EEN の配下に存在する EE を把握しておかなければならない．EEN は配下に存在する EE の情報を格納するために ECT(EE Check Table)を作成する．ECT は IP アドレス，CCGI 番号と鍵バージョンが記されているグループ鍵情報によって構成される．これらの情報は EES が電源投入時に自動的に REI パケットによって EEN に伝えられるため，DPRP ネゴシエーションが開始する前に生成されるようになっている．図 4 のネットワーク構成において EES3 から EES2 への通信が発生した場合，DPRP ネゴシエーションは図 8 のように ACG と MCI によって行われる．中間 EE では制御パケットの宛先 IP アドレスとグループ鍵情報を ECT と比較して検証を行う．検証により一致するデータが確認できた場合，中間 EE となる EEN は確かに配下に存在する EE へ送られるとして中継する．End-End ではグループ鍵による認証処理を行う．これらの処理によって PIT は安全に生成される．

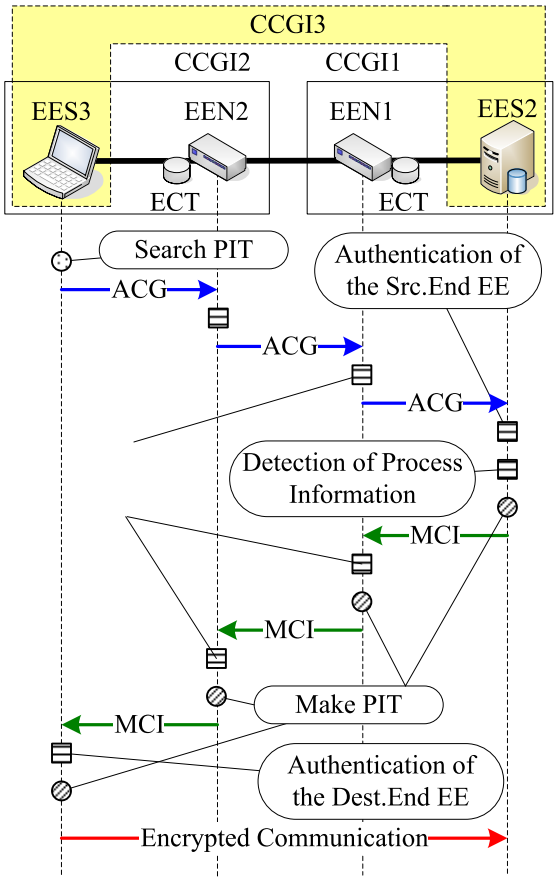


図 9　改良した DPRP シーケンス

3.3 制御パケット

改良方式のネゴシエーションを行うために使われる REI，ACG，MCI の 3 種類の制御パケットの詳細を説明する．

(1)REI

REI は EEN に ACG および MCI を検証するために必要な ECT を作成および削除する制御パケットで，UDP を利用する．EES の電源投入時とネットワークを移動したときに移動先のネットワークに接続した時に REI を自動的に送信される．まず電源が入るとユーザは IC カードと生体情報を利用してログインを行う．正常なユーザと判断されると，MS に対して鍵要求を行ってグループ鍵を取得する．この後に REI を自動的に送信する．REI は EES が位置するサブネットワークを構成する EEN に届かなければならない．例えば図 9 において CCGI3 を構成する EEN3 のように配下に EEN が存在しない場合は，そのサブネットワークに存在する EES は REI をブロードキャストすれば必ず EEN3 に届く．しかし CCGI1 を構成する EEN1 のように配下に EEN が存在する多段構成の場合，CCGI2 のように内部のサブネットワークに存在する EES3 が REI をブロードキャストすると，EEN2 には届くが，EEN はルータでもあるため REI が破棄されて EEN1 には届かない．この場合，T3-EES3 間の通信は終端 EE が EEN3 と EEN2，中間 EE は EEN1 となり，制御パケットが EEN1 で検証されるのだが，EEN1 の ECT には EES3 の情報が存在しないため破棄されてしまう．そのため REI はブロードキャストするのではなく，EEN1 にも届くようにある特定の端末へ向けて送信されなければならない．企業ネットワークを図 10 のように木構造として考えた場合，EES が送信する REI の宛先端末は以下の 4 通りが考えられる．

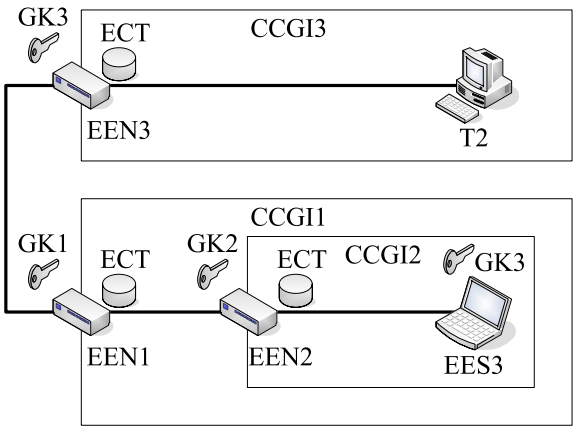


図 8　EEN の多段構成

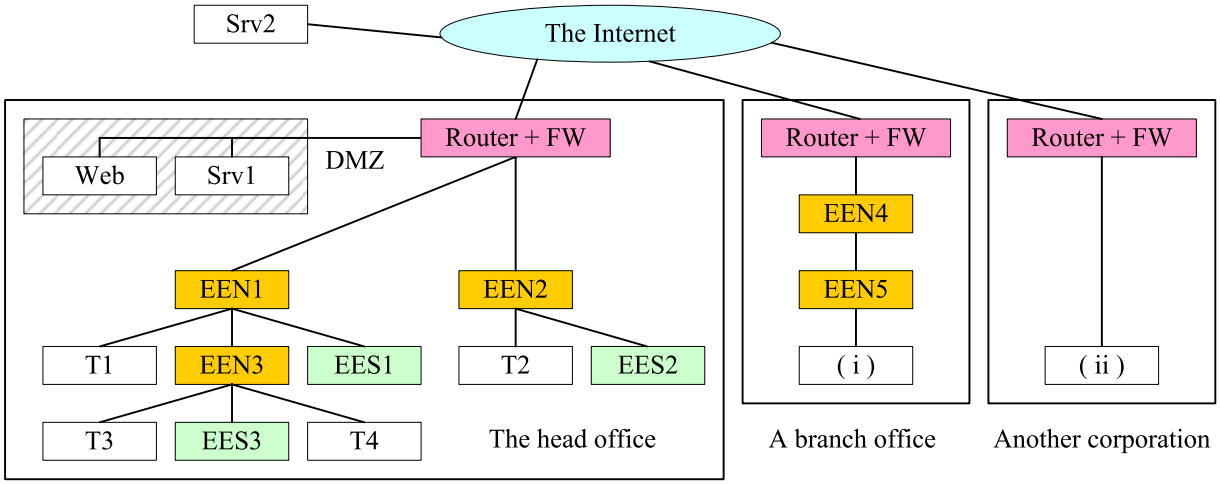


図 10　企業ネットワーク構成の木構造表現

① EES が位置する最上位の EEN

EES3 の場合，最上位に位置するのは EEN1 となる．EEN1 宛へ REI を送信すれば，EES3-EEN1 間に存在する全ての EEN を通過する．そのため，EEN3，EEN1 共に REI を受信することが可能である．EES1 の場合も同様に EEN1 宛，EES2 の場合は EEN2 宛となる．このように EES が位置する場所によって最上位 EEN の IP アドレスが異なるため，ユーザが移動した場合に最上位 EEN の IP アドレスを知る必要がある．また EEN1 より上位に新たな EEN が設置された場合，EES が REI を送信するべき端末が変わるため，設定の変更が必要となる．

② DMZ に位置するサーバ

企業の DMZ は最上位 EEN より上位に位置することが一般的である．DMZ に特性のサーバ Srv1 を設置する．各 EES は Srv1 宛に REI を送信すれば，EES-Srv1 間に存在する全ての EEN を通過する．このため，新たな EEN を設置した場合でも確実に最上位 EEN まで REI が送られることになる．ただし企業ネットワークにおいて管理者の立場からすると，DMZ に新たなサーバを設置することはセキュリティ上あまり好ましくないため，DMZ に新たなサーバを設置することが難しい場合がある．

③ ルータまたはファイアウォール FW

企業ネットワークとインターネットを繋ぐルータやファイアウォールは②と同様に最上位 EEN より上位に位置するのが一般的である．各 EES はルータ宛に REI を送信すれば，EES-Router 間に存在する全ての EEN を通過する．②と異なり REI を受信するサーバを新規に用意しなくてもよい．例えば EES3 が出張などで本社から支社や他企業へ移動した場合を考える．支社に移動した場合，本社と同様に FPN 環境が構築されており，EEN 等が設置されているとする．移動した場所が図 10 の(i)とすると，EES3 は支社のルータ宛に REI を送信しなければならない．ネットワーク管理者でない一般の従業員の場合，通常はルータの IP アドレスなど知らないため設定することができない．EES 設定のためにルータの IP アドレスが公開されていれば可能だが，他企業や FPN 環境でないネットワークへ移動した場合，(ii)に位置することになる．この場合，確実にルータの IP アドレスを知ることは保証されない．

④ グローバルに公開されているサーバ

企業ネットワーク内ではなく，インターネット上で一般に公開されているサーバ Srv2 を用意する．各 EES は Srv2 宛へ REI を送信すれば，EES-Srv2 間に存在する全ての EEN を通過する．ここでは Srv2 は FPN に関するサービスを提供する企業が公開しているとする．送信先がグローバルに公開されているため，全ての EES は IP アドレスを予め設定しておけば，ユーザが移動した場合でもインターネットに繋がる環境があれば，設定を変えることなく確実に Srv2 宛へ REI は送信される．REI パケットは UDP を利用しているため，FW が通常塞いでいるポートへ指定すれば，Srv2 宛へ送信されても FW において REI は破棄されるので，Srv2 に REI が届くことはない．そのため Srv2 に負荷は発生せず，グローバルアドレスがあれば Srv2 を比較的容易に設置することが可能と考えられる．

以上の考察より，REI は④の公開サーバ宛へ送信する．REI が EEN に伝える情報は EES の IP アドレスと帰属する CCGI 番号と鍵バージョンを記したグループ鍵情報である．単純にこれらの情報を送信して ECT を生成すると，悪意のあるユーザによって不正な情報が送信され不正な ECT が生成される恐れがある．その

ため, REI はディジタル署名を付加して送信される．3.1 の検討案 2 で説明した方法でディジタル署名付き REI を生成する．署名生成時に EES の秘密鍵 Prx と暗号化データ Eprs[Pux]を使用する．REI は図 11 のように生成および検証される．ここで EES の IP アドレスとグループ鍵情報をまとめて D と表記する．EES は MD5 で D のハッシュ H(D)を生成して自分の秘密鍵 Prx で暗号化する. 暗号化したものがディジタル署名となり Eprx[H(D)]と表す. D にディジタル署名と MS のよって配送されている認証情報 Eprs[Pux]を付加して REI が送信される. EEN が REI を受信すると MD5 で D のハッシュ H(D)を生成する．その後 REI の認証情報 Eprs[Pux]を MS の公開鍵 Pus で復号して EES の公開鍵 Pux を取得してから, ディジタル署名を復号してハッシュ H(D)を取得する．取得した H(D)と生成した H(D)を比較検証して一致していれば, 受信した REI が MS によって証明された EES からのものであるといえ，確実に認証することができる.

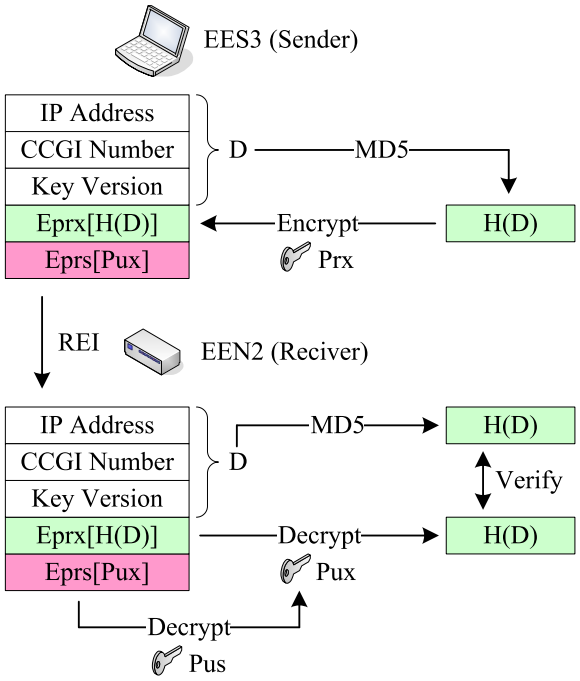


図 11　REI 生成と検証

REI が認証されると EEN は伝えられた情報 D を ECT に追加して応答メッセージを返信する．ECT はハッシュテーブルとして作成されているため，IP アドレスをハッシュキーとして登録を行う．ECT には IP アドレス，グループ鍵情報とは別に生存時間が設定される. 生存時間は EEN ごとに設定され，ECT に追加された情報は生存時間を経過すると自動的に削除される．EEN は REI の応答メッセージによって生存時間を EES に伝える．EES は生存時間を超過する前に再度 REI を自動的に送信して ECT を更新する．この仕組みは DHCP[7]の仕組みと類似している.

EES が電源を切る時や, ネットワークから明示的に切断する場合に，EEN に情報が残ったままだと実際には EEN の配下に存在しないが, あたかも存在しているかのように EEN は判断してしまう. そのため EES は REI によって ECT の情報を削除しなければならない. REI の生成および検証は ECT に情報を追加する場合と同様に行われ, 認証されると該当する情報が削除される. EES がネットワークから明示的に切断しなかった場合は, 生存時間が超過した時点で自動的に削除される．これは移動端末から LAN ケーブルなどが抜かれた場合や，無線 LAN 接続の状態で他のアクセスポイント AP へ移動してしまった場合が考えられる.

(2)ACG

ACG は始点 EE から通信相手へ送信され, 終点 EE 間で認証を行う制御パケットで, ICMP を利用する．従来方式では DDE，RCN による 1 往復のネゴシエーションによって CCGI 情報の確認と動作処理情報の決定を行っていた. 改良方式ではこの動作を ACG によって単方向で行うことが可能である．ACG は始点 EE が帰属する CCGI 情報，通信経路上の各 EE の設定情報から構成されている.

始点 EE は自分が保持するグループ鍵を使って CCGI 情報を生成する．CCGI 情報は CCGI 番号，鍵バージョンと認証データが含まれる．この CCGI 情報を GMAP（Group Message Authentication Package）と定義する．認証データは終端 EE 同士で認証を行うためのデータで, これを GMAC（Group Message Authentication Code）という．GMAC は図 12 のように，始点 EE が発生させる 128bit 乱数 RN と HMAC-MD5 によって生成される乱数 RN の MAC128bit を組み合わせたものを指す．MAC を生成するときに HMAC-MD5 を利用しているが, この時に暗号鍵が必要なのでグループ鍵 GKx を使用する．GMAC を表記すると GMAC(GKx)または [RN, MAC(RN)]となる． ACG を送信する始点 EE または受信する各 EE は自らの EE 設定情報を ACG に追加する．EE 設定情報は，各 EE の IP アドレス，グループ鍵情報，動作モード,

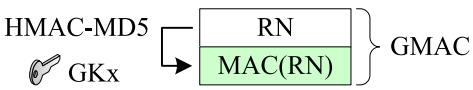


図 12　GMAC 生成

動作処理情報から構成される．ACG によって伝えられる EE 設定情報のうち，動作処理情報はこの段階で決定していないので空白にして送信する．

終点 EE が ACG を受信すると，GMAP に記されているグループ鍵情報と一致する鍵を保持していれば，GMAC を認証する．認証方法の詳細は 3.5 で説明する．認証できた後に，ACG によって伝えられた通信経路上の EE 設定情報に基づいて動作処理情報を決定する．

(3)MCI

MCI は終点 EE で決定した動作処理情報を始点 EE へ送信され，各 EE で PIT を生成する制御パケットで，UDP を利用する．従来方式では DCN，MCI による 1 往復のネゴシエーションによって終端 EE 間の認証および動作処理情報テーブルの生成を行っていた．改良方式ではこの動作を MCI の単方向で行うことが可能である．MCI は終点 EE が認証した CCGI 情報，決定した動作処理情報を含めた EE 設定情報から構成されている．

終点 EE は ACG を認証できたグループ鍵で GMAP を ACG の時と同様に生成する．その後 ACG によって伝えられた EE 設定情報に，決定した動作処理情報を書き込んで MCI に付加する．終点 EE は動作処理情報テーブル PIT を生成してから MCI を送信する．中間 EE が MCI を受信すると，ECT によるパケット検証によって認められた場合，PIT を生成して中継処理を行う．始点 EE が MCI を受信すると，GMAP に記されているグループ鍵情報と一致する鍵を保持していれば，GMAC を認証する．認証は ACG と同様にして行われる．認証できたら PIT を生成する．

3.4 制御パケットの検証

改良方式で追加された新たな機能として，中間 EE による制御パケットの検証があげられる．従来方式では無条件に中継していたが，改良方式では REI によって生成された ECT を使用して検証を行う．検証は制御パケットと ECT の IP アドレス，グループ鍵情報を比較して行われる．検証フローは図 13 のように表される．図中の数字は一致しなかった場合に実行する処理内容で，この節の最後にまとめて説明する．

まず IP アドレスの比較から行う．制御パケットの IP アドレスからハッシュキーを求めて ECT を検索する．検索の結果，一致するデータが無かった場合は①の処理を行う．一致したデータがあった場合，CCGI 番号，鍵バージョンと順に比較を行う．3 項目全て一致した場合は，チェックしている制御パケットが MS によって証明された EES へ送信されるといえるので，制御パケットの中継処理を行う．

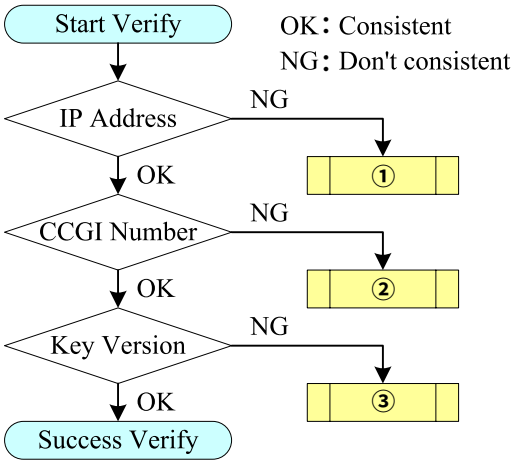


図 13　制御パケットの検証フロー

ECT と完全に一致したデータが無かった場合，チェック項目により次の①から③の処理のいずれかが実行される．この処理説明の中では，制御パケットの宛先 IP アドレスが設定されている端末が EE の場合を宛先 EE，グループ鍵を持たない通常の端末の場合を宛先端末と呼ぶ．

①IP アドレスが一致しない場合

宛先 EE は存在しないが宛先端末は存在する場合と，宛先 EE と宛先端末が共に存在しない場合の 2 つの可能性が考えられる．前者の場合，宛先端末は REI を送信しないので ECT に IP アドレスが通知されない．しかし端末としては存在しているため，制御パケットを検証している EEN が終端 EE になる．よって ECT による検証ではなく，グループ鍵による認証処理を行う．後者の場合，EEN の配下に指定された IP アドレスを持つ端末および EE が存在しないことになるため，この制御パケットは不正なパケットとして識別しなければならない．しかし EEN が把握しているのは配下にいる EE だけで，通常の端末の存在はわからない．そのため前者と同様に制御パケットを検証している EEN が終端 EE として認証処理を行う．認証されなかった場合は破棄され，認証されても宛先端末または宛先 EE が存在しないので，結果的に制御パケットは破棄される．

②CCGI 番号が一致しない場合

宛先端末が制御パケットの送信元と同じ CCGI に属していないため，通信は許可されない．そのため制御パケットは破棄され，PIT に

破棄情報が追加される.

③鍵バージョンが一致しない場合

宛先端末と同じ CCGI に属するが, 鍵のバージョンが一致しないため制御パケットは破棄されるが，PIT に破棄情報を追加せず，バージョンの古いグループ鍵を保持している EE へ鍵が更新されていることを通知する.

3.5 認証方法

従来方式の認証方法は図 14 のように暗号化した乱数のやりとりによって行っていた. そのため片方向の認証を行うために 1 往復ネゴシエーションを行う必要があった. 改良方式では GMAC の導入により図 15 のように片方向の認証を行うために単方向ネゴシエーションで可能になった．また IPsec でも利用されている MAC 技術を採用したことで，送信元の認証に加えてメッセージの偽造および改竄の検出が可能になり, 安全性と信頼性が向上されている.

GMAC の認証方法は生成時と逆の手順を行えばよい．受信した GMAP に記されているグループ鍵情報を基にして，EE は保持するグループ鍵から情報と一致するものを決定する. グループ鍵が決定すると，GMAC の乱数 RN から HMAC-MD5 により MAC を生成する. HMAC-MD5 に使用する暗号鍵は決定したグループ鍵である. これと取得した MAC と比較して認証を行う. グループ鍵が見つからなかった場合は, 制御パケットの受信側が送信側と同じグループ鍵を保持していない場合と, 鍵バージョンが異なる場合の 2 つの可能性が考えられる. 前者の場合，同じ CCGI に属していないことになるため, 認証失敗となり制御パケットを破棄して，PIT に破棄情報を追加する．後者の場合, 現在処理している制御パケットは破棄されるが，PIT に破棄情報は追加せず，バージョンの古いグループ鍵を保持している EE へ鍵が更新されていることを通知する.

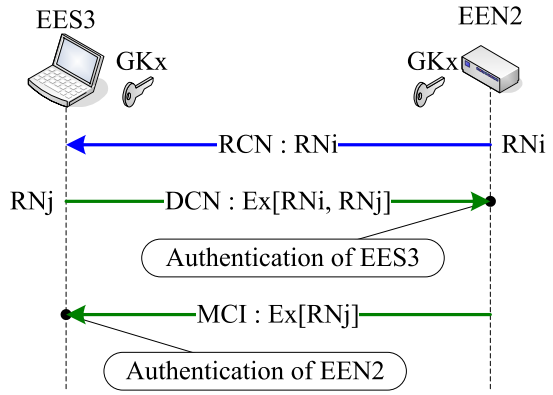


図 15　既存方式の認証方法

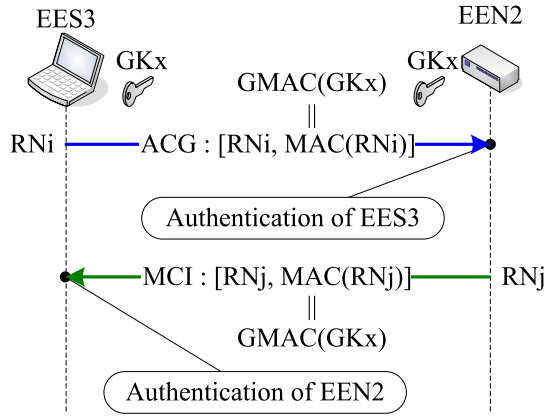


図 15　改良方式の認証方法

3.6 動作処理情報

動作処理情報は通信パケットの処理内容を定めたものである．ACG によって伝えられた通信経路上の EE 設定情報によって動作処理情報を決定する．決定した情報は ACG で伝えられた EE 設定情報に追加して，これを MCI に付加する．MCI を受信した EE は ECT による検証またはグループ鍵による認証を行った後に，EE 設定情報の動作処理情報から動作処理情報テーブル PIT を生成する. PIT は送信元 IP アドレス（sIP），送信元ポート番号（sPrt），宛先 IP アドレス（dIP），宛先ポート（dPrt），グループ鍵情報（No，Ver），動作処理情報（Proc）とカウンタフィールド（CF）から構成される. 動作処理情報は暗号化（Enc），復号化（Dec），転送（Fwd），破棄（Dst），作成中（Cre）を示すデータからなる. 図 8 のネゴシエーションを行った場合，EE に表 5 から表 8 の PIT がそれぞれ生成される. 受信した EE 設定情報から両方向のデータを追加する．EES3 から EES2 の方向の DPRP ネゴシエーションで双方向の動作処理情報を生成する.

表 5　EES3 の動作処理情報テーブル

sIP	sPrt	dIP	dPrt	No	Ver	Proc	CF
S3	xxx	S2	yyy	3	zzz	Enc	0
S2	yyy	S3	xxx	3	zzz	Dec	0

表 6　EEN2 の動作処理情報テーブル

sIP	sPrt	dIP	dPrt	No	Ver	Proc	CF
S3	xxx	S2	yyy	3	zzz	Fwd	0
S2	yyy	S3	xxx	3	zzz	Fwd	0

表 7　EEN1 の動作処理情報テーブル

sIP	sPrt	dIP	dPrt	No	Ver	Proc	CF
S3	xxx	S2	yyy	3	zzz	Fwd	0
S2	yyy	S3	xxx	3	zzz	Fwd	0

表 8　EES2 の動作処理情報テーブル

sIP	sPrt	dIP	dPrt	No	Ver	Proc	CF
S3	xxx	S2	yyy	3	zzz	Dec	0
S2	yyy	S3	xxx	3	zzz	Enc	0

※sIP; src IP Address　sPrt; src Port
dIP; dest IP Address　dPrt; dest Port
No; CCGI No　Ver; Key Version
Proc; Process Information
CF; Counter Field (src:sorce, dest:destination)

EE は通信パケットを受信するたびに PIT を検索して，一致するデータが存在した場合はその動作処理情報に基づいてパケットが処理される．このとき，カウンタフィールドをインクリメントする．PIT はタイマーが動作しており，設定された時間が経過するたびに各データのカウンタフィールドの値をチェックする．カウンタフィールド値が，各 EE で予め設定されている基準値に達していないデータは，通信頻度が少ないため，データを削除する．自動的に PIT データを削除する機能を持たせることで，余分な情報を持たなくて済むためセキュリティホールとなる危険性を排除することができる．

3.7 物理的位置透過性

FPN はユーザの移動に対応したシステムで，社員が出張などで自社イントラネット内から移動した場合や，部署間の移動の場合にも管理者の負荷は発生しない．一度定義した CCGI 情報は，ユーザがどこへ移動しようとも保持され，これをロケーションフリー（物理的位置透過性）と呼ぶ．図 16 のように EES3 が EEN2 のサブネットから EEN4 のサブネットへ移動した場合，EES3 は REI パケットを送信して EEN4 へ EE 設定情報を伝える．EEN4 はディジタル署名検証により，MS に証明された EE からのパケットであると認証できるため，ECT に EES3 の情報を追加する．一方，EEN2 が保持する ECT には移動前の EES3 の情報が保存されている．この情報は ECT 生存時間が超過することで自動的に削除される．移動後の EES3 が EES2 と通信する場合，新たに DPRP ネゴシエーションが行われる．この時，通信経路上に存在する EE の組み合わせは，始点 EE が EES3，中間 EE は EEN4，EEN1，終点 EE は EES2 となる．EES3-EES2 間の新しい動作処理情報が各 EE の PIT に追加される．

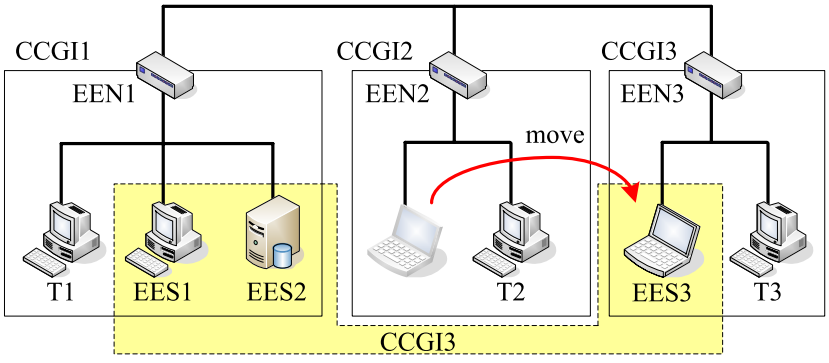


図 16　ロケーションフリー

第 4 章 実装

4.1 実装の概要

DPRP は IP 層に実装され，アプリケーションに依存することなく，全ての IP 通信をトリガーにして DPRP ネゴシエーションを行うことができる．DPRP は FPN 環境を実現するために必要な GSCIP パッケージ（GSCIP を実現するソフトウェア群）の 1 モジュールとして組み込まれる．

実装対象となる OS は FreeBSD である．FreeBSD は PC-UNIX の一種で，無料に入手できソースが公開されていること，安定性の高さ，また IP 層に関する情報や処理フローが入手しやすいため選定した[31]．図 17 のように IP 層で行われる既存の処理に一切の変更を加えず，パケットの種類を判別してから，GSCIP パッケージに処理を渡す形を採用している．従来の DPRP では DPRP 開始トリガーとなるパケットについて明確な定義がされておらず，IP パケットであれば基本的に DPRP 開始トリガーになっていた．しかし DPRP ネゴシエーションが原因で本来全ての端末に送信されなければならない TCP/IP 制御パケットが破棄される可能性が考えられる．このため，受信したパケットが RIP[2][11], OSPF[3][8], DHCP や一部の ICMP 等の場合に限り，DPRP や暗号処理モジュールの適用を除外して通常の IP 層における処理が行われる．

EEN が保持する ECT や全ての EE が保持する PIT は，パケットを受信するたびに検索することになる．そのため高速に検索できなければ通信初期遅延に影響を及ぼす．また DPRP 以外の GSCIP ソフトウェア群から参照される場合もあり得るため，カーネルの介在なしにプロセス間でデータの受け渡しが可能で，最も高速な IPC（Interprocess Communication）である共有メモリ（Shared Memory）上にハッシュテーブ

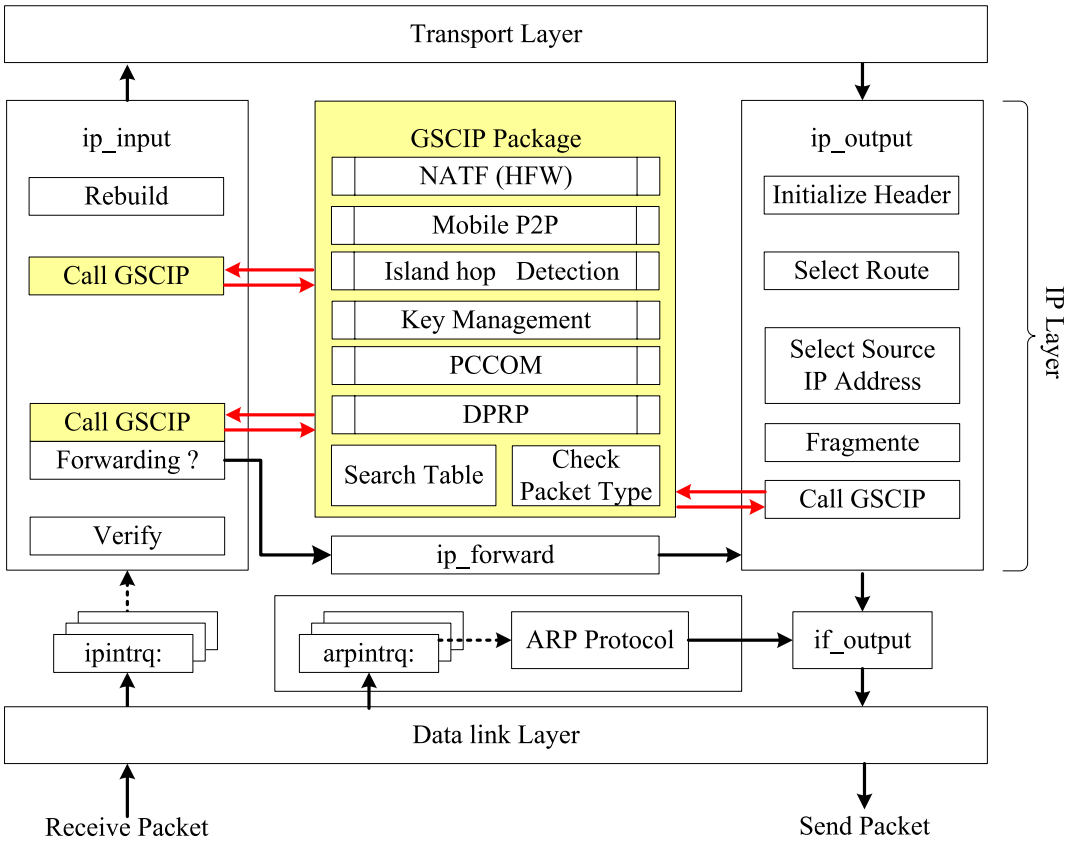


図 17　GSCIP パッケージの実装と動作フロー

ルとして構築する[32]．ハッシュテーブルの検索キーとなるのは送信元 IP アドレスとポート番号，宛先 IP アドレスとポート番号の 4 項目である．

GSCIP に実装される DPRP は IP 層だけでなく，アプリケーション層で動作するものもある．REI パケットの定期送信とネットワーク接続時送信，ECT および PIT データの削除などはデーモン化して動作させる．ECT デーモンは REI パケットのディジタル署名検証を行う機能とデータの生成および削除を行う．ECT デーモンは共有メモリ上にハッシュテーブルとして生成されるとハッシュキーの値と，生成時の時間に設定されている生存時間を加えた期限時間をアプリケーションで管理する．ECT デーモンは管理しているデータの期限時間と一致したデータのハッシュ値から，ECT の該当データを削除する．PIT デーモンは PIT に不要なデータの削除に関する機能のみ実装させる．PIT デーモンは通信時に参照されたデータのカウンタフィールドをインクリメントする．デーモンはタイマーが起動しており，予め設定された時間ごとに PIT のカウンタフィールドをチェックして，規定数に達していないデータは通信がほとんど行われてないと見なして削除する．

DPRP や暗号処理モジュールで使用される MD5 や HMAC などの関数は OpenSSL[26]を利用する．OpenSSL は SSL v2/v3（Secure Sockets Layer）と世界標準の暗号プロトコル TLS v1（Transport Layer Security）を実装した，強固で商用に耐えうる暗号ライブラリで，基本的に入手と商用および非商用の利用は自由に行える．そのため端末には OpenSSL と GSCIP パッケージがインストールされることになる．

4.2 DPRP モジュール

GSCIP パッケージに処理が移り DPRP モジュールがロードされるとき，受信したパケットの種類や暗号装置の種類などにより処理内容が異なる．DPRP ネゴシエーションを行う処理を細分化して，処理内容にあわせて組み合わせを行う．図 18 のようにサブモジュール化され，①自端末グループ鍵情報取得，②終端 EE 間で行うグループ認証，③中間 EE で行うパケットチェック，④ACG 生成，⑤ACG への自端末 EE 設定情報追加，⑥MCI 生成，⑦PIT 生成，⑧エラー生成処理と分割される．EES と EEN では果たす役割が異なるため処理内容も分かれている．双方とも最初に①の処理を行う．

EEN において DPRP が実行されるのは 5 通りの組み合わせがある．

ⅰ）受信パケットが自宛で ACG の場合

この場合さらに，始点 EE と終点 EE が同じ端末の場合と違う端末の場合の 2 パターンに分かれる．前者の場合は通常の端末と EEN の 2 台だけの通信モデルになる．このような通信モデルの場合，EEN の動作モードが閉域モードの場合，EEN の外に位置する端末は中に入れないため通信が拒否される．仮に開放モードの場合は，②→⑦の順に処理が行われる．後者の場合は②→⑥→⑦の順に処理が行われる．②の認証処理において失敗した場合，原因を伝えるために⑧エラー生成処理を行う．

ⅱ）受信パケットが自宛で MCI の場合

②→⑦の順に処理が行われる．

ⅲ）受信パケットが他宛で ACG の場合

この場合，通信経路上の EEN の位置により動作が異なる．中間 EE の位置の場合，③→⑤の順に処理を行い，ACG を中継する．終点 EE の場合，ACG パケットタイプが通知の場合は上記と全く同様にして行うが，ACG パケットタイプが応答の場合は②→⑥→⑦の順に処理

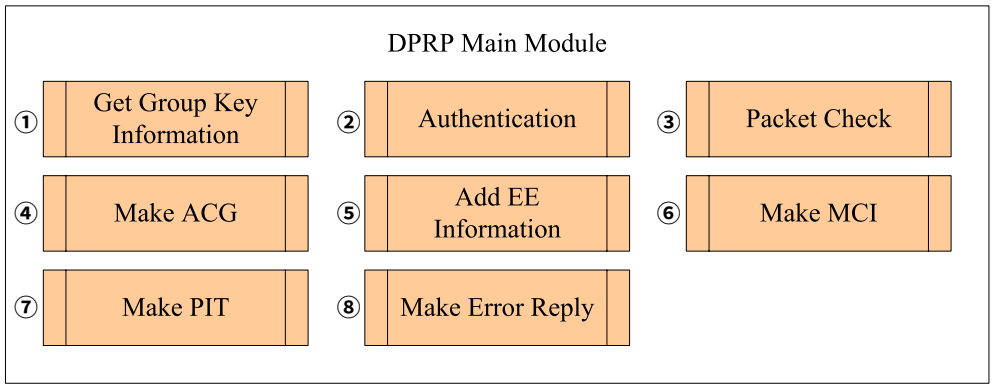


図 18　DPRP モジュール構成

が行われる.

ⅳ）受信パケットが他宛で MCI の場合

③→⑥の順に処理を行い, MCI を中継する.

ⅴ）受信パケットが他宛で上記以外の場合

DPRP ネゴシエーションを開始するために④を行ってから ACG を送信する.

EES において DPRP が実行されるのは 3 通りの組み合わせがある.

ⅰ）受信パケットが ACG の場合

②の処理を行い, 認証に成功した場合または認証に失敗して EES が開放モードの場合は, ⑦を行う. 認証に失敗して閉域モードの場合は, ACG は破棄されて⑧を行う.

ⅱ）受信パケットが MCI の場合

ACG と同様に②を行った結果，認証に成功した場合または認証に失敗して EES が開放モードの場合は，⑦を行う．認証に失敗して閉域モードの場合は, MCI は破棄されて⑧を行う.

ⅲ）パケット送信時 PIT にデータがない場合

DPRP ネゴシエーションを開始するために④を行ってから ACG を送信する.

4.3 評価プログラム

DPRP は GSCIP パッケージに含まれるため, 本来は鍵配送や暗号処理モジュールと連携して動作する．そのため IP 層に実装する形で開発を進めた場合, 他のモジュールとの連携などを考慮しなければならず DPRP 個別の性能評価が難しい. また動作確認をするためにその都度, FreeBSD カーネルの再構築を行うのは現実的ではないので，DPRP の動作確認および性能評価を行うためのテストプログラムを開発する. 評価プログラムはソケットを利用したクライアント／サーバシステムの形式で IP 層に実装されるものとほぼ同じ動作をする．中間 EE では BPF（BSD Packet Filter）を利用して通過する制御パケットを ECT によってチェックする．BPF は FreeBSD に実装されているインタフェースで，Ethernet 等のデータリンク中を流れるパケットのモニタリングや, データリンクフレームの送受信が可能である[30]．ただし BPF を利用してもパケットのコピーを取得して評価プログラムでチェックするため， ECT によるチェックで通過が認められなくても, 元のパケットは既に流れてしまっている. そのため，テストプログラムでは中間 EE の動作確認は限界があり, パケットのチェック時間の計測は行える.

現段階で DPRP の正式な仕様が決定していないため，一部分しか完成していない．図 18 の①，②，④，⑥に該当する機能は実装しているため，ACG 生成時間や認証に必要な時間などの計測が行える．（評価プログラムのソースコードは付録に添付）

第５章 評価

5.1 検討案の機能評価

第 3 章で検討した 4 案の比較評価を行う. テストプログラムが完成していないため, ここでは性能評価ではなく，機能評価を行う.

検討案 1 は始点 EE に通信経路上の EE を通過できるように, 予め鍵を配送する方式である. これにより中間 EE では制御パケットの認証を行うことが可能になり, 不正なパケットを早期段階で発見，破棄することができる．また 2 往復のネゴシエーションをしていた従来方式に対して，検討案 1 では 1.5 往復にすることが可能になった. シーケンス数が減少したことにより, ネットワークトラフィックの軽減につながると考えられる. 一方問題点はグループの重複帰属である．始点 EE に複数のグループ鍵を配送させたため, 表 3 の CCGI 情報が設定されるため, ネットワーク構成および CCGI 構成は図 19 のようになる. この鍵の持たせ方は EES3 から EES2 の通信が可能になる場合で，逆方向の通信をする場合である. EES3 は CCGI3 に加えて CCGI1，CCGI2 に重複帰属することになる．図 19 から判断できるように CCGI 構成が複雑になり, 管理者が行う CCGI 設定および管理が難しくなると予想できる. 検討案 1 における通信マトリックスは表 9 のように表される. EES3 が EEN1 を通過するために GK1 を保持したことから，本来なら通信してはいけない T1 と通信ができてしまう．EES3-EEN1 間が GK1 により暗号化され, その後平文で T1 へ達する.

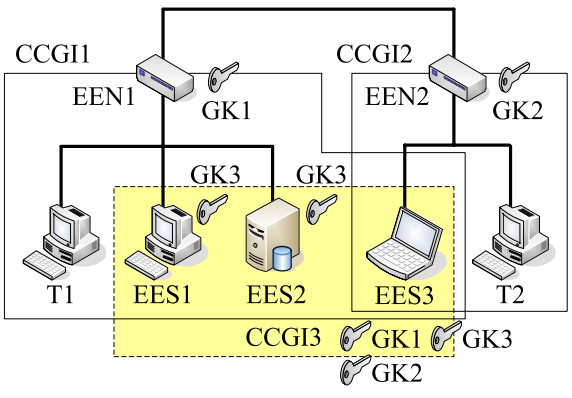


図 19　検討案 1 における CCGI 構成

逆に T1 から EES3 への通信は始点 EE となる EEN1 が GK2 を保持していないため，通信は拒否される．しかし，EES3 から T1 への DPRP ネゴシエーションが行われると PIT データが生成される. このとき逆方向のデータも生成されるため，このデータが存在する間は TI から EES3 への通信が可能になってしまう．このように常に一定した動作処理情報が生成されないため, 予期しない通信が行われる可能性がある. 図 19 および表 9 は EES3 が EES2 へ通信する場合の構成であるため, 他の端末と通信する場合や, ユーザの移動を考慮すると更なるグループ鍵を保持させなければならず，検討案 1 は現実的でない.

表 9　検討案 1 の通信マトリックス

	EES1	EES2	EES3	T1	T2
EES1		E3	E3	P	×
EES2	E3		E3	×	×
EES3	E3	E3		E1*	P
T1	P	×	×		×
T2	×	×	P	×	

※Ex; Encrypted communication by GKx
Ex*; Partial Encrypted communication by GKx
P; Communication by clear text
×; Impossibility communication

検討案 2 は中間 EE においてディジタル署名検証を行う方式である．これにより検討案 1 と同様に制御パケットの検証を行うことが可能になり，不正なパケットの早期発見，破棄することができる. 鍵配送時に行われるディジタル署名検証機能を利用するため，各 EE に新たに保持させるものはない. ディジタル署名を検証するのはどの EE でも行えるため，シーケンスは 2 往復から 1 往復へ減らせる. ただしディジタル署名のため公開鍵暗号を利用することになる. 一般的に共通鍵暗号に比べて処理時間が遅いため，中間 EE で毎回行うことにより，通信の初期遅延の増加が予想される.

検討案 3 は全ての EE にグループ鍵とは別の共通の暗号鍵を持たせて，中間 EE を認証する方式である．検討案 2 と違い MAC を利用して中間 EE を認証するために，処理速度は従来方

式と比較した場合は遅くなっているが, 検討案 2 と比較した場合は早くなっている．この方式では EE が全て共通の暗号鍵を保持しているので，通常の端末が偽造パケットを生成しても，中間 EE で認証されず破棄されるが，悪意を持ったユーザが EE で偽造パケットを生成して送信した場合, 中間 EE では認証され中継される．そのためセキュリティ面での心配がある．

改良方式（検討案 4）は予め ECT を保持させて，中間 EE を通過する制御パケットの検証を行う方式である. ディジタル署名検証を利用して生成された ECT と制御パケットを比較することで，MS に証明された EE から送信されたパケットかどうか，MS に証明された EE へ送信されるパケットかどうか検証ができる. また検討案 2，検討案 3 のように中間 EE で認証処理を行わず，ハッシュテーブルの ECT と比較するだけなので，DPRP ネゴシエーションにおける初期遅延を極力抑えることが可能である．

従来方式と各検討案の機能評価をまとめると表 10 のようになる．評価項目は中継処理, 処理時間, シーケンス数, 管理負荷, 追加設定, 認証方法，安全性の向上についてまとめた．CCGI 管理負荷について検討案 2，改良方式は従来方式と同様に行われる. 検討案 3 はこれらに共通暗号鍵の管理が追加されるため, 負担がやや増加する. 検討案 1 はユーザの移動や中間 EE の位置を把握して設定しなければならず，非常に難しい．EE に追加する設定について，検討案 1 は CCGI 初期情報の定義時に中間 EE を通過させるため複数のグループ鍵を追加する．検討案 3 は共通秘密鍵を追加する．検討案 2，改良方式はディジタル署名検証を利用するため秘密鍵，公開鍵情報などを使用する．これらの情報は MS からの鍵配送の認証にも利用されるため，DPRP における特別な追加事項ではない．改良方式はこれに加えて，ECT データを新たに追加する．End-End で行われる認証方法は, 従来方式から MAC を利用した GMAC を使用することで, 安全性と信頼性を確保することができた. 従来方式を基準として安全性の向上に注目すると, ディジタル署名検証を利用する検討案 2 と改良方式が最も高いセキュリティを確保することができる．

以上の考察から, セキュリティ強度や導入性など総合的に評価すると，4 案の中では改良方式が最も優れた方式であることがわかる．

5.2 改良方式の性能評価

改良方式の有効性を検証するために, 評価プログラムを利用して性能を評価する．DPRP の仕様が確定していないため, 評価プログラムは DPRP ネゴシエーションの一部分のみ評価可能である．評価を行う実験装置は表 11 に示す

表 10　各検討案の機能比較評価

	従来方式	検討案 1	検討案 2	検討案 3	改良方式
中間 EE におけるパケットの中継処理	無条件中継	グループ鍵認証	ディジタル署名検証	MAC 認証	ECT 検証
DPRP ネゴシエーションの処理時間	◎	△	×	△	○
シーケンス数	2 往復	1.5 往復	1 往復	1 往復	1 往復
CCGI 管理負荷	○	×	○	△	○
EE に対する追加設定	―	グループ鍵（複数）	EE 秘密鍵 MS 公開鍵 認証情報 （鍵配送時に使用する鍵を利用）	共通秘密鍵	ECT EE 秘密鍵 MS 公開鍵 認証情報 （鍵配送時に使用する鍵を利用）
End-End 認証	暗号化乱数の交換	GMAC 認証	GMAC 認証	GMAC 認証	GMAC 認証
安全性の向上	―	△	◎	○	◎

仕様のものを利用した．暗号ライブラリは OpenSSL-0.9.6l を利用する．使用するハッシュアルゴリズムは MD5, HMAC-MD5 の 2 つである. なお評価プログラムは C 言語で作成して，処理時間計測は GMAC 生成，ACG パケット生成，グループ鍵認証の 3 項目を実施した．

表 11　実験装置の性能

	PC1	PC2
CPU	Pentium4 2.4GHz	PentiumII 233MHz
RAM	1024 MB	196MB
OS	FreeBSD 5.1-Release	FreeBSD 5.2-Release

GMAC 生成の処理計測は make_gmac 関数の処理時間と一致する. make_gmac 関数は 128bit の乱数を発生させて，グループ鍵を使用して HMAC-MD5 から MAC を生成する．乱数と MAC を結合して GMAC を生成する．ACG パケット生成の処理計測は make_acg 関数の処理時間と一致する．make_acg 関数は IP ヘッダ，ICMP ヘッダ，ACG ヘッダ，GMAP，ACG 情報ヘッダ，ACG 情報が順に生成され，ACG パケットが生成される．GMAC および GMAP は保持しているグループ鍵の数だけ生成する. 実験ではグループ鍵を 3 つ保持させた. グループ鍵の認証は 1 つの GMAC を生成した後，生成時に使用したグループ鍵を使用して認証処理を行った．認証方法は生成した GMAC の前半 16Byte を取得して HMAC-MD5 により MAC を生成する．生成した MAC と GMAC の後半 16Byte を比較する．

実験条件は make_gmac 関数，make_acg 関数を 10,000 回繰り返し, 1 回当たりの処理時間を導く．これを 10 回試行した場合の平均値を算出する．処理結果は表 12 のようになった．

表 12　性能評価

	PC1 [μ sec]	PC2 [μ sec]
GMAC 生成	12.185	90.547
ACG 生成	39.609	288.203
グループ鍵認証	9.219	63.984

上記の結果から，ACG 生成処理の大部分は GMAC 生成の処理が行われていることがわかる．PC1 では 39.609[μ sec]のうち，GMAC 生成が 36.555[μ sec]（約 92%）を，PC2 では 288.203[μ sec]のうち, GMAC 生成が 271.641[μ sec]（約 94%）を占めている．社員のグループ帰属数を考慮した場合，部署単位で 1，支社単位で 1，グループ単位で 1～2 と見積もった場合でもグループ鍵を 4 つ保持することになる．PC1 の場合，DPRP ネゴシエーションによる通信初期遅延は約 0.1～0.2[msec]+ネットワーク遅延になると推測できる．PC2 の場合，ACG 生成に約 0.4[msec]の処理時間がかかることが予測される. この値から DPRP ネゴシエーションによる通信初期遅延は約 1.0～1.3[sec]+ネットワーク遅延になると推測される. ここまで遅延が大きくなると, アプリケーションに影響を与えることが考えられる．PC2 のスペックは EEN を想定している. 現在の IPsec セキュリティゲートウェイの CPU は約 150～200MHz で動作しており，MD5 などによるハッシュ値生成や IPsec 処理は全て暗号アクセラレータ LSI により処理される. ソフトウェア処理に比べてハードウェア処理の場合, ハッシュ生成などの計算処理は約 100～200 倍の速さで処理が可能であり[27]，IPsec の全処理は約 10 倍以上の高速化が実現されている[28]．そのため，EEN に暗号アクセラレータ LSI を実装させることにより, PC2 の処理性能は運用上問題ない値となる. また小・中規模の企業であれば，社員 1 人当たりのグループ帰属数が 1～2 つになるため，初期遅延は表 12 の値より小さくなると考えられる．

第６章 総括

FPN 環境下の通信に先立って行われる DPRP ネゴシエーションの課題を解決するために，中間 EE に制御パケットの検証を行う改良の検討を行った．具体的な実現手段として，各 EEN は配下に存在する全ての EE の設定情報を把握して ECT を保持して，中間 EE となる EEN は制御パケットと ECT を比較して検証を行った．このデータを生成するときに，REI によるディジタル署名検証機能を利用することで，MS が認証した EE から送信されたパケットであることを検証することが可能になった．この結果，確実に制御パケットの認証処理が可能になり，安全に PIT の作成が可能になった．中間 EE で検証することにより不正なパケットを早期段階で発見および破棄できるため，DoS 攻撃を未然に防ぐ効果も期待できる．また終端 EE 間で行うグループ認証処理は従来の暗号化された乱数のやりとりから，送信元の認証とパケットの改竄検出が可能な HMAC-MD5 を利用することで，安全性と信頼性を向上させることができた．これらの改良により，セキュリティの効果が見込めた．またテストプログラムを利用して，ACG 生成結果などから DPRP ネゴシエーションの遅延を推測して，性能上問題のないことが示された．

今後は柔軟なネットワーク構成に対応できるように DPRP の改良を進める．複雑な通信経路の場合の動作処理情報の決定方法の検討が必要である．現段階での ECT，PIT 検索は通信端末間の IP アドレスとポート番号の組などで行っているため，第三者が IP アドレスを偽造した場合は，通信経路上の EE が保持する ECT，PIT 情報を不正に利用して通信パケットを送信する可能性がある．この場合，第三者が PIT に記述してある IP アドレスとポート番号の組に一致するグループ鍵を保持していない限り，送信された通信パケットは受信時に不正パケットだと判断でき，破棄することが可能であるが，パケット自体は中間 EE を通過してしまう．そのため IP アドレス偽造対策や制御パケット自体の完全性保証なども今後考慮しなければならない．またテストプログラムを完成させて，今回検討した 4 案のネゴシエーションを実際に動作させて性能評価を行う．次の段階として GSCIP パッケージを実装させるため，FreeBSD カーネルの改造を行い，動作確認を行う．FreeBSD での動作が確認されれば，別の PC-UNIX で最近著しい発展をしている Linux への移植を検討する．現段階では NAT に関して全く考慮に入れていないが，GSCIP の構想に含まれる NATF を利用することでイントラネット内に限定されず，プライベート空間，グローバル空間の隔たり無く DPRP が実行可能になり，FPN 環境の実現に繋がると思われる．

参 考 文 献

[1] R.Richardson，”Issues and Trends: 2003 CSI/FBI Computer Crime and Security Survey”，CSI Press Release, June 2003.

[2] C.L.Hedrick,“Routing Information Protocol”, RFC1058，June 1988.

[3] J.Moy，“OSPF specification”，RFC1131，October 1989.

[4] R.Rivest，“The MD5 Message-Digest Algorithm”，RFC1321，April 1992.

[5] M.Leech, M.Ganis, Y.Lee, R.Kuris, D.Koblas, L.Jones，“SOCKS Protocol Version 5”，RFC1928，March 1996.

[6] H.Krawczyk, M.Bellare, R.Canetti,“HMAC: Keyed-Hashing for Message Authentication”, RFC2104，February 1997.

[7] R. Droms，“Dynamic Host Configuration Protocol”，RFC2131，March 1997.

[8] J.Moy，“OSPF Version 2”，RFC2328，April 1998.

[9] S. Kent，R. Atkinson，“Security Architecture for the Internet Protocol”，RFC2401，November 1998.

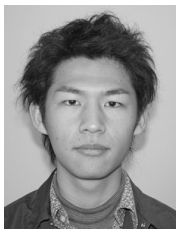
[10] D.Harkins，S.Carrel，“The Internet Key Exchange (IKE)”, RFC2409, November 1998.

[11] G.Malkin，“RIP Version 2”，RFC2453，November 1998.

[12] E.Rosen，Y.Rekhter，“BGP/MPLS VPNs”，RFC2547，March 1999.

[13] P.Hoffman，“SMTP Service Extension for Secure SMTP over Transport Layer Security”，

RFC3207，February 2002.
[14] 渡邊晃，井手口哲夫，笹瀬巌，“イントラネット閉域通信グループの物理的位置透過性を可能にする動的処理解決プロトコルの提案”，信学論（D-I），vol.J84-D-I, No.3，pp.269-284，March 2001.
[15] 渡邊晃，岡崎直宣，朴美娘，井手口哲夫，笹瀬巌，“イントラネット閉域通信グループの構築に適した安全な鍵配送方式とその運用管理方式”，電気学会論文誌 C，vol.121-C, No.9, pp.1429-1438, Sept. 2001.
[16] 竹内元規，渡邊晃，“移動体通信におけるコネクションを維持した通信方式の研究”，情報処理学会第 66 回全国大会 講演論文集 3-463，March 2004.
[17] 加藤尚樹，渡邊晃，“NAT を意識しない個人ネットワークを管理する Home Fire Wall の提案”，情報処理学会第 66 回全国大会 講演論文集 3-469，March 2004.
[18] 鈴木秀和，渡邊晃，“GSCIP を構成する DPRP の仕組みの検討”，情報処理学会第 66 回全国大会 講演論文集 3-479，March 2004.
[19] 竹尾大輔，渡邊晃，“GSCIP を構成する渡り歩き検出機能の仕組みの検討”，情報処理学会第 66 回全国大会 講演論文集 3-481，March 2004.
[20] 増田真也，渡邊晃，“閉域通信グループにおける暗号通信方式の検討”，情報処理学会第 66 回全国大会 講演論文集 3-481，March 2004.
[21] 保母雅敏，渡邊晃，“多段構成ネットワークにおける鍵配送方式の一検討”，情報処理学会第 66 回全国大会 講演論文集 3-495，March 2004.
[22] 前羽理克，渡邊晃，“生体認証を利用したセキュアネットワーク通信”，情報処理学会第 66 回全国大会 講演論文集 3-503，March 2004.
[23] 鈴木秀和，渡邊晃，“イントラネットに柔軟な閉域通信グループを実現する動的処理解決プロトコル DPRP の検討”，電気関係学会東海支部連合大会 pp.359，October 2003.
[24] ITAA，http://www.itaa.org/news/pr/PressRelease.cfm?ReleaseID=1055867633，2003/06/17.
[25] 名城大学理工学部 渡邊研究室，http://www-is.meijo-u.ac.jp/~watanabe/ 研究紹介内，2003.
[26] OpenSSL Project，http://www.openssl.org/
[27] 富士通 LSI ソリューション株式会社，http://pr.fujitsu.com/jp/news/2003/07/1.html
[28] 株式会社日立超 LSI システムズ，http://www.hitachi-ul.co.jp/PROD/MICOM/index.html，2003/11/06
[29] 馬場達也，マスタリング IPsec，オライリー・ジャパン，2001.
[30] 村山公保，基礎からわかる TCP/IP ネットワーク実験プログラミング－Linux/FreeBSD 対応，オーム社，2001.
[31] Gary R. Wright，W. Richard Stevens，詳細 TCP/IP Vol. 2 実装，ピアソン・エデュケーション，2002.
[32] W. Richard Stevens，UNIX ネットワークプログラミング第 2 版 Vol.2，ピアソン・エデュケーション，2002.



鈴木　秀和（渡邊研究室 B4）
2004 年名城大学理工学部情報科学科卒業．同年，同大学院理工学研究科情報科学専攻修士課程進学．ネットワークセキュリティの研究に従事．2001，2002, 2003 年度情報科学科 Web コンクール最優秀賞，2004 年度優秀賞受賞．2002 年度情報科学科プログラミングコンテスト優秀賞/技術賞受賞．同大学 TEC 所属．情報処理学会会員．

渡邊　　晃（教授）
1974 年慶応大学電気工学科卒業．1976 年同大学院修士課程修了．同年三菱電機（株）入社．以来，同社情報技術総合研究所にて LAN，ネットワークセキュリティ等の研究開発に従事．新エネルギー・産業技術総合開発機構（NEDO）を経て，現在，名城大学理工学部教授．情報処理学会会員．電子情報通信学会会員．

謝辞

本研究を遂行するにあたり，多大なるご指導，ご鞭撻を賜りました渡邊晃教授に心より感謝します．また有益なご助言，ご検討いただきました渡邊研究室の学部生の皆さんに深く感謝いたします．

付録

A. DPRP モジュール設計とインタフェース

A-1. DPRP メインモジュール

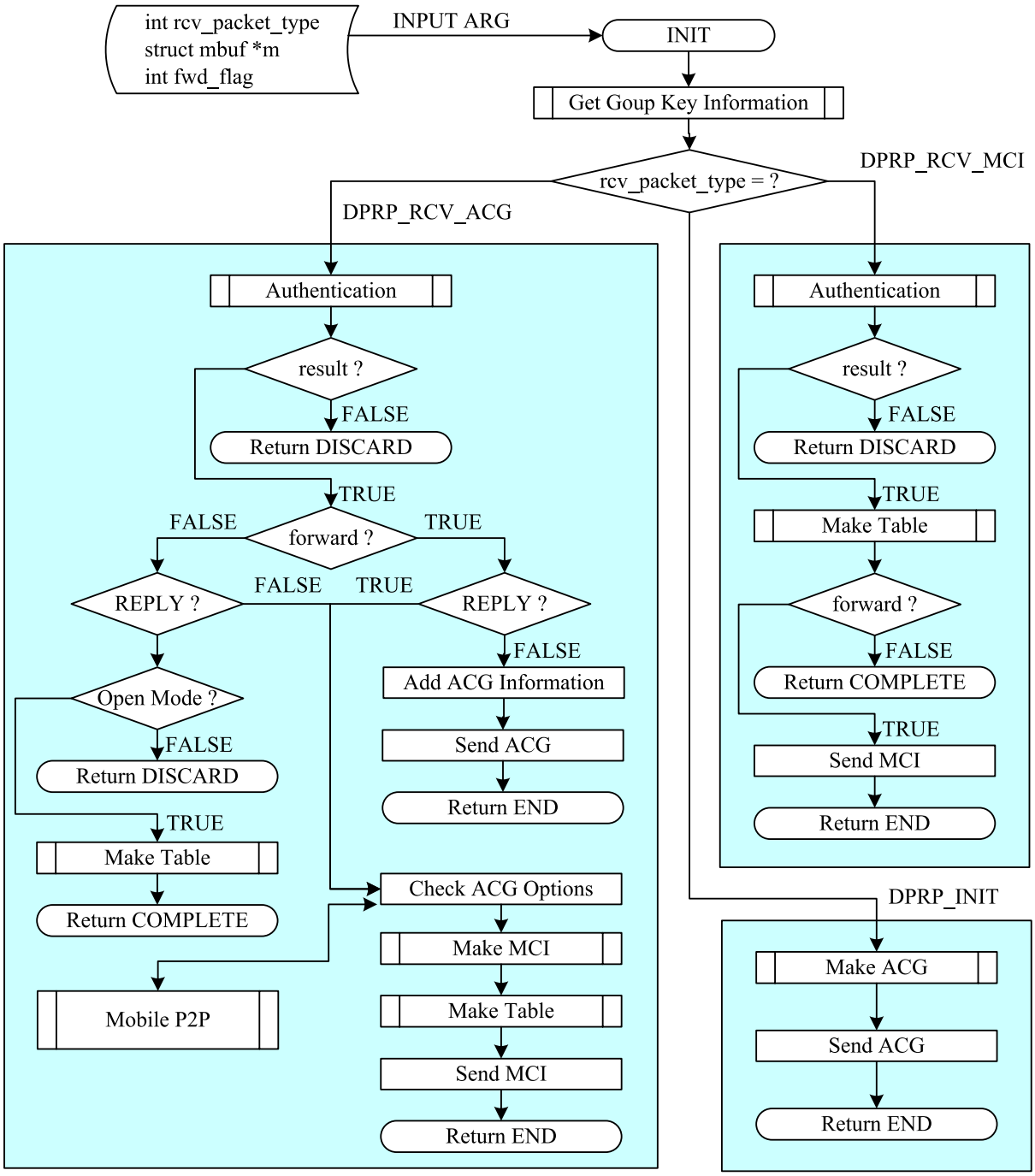


図 A-1　DPRP メインモジュールのフローチャート

Input argument

struct mbuf *m	パケットデータ
int recv_packet_type	処理切り替え
DPRP_RCV_ACG	ACG/ACG REPLY 受信
DPRP_RCV_MCI	MCI 受信
DPRP_INIT	上記以外のパケット受信
Int fwd_flag	転送フラグ

Return value

END	DPRP 処理の正常終了
COMPLETE	保持パケット処理
DISCARD	受信パケット破棄

A-2. サブモジュール　－　Authentication

int authenticate_gmac (u_char *gmac , u_char *g_key)

Input argument

u_char *gmac	GMAC データ
u_char *g_key	グループ鍵

Return value

TRUE	認証成功
FALSE	認証失敗

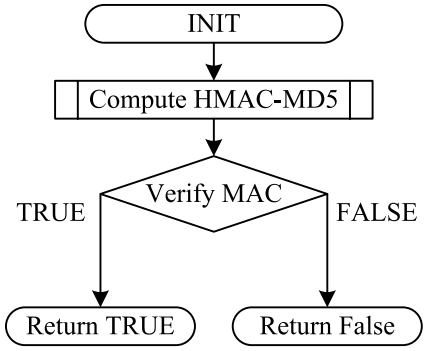


図 A-2　Authentication module

A-3. サブモジュール　－　Make ACG

int make_acg (struct mbuf *m, GKEY_INFO *gki, int gkey_count)

Input argument

struct mbuf *m	パケットデータ
GKEY_INFO *gki	グループ鍵情報
int gkey_count	グループ鍵の数

Return value

ip_len	ACG パケット長

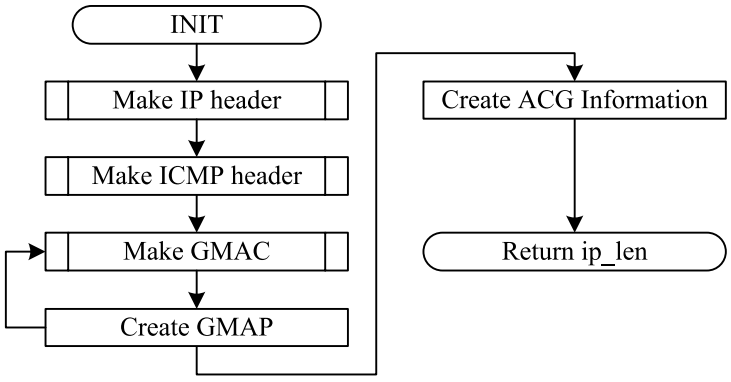


図 A-3　Make ACG module

A-4. サブモジュール　－　Make MCI

int make_mci (struct mbuf *m , GKEY_INFO *gki, int gkey_count)

Input argument

struct mbuf *m	パケットデータ
GKEY_INFO *gki	グループ鍵情報
int gkey_count	グループ鍵の数

Return value

ip_len	MCI パケット長

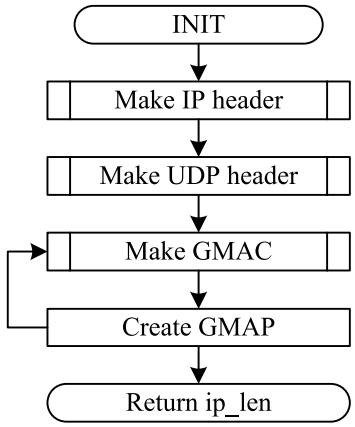


図 A-4　Make MCI module

A-5. サブモジュール　－　Make PIT

void make_table (int src_ip , int dst_ip , int process , GKEY_INFO *gki)

Input argument

int src_ip	送信元 IP アドレス
int dst_ip	宛先 IP アドレス
int process	動作処理内容
GKEY_INFO *gki	グループ鍵情報

Return value

－	－

- PROC_ENCRYPT 暗号化
- PROC_DECRYPT 復号化
- PROC_FORWARD 転送
- PROC_CREATE 作成中
- PROC_DISCARD 破棄

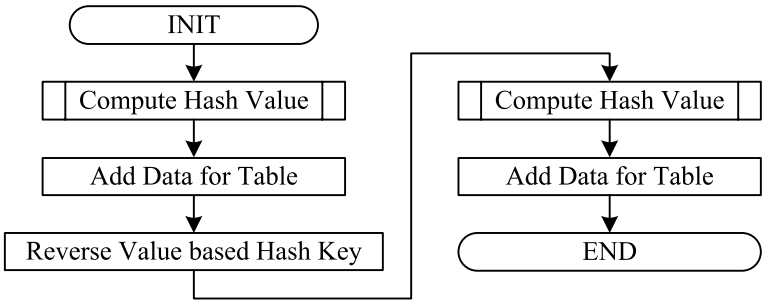
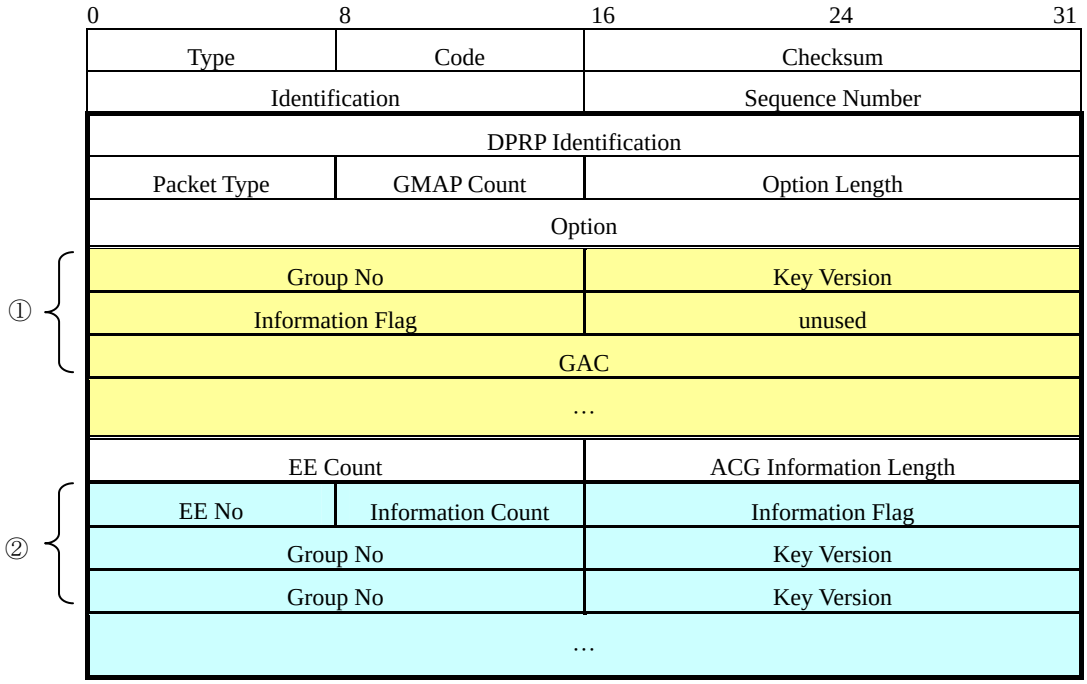


図 A-5　Make PIT module

A-6. DPRP 制御パケット　－　ACG



フィールド名	bit	内容
DPRP Identification	128	DPRP 制御パケットと認識させるための定義情報
Packet Type	8	ACG パケットタイプ（通常/移動体通信処理）
GMAP Count	8	GMAP の個数
Option Length	16	オプションフィールドの長さ
Option	可変	オプションフィールド
GMAP	320	GMAP（①）
Group No	16	グループ番号
Key Version	16	鍵バージョン
Information Flag	16	情報フラグフィールド
Unused	16	未使用
GAC	256	GAC（乱数と HMAC-MD5 値をグループ鍵で暗号化）
EE Count	16	現在までに通過してきた EE の個数
ACG Information Length	16	ACG 情報全体の長さ
ACG Information Data	可変	ACG 情報（②）
EE No	8	EE の番号（始点=1 とし EE を通過する度にインクリメント）
Information Count	8	知らせる情報の個数（=認証できた個数）
Information Flag	16	情報フラグフィールド
Group No	16	グループ番号（認証できたもの）
Key Version	16	鍵バージョン（認証できたもの）

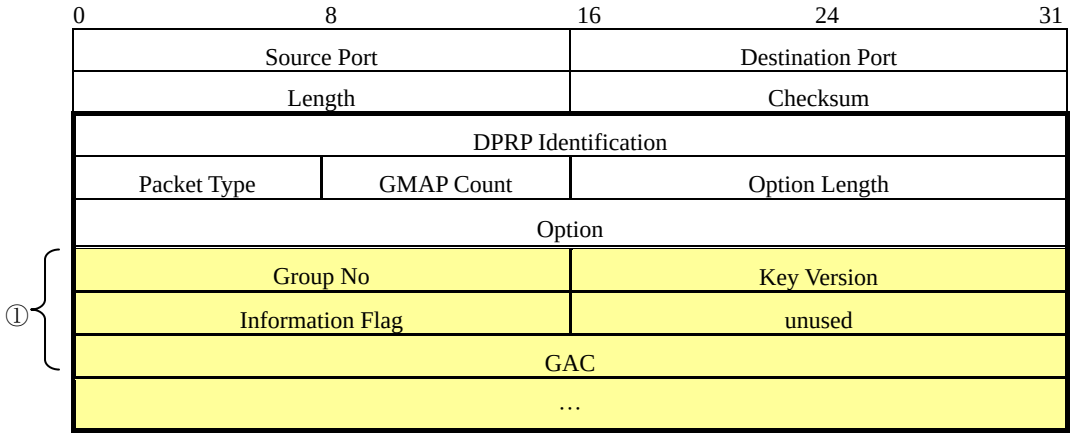
※　パケットフォーマットの太枠が ICMP データ，黄色が GMAP，水色が ACG Information を示す

※　ACG パケットの GMAP に含まれる情報フラグフィールドは値の設定を行わなくてもよい

※　①，②の後ろにある「…」フィールドは①，②がそれぞれ続くことを示す（環境により異なる）

※　上表は ICMP データ部分のみのフィールド

A-7. DPRP 制御パケット　－　MCI



フィールド名	bit	内容
DPRP Identification	128	DPRP 制御パケットと認識させるための定義情報
Packet Type	8	MCI パケットタイプ（通常/移動体通信処理）
GMAP Count	8	GMAP の個数
Option Length	16	オプションフィールドの長さ
Option	可変	オプションフィールド
GMAP	320	GMAP（①）
Group No	16	グループ番号
Key Version	16	鍵バージョン
Information Flag	16	情報フラグフィールド
Unused	16	未使用
GAC	256	GAC（乱数と HMAC-MD5 値をグループ鍵で暗号化）

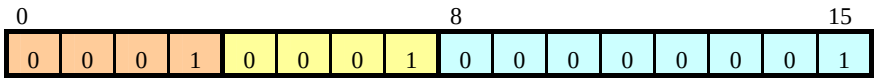
※　パケットフォーマットの太枠が UDP データ，黄色が GMAP を示す

※　①の後ろにある「…」フィールドは①がそれぞれ続くことを示す（個数は環境により異なる）

※　上表は UDP データ部分のみのフィールド

A-8. 情報フラグフィールド

このフラグは EE の状態や動作処理内容を示すために設定される．サイズは 16bit あり，上位 8bit の前半は始点/中間/終端という EE の位置，後半は開放/閉域という動作モードを示し，下位 8bit は暗号/復号/透過/作成中などの動作処理情報を示す．



0	予約		
1	終点 EE	#define EE_END	0x04
2	中間 EE	#define EE_MIDDLE	0x02
3	始点 EE	#define EE_INIT	0x01
4	予約		
5	予約		
6	開放モード	#define EE_OPEN	0x02
7	閉域モード	#define EE_CLOSE	0x01

8	予約		
9	予約		
10	予約		
11	破棄	#define PROC_DISCARD	0x0010
12	作成中	#define PROC_CREATE	0x0008
13	透過	#define PROC_FWD	0x0004
14	復号化	#define PROC_DECRYPT	0x0002
15	暗号化	#define PROC_ENCRYPT	0x0001

B. 評価プログラム

DPRP ネゴシエーションの動作確認と処理速度を計測する評価プログラムの概要について述べる．プログラムは“./dprpc [IP Address] [Port No.]”で実行する．現時点でグループ鍵情報はテキストファイル gkey_info.txt から読み込む．ファイルは左から順に CCGI 番号，鍵バージョン，グループ鍵となっている．

```
1  100  d0c6dd891fdf1695144e6e7eb8be00d9
2  200  374b972a549e5afc1765dec0fe6b265a
3  300  1630ca9d17348a2f619e99902210cb95
```

図 B-1　gkey_info.txt

B-1. dprpc.c

```
/*********************************************************************
                        dprp.c  -  description
                           -------------------
    begin                : 2003 年 12 月 10 日 曜日 05 時 50 分 31 秒 JST
    copyright            : (C) 2004 by Hidekazu Suzuki
    email                : a300j075@ccmailg.meijo-u.ac.jp

*********************************************************************/

/*********************************************************************
 *                                                                   *
 *   This program is free software; you can redistribute it and/or modify   *
 *   it under the terms of the GNU General Public License as published by   *
 *   the Free Software Foundation; either version 2 of the License, or       *
 *   (at your option) any later version.                                     *
 *                                                                   *
 *********************************************************************/

#include "includes.h"

#define MAXSIZE    4096
#define OPTNUM     8
#define ON 1
#define OFF 0

#define  GKEY_FILE      "gkey_info.txt"

enum {ETHER, ARP, IP, TCP, UDP, ICMP, DUMP, ALL};

#ifndef __linux
int open_bpf(char *ifname);
#endif

void print_ethernet(struct ether_header *eth);
void print_arp(struct ether_arp *arp);
void print_ip(struct ip *ip);
void print_icmp(struct icmp *icmp);
void print_tcp(struct tcphdr *tcp);
void print_udp(struct udphdr *udp);

void dump_packet(unsigned char *buff, int len);
int     get_gkey_info(GKEY_INFO_P *gkip, int *gkey_count);

char *mac_ntoa(u_char *d);
char *tcp_ftoa(int flag);
char *ip_ttoa(int flag);
char *ip_ftoa(int flag);
void help(char *cmd);

int main(int argc, char **argv)
{
      int c;                        /* getopt()で取得した文字 */
      int opt[OPTNUM];          /* 表示オプションのフラグ */
      extern int optind;          /* getopt()のグローバル変数 */
      char buff[MAXSIZE];      /* データ送受信バッファ */

      /* 表示するパケットの種類*/
      opt[ETHER] = OFF;
      opt[ARP]     = OFF;
      opt[IP]      = ON;
      opt[TCP]     = OFF;
      opt[UDP]     = ON;
      opt[ICMP]    = ON;
      opt[DUMP]    = OFF;
      opt[ALL]     = OFF;

      /* コマンドラインオプションの検査 */
      while ((c = getopt(argc, argv, "cs")) != EOF) {
            switch (c) {
                  case 'c': /* クライアントモード */
                        printf("DPRP ClientMode:¥n");
                        printf("    0 : View Help¥n");
                        printf("    1 : Send ACG Packet¥n");
                        printf("    2 : Send MCI Packet¥n");
                        printf("    9 : Quit¥n¥n");

                        struct sockaddr_in      dest;      /* サーバのアドレス */
                        GKEY_INFO_P gkip[5];      /* グループ鍵情報格納変数 */
                        int         gkey_count = 0;      /* 取得したグループ鍵の数 */
                        int         dst_ip;                  /* サーバの IP アドレス */
                        int         size;                     /* 各種バイト数 */
                        int         s;                        /*ソケットファイルディスクリプタ*/
                        int         n;                        /* 入力コマンド番号 */
                        int         on = 1;                  /* ON */
                        int         result;                  /* 戻り値格納変数 */

                        /* 入力コマンド名 */
                        enum {CMD_HELP, CMD_ACG, CMD_MCI, CMD_QUIT=9};

                        optind--;
                        /* サーバの IP アドレスを調べる */
                        if ((dst_ip = inet_addr(argv[optind])) == INADDR_NONE) {
                              struct hostent *he;           /* ホスト情報 */

                              if ((he = gethostbyname(argv[optind])) == NULL) {
                                    fprintf(stderr, "gethostbyname error¥n");
                                    exit(EXIT_FAILURE);
                              }
                              memcpy((char *)&dst_ip, (char *)he->h_addr, he->h_length);
                        }

                        /* RAW ソケットのオープン */
                        if ((s = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
                              perror("socket(SOCK_RAW)");
                              exit(EXIT_FAILURE);
                        }

                        if (setsockopt(s, IPPROTO_IP, IP_HDRINCL, (char *)&on, sizeof(on))
< 0) {

                              perror("setsockopt(IP_HDRINCL)");
                              exit(EXIT_FAILURE);
                        }

                        /* グループ鍵情報の取得 */
                        result = get_gkey_info(gkip, &gkey_count);
                        if (result == FAILURE)    return DISCARD;
                        printf("gkey count : %d¥n", gkey_count);

                        /* クライアント処理メインルーチン */
                        while (1) {
                              /* 入力コマンドの取得 */
                              printf("CMD > ");
                              scanf("%d", &n);

                              /* 入力コマンドが"9"の場合，終了する */
                              if (n == CMD_QUIT)
                                    printf("¥nDPRP Quit...¥n¥n");
                                    break;

                              switch (n) {
                                    case CMD_ACG:
                                          printf("¥nCreate ACG Packet...¥n¥n");
                                          /* ACG パケット生成関数のロード */
/*                                        size = make_acg(buff, dst_ip, gkip, gkey_count);

                                          /* 送信アドレスの設定 */
                                          memset((char *)&dest, 0, sizeof(dest));
                                          dest.sin_family              = AF_INET;
                                          dest.sin_addr.a_addr  = dst_ip;

                                          /* ACG パケット送信 */
                                          if  (sendto(s,  buff,  size,  0,  (struct  sockaddr  *)&dest,
sizeof(dest)) < 0) {
```

```
                                perror("sendto");
                                exit(EXIT_FAILURE);
                            }
*/                          printf("¥nSend This Packet...¥n¥n");
                            break;
                        case CMD_MCI:
                            printf("¥nCreate MCI Packet...¥n¥n");

                            printf("¥nSend This Packet...¥n¥n");
                            break;
                        case CMD_HELP:
                        default:
                            printf("   0 : View Help¥n");
                            printf("   1 : Send ACG Packet¥n");
                            printf("   2 : Send MCI Packet¥n");
                            printf("   9 : Quit¥n¥n");
                            break;
                    }
                }

                break;
            case 's':  /* サーバモード */
                printf("DPRP ServerMode:¥n");

                struct ether_header *eth;  /* Ethernet ヘッダ構造体 */
                struct ether_arp *arp;     /* ARP パケット構造体 */
                struct ip *ip;             /* IP ヘッダ構造体 */
                struct icmp *icmp;         /* ICMP パケット構造体 */
                struct tcphdr *tcp;        /* TCP ヘッダ構造体 */
                struct udphdr *udp;        /* UDP ヘッダ構造体 */
                int len;                   /* 受信したデータの長さ */
                int disp;                  /* 画面に出力したかどうかのフラグ */
                char *p;                   /* ヘッダの先頭を表す作業用ポインタ */
                char *p0;                  /* パケットの先頭を表すポインタ */
                char ifname[256] = "xl0"; /* FreeBSD のインタフェース名 */
            #ifndef __linux
                int bpf_len;               /* BPF での受信データの長さ */
                struct bpf_hdr *bp;        /* BPF ヘッダ構造体 */
            #endif


            #ifdef __linux
                if ((s = socket(AF_INET, SOCK_PACKET, htons(ETH_P_ALL))) < 0) {
                    perror("socket");
                    exit(EXIT_FAILURE);
                }

                if (strcmp(ifname, "xl0") != 0) {
                    struct sockaddr sa;
                    memset(&sa, 0, sizeof sa);
                    sa.sa_family = AF_INET;
                    strcpy(sa.sa_data, ifname);
                    if (bind(s, &sa, sizeof sa) < 0) {
                        perror("bind");
                        exit(EXIT_FAILURE);
                    }
                }
            #else
                if ((s = open_bpf(ifname)) < 0)
                    exit(EXIT_FAILURE);
                bpf_len=0;
            #endif

                while (1) {
            #ifndef __linux
                    /*   BPF からの入力 */
                    if (bpf_len <= 0) {
                        /* 複数のパケットを一括取りだし */
                        if ((bpf_len = read(s, buff, MAXSIZE)) < 0) {
                            perror("read");
                            exit(EXIT_FAILURE);
                        }
                        bp = (struct bpf_hdr *)buff;
                    } else {
                        /* BPF の次のパケットへポインタを移動 */
                        bp = (struct bpf_hdr *)((char *) bp + bp->bh_hdrlen +
bp->bh_caplen);
                        bp = (struct bpf_hdr *)BPF_WORDALIGN((int)bp);
                    }
                    /* Ethernet ヘッダの先頭にポインタをセット */
                    p = p0 = (char *)bp + bp->bh_hdrlen;
                    len    = bp->bh_caplen;
            #ifdef DEBUG
                    /* BPF ヘッダ構造の値を表示 */
                    printf("bpf_len=%d,",   bpf_len);
                    printf("hdrlen=%d,",    bp->bh_hdrlen);
                    printf("caplen=%d,",    bp->bh_caplen);
                    printf("datalen=%d¥n", bp->bh_datalen);
            #endif
                    /* 次の while ループのための処理 */
                    bpf_len -= BPF_WORDALIGN(bp->bh_hdrlen + bp->bh_caplen);
            #else
                    /* Linux SOCK_PACKET からの入力 */
                    if ((len = read(s, buff, MAXSIZE)) < 0) {
                        perror("read");
                        exit(EXIT_FAILURE);
                    }
                    /* Ethernet ヘッダの先頭にポインタをセット */
                    p = p0 = buff;
            #endif
                    /*
                    * パケット表示ルーチン
                    */
                    disp = OFF;       /* 画面に出力したかどうかのフラグ */

                    /* Ethernet ヘッダ構造体の設定 */
                    eth = (struct ether_header *) p;
                    p = p + sizeof (struct ether_header);

                    if (ntohs(eth->ether_type) == ETHERTYPE_ARP) {
                        if (opt[ARP] == ON) {
                            if (opt[ETHER] == ON)
                            print_ethernet(eth);
                            arp = (struct ether_arp *)p;
                            print_arp(arp);
                            disp = ON;
                        }
                    } else if (ntohs(eth->ether_type) == ETHERTYPE_IP) {
                        ip = (struct ip *) p;
                        p = p + ((int)(ip->ip_hl) << 2);

                        if (opt[IP]==ON && opt[TCP]==OFF && opt[UDP]==OFF
&& opt[ICMP]==OFF) {
                            if (opt[ETHER] == ON)
                            print_ethernet(eth);
                            print_ip(ip);
                            disp = ON;
                        }
                        switch (ip->ip_p) {
                            case IPPROTO_TCP:
                                tcp = (struct tcphdr *) p;
                                p = p + ((int)(tcp->th_off) << 2);
                                if (opt[TCP] == ON) {
                                    if (opt[IP] == ON) {
                                        if (opt[ETHER] == ON)
                                            print_ethernet(eth);
                                        print_ip(ip);
                                    }
                                    print_tcp(tcp);
                                    disp = ON;
                                }
                                break;
                            case IPPROTO_UDP:
                                udp = (struct udphdr *) p;
                                p = p + sizeof(struct udphdr);
                                if (opt[UDP] == ON) {
                                    if (opt[IP] == ON) {
                                        if (opt[ETHER] == ON)
                                            print_ethernet(eth);
                                        print_ip(ip);
                                    }
                                    print_udp(udp);
                                    disp = ON;
                                }
                                break;
                            case IPPROTO_ICMP:
                                icmp = (struct icmp *) p;
                                p = p + sizeof(struct udphdr);
                                if (opt[ICMP] == ON) {
                                    if (opt[IP] == ON) {
                                        if (opt[ETHER] == ON)
                                            print_ethernet(eth);
                                        print_ip(ip);
                                    }
                                    print_icmp(icmp);
                                    disp = ON;
                                }
                                break;
                            default:
                                if (opt[ALL] == ON) {
                                    if (opt[IP] == ON) {
                                        if (opt[ETHER] == ON)
                                            print_ethernet(eth);
                                        print_ip(ip);
                                    }
                                    printf("Protocol: unknown¥n");
                                    disp = ON;
                                }
                                break;
                        }
                    } else {
                        if (opt[ALL] == ON) {
                            if (opt[ETHER] == ON)
                                print_ethernet(eth);
                            printf("Protocol: unknown¥n");
                            disp = ON;
                        }
                    }
                    if (disp == ON) {
                        if (opt[DUMP] == ON)
                            dump_packet(p0, len);
                        printf("¥n");
                    }
                }

                break;
            case 'h': /* ヘルプ */
            case '?':
```

```
                default:
                    help(argv[0]);
                    exit(EXIT_FAILURE);
                    break;
            }
        }

        if (optind < argc) {
            while (optind < argc)
                printf("%s ", argv[optind++]);
            printf("¥n");
            help(argv[0]);
            exit(EXIT_FAILURE);
        }

        return EXIT_SUCCESS;
}

/*
 * char *mac_ntoa(u_char *d);
 * 機能
 *     配列に格納されている MAC アドレスを文字列に変換
 * 引き数
 *     u_char *d;   MAC アドレスが格納されている領域の先頭アドレス
 * 戻り値
 *     文字列に変換された MAC アドレス
 */
char *mac_ntoa(u_char *d)
{
    static char str[50];   /* 文字列に変換した MAC アドレスを保存 */

    sprintf(str, "%02x:%02x:%02x:%02x:%02x:%02x",
            d[0], d[1], d[2], d[3], d[4], d[5]);

    return str;
}

/*
 * void print_ethernet(struct ether_header *eth);
 * 機能
 *      Ethernet ヘッダの表示
 * 引き数
 *      struct ether_header *eth; Ethernet ヘッダ構造体へのポインタ
 * 戻り値
 *      なし
 */
void print_ethernet(struct ether_header *eth)
{
    int type = ntohs(eth->ether_type);      /* Ethernet タイプ */

    if (type <= 1500)
        printf("IEEE 802.3 Ethernet Frame:¥n");
    else
        printf("Ethernet Frame:¥n");

    printf("+-------------------------+-------------------------"
           "+-------------------------+¥n");
    printf("|Destination MAC Address:                                "
           "|             %17s|¥n", mac_ntoa(eth->ether_dhost));
    printf("+-------------------------+-------------------------"
           "+-------------------------+¥n");
    printf("|Source MAC Address:                                     "
           "|             %17s|¥n", mac_ntoa(eth->ether_shost));
    printf("+-------------------------+-------------------------"
           "+-------------------------+¥n");
    if (type < 1500)
        printf("|Length:                  %5u|¥n", type);
    else
        printf("|Ethernet Type:       0x%04x|¥n", type);
    printf("+-------------------------+¥n");
}

/*
 * void print_arp(struct ether_arp *arp);
 * 機能
 *      ARP パケットの表示
 * 引き数
 *      struct ether_arp *arp;   ARP パケット構造体へのポインタ
 * 戻り値
 *      なし
 */
void print_arp(struct ether_arp *arp)
{
    static char *arp_operation[] ={
        "Undefine",
        "(ARP Request)",
        "(ARP Reply)",
        "(RARP Request)",
        "(RARP Reply)"
    };    /* オペレーションの種類を表す文字列 */
    int op = ntohs(arp->ea_hdr.ar_op);    /* ARP オペレーション */

    if (op <= 0 || 5 < op)
        op = 0;

    printf("Protocol: ARP¥n");
    printf("+-------------------------+-------------------------+¥n");
    printf("| Hard Type: %2u%-11s| Protocol:0x%04x%-9s|¥n",
            ntohs(arp->ea_hdr.ar_hrd),
            (ntohs(arp->ea_hdr.ar_hrd)==ARPHRD_ETHER)?"(Ethernet)":"(Not Ether)",
            ntohs(arp->ea_hdr.ar_pro),
            (ntohs(arp->ea_hdr.ar_pro) == ETHERTYPE_IP)?"(IP)":"(Not IP)");
    printf("+-----------+-----------+-------------------------+¥n");
    printf("| HardLen:%3u| Addr Len:%2u| OP: %4d%16s|¥n",
            arp->ea_hdr.ar_hln, arp->ea_hdr.ar_pln, ntohs(arp->ea_hdr.ar_op),
            arp_operation[op]);
    printf("+-----------+-----------+-------------------------"
           "+-------------------------+¥n");
    printf("| Source MAC Address:                                    "
           "|             %17s|¥n", mac_ntoa(arp->arp_sha));
    printf("+-------------------------------------------------------"
           "+-------------------------+¥n");
    printf("| Source IP Address:                      %15s|¥n",
            inet_ntoa(*(struct in_addr *)&arp->arp_spa));
    printf("+-------------------------------------------------------"
           "+-------------------------+¥n");
    printf("| Destination MAC Address:                               "
           "|             %17s|¥n", mac_ntoa(arp->arp_tha));
    printf("+-------------------------------------------------------"
           "+-------------------------+¥n");
    printf("| Destination IP Address:                 %15s|¥n",
            inet_ntoa(*(struct in_addr *)&arp->arp_tpa));
    printf("+-------------------------------------------------------+¥n");
}

/*
 * void print_ip(struct ip *ip);
 * 機能
 *      IP ヘッダの表示
 * 引き数
 *      struct ip *ip;     IP ヘッダ構造体へのポインタ
 * 戻り値
 *      なし
 */
void print_ip(struct ip *ip)
{
    printf("Protocol: IP¥n");
    printf("+-----+------+-----------+-------------------------+¥n");
    printf("| IV:%1u| HL:%2u| T: %8s| Total Length: %10u|¥n",
            ip->ip_v, ip->ip_hl, ip_ttoa(ip->ip_tos), ntohs(ip->ip_len));
    printf("+-----+------+-----------+-------+-----------------+¥n");
    printf("| Identifier:             %5u| FF:%3s|FO:             %5u|¥n",
            ntohs(ip->ip_id), ip_ftoa(ntohs(ip->ip_off)),
            ntohs(ip->ip_off) &IP_OFFMASK);
    printf("+-----------+-----------+-------+-----------------+¥n");
    printf("| TTL:      %3u| Pro:      %3u| Header Checksum:     %5u|¥n",
            ip->ip_ttl, ip->ip_p, ntohs(ip->ip_sum));
    printf("+-----------+-----------+-------------------------+¥n");
    printf("| Source IP Address:                      %15s|¥n",
            inet_ntoa(*(struct in_addr *)&(ip->ip_src)));
    printf("+-------------------------------------------------------+¥n");
    printf("| Destination IP Address:                 %15s|¥n",
            inet_ntoa(*(struct in_addr *)&(ip->ip_dst)));
    printf("+-------------------------------------------------------+¥n");
}

/*
 * char *ip_ftoa(int flag);
 * 機能
 *      IP ヘッダのフラグメントビットを文字列に変換
 * 引き数
 *      int flag;    フラグメントフィールドの値
 * 戻り値
 *      char *          変換された文字列
 */
char *ip_ftoa(int flag)
{
    static int   f[] = {'R', 'D', 'M'};    /* フラグメントフラグを表す文字 */
    static char str[17];                   /* 戻り値を格納するバッファ */
    u_int mask = 0x8000;                   /* マスク */
    int i;                                 /* ループ変数 */

    for (i = 0; i < 3; i++) {
        if (((flag<< i) & mask) != 0)
            str[i] = f[i];
        else
            str[i] = '0';
    }
    str[i] = '¥0';

    return str;
}

/*
 * char *ip_ttoa(int flag);
 * 機能
 *      IP ヘッダの TOS フィールドを文字列に変換
 * 引き数
 *      int flag;     TOS フィールドの値
 * 戻り値
 *      char *          変換された文字列
 */
char *ip_ttoa(int flag)
{
    static int   f[] = {'1', '1', '1', 'D', 'T', 'R', 'C', 'X'};
                                    /* TOS フィールドを表す文字 */
```

```
  static char str[17];        /* 戻り値を格納するバッファ */
  u_int mask = 0x80;          /* TOS フィールドを取り出すマスク */
  int i;                      /* ループ変数 */

  for (i = 0; i < 8; i++) {
    if (((flag<< i) & mask) != 0)
      str[i] = f[i];
    else
      str[i] = '0';
  }
  str[i] = '¥0';

  return str;
}

/*
 * void print_icmp(struct icmp *icmp);
 * 機能
 *      IP ヘッダの表示
 * 引き数
 *      struct icmp *icmp;
 * 戻り値
 *      なし
 */
void print_icmp(struct icmp *icmp)
{
  static char *type_name[] ={
    "Echo Reply",                 /* Type   0 */
    "Undefine",                   /* Type   1 */
    "Undefine",                   /* Type   2 */
    "Destination Unreachable",    /* Type   3 */
    "Source Quench",              /* Type   4 */
    "Redirect (change route)",    /* Type   5 */
    "Undefine",                   /* Type   6 */
    "Undefine",                   /* Type   7 */
    "Echo Request",               /* Type   8 */
    "Undefine",                   /* Type   9 */
    "Undefine",                   /* Type 10 */
    "Time Exceeded",              /* Type 11 */
    "Parameter Problem",          /* Type 12 */
    "Timestamp Request",          /* Type 13 */
    "Timestamp Reply",            /* Type 14 */
    "Information Request",        /* Type 15 */
    "Information Reply",          /* Type 16 */
    "Address Mask Request",       /* Type 17 */
    "Address Mask Reply",         /* Type 18 */
    "Unknown"                     /* Unknown */
  };                              /* ICMP のタイプを表す文字列 */
  int type = icmp->icmp_type;     /* ICMP タイプ */

  if (type < 0 || type > 18)
    type = 19;

  printf("Protocol: ICMP (%s)¥n", type_name[type]);

  printf("+------------+------------+-------------------------+¥n");
  printf("|Type:      %3u| Code:     %3u| Checksum:                %5u|¥n",
          icmp->icmp_type, icmp->icmp_code, ntohs(icmp->icmp_cksum));
  printf("+------------+------------+-------------------------+¥n");

  if (icmp->icmp_type == 0 || icmp->icmp_type == 8) {
      printf("|Identification:       %5u| Sequence Number:     %5u|¥n",
              ntohs(icmp->icmp_id), ntohs(icmp->icmp_seq));
      printf("+-------------------------+-------------------------+¥n");
  } else if (icmp->icmp_type == 3) {
    if (icmp->icmp_code == 4) {
      printf("|void:                %5u| Next MTU:                %5u|¥n",
              ntohs(icmp->icmp_pmvoid), ntohs(icmp->icmp_nextmtu));
      printf("+-------------------------+-------------------------+¥n");
    } else {
      printf("| Unused:                                          %10lu|¥n",
              (u_long)ntohl(icmp->icmp_void));
      printf("+-------------------------+-------------------------+¥n");
    }
  } else if (icmp->icmp_type == 5) {
    printf("| Router IP Address:                       %15s|¥n",
            inet_ntoa(*(struct in_addr *)&(icmp->icmp_gwaddr)));
    printf("+-----------------------------------------------------+¥n");
  } else if (icmp->icmp_type == 11) {
    printf("| Unused:                                          %10lu|¥n",
            (u_long)ntohl(icmp->icmp_void));
    printf("+-----------------------------------------------------+¥n");
  }

  if (icmp->icmp_type == 3 || icmp->icmp_type == 5 || icmp->icmp_type == 11)
    print_ip((struct ip *)(((char *) icmp) + 8));
}

/*
 * void print_tcp(struct tcphdr *tcp);
 * 機能
 *      TCP ヘッダの表示
 * 引き数
 *      struct tcphdr *tcp;   TCP ヘッダ構造体
 * 戻り値
 *      なし
 */
void print_tcp(struct tcphdr *tcp)
{
  printf("protocol: TCP¥n");
  printf("+-------------------------+-------------------------+¥n");
  printf("| Source Port:          %5u| Destination Port:   %5u|¥n",
          ntohs(tcp->th_sport), ntohs(tcp->th_dport));
  printf("+-------------------------+-------------------------+¥n");
  printf("| Sequence Number:                                 %10lu|¥n",
          (u_long)ntohl(tcp->th_seq));
  printf("+-----------------------------------------------------+¥n");
  printf("| Acknowledgement Number:                          %10lu|¥n",
          (u_long)ntohl(tcp->th_ack));
  printf("+------+---------+--------+-------------------------+¥n");
  printf("| DO:%2u| Reserved|F:%6s| Window Size:          %5u|¥n",
          tcp->th_off, tcp_ftoa(tcp->th_flags), ntohs(tcp->th_win));
  printf("+------+---------+--------+-------------------------+¥n");
  printf("| Checksum:             %5u| Urgent Pointer:      %5u|¥n",
          ntohs(tcp->th_sum), ntohs(tcp->th_urp));
  printf("+-------------------------+-------------------------+¥n");
}

/*
 * char *tcp_ftoa(int flag);
 * 機能
 *      TCP ヘッダのコントロールフラグを文字列に変換
 * 引き数
 *      int flag          TCP のコントロールフラグ
 * 戻り値
 *      char *            変換された文字列
 */
char *tcp_ftoa(int flag)
{
  static int   f[] = {'U', 'A', 'P', 'R', 'S', 'F'};
                              /* TCP のフラグを表す文字 */
  static char str[17];        /* 戻り値を格納するバッファ */
  u_int mask = 1 << 5;
  int i;

  for (i = 0; i < 6; i++) {
    if (((flag << i) & mask) != 0)
      str[i] = f[i];
    else
      str[i] = '0';
  }
  str[i] = '¥0';

  return str;
}


/*
 * void print_udp(struct udphdr *udp);
 * 機能
 *      UDP ヘッダを表示
 * 引き数
 *      struct udphdr *udp;    UDP ヘッダ構造体へのポインタ
 * 戻り値
 *      なし
 */
void print_udp(struct udphdr *udp)
{
  printf("Protocol: UDP¥n");
  printf("+-------------------------+-------------------------+¥n");
  printf("|Source Port:           %5u| Dest Port:           %5u|¥n",
          ntohs(udp->uh_sport), ntohs(udp->uh_dport));
  printf("+-------------------------+-------------------------+¥n");
  printf("|Length:                %5u| Ckecksum:            %5u|¥n",
          ntohs(udp->uh_ulen), ntohs(udp->uh_sum));
  printf("+-------------------------+-------------------------+¥n");
}

/*
 * void dump_packet(unsigned char *buff, int len);
 * 機能
 *      Ethernet フレーム先頭から 16 進数ダンプ(アスキー文字表示)
 * 引き数
 *      unsigned char *buff;   ダンプするデータの先頭アドレス
 *      int len;               ダンプするバイト数
 * 戻り値
 *      なし
 */
void dump_packet(unsigned char *buff, int len)
{
  int i, j;   /* ループ変数 */

  printf("Frame Dump:¥n");
  for (i = 0; i < len; i += 16) {
    /* 16 進数ダンプ */
    for (j = i; j < i + 16 && j < len; j++) {
      printf("%02x", buff[j]);
      if (j % 2 == 1)
        printf(" ");
    }

    /* 最後の行の端数を整列 */
    if (j == len && len % 16 != 0)
      for (j = 0; j < 40 - (len % 16)*2.5; j++)
        printf(" ");
      printf(": ");
```

```
        /* アスキー文字表示 */
        for (j = i; j < i + 16 && j < len; j++) {
          if ((buff[j] >= 0x20) && (buff[j] <= 0x7e))
            putchar(buff[j]);
          else
            printf(".");
        }

        printf("¥n");
    }
    fflush(stdout);
}

#ifndef __linux
/*
 * int open_bpf(char *ifname);
 * 機能
 *      BPF をオープンする
 * 引き数
 *      char *ifname;   インタフェース名
 * 戻り値
 *      int             ファイルディスクリプタ
 */
int open_bpf(char *ifname)
{
    char buf[256];      /* 文字列格納用 */
    int bpfd;           /* ファイルディスクリプタ */
    struct ifreq ifr;   /* インタフェース属性構造体 */
    int i;              /* ループ変数*/

    /* BPF デバイスファイルのオープン */
    for (i = 0; i < 4; i++) {
      sprintf(buf, "/dev/bpf%d", i);
      if ((bpfd = open(buf, O_RDWR, 0)) > 0)
        goto bpf_ok;
    }
    fprintf(stderr, "cannot open BPF¥n");
    return -1;

 bpf_ok:
    /* インタフェース名の設定 */
    strcpy(ifr.ifr_name, ifname);
    if (ioctl(bpfd, BIOCSETIF, &ifr) < 0) {
      sprintf(buf, "ioctl(BIOCSETIF, '%s')", ifname);
      perror(buf);
      return -1;
    }
    fprintf(stderr, "BPF read from '%s' (%s)¥n", ifr.ifr_name, buf);

    /* promiscuous モード */
    if (ioctl(bpfd, BIOCPROMISC, NULL) < 0) {
      perror("ioctl(BIOCPROMISC)");
      return -1;
    }

    /* 即時モード */
    i = 1;
    if (ioctl(bpfd, BIOCIMMEDIATE, &i) < 0) {
      perror("ioctl(BIOCIMMEDIATE)");
      return -1;
    }

    return bpfd;
}
#endif

void help(char *cmd)
{
    fprintf(stderr, "usage: %s [-aedh] [-i ifname] [-p protocols]¥n", cmd);
    fprintf(stderr, "protocols: arp ip icmp tcp udp other¥n");
#ifdef __linux
    fprintf(stderr, "default: %s -p arp ip icmp tcp udp¥n", cmd);
#else
    fprintf(stderr, "default: %s -i xl0 -p arp ip icmp tcp udp¥n", cmd);
#endif
}

/* グループ鍵情報取得 */
int get_gkey_info(GKEY_INFO_P *gkip, int *gkey_count)
{
        FILE    *fp;
        charbuff[50];
        int     no, ver, i;
        chargkey[32];

        if((fp = fopen(GKEY_FILE, "r")) == NULL){
                printf("Can't open %s.¥n", GKEY_FILE);
                exit(EXIT_FAILURE);
        }

        i=0;
        while(fgets(buff, 50, fp) != NULL){
                sscanf(buff, "%hd       %hd       %s¥n", &no, &ver, gkey);
                (gkip+i)->gkey_info.g_no        = no;
                (gkip+i)->gkey_info.key_ver  = ver;
                strcpy((gkip+i)->gkey, gkey);

                printf("g_no[%d]:%d¥n", i, (gkip+i)->gkey_info.g_no);
                printf("key_ver[%d]  :%d¥n", i, (gkip+i)->gkey_info.key_ver);
                printf("g_key[%d]    :%s¥n¥n", i, (gkip+i)->gkey);

                i++;
        }
        fclose(fp);

        *gkey_count = i;

        return 0;
}
```

B-2. acg.c

```
/*************************************************************************
                        acg.c  -  description
                           -------------------
    begin                : 2003 年 12 月 10 日 曜日 05 時 50 分 31 秒 JST
    copyright            : (C) 2004 by Hidekazu Suzuki
    email                : a300j075@ccmailg.meijo-u.ac.jp

*************************************************************************/

/*************************************************************************
 *                                                                       *
 *    This program is free software; you can redistribute it and/or modify   *
 *    it under the terms of the GNU General Public License as published by   *
 *    the Free Software Foundation; either version 2 of the License, or      *
 *    (at your option) any later version.                                    *
 *                                                                       *
*************************************************************************/

 /*
  * Include files
  */
#ifndef  __INCLUDES
#include "includes.h"
#define  __INCLUDES
#endif

#include "hmacmd5.h"

 /*
  * Define variables
  */


 /*
  * Define functions
  */
void make_gmac(u_char *gmac, u_char *key);
int authenticate_gmac(u_char *gmac, u_char *key);
void make_ip_header(struct ip *ip, int src_ip, int dst_ip, int proto, int iplen);
void make_icmp8_header(struct icmp *icmp, int len);
u_short checksum(u_short *data, int len);
static char *pt(u_char *, int);

/*
 *   int make_acg(u_char *buff, int dst_ip, GKEY_INFO_P *gkip, int gkey_count);
 *   機能
 *       ACG パケット生成 メインルーチン
 *   引き数
 *       u_char *buff;          生成される ACG パケットの格納変数
 *       int dst_ip;            パケットの宛先 IP アドレス
 *       GKEY_INFO_P *gkip;     グループ鍵情報
 *       int gkey_count;        グループ鍵の数
 *   戻り値
 *       int                    終了判定
 */
int make_acg(u_char *buff, int dst_ip, GKEY_INFO_P *gkip, int gkey_count)
{
        struct sockaddr_in      send_sa;
        struct ip               *ip;
        struct icmp             *icmp;
        ACG_H                   *acg_h;
        u_char                  gmac[GMAC_LENGTH];
        GMAP                    *gmap;
        ACG_INFO_H              *acg_info_h;
        ACG_INFO                *acg_info;
        GKEY_INFO               *gki;

        int                     i,j, s;
        int                     ip_len, icmp_len, acg_opt_len, acg_info_len;
        int                     plen, size;
        const int               on = 1;

        /* 値の初期化 */
        acg_opt_len = 0;

        /* 各ヘッダのポインタを設定

        IP(20)+ICMP(8)+ACG_H(20+a)+GMAP(40x)+ACG_INFO_H(4)+ACG_INFO(4+4
x) */
        /* IP ヘッダ */
        ip              = (struct ip    *)(buff);
```

```
    ip_len          = sizeof(struct ip);
    plen            = ip_len;
    printf("IP          :%d¥n", plen);
    /* ICMP ヘッダ */
    icmp            = (struct icmp     *)(buff+plen);
    icmp_len        = sizeof(struct udphdr);
    plen            += icmp_len;
    printf("~ICMP           :%d¥n", plen);
    /* ACG ヘッダ */
    acg_h           = (ACG_H        *)(buff+plen);
    size            = sizeof(ACG_H)+acg_opt_len;
    plen            += size;
    icmp_len        += size;
    printf("~ACG_H      :%d¥n", plen);
    /* GMAP */
    gmap            = (GMAP                 *)(buff+plen);
    size            = sizeof(GMAP) * gkey_count;
    plen            += size;
    icmp_len        += size;
    printf("~GMAP       :%d (count=%d)¥n", plen, gkey_count);
    /* ACG Information ヘッダ */
    acg_info_h   = (ACG_INFO_H     *)(buff+plen);
    size            = sizeof(ACG_INFO_H);
    plen            += size;
    icmp_len        += size;
    printf("~ACG_INFO_H   :%d¥n", plen);
    /* ACG Information */
    acg_info        = (ACG_INFO         *)(buff+plen);
    size            = sizeof(ACG_INFO);
    plen            += size;
    acg_info_len= size;
    printf("~ACG_INFO :%d¥n", plen);
    /* グループ鍵情報 */
    gki             = (GKEY_INFO *)(buff+plen);
    size            = sizeof(GKEY_INFO) * gkey_count;
    plen            += size;
    acg_info_len+= size;
    icmp_len        = icmp_len + acg_info_len;
    ip_len          = ip_len + icmp_len;
    printf("Total Size:%d¥n", plen);

    /* 各データグラム長の表示 */
    printf("¥n");
    printf("IP len          :%d¥n", ip_len);
    printf("ICMP len :%d¥n", icmp_len);
    printf("ACG Opt len  :%d¥n", acg_opt_len);
    printf("ACG Info len :%d¥n¥n", acg_info_len);

    /* IP ヘッダの生成 */
    make_ip_header(ip, 0, dst_ip, IPPROTO_ICMP, ip_len);

    /* ICMP ECHO ヘッダの生成 */
    make_icmp8_header(icmp, icmp_len);

    printf("GMAC List¥n");
    /* GMAP の生成ルーチン */
    for(i=0; i<gkey_count; i++) {
        /* GMAC の生成 */
        make_gmac(gmac, (gkip+i)->gkey);

        /* GMAP の生成 */
        (gmap+i)->gki.g_no      = htons((gkip+i)->gki.g_no);
        (gmap+i)->gki.key_ver   = htons((gkip+i)->gki.key_ver);
        (gmap+i)->info_flag     = 7;
        (gmap+i)->unused        = 7;
        strcpy((gmap+i)->gmac, gmac);

        /* ACG Information に載せるグループ鍵情報の設定 */
        (gki+i)->g_no           = (gkip+i)->gki.g_no;
        (gki+i)->key_ver        = (gkip+i)->gki.key_ver;
    }

    /* ACG Information の生成 */
    acg_info->ee_no             = 1;
    acg_info->info_count        = gkey_count;
    acg_info->info_flag         = 10;

    /* ACG ヘッダの生成 */
    strcpy(acg_h->dprp_id, "watanabe");
    acg_h->packet_type  = DPRP_ACG;
    acg_h->gmap_count  = gkey_count;
    acg_h->option_len    = acg_opt_len;

    /* ACG Information ヘッダの生成 */
    acg_info_h->ee_count            =1;
    acg_info_h->acg_info_len    = acg_info_len;

    return ip_len;
}

/*
 *  void make_gmac(u_char *gmac, u_char *key);
 *  機能
 *      GMAC 生成
 *  引き数
 *      u_char *gmac;   生成される GMAC の格納変数
 *      u_char *key;GMAC 生成に使用するグループ鍵
 *  戻り値
 *      なし
 */
void make_gmac(u_char *gmac, u_char *key)
{
    u_char  rn[RND_LENGTH];
    u_char  *hmac;
    int     result;

    printf(" GKey    : %2d byte (ASCII)    %s¥n", strlen(key), key);

    /* 128bit の乱数を生成 */
    make_rnd(rn);           /* return size is 17 Byte */
    rn[16] = NULL;          /* size of rn change 17 Byte to 16 Byte */
    printf(" rnd   : %2d byte (BINARY)   %s¥n", strlen(rn), pt(rn, 16));

    /* Generate HMAC-MD5 of 128bit Random Numbers */
    hmac = hmac_md5(key, rn);
    printf(" hmacmd5: %2d byte (BINARY)   %s¥n", strlen(hmac), pt(hmac, 16));

    /* Combine Random Number and HMAC-MD5 */
    strcpy(gmac, rn);
    strcat(gmac, hmac);
    printf(" GMAC   : %2d byte (BINARY)   %s¥n¥n", strlen(gmac), pt(gmac, 32));
}

/*
 *  int authenticate_gmac(u_char *gmac, u_char *key);
 *  機能
 *      GMAC 認証
 *  引き数
 *      u_char *gmac;   認証する GMAC の格納変数
 *      u_char *key;MAC 生成に使用するグループ鍵
 *  戻り値
 *      int  result;        認証結果
 */
int authenticate_gmac(u_char *gmac, u_char *key)
{
    u_char  rn[RND_LENGTH];
    u_char  *mac;
    u_char  *hmac;
    int     i;
    int     result;

    /* GMAC から乱数と MAC の分離 */
    strncpy(rn, gmac, RND_LENGTH);
    rn[16] = NULL;

    /* Generate HMAC-MD5 of 128bit Random Numbers */
    hmac = hmac_md5(key, rn);

    /* MAC の比較 */
    printf("GMAC                : %s¥n", pt(gmac, 32));
    printf("乱数 RN             : %s¥n", pt(rn, 16));
    printf("生成した MAC    : %s¥n¥n", pt(hmac, 16));
    result = 0;
    for (i=0; i<16; i++) {
        if (strcmp(&hmac[i], &gmac[16+i]) != 0) {
            result = 1;
        }
    }
    return result;
}

/*
 *  void make_ip_header(struct ip *ip, int src_ip, int dst_ip, int proto, int iplen);
 *  機能
 *      IP ヘッダ生成
 *  引き数
 *      struct ip *ip; 作成する IP ヘッダの先頭アドレス
 *      int src_ip;         送信元 IP アドレス
 *      int dst_ip;         宛先 IP アドレス
 *      int proto;          プロトコル
 *      int iplen;          IP データグラムの全長
 *  戻り値
 *      なし
 */
void make_ip_header(struct ip *ip, int src_ip, int dst_ip, int proto, int iplen)
{
    memset((char *)ip, 0, sizeof(struct ip));

    /* IP ヘッダの作成 */
    ip->ip_v    = IPVERSION;
    ip->ip_hl   = sizeof(struct ip)>>2;
    ip->ip_id   = htons(0);
    ip->ip_off  = 0;

#ifdef __linux
    /* Linux の Raw IP の場合 */
    ip->ip_len  = htons(iplen);
    ip->ip_off  = htons(IP_DF);
#else
    /* BSD の Raw IP の場合 */
    ip->ip_len  = iplen;
    ip->ip_off  = IP_DF;
#endif
    ip->ip_ttl  = 3;
    ip->ip_p    = proto;
    ip->ip_src.s_addr   = src_ip;
```

```
    ip->ip_dst.s_addr      = dst_ip;

    /* チェックサムの計算 */
    ip->ip_sum   = 0;
    ip->ip_sum   = checksum((u_short *)ip, sizeof(struct ip));
}

/*
 *  void make_icmp8_header(struct icmp *icmp, int len);
 *  機能
 *      ICMP ECHO パケット生成
 *  引き数
 *      struct icmp *icmp;    作成する ICMP ヘッダの先頭アドレス
 *      int len;              ICMP ECHO のパケット長
 *  戻り値
 *      なし
 */
void make_icmp8_header(struct icmp *icmp, int len)
{
    memset((char *)icmp, 0, len);

    /* Make ICMP header */
    icmp->icmp_type      = ICMP_ECHO;
    icmp->icmp_code      = 0;
    icmp->icmp_id        = 0;
    icmp->icmp_seq       = 0;

    /* Compute Checksum */
    icmp->icmp_cksum   = 0;
    icmp->icmp_cksum   = checksum((u_short *)icmp, len);
}

/*
 *  u_short checksum(u_short *data, int len);
 *  機能
 *      チェックサムの計算
 *  引き数
 *      u_short *data;    チェックサムを求めるデータ
 *      int len;          データのバイト数
 *  戻り値
 *      u_short           チェックサムの値(補数値)
 */
u_short checksum(u_short *data, int len)
{
    /* 求めるチェックサム */
    u_long   sum = 0;

    /* 2 バイトずつ加算 */
    for(; len > 1; len -=2) {
        sum += *data++;
        if(sum & 0x80000000)
            sum = (sum & 0xfff)+(sum >> 16);
    }

    /* データ長が奇数バイトの場合の処理 */
    if(len == 1) {
        u_short i = 0;
        *(u_char *)(&i) = *(u_char *)data;
        sum += i;
    }

    /* 桁あふれを折り返す */
    while(sum >> 16)
        sum = (sum & 0xfff)+(sum >> 16);

    return (sum == 0xffff)?sum:~sum;
}
```

B-3. hmacmd5.c

```
#include "hmacmd5.h"

u_char *hmac_md5(u_char *, u_char *);
void make_rnd(u_char *);

/* HMAC-MD5 Function */
u_char *hmac_md5(u_char *g_key, u_char *data)
{
    u_char   *digest;

    digest = HMAC(EVP_md5(), g_key, strlen(g_key), data, strlen(data), NULL, NULL);

    return digest;
}

/* Make 128bit Random Numbers */
void make_rnd(u_char *md)
{
    u_char   data[5];
    MD5_CTX  ctx;
    u_int    r;

    srand((u_int)time(NULL));
    r = rand() % 65536;

    sprintf(data, "%04X", r);

    MD5_Init(&ctx);
    MD5_Update(&ctx, data, strlen(data));
    MD5_Final(md, &ctx);
}
```

B-4. hmacmd5.h

```
#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <openssl/hmac.h>

#define GKEY_LENGTH     32   // Length of Group Key (256bit:32Byte)
#define RND_LENGTH   16   /* Length of Random Numbers (128bit:16Byte) */

/* Function of hmacmd5.c */

void      make_rnd(u_char *);
u_char   *hmac_md5(u_char *, u_char *);
```

B-5. includes.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <time.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <openssl/hmac.h>
#include <net/ethernet.h>

#define   __FAVOR_BSD
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>

#ifdef __linux
#include <linux/sockios.h>
#include <linux/if.h>
#else
#include <sys/ioctl.h>
#include <net/bpf.h>
#include <net/if.h>
#include <fcntl.h>
#endif

/* DPRP Input arguments : rcv_packet_type */
#define  DPRP_RCV_ACG    1    /* Flag : Receive ACG Packet */
#define  DPRP_RCV_MCI    2    /* Flag : Receive MCI Packet */
#define  DPRP_INIT       3    /* Flag : Receive Other Packets */

/* DPRP Input arguments : fwd_flag */
#define FLAG_FWD         0    /* Flag : forwarding */
#define FLAG_NFWD        -1   /* Flag : Not forwarding */

/* DPRP Return values */
#define END              0    /* Return value of System modules : End */
#define COMPLETE         1    /* Return value of System modules : Complete */
#define DISCARD          2    /* Return value of System modules : Discard */

#define TRUE             0    /* Return value of Sub modules : True */
#define FALSE            -1   /* Return value of Sub modules : False */
#define  SUCCESS         0    /* Return value of Main module : Success */
#define  FAILURE         -1   /* Return value of Main module : Failure */

/* Encryption Flag on Authenticate process */
#define  ENCRYPT         1    /* Encrypt */
#define  DECRYPT         0    /* Decrypt */

/* Definition Length */
#define GKEY_LENGTH      32   /* Length of Group Key (256bit:32Byte) */
#define RND_LENGTH       16   /* Length of Random Numbers (128bit:16Byte) */
#define HMAC_LENGTH      16   /* Length of HMAC-MD5 (128bit:16Byte) */
#define GMAC_LENGTH      32   /* Length of GMAC (256bit:32Byte) */
#define IV_LENGTH        16   /* Length of IV (128bit:16Byte) */
#define DPRP_ID_LENGTH   16   /* Length of DPRP Identification */

/* Definition Packet Types */
```

```
#define DPRP_ACG                1    /* Type : ACG (Normal) */
#define DPRP_ACG_OPTION         2    /* Type : ACG (add Option) */
#define DPRP_MCI                3    /* Type : MCI */

/* Define Structures */

/* Group Key Information */
typedef struct gkey_info {
      u_short  g_no;
      u_short  key_ver;
}GKEY_INFO;

/* Group Key Information Package */
typedef struct gkey_info_package {
      GKEY_INFO     gki;
      u_char        gkey[GKEY_LENGTH];
}GKEY_INFO_P;

/* GMAP : Group Message Authentication Package */
typedef struct gmap {
      GKEY_INFO     gki;
      u_short       info_flag;
      u_short       unused;
      u_char        gmac[GMAC_LENGTH];
}GMAP;

/* ACG header */
typedef struct acg_header {
      u_char        dprp_id[DPRP_ID_LENGTH];
      u_char        packet_type;
      u_char        gmap_count;
      u_short       option_len;
}ACG_H;

/* ACG Information header */
typedef struct acg_info_header {
      u_short       ee_count;
      u_short       acg_info_len;
}ACG_INFO_H;

/* ACG Information */
typedef struct acg_info {
      u_char        ee_no;
      u_char        info_count;
      u_short       info_flag;
}ACG_INFO;


static u_char iv[IV_LENGTH] = "123456789abcdef";

/* View HMAC-MD5 Value */
static char *pt(u_char *md, int len)
{
      int i;
      static char buf[80];

      for (i = 0; i < len; i++)
            sprintf(&(buf[i * 2]), "%02X", md[i]);
      return buf;
}

static double proctime(time_t start, time_t finish, int type)
{
      switch (type) {
            case 1:   /* sec */
                  return (double)((finish-start)/CLOCKS_PER_SEC);
                  break;
            case 2:   /* msec */
                  return (double)((finish-start)*1000/CLOCKS_PER_SEC);
                  break;
            case 3:   /* usec */
                  return (double)((finish-start)*1000000/CLOCKS_PER_SEC);
                  break;
      }
}
```

B-6. makefile

```
dprpc: dprpc.c acg.o hmacmd5.o includes.h
      gcc -o dprpc dprpc.c acg.o hmacmd5.o -lcrypto

hmacmd5.o: hmacmd5.c hmacmd5.h includes.h
      gcc -c hmacmd5.c

acg.o: acg.c acg.h includes.h
      gcc -c acg.c

clean:
      rm hmacmd5.o
      rm acg.o
      rm dprpc
```

B-7. 実行結果（性能）

```
vm-hydekazu# ./dprpc 172.18.16.41
  0 : View Help
  1 : View Group Key Information
  2 : Test GMAC create time
  3 : Test GMAC authenticate time
  4 : Send ACG Packet
  5 : Send MCI Packet
  9 : Quit
CMD > 1
Group Key Information
  Group No   Key Version   Group Key
         1           100   d0c6dd891fdf1695144e6e7eb8be00d9 (32 Byte)
         2           200   374b972a549e5afc1765dec0fe6b265a    (32 Byte)
         3           300   1630ca9d17348a2f619e99902210cb95 (32 Byte)
CMD > 2
Create GMAC...
GMAC 生成時間 : 218750.000000[usec] 試行回数:10000
      平均時間 : 21.875000[usec]
CMD > 3
Authenticate GMAC...

認証成功
GMAC 認証時間 : 101562.000000[usec] 試行回数:10000
      平均時間 : 10.156200[usec]
```

図 B-2　評価プログラム実行画面 1-1

```
CMD > 4
Create ACG Packet...
Send This Packet...
ACG  生成時間 : 687500.000000[usec] (GMAC count:3) 試行回数:10000
     平均時間 : 68.750000[usec]
CMD > 5
Create MCI Packet...
Send This Packet...
CMD > 9
DPRP Client Quit...
```

図 B-3　評価プログラム実行画面 1-2

B-8. 実行結果（パケット出力）

```
vm-hydekazu# ./dprpc 172.18.16.41
gkey count : 3
  0 : View Help
  1 : View Group Key Information
  2 : Test GMAC create time
  3 : Test GMAC authenticate time
  4 : Send ACG Packet
  5 : Send MCI Packet
  9 : Quit
CMD > 4
Create ACG Packet...
IP           :20
~ICMP        :28
~ACG_H       :48
~GMAP        :168 (count=3)
~ACG_INFO_H  :172
~ACG_INFO    :176
Total Size   :188

IP len       :188
ICMP len     :168
ACG Opt len  :0
ACG Info len :16

GMAC List
 GKey      : 33 byte (ASCII)    d0c6dd891fdf1695144e6e7eb8be00d9
 rnd       : 16 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F0
 hmacmd5   : 16 byte (BINARY)   0C5AD42CA466F5EE48450B1DADADD624
 GMAC      : 32 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F00C5AD42CA466F5EE48450B1DADADD624

 GKey      : 33 byte (ASCII)    374b972a549e5afc1765dec0fe6b265a
 rnd       : 16 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F0
 hmacmd5   : 16 byte (BINARY)   6E98C5D6D5F877FB551065197C2D04A7
 GMAC      : 32 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F06E98C5D6D5F877FB551065197C2D04A7

 GKey      : 32 byte (ASCII)    1630ca9d17348a2f619e99902210cb95
 rnd       : 16 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F0
 hmacmd5   : 16 byte (BINARY)   91D2E3A2C797B3395573983512092D5B
 GMAC      : 32 byte (BINARY)   FB5A1C9786BF0F68E3C691671D7535F091D2E3A2C797B3395573983512092D5B

Protocol: IP
+-------+-------+---------------+-------------------------------+
| IV: 4 | HL:  5| T:     00000000| Total Length:             48128|
+-------+-------+---------------+---------+---------------------+
| Identifier:                   0| FF:000  |FO:                   64|
+---------------+---------------+---------+---------------------+
| TTL:         3| Pro:          1| Header Checksum:          16258|
+---------------+---------------+-------------------------------+
| Source IP Address:                                    0.0.0.0|
+---------------------------------------------------------------+
| Destination IP Address:                           172.18.16.41|
+---------------------------------------------------------------+
ACG: ACG header
+---------------------------------------------------------------+
|DPRP ID:                         776174616E6162650000000000000000|
+---------------+---------------+-------------------------------+
|Type:         1| GMAP:         3| Option Len:                  0|
+---------------+---------------+-------------------------------+
```

図 B-4　評価プログラム実行画面 2-1

```
ACG: GMAP
+---------------+---------------+---------------------------------+
|Group No:                    1| Key Ver:                      100|
+---------------+---------------+---------------------------------+
|Info Flag:                1792| Unused:                      1792|
+---------------+---------------+---------------------------------+
|GMAC:                                                            |
|FB5A1C9786BF0F68E3C691671D7535F00C5AD42CA466F5EE48450B1DADADD624 |
+---------------+---------------+---------------------------------+
ACG: GMAP
+---------------+---------------+---------------------------------+
|Group No:                    2| Key Ver:                      200|
+---------------+---------------+---------------------------------+
|Info Flag:                1792| Unused:                      1792|
+---------------+---------------+---------------------------------+
|GMAC:                                                            |
|FB5A1C9786BF0F68E3C691671D7535F06E98C5D6D5F877FB551065197C2D04A7 |
+---------------+---------------+---------------------------------+
ACG: GMAP
+---------------+---------------+---------------------------------+
|Group No:                    3| Key Ver:                      300|
+---------------+---------------+---------------------------------+
|Info Flag:                1792| Unused:                      1792|
+---------------+---------------+---------------------------------+
|GMAC:                                                            |
|FB5A1C9786BF0F68E3C691671D7535F091D2E3A2C797B3395573983512092D5B |
+---------------+---------------+---------------------------------+
ACG: ACG Information header
+-------------------------------+---------------------------------+
|EE Count:                  256| ACG Info Len:                4096|
+-------------------------------+---------------------------------+
ACG: ACG Information
+---------------+---------------+---------------------------------+
|EE No:        1| Count:        3| Info Flag:                   2560|
+---------------+---------------+---------------------------------+
|Group No:                  256| Key Ver:                    25600|
+-------------------------------+---------------------------------+
|Group No:                  512| Key Ver:                    51200|
+-------------------------------+---------------------------------+
|Group No:                  768| Key Ver:                    11265|
+-------------------------------+---------------------------------+

Frame Dump:
4500 bc00 0000 0040 0301 3f82 0000 0000 : E......@..?.....
ac12 1029 0800 f7ff 0000 0000 7761 7461 : ...).........wata
6e61 6265 0000 0000 0000 0000 0103 0000 : nabe............
0001 0064 0700 0700 fb5a 1c97 86bf 0f68 : ...d.....Z.....h
e3c6 9167 1d75 35f0 0c5a d42c a466 f5ee : ...g.u5..Z.,.f..
4845 0b1d adad d624 0002 00c8 0700 0700 : HE.....$........
fb5a 1c97 86bf 0f68 e3c6 9167 1d75 35f0 : .Z.....h...g.u5.
6e98 c5d6 d5f8 77fb 5510 6519 7c2d 04a7 : n.....w.U.e.|-..
0003 012c 0700 0700 fb5a 1c97 86bf 0f68 : ...,.....Z.....h
e3c6 9167 1d75 35f0 91d2 e3a2 c797 b339 : ...g.u5........9
5573 9835 1209 2d5b 0100 1000 0103 0a00 : Us.5..-[........
0100 6400 0200 c800 0300 2c01           : ..d........,.

Send This Packet...
```

図 B-5　評価プログラム実行画面 2-2