Linux Conference 2002

FLIM/SEMI における電子メールセキュリティ

上野乃毅　岡田健一　小林修平

Implementing Email Security for FLIM/SEMI

Daiki Ueno, Kenichi Okada, and Shuhei Kobayashi

概 要

電子メールプロトコルにおける認証や通信路の暗号化、および電子メール本文の暗号化と署名のためのライブラリを開発した。Emacs 上で動作するこれらの実装は、FLIM/SEMI と共に配布され広く利用されている。

本論文では、開発に携わった経験から、セキュリティ機能のライブラリとしての実装の実際と、安全なアプリケーションの構築に向けた指針を解説する。

1　背景と意義

主要なサーバ実装の採用により、電子メールプロトコルにおける認証や通信路の暗号化といった基本的なセキュリティ機能が身近なものとなりつつある。

これらの基礎となる枠組として、SMTP のみならず様々なインターネットプロトコルに容易に組み込めるように設計された標準規格が提案されている。

我々は 1999 年から様々な標準規格に準拠したセキュリティ機能のライブラリ化に取り組んでいる。実装は Emacs 上で動作し、FLIM/SEMI[1, 2] の拡張として提供される。FLIM/SEMI は守岡知彦氏らを中心に開発が進められてきた Emacs でインターネットメッセージを取り扱うための汎用のライブラリである。FLIM はユーザインターフェースに関わらない低位の機能を提供し、SEMI は MIME 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を提供する。現時点で最新のリリース版である FLIM 1.14.4 および SEMI 1.14.4 には、開発の成果として HMAC[3], SASL[4], OpenPGP[5, 6],S/MIME[7, 8, 9, 10] に対応した実装が含まれている。また、STARTTLS[11] への対応は個別のパッケージとして GNU Emacs 21.2 に取り込まれている。

FLIM/SEMI のセキュリティ関連モジュールは、基本機能の実装完了後、特に大きな改編もなく約二年間継続して保守がなされてきた。その間に Wanderlust[12] をはじめとする FLIM/SEMI 対応 MUA の機能の一部として実用に耐えてきたことから、実装は比較的枯れているといえる。

今日成果を論文としてまとめる意義は二点ある。一つは、煩雑で泥臭い部分が多いとみなされてきたセキュリティの分野において、簡潔で実用に即した設計を採用した最新のセキュリティプロトコルを紹介すること、もう一つは、実装の内部構造を詳細に解説することで、Emacs Lisp 以外のプログラミング言語によるライブラリの実装を奨励することである。

我々のライブラリで網羅する標準規格と、セキュリティ諸分野との対応を、図 1に示す。

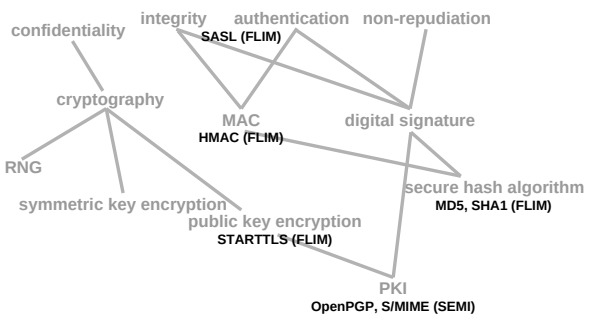

図 1: FLIM/SEMI で網羅する標準規格

以降では、HMAC, SASL, STARTTLS, OpenPGP, S/MIME の各標準規格の概要を実装を踏まえて解説する。続いて、Emacs Lisp で安全なプログラムを書くための留意点を挙げる。次に GNU Emacs への統合作業の現状や関連プロジェクトとの協力体制を示し、最後に次期実装への展望を述べる。

2 HMAC

HMAC は RFC2104 で標準化されたメッセージ認証コード (Message Authentication Code, 以降 MAC と表記) アルゴリズムの一つである。MAC アルゴリズムとは、鍵に依存する一方向ハッシュ関数で、通信者同士が秘密鍵を共有しメッセージ送信者とデータの一貫性を同時に検証するために用いることができる。通常のメッセージダイジェストのアルゴリズムと HMAC アルゴリズムによる署名の流れを図 2に示した。上が通常のメッセージダイジェスト、下が HMAC による署名の流れである。

通常のメッセージダイジェストのアルゴリズムでは、改竄後のデータに対してメッセージダイジェストを生成することにより、受信者に知られることなくデータを改竄することができる。これに対し、MAC アルゴリズムでは署名の生成に秘密鍵の情報を利用する。正しい署名を生成するためにはあらかじめ同一の秘密鍵を共有していなければならないため、第三者によるデータの改竄を防ぐことができる。

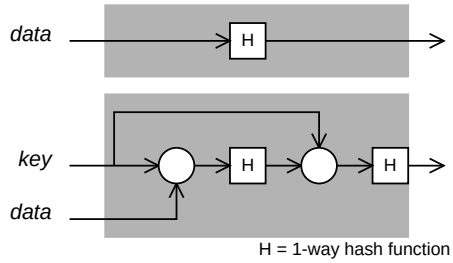

図 2: メッセージダイジェストのアルゴリズムと HMAC

HMAC は内部のハッシュ関数を切り替え可能にする柔軟な設計を採用した標準規格であり、後述する SASL の認証メカニズムの一部として、また NNTP における記事のキャンセルロックや、DNS のトランザクション署名 [13]、主に SPAM の防止に用いられるタグ付き配送エージェント [14] のアドレスタグの生成など幅広い用途に応用されている。FLIM における HMAC の実装は主に小林が担当した。初期の HMAC の実装には、内部のハッシュ関数に依存しない枠組の他に、標準的なハッシュ関数である MD5 および SHA-1 の実装が含まれた。ハッシュ関数の実行速度と可搬性を両立させるために、Emacs のダイナミックローディング拡張 [15] を利用した実装も用意され、実装を選択する仕組みが開発された。GNU Emacs 21.1 以降は組込みの関数として MD5 の実装が含まれている。

3 SASL

SASL は任意のアプリケーションプロトコルに認証・一貫性検査・機密性保持機能を追加するためのプロトコルである。その規格は大きく分けて三つの部分からなる。一つは承認を獲得するための複数回のプロトコル応答定義、二つ目は各種アプリケーションプロトコルへの組み込み手段を定義したプロファイル、最後に内部の認証アルゴリズムの詳細をカプセル化する SASL 認証メカニズムの定義である。これにより暗号法に関する詳細な知識を必要とすることなく、セキュリティ機能をアプリケーションプロトコルに容易に組込むことができる。

SASL に対応したクライアントがサーバとの間で承認を獲得し安全な通信路を確立する過程を図 3に示す。

クライアントは最初に利用可能な認証メカニズムの一覧を取得する。次に適当な認証メカニズムを選択し、SASL のネゴシエーションを開始する。承認を得るまで複数回の応答を繰り返し、最終的に承認を獲得するという流れになる。一連のネゴシエーションは、アプリケーションプロトコルに組み込まれた SASL プロファイルに従い、適切に符号化される。

ここで注意すべきは、承認を得るまでの複数回の応答のどの段階で機密情報を与える必要があるか否かは利用する認証メカニズムの定義に依存するということである。例えば、ANONYMOUS 認証メカニズムでは一切の機密情報を必要とせず、PLAIN では応答の最初に平文で機密情報を送り、CRAM-MD5 や DIGEST-MD5 ではサーバのチャレンジを解釈した後で機密情報から計算したレスポンスを送信するというように、機密情報が異なる段階で扱われることを考慮する必要がある。

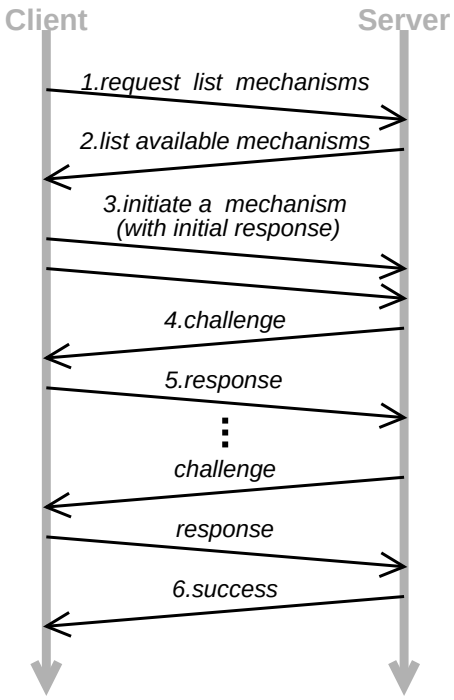

図 3: 承認を獲得するためのプロトコル応答

ネゴシエーションを開始する前に承認の獲得に必要なクライアントの機密情報を全て揃えておく方針は、柔軟性がないばかりか特定の段階で応答が中断された場合に脆弱性が増すことになる。このようなことから、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必要となる。しかしながら、Emacs Lisp には継続を実現する手段が用意されていないため、簡単なコルーチンを実現することで解決を図った。SASL クライアント・フレームワークの基礎となる三種類のデータ型を表 1に簡単にまとめる。表中の sasl-step が、認証の過程で呼出される関数の戻り番地と返り値をカプセル化したオブジェクトである。

`sasl-mechanism`	認証メカニズムの定義
`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
`sasl-step`	次の応答段階と直前の段階の返り値をカプセル化したオブジェクト

表 1: SASL クライアント・フレームワークの基本データ型

本フレームワークの典型的な利用例を図 4に示した。この例は FLIM の smtp.el から一部抜粋、改変したものであり、SMTP の SASL プロファイルに基づき、AUTH コマンドを実装している。“sasl-” から始まる名前の関数呼出しが実際に本フレームワークを利用している箇所である。以下では、コード中のコメントの番号に沿って、承認を獲得するまでの流れを示す。

1. 認証メカニズムの選択
2. クライアント情報の設定
3. クライアントが送信する最初のレスポンスを取得する。これは認証メカニズムの定義に依存する
4. サーバへの SASL 認証の開始通知
5. 承認を得るためのチャレンジ・レスポンスのループの開始
   (a) 承認の獲得
   (b) 次の段階に進む

フレームワークとしてまとめ上げたことにより、コードの追加量は最低限に抑えることができた。このため、FLIM 附属の SMTP 実装のみならず、非同期に通信を行う Mew[16] の SMTP 実装にも容易に適用することができた。

初期の FLIM の SASL 実装は岡田が担当した。この段階では仕様の明らかにされている各種認証メカニズムのアルゴリズムの詳細が実装された。その後、フレームワークにまとめる作業を主に上野が担当した。

FLIM 1.14.4 で対応している認証メカニズムは以下の 7 つである。

- ANONYMOUS[17]
- PLAIN[11]
- CRAM-MD5[18]
- DIGEST-MD5[19]
- SCRAM-MD5[20]
- NTLM (NT LanManager)
- LOGIN

SASL には認証の機能だけではなく、安全な通信路を確立する機能も含まれている。SASL における通信路の安全性のレベルはセキュリティ強度要素 (Security

```
;; 1. Find a preferred SASL mechanism.
(let ((mechanism (sasl-find-mechanism mechanisms))
      client name step response)
  ;; 2. Set up a SASL client.
  (setq client (sasl-make-client
                mechanism smtp-sasl-user-name
                "smtp" smtp-server)
        ;; 3. Retrieve an initial response.
        step (sasl-next-step client nil))
  ;; 4. Instigate server to start authentication.
  (smtp-send-command
   connection
   (format "AUTH %s" name))
  ;; 5. Challenge and response loop.
  (catch 'done
    (while t
      (setq response (smtp-read-response connection))
      (when (= (car response) 235)
        ;; 5(a). Authentication process finished.
        (setq step (sasl-next-step client step))
        (if (null step)
            (throw 'done nil)))
      (if (/= (car response) 334)
          (smtp-response-error response))
      ;; 5(b). Step forward the next authentication process.
      (sasl-step-set-data
       step
       (base64-decode-string (nth 1 response)))
      (setq step (sasl-next-step client step))
      (smtp-send-command
       connection
       (if (sasl-step-data step)
           (base64-encode-string (sasl-step-data step) t)
         ""))))))
```

図 4: SMTP 実装への SASL の組み込み例

Strength Factor, 以降 SSF と表記) と呼ばれる数値により分類することができる。対応可能な SSF 値は認証メカニズムの定義に依存するが、現在の FLIM の実装では認証のみ (SSF = 0) に対応している。

0	認証のみ
1	認証 + 一貫性検査
2	認証 + 一貫性検査 + 機密性保持

表 2: SASL のセキュリティ強度要素

4 STARTTLS

STARTTLS は SASL と同様に、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である。単なる SSL によるトンネル接続とは異なり、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のが STARTTLS プロトコルの大きな特色である。

SASL と同様に、STARTTLS にも各種アプリケーションプロトコルへの組み込み手段を定義したプロファイルが存在する [21, 11]。運用に際しては STARTTLS による通信路の暗号化と SASL の認証を併用するのが一般的である。図 5に、STARTTLS と SASL を併用して承認を得るまでの過程を示す。

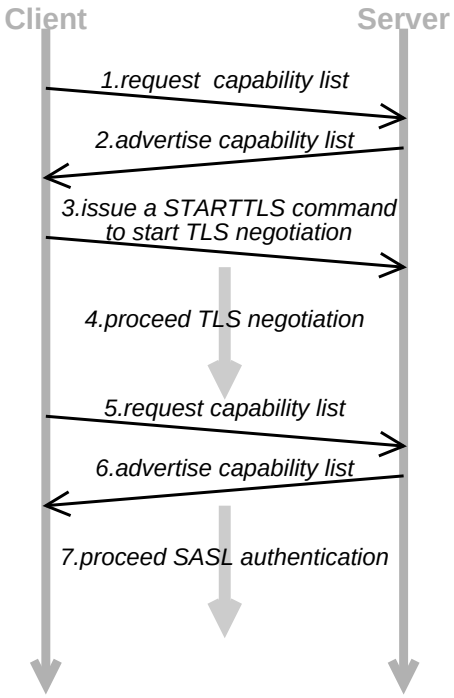

図 5: STARTTLS と SASL の併用

クライアントは最初にサーバの特性一覧を問い合わせ、サーバが STARTTLS に対応していることを確認する。次に STARTTLS プロファイルに従って TLS のネゴシエーションを開始する。TLS による安全な通信路が確立した時点で、再度サーバに特性一覧を問い合わせる。これは、最初に平文でネットワークを流れたサーバの特性一覧が、man-in-the-middle アタックにより書換えられている可能性があるためである。

FLIM に附属する SMTP クライアントには、外部プログラム starttls を利用した STARTTLS プロトコルへの対応が含まれている。Simon Josefsson は SecurEmacs プロジェクト [22] で、GNU Emacs への GNUTLS[23] のバインディングを提案しており、我々の STARTTLS 実装と互換性のある API を提供している。

5 OpenPGP

OpenPGP は電子メールの暗号化や署名といったユーザレベルのプライバシー保護を目的とした相互運用可能なプロトコルである。PGP メッセージ交換フォーマット (RFC1991) の流れを組む交換可能なメッセージ形式が RFC2440 で標準化されている。

MIME の枠組に沿って、OpenPGP メッセージを電子メールで利用するためには、RFC3156 に準拠したセキュリティ・マルチパート形式 [24](以降 OpenPGP/MIME と表記) に従うのが一般的である。Emacs から外部の PGP コマンドを実行する方式で PGP の機能を利用する実装としては既に Mailcrypt や gpg.el が存在するが、前者は OpenPGP/MIME で前提とされる分割署名を扱えないことや、後者のライセンスが GPL と矛盾することから、SEMI では自前のライブラリを揃える方針をとった。

SEMI での実装は大きく分けて三階層に分けられる。一つは OpenPGP のメッセージ形式を解釈する低位のライブラリ、二つ目は GNU プライバシーガード等の外部の PGP コマンドを統一的な手段で呼び出すライブラリ、最後に OpenPGP/MIME メッセージを再生するユーザインターフェースの実装である。

利用者の環境に複数の PGP コマンドがインストールされている場合の振舞いに関しては、解析した OpenPGP メッセージから、それを扱うことができる PGP コマンドを実行時に選択する方式を採用している。この実装には APEL に含まれる状態木と状態/状況連想リストを操作するライブラリである calist.el の機能を用いている。これにより、各 PGP コマンドの機能性を宣言的に記述することが可能となった。

暗号化・復号化・署名の検証などの PGP 機能の抽象化には FLIM のオブジェクトシステムである luna を用いている。PGP の個々の機能は luna の提供する CLOS 風な文法を用いて pgg-scheme 型のオブジェクトを第一引数に取る総称関数としてまとめられている。新たな PGP 実装に対応するためには、pgg-scheme を継承したクラスを定義すれば良い。具体的な手順を以下にまとめる。

1. 新たな PGP 実装の機能性を記述する。具体的には状態木 pgg-verify-condition/pgg-decrypt-condition を設定する
2. pgg-scheme を継承したクラスを定義し、新たな PGP 実装への呼出しをカプセル化する
3. pgg-scheme の登録手続きに従う
   (a) シングルトン性を保証するための関数 pgg-make-scheme-* を書く
   (b) フィーチャ pgg-* を宣言する

6 S/MIME

S/MIME は OpenPGP と同様に、電子メールの暗号化や署名といったユーザレベルのプライバシー保護を目的としたプロトコルである。マイクロソフトの Outlook Express のような一般的なメール・ユーザ・エージェントに広く採用されている。S/MIME のメッセージ形式は RFC2633 で標準化されているが、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 をベースとした既存の MUA への組込みは非常に難しい。SEMI には上野による S/MIME の実装が含まれているが、Gnus[25] に附属する実装との統合を現在検討中である。

7 安全な Emacs Lisp プログラムを書くための留意点

安全なアプリケーションの構築には様々な泥臭い問題がつきものである。開発を通じて、Emacs で機密情報を扱う上で注意しなければならない様々な問題に直面した。以降ではその中で主なものを紹介する。

一時ファイルのパーミッションの保護

外部プログラムの中にはファイルを介してデータを受渡すものが少なくない。call-process-region のように一見安全に見えても Emacs の内部で一時ファイルを生成する関数も存在する。

このような関数や外部プログラムを利用する場合に機密情報の漏洩を防ぐには unwind-protect と set-default-file-modes を組み合わせた以下のイデオムが一般的に有効である。

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

Lisp 文字列の汚染

core ファイルを解析することで、プロセス内で使用した機密情報を第三者に盗まれる可能性が考えられる。一般に実行時のプロセスイメージは OS の保護下にあるが、席を外している隙に端末を横取りされた場合にはなす術がない。

これを防ぐため、機密情報に対する操作を終えた後は即座に無意味な文字列で埋める。unwind-protect を利用した以下のイデオムが一般的に有効である。

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

8 これまでの展開と効果

STARTTLS 対応などの一部のセキュリティ機能の実装については、Gnus の一部という形で Free Software Foundation に寄贈された。

Simon Josefsson は FLIM から SASL ライブラリを分離し、XEmacs のパッケージ化および保守管理を行っている。また HMAC の低位ライブラリである SHA-1 および MD5 アルゴリズムの実装は Ray Jones の Ecrypto[26] と併せて XEmacs のパッケージリポジトリに含まれている。

9 今後の展望

FLIM/SEMI を取り巻く活発なプロジェクトとの連携を考えると、ライブラリ構成に大きな改編の必要が予想される。特に FLIM/SEMI の機能を利用しない MUA での需要に対応するために、低位 OpenPGP ライブラリの SEMI からの分離は比較的早い段階でなされるべきであると考えている。現在、Simon Josefsson と上野が共同で作業中である。

SASL 対応部分の本質的な改良の余地としては、1 以上の SSF 値への対応が挙げられる。このために次期実装では luna による通信路の抽象化を行う予定である。GSSAPI[27] や EXTERNAL[11] といった未実装の認証メカニズムへの対応も考えている。

OpenPGP 対応部分の改良の余地としては、OpenPGP/MIME の表現力を最大限に活用することが挙げられる。また、慣用暗号や圧縮形式を直接扱うことが可能な OpenPGP ユーザインターフェースの実現が望まれていることから、特に GNU プライバシーガードとの親和性の改善が必須であると考えている。この目的のために、GPGME[28] を Emacs のダイナミックローディング拡張を介して利用する実装も検討中である。GPGME は GNU プライバシーガードの機能を外部のアプリケーションから利用するためのライブラリである。

以上のような課題を踏まえ、近い将来の GNU Emacs への FLIM/SEMI 機能の統合に向けて積極的に改良を続けていくつもりである。

10 まとめ

広く普及した Emacs 用 MIME ライブラリである FLIM/SEMI に、最新の標準に準拠したセキュリティ機能を実装した。ほとんどの機能は MUA から透過的に利用することが可能であり、フレームワークとしての設計を採用することで、様々な Emacs プログラムから容易に利用可能となった。

本論文の成果は平成 12 年度未踏ソフトウェア創造事業において、上林 PM 採択のプロジェクト「GNU Emacsen における複数の MUA で共有可能な Internet Message のための汎用部品の開発」の一環として情報処理振興事業協会の助成を受けた。

GNU Emacs 本体への FLIM/SEMI 機能の統合作業は現在も継続中であり、LEMI プロジェクト [29] において、単独で GNU Emacs 21 上の RMAIL と mh-e に MIME 関連機能を提供するパッケージを公開している。

謝辞

FLIM/SEMI の原作者である守岡知彦氏をはじめ、emacs-mime-ja,en(旧 tm-ja,en) メーリングリストの方々には、動作試験を含め貴重な助言を頂いたことをここに感謝する。

参考文献

[1] FLIM. http://www.kanji.zinbun.kyoto-u.ac.jp/ tomo/elisp/FLIM/.

[2] SEMI. http://www.kanji.zinbun.kyoto-u.ac.jp/ tomo/elisp/SEMI/.

[3] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104: HMAC: Keyed-Hashing for Message Authentication, Feb. 1997.

[4] J. Myers. RFC 2222: Simple Authentication and Security Layer (SASL), Oct. 1997.

[5] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. RFC2440: OpenPGP Message Format, Nov. 1998.

[6] M. Elkins, D. Del Torto, R. Levien, and T. Roessler. RFC3156: MIME Security with OpenPGP, Aug. 2001.

[7] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. RFC2311: S/MIME Version 2 Message Specification, Mar. 1998.

[8] S. Dusse, P. Hoffman, B. Ramsdell, and J. Weinstein. RFC2312: S/MIME Version 2 Certificate Handling, Mar. 1998.

[9] B. Ramsdell. RFC2633: S/MIME Version 3 Message Specification, Jun. 1999.

[10] B. Ramsdell. RFC2632: S/MIME Version 3 Certificate Handling, Jun. 1999.

[11] C. Newman. RFC 2595: Using TLS with IMAP, POP3 and ACAP, Jun. 1999.

[12] Yuuichi Teranishi. Wanderlust – Yet Another Message Interface on Emacsen –. http://www.gohome.org/wl/.

[13] P. Vixie, O. Gudmundsson, D. Eastlake, and B. Wellington. RFC 2845: Secret Key Transaction Authentication for DNS (TSIG), May. 2000.

[14] J. Mastaler. Tagged Message Delivery Agent (TMDA). http://tmda.net/.

[15] Mule with Dynamic Loading Facility. http://www.m17n.org/mule/dynamic-loading/.

[16] Mew Official Homepage (In Japanese). http://www.mew.org.

[17] C. Newman. RFC 2245: Anonymous SASL Mechanism, Nov. 1997.

[18] J. Klensin, R. Catoe, and P. Krumviede. RFC 2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response, Sep. 1997.

[19] P. Leach and C. Newman. RFC2831: Using Digest Authentication as a SASL Mechanism, May. 2000.

[20] C. Newman. draft-newman-auth-scram-03.txt: Salted Challenge Response Authentication Mechanism, Mar. 1998.

[21] P. Hoffman. RFC2487: SMTP Service Extension for Secure SMTP over TLS, Jan. 1999.

[22] Simon Josefsson. SecurEmacs - Security functionality for emacs. http://josefsson.org/securemacs/.

[23] The GNU Transport Layer Security Library. http://www.gnu.org/software/gnutls/.

[24] J. Galvin, S. Murphy, S. Crocker, and N. Freed. RFC 1847: Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted, Oct. 1995.

[25] Lars Magne Ingebrigtsen. Gnus Newsreader Homepage. http://www.gnus.org.

[26] Ray Jones. Ecrypto. http://web.mit.edu/thouis/ecrypto/.

[27] J. Myers. draft-ietf-cat-sasl-gssapi-05.txt: SASL GSSAPI mechanisms, May. 2001.

[28] GPGME - GnuPG Made Easy. http://www.gnupg.org/gpgme.html.

[29] Lemi projects. http://mousai.as.wakwak.ne.jp/projects/lemi/.