IT- und Netzwerksicherheit
WiSe 2016/17
Praktisches Übungsblatt Nr. 3

Nils Aschenbruck

Jan Bauer, Alexander Bothe, Florian Krampe, Thomas Hänel,

Timmy Schüller, Bertram Schütz, Matthias Schwamborn, Stefanie Thieme

Constantin Schraeder

Veröffentlichung	28.11.2016
Abgabe	06.12.2016

Allgemeine Informationen zu den praktischen Aufgaben:

- Lesen Sie sich das Aufgabenblatt **aufmerksam von vorne bis hinten** durch, bevor Sie mit der Bearbeitung beginnen!
- Die erfolgreiche Bearbeitung von 4 der 6 praktischen Aufgaben ist für die Zulassung zur Prüfung erforderlich.
- Alle praktischen Aufgaben müssen **alleine** bearbeitet werden.
- **Code-Plagiate werden nicht toleriert!** Gleiches gilt, falls Code von anderen Quellen ohne Referenz oder ungewöhnlich viel Code (auch mit Referenz) übernommen wird. Dies betrifft alle beteiligten Teilnehmer.
- Die Abgabe der Aufgabe muss am jeweiligen Abgabetermin **bis spätestens 23:59 Uhr** erfolgt sein. Im allgemeinen Fall muss dies via **StudIP** geschehen. Zugehörige Dateien sind in ein ZIP Archiv zu packen und im Aufgabenbereich der Veranstaltung hochzuladen. Der Name der Datei muss nach folgendem Schema gewählt sein:

  ITS_WiSe_201617_PA3_*MatrNr*

  wobei ***MatrNr*** durch die jeweilige Matrikelnummer ersetzt werden muss. Beim Entpacken des Archivs muss automatisch ein **Unterordner nach dem gleichen Namensschema** erzeugt werden. Markieren Sie nach dem Upload die jeweilige Aufgabe als fertig. Beachten Sie mögliche Änderungen der Abgabemodalitäten bei einzelnen Aufgaben.
- **Befolgen Sie alle aufgeführten Konventionen** für Dateinamen, Ordnerstruktur, Programmparameter, etc. Ansonsten verlängern Sie unnötig die Korrekturdauer und verzögern somit die Ergebnisverkündung.
- Bei Fragen von allgemeinem Interesse nutzen Sie bitte die **Mailingliste** zur Vorlesung. Alternativ können Sie auch in die wöchentliche **Tutoren-Fragestunde** kommen, wobei vorab eine kurze Problembeschreibung per E-Mail an den Tutor (**cschraed@uos.de**) erfolgen sollte.

Praktische Aufgabe 3: Pretty Good Privacy

In der Vorlesung haben Sie PGP als praktische Anwendung kryptographischer Verfahren wie RSA, AES oder 3DES kennengelernt. Diese praktische Aufgabe hat das Ziel, Sie noch vertrauter mit PGP zu machen.

Ziel der Aufgabe ist das Erstellen eines „Trusted Log Servers“ und einem dazugehörigen Client. Der Server empfängt vom Client signierte Nachrichten, prüft die Korrektheit der Signatur und gibt die Nachricht nach korrekter Prüfung aus. Um die Aufgabe etwas zu vereinfachen, soll das Management der dazu nötigen Schlüssel nicht im Code, sondern auf der Kommandozeile stattfinden.

Vorbereitung: Schlüsselmanagement

Richten Sie sich die frei verfügbare OpenPGP Implementierung GnuPG [2] ein und generieren Sie sich ein PGP-Schlüsselpaar für Ihre UOS-Adresse. Dabei können Sie Ihre Verschlüsselungsalgorithmen frei wählen und können optional weitere eigene E-Mail-Adressen (IDs) zu diesem Paar hinzufügen. Das Schlüsselpaar wird automatisch dem lokalen Schlüsselbund hinzugefügt, auf den Sie später einfach im Code zugreifen können.

Der neu erzeugte öffentliche Schlüssel sollte nun an einen Key-Server gesendet werden, um ihn potentiellen Kommunikationspartnern verfügbar zu machen. Es existieren diverse Key-Server, die sich untereinander synchronisieren und dabei neue Schlüssel verbreiten. Diese Synchronisation benötigt jedoch in der Regel eine gewisse Zeit. Aus diesem Grund empfehlen wir den Key-Server pgp.mit.edu. Damit Sie den übertragenen Schlüssel bei Bedarf wieder von den Key-Servern zurückziehen können, benötigen Sie einen weiteren Schlüssel, den Revoke-Key.

Da Sie im Rahmen dieser Aufgabe effektiv nicht mit anderen kommunizieren müssen, ist das Hochladen des öffentlichen Schlüssels nicht zwingend nötig. Sofern der Server nicht lokal läuft, muss der öffentliche Schlüssel allerdings in den GPG Schlüsselbund des entsprechenden Systems importiert werden. Dies kann entweder durch schlichtes Transferieren des öffentlichen Schlüssels via E-Mail/USB-Stick etc. geschehen oder aber über besagten Key-Server.

Mithilfe Ihres privaten Schlüssels können Sie Nachrichten signieren. Der (hochgeladene) öffentliche Schlüssel erlaubt es nun Ihren Kommunikationspartnern, diese Signatur überprüfen zu können. Zudem können Ihnen nun verschlüsselte Nachrichten gesendet werden. Analog müssen Sie, um jemandem eine verschlüsselte Nachricht senden zu können, dessen öffentlichen Schlüssel kennen.

Allgemeine Hinweise und Tipps

- Sobald Sie Ihren öffentlichen Schlüssel auf einen Key-Server hochgeladen haben sind Sie in der Lage signierte und verschlüsselte E-Mails zu verschicken bzw. zu empfangen. Plugins wie z.B. Enigmail [4] bieten eine direkte Integration von PGP für E-Mail-Programme und vereinfachen die Nutzung somit wesentlich.
- Neben den frei verfügbaren PGP-Programmen GnuPG[2]/OpenPGP[1] für Linux, MacOSX und Windows gibt es inzwischen auch Apps für Android und iOS mit deren Hilfe E-Mails auch über das Smartphone signiert und verschlüsselt werden können.

Programmieraufgabe: Trusted Log Server

Schreiben Sie einen einfachen UDP Client, der signierte Nachrichten an einen Server senden kann. Beim Start des Clienten soll per Parameter die zu versendende Nachricht und der vollständige Name der Person angegeben werden, in dessen Namen signiert werden soll. Außerdem sollen die Adresse und Port des Servers übergeben werden. Orientieren Sie Ihren Programmaufruf an folgendem Muster:

```
./pa3_client SERVER_ADRESSE SERVER_PORT NUTZERNAME "ZU VERSCHLÜSSELNDE NACHRICHT"
```

Nutzen Sie GnuPG Made Easy (GPGME) [3], um die GPG Umgebung zu initialisieren und einen Kontext zu erstellen. Laden Sie außerdem den zum übergebenen Namen gehörigen privaten Schlüssel. Anschließend soll der Client Nachrichten über die Kommandozeile einlesen. Jede Nachricht soll mittels GPGME signiert und anschließend an den Server übertragen werden. Achten Sie darauf, die Nachricht beim signieren nicht zu komprimieren, sodass diese als Klartext lesbar bleibt. Für die Übertragung reicht es aus, die signierte Nachricht in einen Buffer zu schreiben und über einen UDP Socket zu versenden.

Programmieren Sie außerdem einen dazugehörigen UDP Server. Der Server benötigt nur den Port, auf dem Nachrichten eingehen sollen, als Übergabeparameter:

```
./pa3_server SERVER_PORT
```

Zu Beginn wird, wie zuvor beim Client, die GPG Umgebung initialisiert. Danach lauscht der Server auf einem entsprechend vorbereiteten Socket auf Nachrichten. Bekommt er eine Nachricht, so soll die Signatur mittels GPGME überprüft werden. Ist die Signatur korrekt, wird die Nachricht zusammen mit dem Namen des Senders auf der Kommandozeile ausgegeben. Schlägt die Verifizierung fehl, so soll eine entsprechende Warnung ausgegeben werden.

GPG Made Easy

Um die Programmieraufgabe zu lösen, sollten die folgenden Seiten aus der GPGME Dokumentation [3] genügen:

Fehlerbehandlung:
`https://www.gnupg.org/documentation/manuals/gpgme/Error-Handling.html`

Kontextinitialisierung:
`https://www.gnupg.org/documentation/manuals/gpgme/Creating-Contexts.html`

Datenhandhabung:
`https://www.gnupg.org/documentation/manuals/gpgme/Exchanging-Data.html`
`https://www.gnupg.org/documentation/manuals/gpgme/Memory-Based-Data-Buffers.html`
`https://www.gnupg.org/documentation/manuals/gpgme/Data-Buffer-I_002fO-Operations.html`
`https://www.gnupg.org/documentation/manuals/gpgme/Destroying-Data-Buffers.html`

Schlüsselhandhabung:
`https://www.gnupg.org/documentation/manuals/gpgme/Key-objects.html`
`https://www.gnupg.org/documentation/manuals/gpgme/Listing-Keys.html`

Krypto Operationen:
`https://www.gnupg.org/documentation/manuals/gpgme/Crypto-Operations.html`

> Signieren:
> `https://www.gnupg.org/documentation/manuals/gpgme/Sign.html`
> `https://www.gnupg.org/documentation/manuals/gpgme/Selecting-Signers.html`
> `https://www.gnupg.org/documentation/manuals/gpgme/Creating-a-Signature.html`
>
> Verifizieren:
> `https://www.gnupg.org/documentation/manuals/gpgme/Verify.html`

Kompilieren vereinfachen

Um das Kompilieren eines Projekts zu vereinfachen, kann das Programm *make* verwendet werden. Schreiben Sie ein einfaches Makefile für Ihr C-Programm, welches das Programm per `default` Target kompiliert. Zudem sollte Ihr Makefile das Target `clean` unterstützen, d.h. durch den Befehl `make clean` sollen Sie Ihr Projekt bereinigen können.

Allgemeine Funktionsprüfung

Stellen Sie auch sicher, dass Ihr Programm **keine Memory Leaks erzeugt**. Um Ihr Programm auf Speicherfehler zu testen, können Sie das Programm *valgrind* [6] verwenden. Um einige systeminhärente Warnmeldungen zu unterdrücken, können Sie die zur

Verfügung gestellte Valgrind-Suppression-Datei (*alpine.supp*) nutzen.

Vorsicht! Wenn Sie diese Anforderungen nicht einhalten, kann dies zum Nichtbestehen der Aufgabe führen.

Happy Coding!

Abgabe:

- ZIP-Archiv (*ITS_WiSe_201617_PA3_MatrNr.zip*), das beim Entpacken (z.B. *unzip*) automatisch den Ordner *ITS_WiSe_201617_PA3_MatrNr* anlegt (s. Allg. Informationen auf Seite 1).
- In diesem Ordner muss sich das vollständige Programm befinden, das sich (mit Ihrem Makefile durch Ausführen von make) kompilieren lässt, d.h.:
  - alle Quelldateien (.c) [+ Headerdateien (.h)]
  - Makefile

Abgabevariante:

- Upload des ZIP-Archives in **StudIP** (Unter *IT- und Netzwerksicherheit* → *Aufgaben* → *PA3* die Datei hinzufügen und anschließend links im Aktionsmenü *Aufgabe als fertig markieren*)

Literatur

[1] http://openpgp.org/

[2] https://www.gnupg.org/

[3] https://www.gnupg.org/documentation/manuals/gpgme/index.html

[4] https://www.enigmail.net/index.php/en/

[5] https://xkcd.com/1181/

[6] http://valgrind.org/