September 20, 2002

Linux Conference 2002

FLIM/SEMI における電子メールセキュリティ

上野 乃毅 <ueno@unixuser.org>

岡田 健一 <okada@opaopa.org>

小林 修平 <shuhei@aqua.ocn.ne.jp>

発表の流れ

- FLIM/SEMI とは?
- 発表の意義
- あらまし
- プロトコルと実装の詳細
- 将来の展望
- まとめ

FLIM/SEMI

- Emacs でインターネットメッセージを扱うための汎用のライブラリ
- Wanderlust, Semi-gnus などの MUA で利用され、広く普及
- 階層化された設計

  FLIM ··· ユーザインターフェースに関わらない低位の機能

  SEMI ··· メッセージの再生や作成のためのユーザインターフェース
- CLOS 風のオブジェクトシステム luna によるオブジェクト指向設計
- 早くから CVS による分散開発体制を取り入れていたため、多数の派生版が存在することで有名
  - FLIM の派生版 ··· SLIM(岡田), LIMIT, CLIME
  - SEMI の派生版 ··· WEMI(山岡), EMIKO(上野), EMY(林), NISEMI(岡田), MEIMI(有沢)

発表の意義

最近のセキュリティプロトコルの紹介

- セキュリティプロトコルの仕組みを理解していないために、メーリングリストなどで同じ質問が繰り返される
- 開発者向けの文書はあるが、利用者向けの文書が不足しているとよく文句を言われる

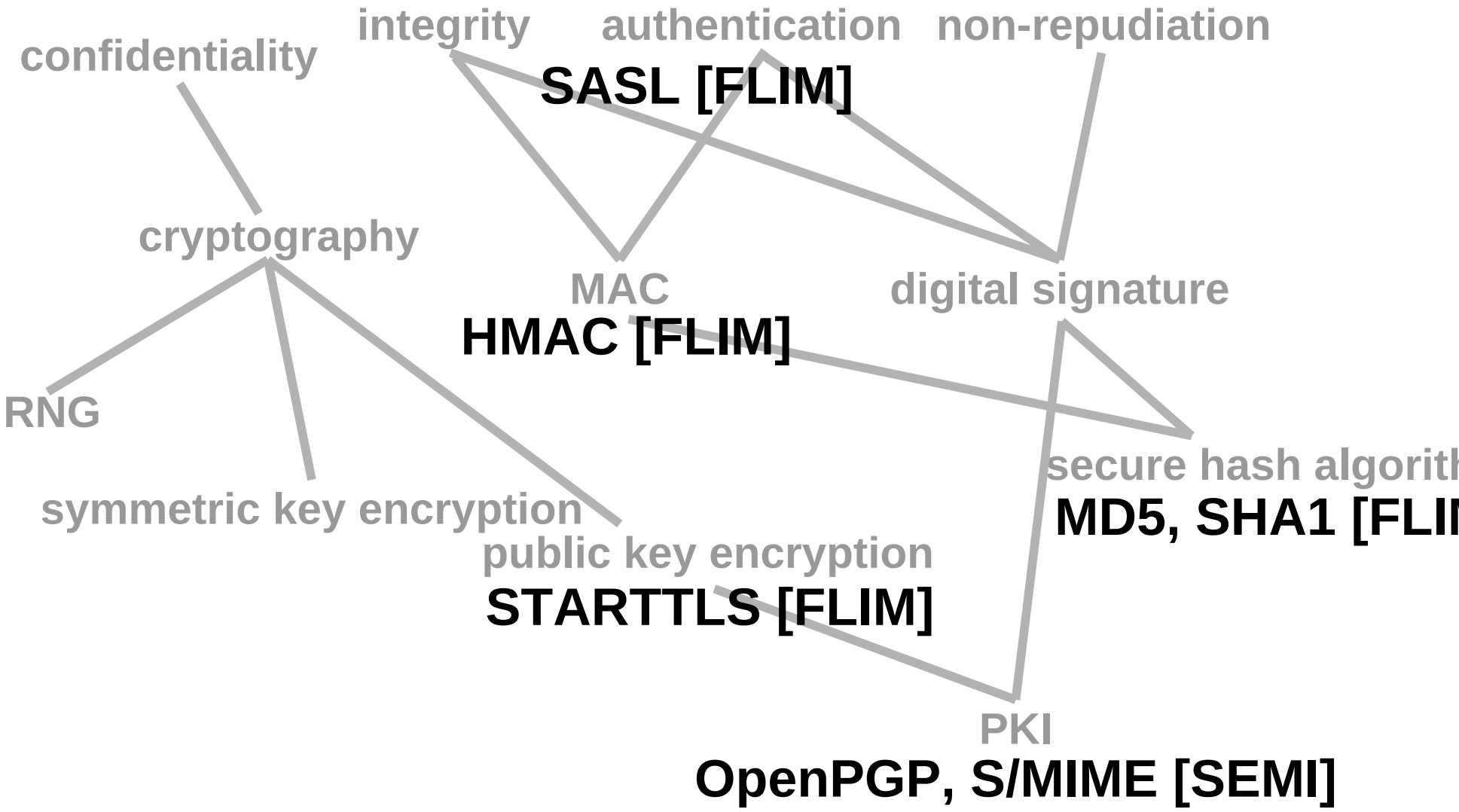
各種プログラミング言語での実装のヒント

- 最近のセキュリティプロトコルは、特定の暗号アルゴリズムに依存しないように設計されているため、実装も同様に柔軟であるべきである
  ⇝ **ライブラリの実装にコツが必要**
- Java や Perl, Python, Emacs Lisp 以外のプログラミング言語には、ライブラリが揃っていない
  ⇝ **誰か Ruby/SASL や O'Caml/SASL を書いてください!**

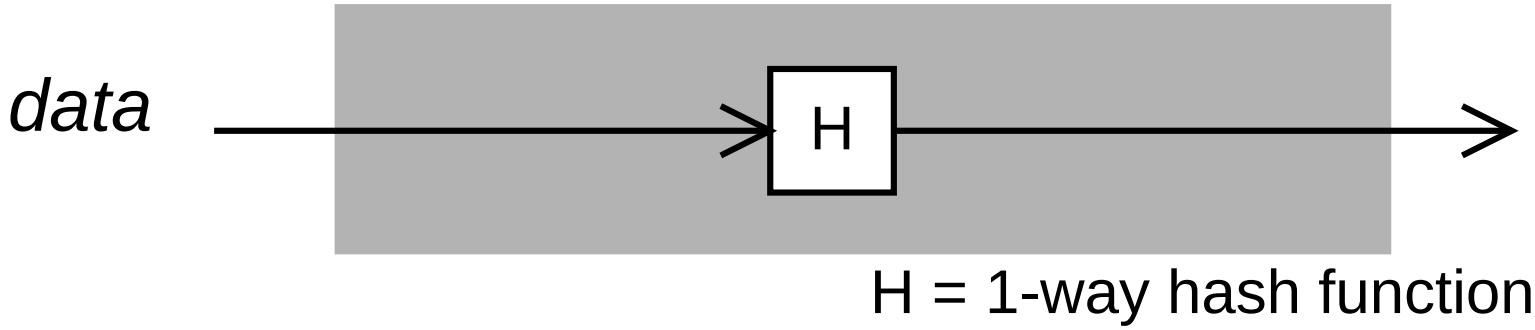
あらまし

1995 年後半	tm (SEMI の前身) の PGP2 対応 (Mailcrypt を利用)
1996 年?	Mailcrypt が PGP5/GnuPG に対応
1998 年?	新しい Mailcrypt を使った SEMI の PGP5/GnuPG 対応
1999 年前半	FLIM のための CLOS 風オブジェクトシステム luna の登場
1999 年後半	PGP/MIME と親和性の高い PGP ライブラリ PGG Mailcrypt のかわりに PGG を使った SEMI の PGP 対応
2000 年前半	FLIM の HMAC, SASL 対応
2000 年後半	STARTTLS のための補助プログラム luna による FLIM の SMTP クライアントの再編成 FLIM の SASL 対応部分のフレームワーク化
2000 年後半	IPA 未踏ソフトウェア創造事業の助成を受けて開発を継続
2001 年後半	Emacs 21.1 のリリース
2005 年?	GNU Emacs へのセキュリティ関連モジュールの統合完了?

セキュリティの諸分野との対応

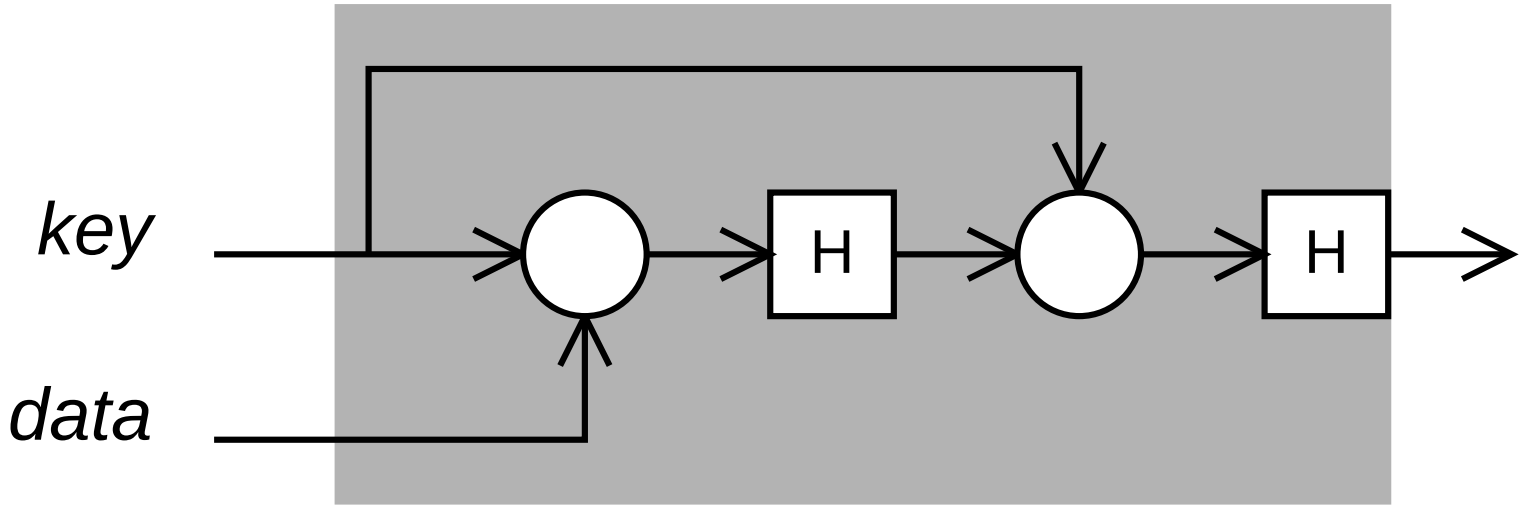


メッセージダイジェスト



- データ と データのハッシュ値 を一緒に送ることで、データが改竄されていないことを保証する仕組み
- データを入手できれば誰でもハッシュ値を計算できる
- 改竄後データと、それに対する正しいハッシュ値を送られた場合には、データの改竄を検出できない
- 主なアルゴリズム: SHA-1, SHA-256, SHA-384, SHA-512 (FIPS 180-2), MD5 (RFC 1321), RIPEMD

メッセージ認証コード (MAC) アルゴリズム



- データ と、データと秘密鍵から計算した署名 を一緒に送ることで、データが改竄されていないことを保証する仕組み
- メッセージダイジェストの応用
- 署名の計算に送信者と受信者で共通の秘密鍵の情報を利用するため、秘密鍵を知らないかぎり、正しい署名を生成できない

RFC 2104: HMAC: Keyed-Hashing for Message Authentication

- MAC アルゴリズムの一種
- 内部のハッシュ関数を取り換え可能
- シンプルなアルゴリズム

  ⇝ **ハッシュ関数のパフォーマンスを大きく損わない**

  $$H(K \oplus \mathrm{opad}, H(K \oplus \mathrm{ipad}, \mathtt{text}))$$

  K: 秘密鍵, H: ハッシュ関数
- さまざまな用途への応用
  - DIGEST-MD5 などの SASL 認証メカニズム
  - DNS のトランザクション署名 (RFC 2845)
  - SPAM 防止のためのアドレスタグ (TMDA, rubyfilter)
  - NNTP における記事のキャンセルロック (draft-ietf-usefor-cancel-lock-01.txt)

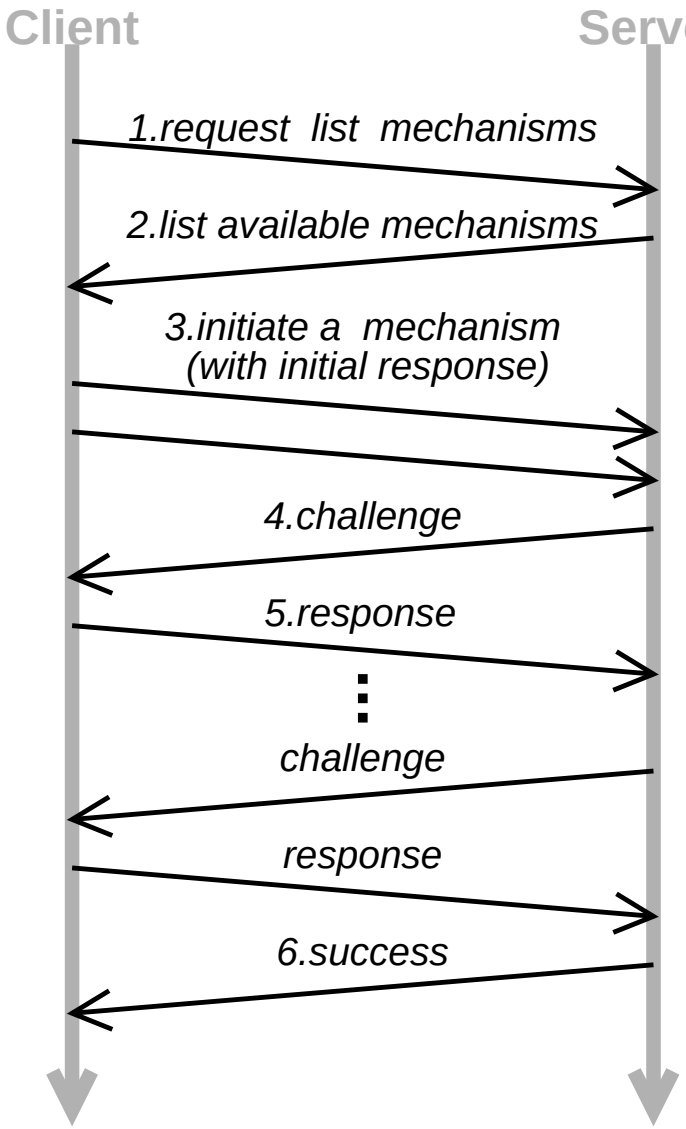
RFC 2222: Simple Authentication and Security Layer (SASL)

- アプリケーションプロトコルに認証・一貫性保護・機密保護機能を追加するためのプロトコル
- 認証アルゴリズムを取り換え可能

規格を構成する 3 つの要素

1. **SASL の仕様そのもの (“SASL specification itself”)**
   承認を獲得するための複数回のプロトコル応答を定義
2. **SASL 認証メカニズム (“SASL mechanisms”)**
   内部の認証アルゴリズムの詳細をカプセル化したもの
3. **プロファイル (“Protocol profiles”)**
   各種アプリケーションプロトコルへの組み込み手段の定義: IMAP (RFC 2060), SMTP (RFC 2554)

SASL のプロトコル応答



1. 2. 利用可能な認証メカニズムの一覧の取
3. 認証メカニズムを選択して認証を開始
4. 5. チャレンジ・レスポンスの繰り返し
6. 承認の獲得

- 認証の各段階はプロファイルに従って、通
に符号化 (e.g. IMAP や SMTP では
から 6. までの各段階を Base64 で符号
- クライアントの機密情報が要求される段階
認証メカニズムの定義によって異なる

SASL クライアント・フレームワークの設計

フレームワークの存在意義

- プログラマに認証アルゴリズムの詳細な知識を要求しない
  - ⇝ **SASL 認証メカニズムを取り換え可能なバックエンドとして実装**
- 認証メカニズムによって機密情報が必要となる段階が異なる
  - ⇝ **応答の途中に利用者に機密情報の入力を促すための仕組みが提供**

フレームワークを構成する 3 種類のデータ型

`sasl-mechanism`	認証メカニズムの抽象
`sasl-client`	機密情報以外の静的に与えられるクライアントの情報を格納
`sasl-step`	次の応答段階の番地と直前の段階の返り値を保持

Emacs Lisp には継続を実現する手段が用意されていないため、簡単なコルーチンの仕組みを実装

SASL クライアント・フレームワークの API

1. 認証メカニズムの選択

```
(let ((sasl-mechanisms '("CRAM-MD5" "DIGEST-MD5")))
  (sasl-find-mechanism server-supported-mechanisms))
```

⇒ **sasl-mechanism オブジェクト**

2. クライアント情報の設定

```
(sasl-make-client mechanism "ueno" "smtp" "localhost"))
```

⇒ **sasl-client オブジェクト**

3. 初期レスポンスの計算

```
(sasl-next-step client nil)
```

⇒ **sasl-step オブジェクト**

4. サーバからのチャレンジの取得 & 次の認証段階へ進む

```
(setq challenge (smtp-read-response connection))
(sasl-step-set-data step (base64-decode-string challenge))
(setq step (sasl-next-step client step))
(smtp-send-command connection (base64-encode-string (sasl-step-data step) t))
```

SASL クライアント・フレームワークの実装

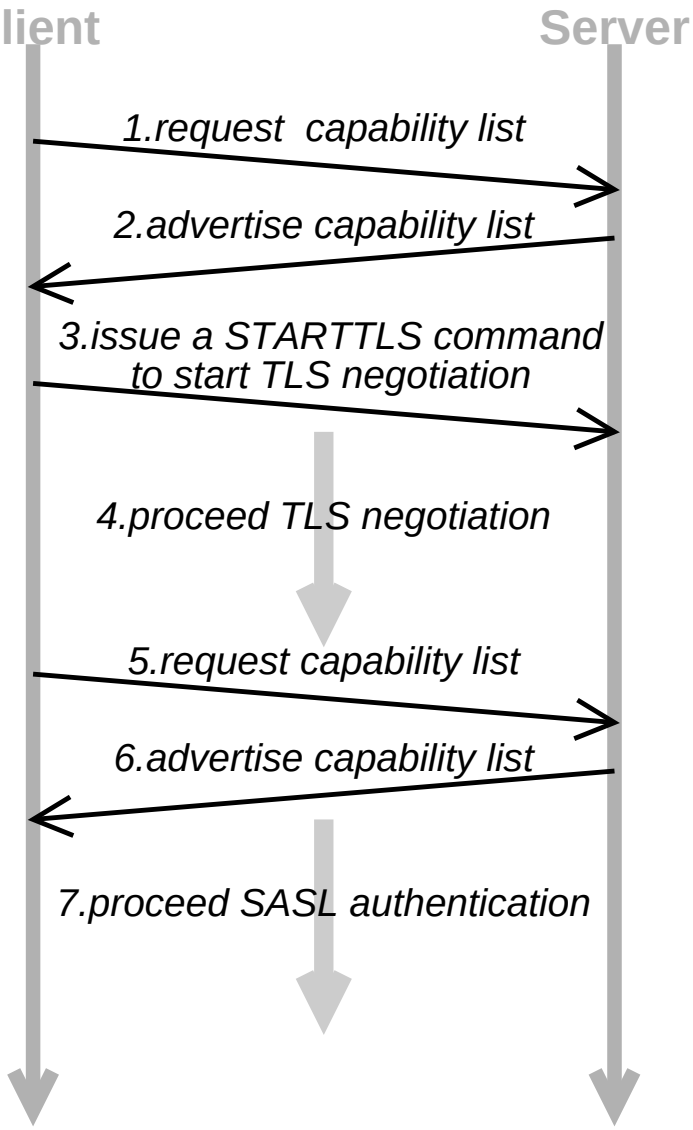
対応している SASL 認証メカニズム

- ANONYMOUS (RFC 2245)
- PLAIN (RFC 2595)
- CRAM-MD5 (RFC 2195)
- DIGEST-MD5 (RFC 2831)
- SCRAM-MD5 (draft-newman-auth-scram-03.txt)
- NTLM (NT LanManager)
- LOGIN

STARTTLS

- 任意のアプリケーションプロトコルに TLS による通信路の暗号化機能を追加するプロトコル
- 単なる SSL によるトンネリングとは異なり、専用のポート番号や接続時の SSL ネゴシエーションは必須ではない
  ⇝ **既に確立した通信路に対して利用者が暗号化を選択できる**
- SASL のような包括的な標準は存在せず、SMTP (RFC 2487), IMAP, POP3, ACAP (RFC 2595) のプロファイルが存在するのみ
- SASL と併用する運用形態が一般的
  ⇝ **機密保護機能に STARTTLS，認証機能に SASL**

STARTTLS と SASL の併用



1. 2. サーバの特性一覧の取得
3. TLS ネゴシエーションの開始要求
4. TLS ネゴシエーション
5. 6. サーバの特性一覧の取得
7. SASL による認証に進む

2. で平文でネットワークを流れたサーバの特性
覧が、man-in-the-middle アタックにより書換
られている可能性があるため、安全な通信路を
立した後に 6. で特性一覧を再度取得する

OpenPGP と S/MIME

電子メールの暗号化や署名といった end-to-end のプライバシー保護

OpenPGP

- 相互運用可能なメッセージ形式が RFC 2440 で標準化
- MIME の枠組に沿って電子メールで利用するためには、RFC 3156 に準拠したセキュリティ・マルチパート形式 (RFC 1847) に従う

S/MIME

- 署名・暗号文のメッセージ形式には PKCS#7 を採用
- MIME の枠組に沿って電子メールで利用するためには、RFC 2633 に準拠した複数の形式に従う
  ⇝ **非 MIME 環境との相互運用性までも考慮した設計であることから MIME を前提とした既存の MUA への組込みは難しい**

PGP ライブラリ (PGG) の設計

既存の PGP ライブラリの問題点

Mailcrypt ··· PGP/MIME で前提とされる分割署名を扱えない

`gpg.el` (`gnus/contrib`) ··· GnuPG しか扱えない

PGG の特長

- バックエンド機構
  - 複数の PGP 実装 (GnuPG, PGP5/PGP2) を選択可能
  - バックエンド API により新たな PGP 実装への対応が容易
- 構成の階層化
  - OpenPGP メッセージ形式の解釈
  - PGP/MIME で利用するうえで必要十分な PGP 機能の抽象
  - PGP/MIME メッセージを再生するユーザインタフェース

S/MIME 機能の実装

SEMI 1.14.3 に含まれているが、現在では obsolete

- 安直に PGG のバックエンドとして実装
- OpenSSL に smime コマンドが同梱される以前の実装
- 現在では OpenSSL を利用した `smime.el` が存在

安全なプログラムを書くための留意点: 一時ファイルのパーミッションの保護

ファイルを介してデータを受渡す外部プログラムや、Emacs 内部で一時ファイルを使用する関数 (e.g. `call-process-region`) を利用すると、機密情報が漏洩する恐れがある

`unwind-protect` と `set-default-file-modes` を組み合わせて、一時ファイルのパーミッションを保護する

```
(let ((orig-mode (default-file-modes)))
  (unwind-protect
      (progn
        (set-default-file-modes 448)     ;#o0700
        ...)
    (set-default-file-modes orig-mode)))
```

安全なプログラムを書くための留意点: Lisp 文字列の汚染

core ファイルや kmem を解析することで、プロセス内で使用した機密情報を第三者に盗まれる恐れがある

- 機密情報を含む文字列を無意味な文字で埋める

```
(unwind-protect
    (progn
      ... operate on credential ...)
  (fillarray credential 0))
```

- concat または substring により文字列が複製されることに注意する
- バッファに機密情報を含む文字列を挿入しない
- ミニバッファの操作履歴に注意する

Emacsen への統合

- Simon Josefesson さんの協力を得て、セキュリティ関連モジュールの一部は Gnus の一部として Emacs 21 に統合済み
- XEmacs のパッケージ
  “sasl” (HMAC, SASL クライアント), “ecrypto” (各種ハッシュアルゴリズム)

LEMI プロジェクト

- <http://mousai.as.wakwak.ne.jp/projects/lemi/>
  - GNU Emacs 21 への FLIM/SEMI の統合
  - APEL の分離
  - 単体での RMAIL と mh-e の MIME 化

今後の展望

ライブラリ構成の再編

- FLIM/SEMI の機能を利用しない MUA での利用を可能にする
  - OpenPGP ライブラリの SEMI からの分離

SASL クライアント・フレームワークの改善

- 一貫性・機密保護機能への対応
  - luna による通信路の抽象化
- 未実装の認証メカニズムへの対応
  GSSAPI, EXTERNAL (RFC 2595)

PGP ライブラリの改善

- GnuPG にターゲットを絞り、全ての機能を活用する
  鍵の管理・選択用インターフェースの提供，慣用暗号の扱い