Integration von GnuPG in MUAs

Werner Koch*

Version 0.7
14. Februar 2001

1 Übersicht

Da GnuPG im wesentlichen in Verbindung mit Email benutzt wrd, ist eine gute Integration mit MUAs (Mail User Agents) essentiell. Hier kann auf die Erfahrung mit der Integration von PGP 2 zurückgegriffen werden; insbesondere wurde das Interface von GnuPG in einer Art und Weise definiert, die eine Integration in andere Programme relativ leicht möglich macht. Besondere Kommandozeilenoptionen unterstützen den Programmierer hier.
Wir geben hier eine schnelle und einfache Anleitung, um es dem Entwickler einer MUA einfach zu machen, GnuPG Unterstützung einzubauen. Wir werden kurz au die Geschichte derartiger Implementierungen eingehen, Vor- und Nachteile einiger Techniken diskutieren und dann eine sinnvolle Arbeitsweise diskutieren.

2 Protokolle

Das von GnuPG unterstützte Protokoll ist OpenPGP, welches auf den Formaten von PGP 2 aufbaut. Es ist im RFC2440[3] definiert. Allerdings sagt dieses Protokoll nichts zur Verwendung in Email aus und man hat hierzu auf weitere Protokolle zurückzugreifen.
Traditionell wurde das in OpenPGP definierte ASCII Armor Encoding direkt auch von Email Programmen benutzt, wobei hier ein MIME-Typ `application/pgp-encrypted` benutzt wird. Dieses sehr simple Protokoll hat jedoch einige entscheidende Nachteile, insbesondrs ist es nicht moglich eine Mail aus mehreren Teilen (Attachments) zusammenzusetzen. Aus diesem Grund und wegen einiger weiterer Probleme, sollte es möglichst nicht benutzt werden. Wenn möglich sollte ein MUA dieses ASCII Armor Format allerdings verstehen; dieses Format zu erzeugen is zu vermieden.
Basierend auf dem MIME Standard[5] wurde das MOSS Protokoll[6] entworfen, welches die Basis für den RFC2015[2] (MIME Security wth Pretty Good Pri-

*wk@gnupg.org

vacy) darstellt. Dieses Protokoll sollte unbedingt zur Anwendung kommen. Im Weiteren wird es auch als PGP/MIME bezeichnet.
PGP/MIME definiert 2 Protokolltypen: PGP Encrypted Data und PGP Signed Data. Beide können auch kombiniert werden. Hier eine kurze Darstellung der Funktionsweise — Details sind dem RFC2015[2] zu entnehmen.
PGP Encrypted Data ist eine MIME Multipart Message mit 2 Teilen in einem *Content-Type* von `multipart/encrypted` mit dem Parameter `protocol="application/pgp-encrypted"`. Der erste Teil hat nun den *Content-Type* `application/pgp-encrypted` und besteht lediglich aus einer Zeile mit dem String `Version: 1`. Der zweite Teil hat den *Content-Type* `application/octet-stream` und besteht aus der OpenPGP ASCII Armored Nachricht. Der Plaintext dieser Nachricht ist nun ein normaler MIME Body und kann durchaus auch wieder eine Multipart Message sein.
PGP Signed Data ist etwas komplizierter, da wir hier sicherstellen müssen, daß die Nachricht auf dem Transport nicht verändert wird. Details hierzu sind wiederum im RFC2015[2] zu finden. Es handelt sich auch hier um eine MIME Multipart Message mit 2 Teilen. Die gesamte Nachricht hat den *Content-Type* von `multipart/signed` mit dem Parameter `protocol="application/pgp-signature"`. Der erste Teil stellt die eigentliche Nachricht dar und kann eine beliebiges Mime Objekt sein. Dieses gesamte Objekt wird signiert. Oft handelt es sich bei diesem Objekt dann auch um ein Multipart Object welches die eigentliche Nachricht mit ihren Attachments darstellt. Der zweite Teile hat den *Content-Type* `application/pgp-signature` und beinhalted eine "Detached Signature" im Textmodus als OpenPGP ASCII Armor. Als Besonderheit muß der Haupt MIME Content-Type noch den Parameter `micalg=foo` enthalten, wobei `foo` hier durch den verwendeten Hash Algorithmus zu ersetzen ist. Unter OpenPGP macht dieser Parameter allerdings wenig Sinn (er stammt aus dem MOSS Framework) und sollte von Anwedungen zwar korrekt gesetzt werden, ansonsten ist er aber zu ignorieren.

3 Historische Verfahren zur GnuPG Integration

Ursprünglich wurde GnuPG, genau wie PGP in MUAs integriert, wobei nur wenig Kommunikation zwischen den Programmen bestand. Aufgrund unterschiedlicher Kommandzeileninterfaces musste GnuPG getrennt von PGP implementiert werden. Der MUA *Mutt* geht seiniger Zeit eine anderen Weg: Dort können die Aufrufe des Backends (PGP oder GnuPG) vom Anwender konfigiertert werden und intern werden sie identische behandelt. Um eine Liste der Public Keys zu erhalten, verwendet *Mutt* intern das Format von GnupG und liefert einen Wrapper für PGP mit, der dessen Ausgaben in das GnuPG Format wandelt.
Das Problem mit all diesen Anwendung ist die eingschränkte Kommunikation, die man erreichen kann. Um eine benutzerfreundlichere Kommuniation mit PGP/GnuPG aufzubauen (z.B. Eingabe der Passphrase) ist ein recht hoher Aufwand notwendig, der in jedem MUA immer wieder neu implementiert werden muss.

4 GPGME als vereinheitlichte Schnittstelle

Um die Integration einfacher zu machen, wurde die Bibliothek *GPGME* entwickelt, die eine Abstraktion der GnuPG Schnittstelle implementiert und eine High-Level Schnittstelle für MUAs und andwere Anwendungen anbietet. Diese Bibliotehk sollte unbedingt verwendet werden. Sie unterliegt der GPL[4], so daß Programme die sie nutzen GPL kompatiblen Lizenzen zu unterliegen haben[1].

5 Vorraussetzungen für einen MUA

Um einen MUA mit GnuPG zu verheiraten, muss dieser unbedingt MIME unterstützen. Es gibt verschiede Techniken, MIME zu implementieren und dadurch ändert sich dann auch die Art und Weise, wie GnuPG/GPGME zu benutzen ist.
Die einfachste (aber auch Resourcen intensivste) Integration ist gegeben, wenn auf die MIME Objekte wahlfrei zugegriffen werden kann und es möglich ist ein MIME Objekt durch ein anderes zu ersetzen. Fast alle heutigen MUAs arebite nach diesem Prinzip. Wir beschreiben im Folgenden ein derartiges System. wobei die
Es wird dringend empfohlen, signieren und verschlüsseln auf MIME Ebene zu trennen und nicht die optionale kombinierte Methode anzuwenden (GPGME unterstützt diese momentan auch noch nicht). Der Vorteil ist die übesichtlichere Implementation und die Möglichkeit eine verschlüsselte Nachricht entschlüsselt zu archivieren, die Signatur dabei aber intakt zu lassen.

6 Entschlüsselung

Die Entschlüsselung ist durchzufühen, sobald ein OpenPGP verschlüsseltes Objekt zur Verarbeitung ansteht. Es ist also zu prüfen, ob der *Content-Type* `multipart/encrypted` vorliegt und ein Parameter `protocol="application/pgp-encrypted"` vorhanden ist. Es sollte hierfür ein entsrpechender Handler implementiert werden.
Es sollte nun geprüft werden, ob der erste Teil den richtigen Content-Type besitzt und eine Zeile mit der richtigen Versionsnummer vorhanden ist. Danach wird der zweite Teil via *GPGME* entschlüsselt und das daraus entstehende MIME Objekt ersetzt dann das ursprüngliche Objekt.
Hier ein Beispiel ohne jegliche Fehlerbehandlung:

```
GpgmeCtx ctx;
GpgmeData ciphertext, plaintext;

gpgme_new (&ctx);
gpgme_data_new_from_mem (&ciphertext, mime_part_2,
                         mime_part_2_len, TRUE );
gpgme_data_new (&plaintext);

gpgme_op_decrypt (ctx, ciphertext, plaintext);
```

[1] falls ich dies in Einzelfällen als Problem herausstellt, sollte der Autor von GPGME kontaktiert werden

```
gpgme_data_release(ciphertext);

gpgme_data_rewind (plaintext);
while ( !gpgme_data_read (dh, buf, sizeof buf, &nread ) ) {
     fwrite (buf, nread, 1, stdout );
}

gpgme_release (ctx);
gpgme_data_release(plaintext);
```

`ciphertext` und `plaintext` enhalten jeweils die vollständigen MIME Objekte. Beispiele hierzu finden sich im RFC2015[2].

7 Verschlüsselung

Eine Verschlüsselung durchzuführen ist prinzipiell recht einfach, wird allerdings dadurch erschwert, daß hier ein UI benötigt wird, welches es erlaubt die Keys der Empfänger auszuwählen. Dies ist ein recht komplexes Thema und wir können hier nicht darauf eingehen.
Als Eingabedaten wird eine Liste der Empfanger benötigt, sowie ein MIME Objekt. Das einfachste MIME Objekt hat den *Content-Type* `text/plain`, es ist aber auch jegliches andere MIME Objekt möglich.
Hier der Beispielcode ohne jegliche Fehlerbehandlung, der eine vollständige RFC822[1] Mail ausgibt.

```
GpgmeCtx ctx;
GpgmeData ciphertext, plaintext;
GpgmeRecipients rset;

gpgme_new (&ctx);
gpgme_set_armor (ctx, 1);

gpgme_data_new_from_mem (&plaintext, mime_object,
                         mime_object_len, TRUE );
err = gpgme_data_new ( &ciphertext );

gpgme_recipients_new (&rset);
gpgme_recipients_add_name (rset, "Bob");,

gpgme_op_encrypt (ctx, rset, plaintext, ciphertext );
gpgme_data_release (plaintext);
gpgme_recipients_release (rset);

print_mail_headers_without_content_lines ();
fputs ( "Content-Type: multipart/encrypted;\r\n"
        "              protocol=\"application/pgp-encrypted\";\r\n"
        "              boundary=\"42=.42=.42=.42\"\r\n"
        "\r\n--42=.42=.42=.42\r\n"
        "Content-Type: application/pgp-encrypted\r\n\r\n"
        "Version: 1\r\n
```

```
        "\r\n--42=.42=.42=.42\r\n"
        "Content-Type: application/octet-stream\r\n\r\n", stdout );

gpgme_data_rewind (ciphertext);
while ( !gpgme_data_read (ciphertext, buf, sizeof buf, &nread ) ) {
     fwrite (buf, nread, 1, stdout );
}
fputs ( "\r\n--42=.42=.42=.42--\r\n", stdout );

gpgme_release (ctx);
gpgme_data_release(plaintext);
```

`print_mail_headers_without_content_lines` sollte alle notwendigen Mail Header mit der Ausnahme der `Content-*:` Zeilen ausgeben. Die Content Zeilen dürfen nicht mit ausgegeben werden, da diese von obigem Coder erzeugt werden. Normalerweise ist es möglich eine existierende Mail zu verschlüsseln, indem die alten Header gespeichert werden, die Content Header benutzt werden um das MIME Objekt zu bilden und dann die obigen Operationen durchgeführt werden.

8 Prüfung einer Signatur

Die Signatur kann geprüft werden, wenn ein OpenPGP signiertes Objekt vorhanden ist. Ein solches Objekt identifiziert sich durch einen *Content-Type* `multipart/signed` und dem dazugehörgen Parameter `protocol="application/pgp-signed"`. Der Parameter `micalg` kann bei der Benuztung von GPGME ignoriert werden. Es sollte hierfür ein entsprechender MIME Handler implementiert werden.
Ein derartiges Objekt besteht aus 2 MIME Teilen: Dem signierten Teil, der ein beliebiges MIME Objekt sein kann, und dem Signatur Teil der den *Content-Type* `application/pgp-signature` hat. Beide Teile werden extrahiert und GPGME zur Prüfung zugeführt. Die MIME Header des ersten Teils sind mit signiert und müssen deswegen auch an GPGME weitergegeben werden. Beim 2. Teil kann dies gemacht werden, da GPGME diese Header ignoriert.
Hier ein Beispiel ohne jegliche Fehlerbehandlung:

```
GpgmeCtx ctx;
GpgmeSigStat Status;
GpgmeData datapart, sigpart;
const char *nota_xml;

gpgme_new (&ctx);
gpgme_data_new_from_mem (&datapart, mime_part_1,
                         mime_part_1_len, TRUE );
gpgme_data_new_from_mem (&sigpart, mime_part_2,
                         mime_part_2_len, TRUE );

gpgme_op_verify (ctx, sigpart, datapart, &status);
gpgme_data_release(datapart);
gpgme_data_release(sigpart);
```

```
show_status (status);
show_extended_status (ctx)
nota_xml = gpgme_get_notation (ctx);
if (nota_xml)
    show_notation_data (nota_xml);

gpgme_release (ctx);
```

Beispiele zu den MIME Objekten finden sich im RFC2015[2].
`show_status` kann lediglich anzeigen, ob die Signatur gültig ist oder nicht und das auch nur wenn es sich um eine einzelne Signatur handelt. Sind mehrere Signaturen mit einem unterschiedlichen Status vorhanden, so wird lediglich ein besonderes Statuswert zurückgegeben. Bessere Informationen wird die oben angedeutete Funktion `show_extended_status` liefern; diese könnte so aussehen:

```
int idx;
GpgmeSigStat status;
time_t created;

for (idx=0; gpgme_get_sig_status(ctx, idx, &status, &created); idx++ ) {
    const char *userid = \"[?]\";
    GpgmeKey key = NULL;

    if ( !gpgme_get_sig_key (ctx, idx, &key) )
        userid = gpgme_key_get_string_attr (key, GPGME_ATTR_USERID,
                                            NULL, 0);

    printf ("%s from \"%s\"\n", status_to_string(status), userid );
    gpgme_key_unref (key);
}
```

Falls Notation Daten vorhanden sind, so wereden sie als ein XML String für alle Signaturen geliefert. Weitere Informationen über den Key können entweder mittels der obigen Funktion oder aber durch `gpgme_key_get_as_xml` ermittelt werden.

9 Signieren

Die Erzeugung einer Signatur hat grosse Ähnlichkeit mit dem Verschlüsseln. Ein beliebiges MIME Objekt kann signiert werden.
Die Vorgehensweise wir Anhand eines Beispiels recht klar; in diesem Beispiel wird ein vorliegendes MIME Objekt in ein signiertes Objekt umgewandelt und auf stdout ausgegeben. Fehlerbehandlung und die Wahl einer sinnvollen Boundary fehlen hier.

```
GpgmeCtx ctx;
GpgmeData data, sig;

gpgme_new (&ctx);
gpgme_set_armor (ctx, 1);
```

```
gpgme_set_textmode (ctx, 1);

gpgme_data_new_from_mem (&data, mime_object,
                         mime_object_len, TRUE );
gpgme_data_new ( &sig );
gpgme_op_sign (ctx, data, sig, GPGME_SIG_MODE_DETACH );

fputs ( "Content-Type: multipart/signed;\r\n"
        "              protocol=\"application/pgp-signature\";\r\n"
        "              boundary=\"42=.42=.42=.42\"\r\n"
        "\r\n--42=.42=.42=.42\r\n", stdout );

gpgme_data_rewind (data);
while ( !gpgme_data_read (data, buf, sizeof buf, &nread ) ) {
     fwrite (buf, nread, 1, stdout );
}
fputs ( "\r\n--42=.42=.42=.42--\r\n"
        "Content-Type: application/pgp-signature\r\n\r\n", stdout);

gpgme_data_rewind (sig);
while ( !gpgme_data_read (sig, buf, sizeof buf, &nread ) ) {
     fwrite (buf, nread, 1, stdout );
}
fputs ( "\r\n--42=.42=.42=.42--\r\n", stdout );

gpgme_release (ctx);
gpgme_data_release(data);
gpgme_data_release(sig);
```

RFC2015[2] und dessen bald erscheinende Überarbeitung definieren noch einige sinnvolle Transformationen des MIME Objekts; z.B. sollten “trailing white spaces” entfernt werden und die Zeilen mit kanonischen Zeilentrennern (CR,LF) versehen sein.

10 Ausblick

Auf die Behandlung von “Passphrases” ist hier bewusst nicht eingegangen worden, da angestrebt wird, dieses von Anwendungen fernzuhalten und durch eine eigene Software, die direkt mit GnuPG zusammenarbeitet, zu lösen.
Die hier beschriebene Schnittstelle ist so allgemein gehalten, das sie in Zukunft auch um andere Verschlüsselungsstandards erweitert werden kann.
Neue Versionen dieses Dokuments werden von Zeit zu Zeit erscheinen.

Literatur

[1] D. Crocker. Standard for the format of arpa internet text messages, August 1982.

[2] M. Elkins. RFC 2015: MIME security with pretty good privacy (PGP), October 1996.

[3] J. Callas et al. RFC 2440: OpenPGP message format, November 1998.

[4] Free Software Foundation. GNU General Public License, version 2, June 1991.

[5] N. Freed and N. Borenstein. Multipurpose internet mail extensions (mime) part one: Format of internet message bodies., November 1996.

[6] J. Galvin, S. Murphy, S. Crocker, and N. Freed. RFC 1847: Security multiparts for mime: Multipart/signed and multipart/encrypted, October 1995.