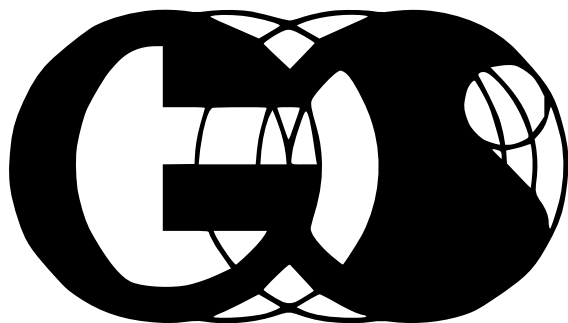


Offene Systeme

Ausgabe 4 (2004)



Gesellschaft für die Anwendung Offener Systeme e. V.

Inhaltsverzeichnis

▶ Editorial	1
Offene Systeme und diebische Elstern	
▶ Frontnachrichten	2
Neuigkeiten aus der freien Welt	
▶ Dieser Satz enthält drei Fehler.	3
Welche Fehler sind es?	
▶ Patentrecht und Landeigentum	5
Ein Vergleich	
▶ Plugins mit wxWidgets	5
Plugins plattformunabhängig – eine Einführung mit wxWidgets	
▶ Kryptographische Skatrunde	10
Die freie Bibliothek libTMCG und eine Referenzimplementierung für Skat	
▶ Über uns	32
Was Sie schon immer über den GAOS e.V. wissen wollten, sich aber nie zu fragen trauten.	
▶ Impressum	32
Autorenverzeichnis und Impressum	

▼ Editorial

Offene Systeme und diebische Elstern

Liebe Leserinnen und Leser,

vielleicht ist es Ihnen bereits aufgefallen: Unsere Zeitschrift trägt einen neuen Namen. „Offene Systeme“ ist zwar nicht sehr originell oder gar von *besonderer Schöpfungshöhe*, aber wir wollen ja keine Wortmarke beim DPMA anmelden, sondern in Zukunft einfach mehr auf Seriosität achten. Die alte Bezeichnung war dafür weniger geeignet und zu introvertiert auf den GAOS-Verein bezogen. Außerdem verbinde ich mit dem Wechsel die große Hoffnung, noch mehr Autoren (z. B. aus dem Bereich der *Freien Software*) anzusprechen. Letztlich geschieht das alles mit dem Ziel, Ihnen, werte Leserschaft, eine noch vielseitigere Lektüre zu bieten.

Die gemeine Elster (lat. *Pica pica*) zählt zu den Rabenvögeln und soll angeblich ein erhöhtes Interesse an glänzenden Gegenständen zeigen. Deshalb bekommt sie vom Volksmund oft die Eigenschaft „diebisch“ attestiert. Die offizielle Software zur elektronischen Steuererklärung (ELSTER), welche seit Anfang 2005 für Unternehmen und Gewerbetreibende Pflicht¹ ist, könnte bald ebenfalls dieses Prädikat tragen. Durch die fehlende Authentifizierung können böswillige Angreifer dem Finanzamt falsche Informationen zuspiesen und letztlich Fehl(ab)buchungen provozieren. Abgesehen von solchen hausgemachten Sicherheitslücken gibt es noch weitere Probleme mit dem ELSTER-Projekt: Zur Zeit steht kostenlos nur die unfreie Lösung „ELSTER-Formular“ (Windows-Plattform) zur Verfügung. Für die Benutzer alternativer Betriebssysteme wie GNU/Linux oder MAC OS waren anscheinend die staatlich finanzierten Entwicklungskapazitäten (ergo Projekt-Budget) erschöpft, so daß jene auf kommerzielle Produkte bzw. eine Windows-Emulation wie WINE zurückgreifen müssen. Auch ein plattformunabhängiges staatliches Web-Portal wird wahrscheinlich erst 2006 fertiggestellt. Die erschreckende Komponente – nicht nur in der ELSTER-Debatte – ist die Ignoranz staatlicher Verantwortungsträger gegenüber einem offenen, ausgewogenen und transparenten Entwicklungsprozeß. Gerade für informationstechnische Neuerungen (Patientenkarte oder biometrische Ausweise² als Beispiel) wäre eine solche Strategie sicher hilf- und lehrreich. Das gilt natürlich auch für die vorangehende Diskussion dieser Maßnahmen. Nicht nur aus Sicht des Datenschutzes vermißt man hier leider zunehmend die Verhältnismäßigkeit der Mittel.

Mit freundlichen und offenen Grüßen,
Ihr HEIKO STAMER.

¹Die elektronische Übertragung (nicht die Software) ist nach einer gewissen Kulanzeit vorgeschrieben, sofern nicht Ausnahmesituationen z. B. wegen *unbilliger Härte* vorliegen.

²Datenschutzfreundliche Lösung: Gerrit Bleumer: *Biometrische Ausweise, Schutz von Personenidentitäten trotz biometrischer Erkennung*, Datenschutz und Datensicherheit 22 (1998)

▼ Frontnachrichten

Neuigkeiten aus der freien Welt

Chemnitzer Linux-Tage 2005

Der Termin der nächsten Chemnitzer Linux-Tage steht fest: Am 5. und 6. März 2005 werden die Fans des freien Betriebssystems Linux wieder den Campus der Technischen Universität Chemnitz bevölkern. Die Organisatoren dieses etablierten Treffs arbeiten bereits auf Hochtouren. Bereits fertig ist die neu gestaltete Homepage. Unter [1] finden Interessenten neben Angaben zu Terminen, Anfahrt und Eintritt auch erste Informationen zu Vorträgen, Workshops, Vorführungen und dem umfangreichen Rahmenprogramm. Im Jahr 2004 besuchten übrigens 2.500 Gäste die Chemnitzer Linux-Tage. Diese Teilnehmerzahl wollen die Veranstalter der TU Chemnitz im März 2005 wieder erreichen.

Einen weiteren Termin sollten sich Referenten, die bei den Chemnitzer Linux-Tagen im März 2005 auftreten wollen, merken: Bis zum 7. Januar 2005 können Beiträge bei den Veranstaltern [2] eingereicht werden. Wie in den Vorjahren finden wieder alle Themen rund um Linux und Open Source ihren Platz im Vortragsprogramm. Einen besonderen Schwerpunkt bildet 2005 das Thema „Der Desktop – Surfen durch den Arbeitsalltag“.

MARIO STEINEBACH, Technische Universität Chemnitz



Wahre Liebe – Linux loves desktop.

Chemnitzer Linux-Tage
am 5. und 6. März 2005

Neues Hörsaal- und Seminargebäude
der TU Chemnitz | Reichenhainer Str. 90

<http://chemnitzer.linux-tage.de>



2. Brandenburger Linux Infotag

Am 22. und 23. April 2005 findet in der Stadt Brandenburg an der Havel der 2. Brandenburger Linux Infotag (BLIT2005) statt. Die Vorbereitungen kommen jetzt langsam in die Gänge ...

Als besonderes Rahmenevent zu dieser Veranstaltung wurde die Idee eines „LUG-Pavillions“ geboren, zu dem wir LUGs aus dem Umkreis (Ostdeutschland) einladen möchten. Idee soll es sein, LUGs die Möglichkeit zu geben sich und ihre Projekte/ Aktivitäten einem größeren Publikum vorzustellen. Desweiteren besteht die Möglichkeit sich aktiv an der Organisation und Gestaltung des BLIT2005 mit Rat/ Ideen sowie vielleicht auch Tat mit-zuwirken.

Impressionen zum vergangenen BLIT2004 findet Ihr unter [4]. Diese Seite wird demnächst sukzessive für die geplante Veranstaltung aktualisiert.

UWE BERGER, BraLUG e.V.

3rd International Linux Audio Conference

Vom 21. bis 24. April 2005 findet in Karlsruhe die „3rd International Linux Audio Conference“ [5] statt. Tagungsort ist wieder das „Zentrum für Kunst und Medientechnologie (ZKM)“. Im Vorjahr gaben sich Linux-Audio-Entwickler wie Steve Harris, Paul Davis, Fernando Pablo Lopez-Lezcano, Dave Phillips, Takashi Iwai oder Matthias Nagorni die Klinke in die Hand, so dass auch 2005 mit interessanten Vorträgen, Präsentationen und Workshops gerechnet werden darf. Der Linux-Audio-Bereich entwickelt sich seit einiger Zeit in beeindruckendem Tempo. Die Programmierer haben professionelle kommerzielle Lösungen ins Visier genommen und arbeiten an Open Source-Alternativen, die auch den Anhängern Freier Software das Einrichten einer voll ausgestatteten digitalen Audioworkstation erlauben soll.

FLORIAN BERGER, fb-audio Leipzig

Radio Garage

Radio Garage ist ein ehrgeiziges Projekt: Auf der Basis mehrjähriger Erfahrungen in den Bereichen Hörfunk, Internet-Streaming, DJ-ing und Web-Services hat ein junges, neunköpfiges Team das erste freie Internet-Radio in Halle gegründet. Geplanter Sendestart ist der 1. April 2005.

Radio Garage ist die ideale Plattform für akustische Experimente aller Art. Ziel ist es, Formate des konventionellen Hörfunks neu zu interpretieren, sie zu erweitern und im besten Falle zu sprengen. Das Team des Senders arbeitet ehrenamtlich. Keine Quote wird die Programmviefalt

von Radio Garage limitieren. Die konsequente Verwendung von Software aus dem Open-Source-Bereich und das breite Spektrum der Internet-Technologie bieten zahlreiche Möglichkeiten zur freien Entfaltung.

Wir heissen jeden, der Lust hat, Erfahrungen in unserem Radio zu sammeln oder sein Können und sein Engagement in unser Projekt einfließen zu lassen, herzlich willkommen. Unter [6] gibt es stets aktuelle Kontaktinformationen.

BENJAMIN VÖLLGER, Radio Garage e. V.

Referenzen

- [1] <http://chemnitzer.linux-tage.de/>
- [2] <http://chemnitzer.linux-tage.de/2005/vortraege/call.html>
- [3] <http://idw-online.de/pages/de/news86889>
- [4] <http://www.linuxinfotag-brb.de/>
- [5] <http://www.zkm.de:81/lac/>
- [6] <http://www.radiogarage.org/>

▼ Dieser Satz enthält drei Fehler.

SVEN TÜRPE: Welche Fehler sind es?

Zwei haben Sie ganz bestimmt entdeckt. Wer sich mit dem dritten schwertut, dem sei verraten: ich habe mich verzählt. Es sind nur zwei, und das ist der dritte Fehler. Aber irgend etwas stimmt nicht, oder? Genau. Jetzt, da wir den dritten Fehler entdeckt haben – ist er weg. Ich habe mich ja doch nicht verzählt. Also sind es doch nur zwei Fehler. Doch halt! Dann stimmt ja wieder alles. Aber ...

Wer sich bei solchen Gedanken ertappt, hat es mit einer *Paradoxie* zu tun. Paradoxien erkennt man daran, daß korrekte logische Schlußfolgerungen aus einer (scheinbar) sinnvollen Aussage zu einem Widerspruch führen. Die Gründe liegen nicht immer so klar auf der Hand wie im selbstwidersprüchlichen Satz: »Dieser Satz ist falsch.« Erkennen kann man Paradoxien aber doch in jedem Fall daran, daß man beim Schlußfolgern in einer Endlosschleife landet.

Paradoxien beschäftigten bereits die Gelehrten der alten Griechen einige Jahrhunderte vor Beginn unserer Zeitrechnung. Zu den bekanntesten Paradoxien aus jener Zeit gehört jenes, das Zenon von Elea formulierte: Achilles, ein guter Läufer, läßt einer Schildkröte einen kleinen Vorsprung – und kann sie nie einholen, obwohl er viel schneller läuft: Kommt er nämlich an den Punkt, wo die Schildkröte eben noch war, ist sie schon wieder ein Stückchen

weitergelaufen. So wird der Abstand immer kleiner, aber nie überholt Achilles den schwachen Gegner, so schnell er auch laufen mag.

Zenons Paradox ergibt sich aus einer mangelhaften Theorie. Wer unendliche Reihen und Grenzwertbetrachtungen kennt, wird sich schnell ausrechnen, daß die Schildkröte im Gedankenexperiment nicht beliebig weit laufen kann, sondern lediglich auf einen Punkt zustrebt, genau jenem Punkt, an dem sie dann eben doch überholt wird.

Viel Zeit verging, bis die Paradoxien im 20. Jahrhundert eine wahre Blütezeit erlebten. Zunächst fand Bertrand Russell im Jahre 1901 ein Problem in der Mengentheorie, an deren sauberer Formalisierung man damals gerade arbeitete. Bildet man nämlich die Menge aller Mengen, die sich nicht selbst enthalten, so kann man, vergleiche den eingangs geschilderten Gedankengang, nicht entscheiden, ob sich diese Menge selbst enthält oder nicht. Das Problem kann man nur umgehen, indem man sie Mengenlehre so aufbaut, daß diese Konstruktion ausgeschlossen ist.

In den 30er Jahren des 20. Jahrhunderts dann machten Kurt Gödel und Alan Turing aus der Not eine Tugend und nutzten analoge Konstruktionen, um den Gödelschen Unvollständigkeitssatz beziehungsweise die Nichtlösbarkeit des Entscheidungsproblems¹ zu beweisen.

Der Gödelsche Unvollständigkeitssatz besagt, daß in einem konsistenten, formalen System der Mathematik, welches ausreicht, die Arithmetik zu beschreiben, Aussagen herleiten lassen, deren Gültigkeit innerhalb dieses Systems weder bewiesen noch widerlegt werden kann.

Alan Turing befaßte sich mit der Frage, welche Probleme sich algorithmisch, anhand einer formalen Rechenvorschrift lösen lassen, und damit mit den prinzipiellen Grenzen des Computers, der damals erst ansatzweise erfunden war. Er bewies, daß es keinen Algorithmus geben kann, der für alle Sätze der Prädikatenlogik erster Ordnung entscheidet, ob die allgemeingültig sind oder nicht. Oder daß es kein Computerprogramm gibt, das für alle Programme entscheidet, ob sie eine Endlosschleife enthalten oder nicht.

Für Details zu beiden Sätzen und ihren Beweisen sei auf die Fachliteratur verwiesen. Für einen ersten Überblick tut es auch die Wikipedia. Beide gehören zum Wichtigsten, was die Mathematik im 20. Jahrhundert hervorgebracht hat. Sie sind die Grundlage der theoretischen Informatik.

Um Politiker zu werden, muß man von alledem nichts verstehen. Leider, denn auch am Anfang des 21. Jahrhunderts bekommen wir es wieder mit Paradoxien zu tun, diesmal im Gesetz. Das ist unschön, denn zu jedem Gesetz, so absurd es auch sein mag, findet sich jemand, der es trotzdem durchsetzen will und eine Zeitlang auch die Macht dazu hat.

¹Den Informatikerinnen unter unseren Leserinnen sicher auch in Form des Halteproblems der Turingmaschine bekannt

Aber der Reihe nach. Eben ist ein neues Urheberrechtsgesetz fertig geworden. Es soll das Urheberrecht den nach dem Siegeszug des Internets zweifellos veränderten Bedingungen anpassen. So lautet denn auch der Titel des Werkes in voller Schönheit: »Gesetz zur Regelung des Urheberrechts in der Informationsgesellschaft«. Klingt gut, ist aber möglicherweise Mist. Das Gesetz enthält nämlich eine Vorschrift, die es in sich hat. Paragraph 95a des neuen Gesetzes bestimmt:

»Wirksame technische Maßnahmen zum Schutz eines nach diesem Gesetz geschützten Werkes oder eines anderen nach diesem Gesetz geschützten Schutzgegenstandes dürfen ohne Zustimmung des Rechtsinhabers nicht umgangen werden, soweit dem Handelnden bekannt ist oder den Umständen nach bekannt sein muss, dass die Umgehung erfolgt, um den Zugang zu einem solchen Werk oder Schutzgegenstand oder deren Nutzung zu ermöglichen.«

Verboten wird auch die Verbreitung dazu geeigneter Werkzeuge, soweit sie vorwiegend dem Zweck der Umgehung dienen.

Hinter dieser Regelung verbirgt sich, Sie ahnen es, eine Paradoxie. Unter einer wirksamen technischen Schutzmaßnahme könnte man sich etwas vorstellen, das das Kopieren verhindert. Dann aber ergibt der Begriff der Umgehung keinen Sinn; die Maßnahme ist ja wirksam, sie kann also nicht umgangen werden. Wurde andererseits eine Maßnahme umgangen, so war sie offensichtlich *nicht* wirksam.

Von der Umgehung einer wirksamen Schutzmaßnahme zu sprechen ergibt also keinen Sinn. Warum tut man es trotzdem? Ganz einfach: Es gibt keine wirksamen technischen Maßnahmen, die in unserer real existierenden Welt das Kopieren zum Beispiel einer Musikaufzeichnung oder eines Films verhindern könnten. Die Musik muß irgendwann von einem Gerät zum Ohr gelangen, der Film zum Auge. Die Aufzeichnung verläßt also stets das technische System, in dem sie gespeichert ist oder transportiert wird. Und für jedes Medium gibt es neben Wiedergabegeräten auch solche, die die Aufzeichnung ermöglichen. Hinzu kommt der Universalcomputer, der zum einen als beliebig programmierbares² Gerät Kopien beliebiger Datenströme herstellen und speichern und zum anderen beliebige Manipulationen daran vornehmen kann. Das ist so gewollt, sonst braucht man nämlich keinen Computer. Und es ist auch der Grund dafür, daß Computer inzwischen zum wichtigsten Wiedergabegerät für viele Medien geworden sind.

Die bisher vorgeschlagenen Möglichkeiten zur technischen Verhinderung des Kopierens ignorieren diese Tatsachen. Sie gehen von der *Closed System Assumption* aus,

²Für Haarspalter: in den von Alan Turing nachgewiesenen Grenzen programmierbares

der Annahme, daß man audiovisuelle Medien in einem geschlossenen System halten könne. Auf diese Weise ließe sich tatsächlich ein wirksamer Schutz realisieren, aber um den Preis, daß das Werk dann tatsächlich stets im System bliebe, also für Menschen nicht sichtbar und nicht hörbar wäre. Womit sich die philosophische Frage stellt, ob es dann überhaupt existiert. Unter Realweltbedingungen hingegen geht der Versuch, das Kopieren digitaler Daten zu verhindern oder gar *Digital Restrictions Management* zu betreiben, regelmäßig in die Hose [6].

Nun schreibt man solche Regelungen aber nicht aus Spaß ins Gesetz. Und selbst wenn es nur ein Scherz wäre, so wird doch mit dem Inkrafttreten schnell Ernst daraus, denn dann gilt es und wird von Richtern interpretiert. Da geht es dann unter anderem um den mutmaßlichen Willen des Gesetzgebers – der sich aus einem paradoxen Gesetz ganz nach Belieben herleiten läßt, egal, welches Ziel man verfolgt. Wir dürfen gespannt sein auf die Prozesse, die sich aus diesem Paragraphen ergeben. Verfassungsgericht, anyone?

Referenzen

- [1] Wikipedia: *Paradox* <http://de.wikipedia.org/wiki/Paradox>
- [2] Wikipedia: *Russell's Paradox* http://www.wikipedia.org/wiki/Russell's_paradox
- [3] Autorenkollektiv: Gesetz zur Regelung des Urheberrechts in der Informationsgesellschaft. <http://www.bmj.bund.de/images/11476.pdf>
- [4] Gregory J. Chaitin: Grenzen der Berechenbarkeit. in: *Spektrum der Wissenschaft*, Februar 2004.
- [5] Volker Grassmuck: *Das Ende des Allzweck-Computers steht bevor*. FIF-Kommunikation 4/2002, <http://waste.informatik.hu-berlin.de/Grassmuck/Texts/drm-fiffko.html>
- [6] Heise Online: *Neue Version von Microsofts E-Book-Reader schon geknackt*. <http://www.heise.de/newsticker/data/jr-09.07.03-000/>
- [7] Heise Online: Sind Privatkopierer nach EU-Recht bald Raubkopierer? <http://www.heise.de/newsticker/data/jk-09.12.03-000/>
- [8] Douglas R. Hofstadter: *Gödel, Escher, Bach. Ein Endloses Geflochtenes Band*. Dtv, 1991.
- [9] László Méré: *Die Grenzen der Vernunft. Kognition, Intuition und komplexes Denken*. Hamburg: Rowohlt, 2002.
- [10] Peter Mühlbauer: *Gesetzbuch zu ... und alle Fragen offen*. <http://www.heise.de/tp/deutsch/special/copy/15718/1.html>

- [11] Wolf-Dieter Roth: *Das neue Geschäftsmodell der Plattenindustrie?* Telepolis, 2003-12-16. <http://www.heise.de/tp/deutsch/special/copy/16331/1.html>
- [12] Matthias Spielkamp: *Mit Technik allein lässt sich DRM nicht durchsetzen.* <http://www.heise.de/tp/deutsch/special/copy/16673/1.html>

sowieso jemand kommen und behaupten, er wäre eher da gewesen, egal was im Grundbuch steht. Nur wenn alle im Dorf sagen, dass der Kerl ein Idiot ist, hast Du eine Chance!“

Selbst in den Staaten gibt es die Regel, dass man seine Rechte an einem Claim verwirkt, wenn man nicht auf ihm wirtschaftet! Mir will nicht in den Kopf, wieso das beim Patent nicht genauso gesehen wird. Oder wird es das?

▼ **Patentrecht und Landeigentum**
 HOLGER KLAWITTER: Ein Vergleich

Ich hatte einen Tagtraum: Der Siedler landet auf einem Kontinent, auf dem es keinen Grundbesitz gibt. Dem Indianer sagt er:

„So wie Ihr lebt, so kann man doch nicht leben. Jeder kann überall wohnen? Ihr müßt Grundbücher einführen, damit man bei Streitigkeiten immer nachschauen kann, wem das Land gehört und wer damit recht hat! Und bei Euch kann man auch nicht handeln! Wir wissen nicht, wo wir unseren Drugstore hinbauen sollen! Für sicheres Handeln muss mein Haus auf meinem Grund stehen können.“

Die Idee klingt auf den ersten Blick nicht schlecht: „Das Ganze ist gerecht und benachteiligt niemanden, denn wer ein neues urbares Stück Land findet, darf es abstecken, und es gehört dann ihm. So kann niemand kommen und ihm den Acker streitig machen.“

Was der Siedler nicht sagt, ist aber das Folgende: „Unser Grundbuch wird so geschrieben: Jeder, der will, kann in dieses Grundbuch irgendwas hineinschreiben. Zum Beispiel: 'Mir gehört das Tal hinter den sieben Bergen da hinten.' Wenn man es etwas geschickt hinschreibt, kann auch als zweiter das selbe Land beanspruchen. Erst wenn es zum Streit kommt, wird vor Gericht entschieden, wer Recht hat.“

So weit her mit der Sicherheit ist es also nicht. Und: „Ach, ja, hier ist übrigens das Grundbuch für Euren Kontinent, das wir vor unserer Reise auf Euren Kontinent erstellt haben. Ich und meine Kollegen haben die großen Täler und Ebenen schon unter uns aufgeteilt. Wenn Ihr neues Land findet und wann immer Ihr zwei Flecken verbinden wollt, werdet Ihr unweigerlich über unser Land müssen, und dann werden wir Zoll nehmen können.“

Da kommt der Indianer schon ins Grübeln, ob das mit dem Grundbuch wirklich so eine gute Idee ist. Was kann er dagegen tun? Auf seiner Scholle mit einer Schrotflinte sitzen? Oder das gefundene Tal geheim halten? Nein, beides ist nicht praktikabel. Irgendwann findet einer das Tal zufällig, und die Schrotflinte hilft nicht viel gegen eine angeheuerte Armee.

Was kann man dem Indianer raten? Ich möchte Ihn sagen: „Führe Dein eigenes Grundbuch! Mach es öffentlich, verteile es! Wenn Du auf Deinem Land Öl findest, wird


▼ **Plugins mit wxWidgets**
 FRANK-MICHAEL SCHLEIF: Plugins plattformunabhängig – eine Einführung mit wxWidgets

Software & Entwicklung

Fast alle Programme aus dem Multimediaumfeld haben eine Besonderheit gemeinsam - sie unterstützen Plugins. Das ist durchaus kein Zufall, sondern eher darin begründet, daß es ein Vielzahl von unterschiedlichen Verfahren gibt, Multimediadaten zu verarbeiten und ständig neue Methoden dazukommen und die alten Methoden erweitert werden. Um den Anwender und auch dem Softwareentwickler möglichst viel Freiheit in der Verwendung, Erstellung und Wartung der entsprechenden Programme zu bieten, ist eine Plug-inschnittstelle eine recht geeignet Lösung. Doch auch in anderen Programmen tauchen neuerdings Plugins auf und die interessante Frage ist – wie funktioniert das eigentlich?

Im folgenden Artikel wird beschrieben, wie man Plugins mit dem Widget-Toolkit wxWidgets programmieren kann und welche Vor- und Nachteile sie haben. Natürlich gibt es immer mehrere Möglichkeiten Plugins und ein Pluginsystem zu erstellen, daher soll auch kurz erklärt werden, warum gerade Plugins mit wxWidgets.

wxWidgets – plattformunabhängig und flexibel

 wxWidgets¹ ist ein Toolkit zur plattformunabhängigen Erstellung von Anwendungen. Im Unterschied zu z. B. Java erhält es dabei das Look & Feel der gewählten Plattform und drückt der Anwendung nicht einen bestimmen optischen oder funktionalen Stempel auf. Wer eine mit wxWidgets erstellte Anwendung schreibt und diese für den MAC erzeugt, erhält eine echte MAC Anwendung und (im Idealfall) durch einfaches Linken gegen die entsprechende MSW-Variante der wxWidgets Bibliothek die gleiche Anwendung mit Look & Feel für Windows.

Damit das alles schön einfach und ohne viel Anpassungen an die einzelnen Plattformen funktioniert bietet wxWidgets für alle Probleme des Programmieralltags (Konsole, Strings, Sockets, Datenbanken, . . . , Plugins) entsprechende Klassen die, wenn man sie ordentlich und konsequent benutzt tatsächlich zu Plattformunabhängigkeit der erstellten Anwendung führen und lediglich für die Zielplattform neu kompiliert werden müssen und dabei gegen die entsprechende wxWidgets-Bibliothek gelinkt werden. Für eine detaillierte Einführung in wxWidgets sei auf [1]

¹Formals wxWindows – man kann sich leicht vorstellen, warum sich der Name geändert hat.

verwiesen, wo man auch Aussagen dazu findet warum es eben auch gute Gründe gibt, nicht einfach nur laut „Java“ zu rufen. Wie gerade gesagt, gibt es extrem viele Klassen zur Unterstützung bei der Programmierung und auch Plugins sind mit einem geringen Maß an Handarbeit schnell erstellt, einsatzbereit und durch einfaches Neukompilieren für andere Plattformen von wxWidgets auch plattformunabhängig verfügbar².

Neben der obligatorischen Wahl von Java für die Erstellung von (mehr oder weniger) plattformunabhängigen Komponenten die man auch als Plugins ansehen kann, gibt es für plattformunabhängige Plugins die naheliegende Wahl von Scriptsprachen, die diesen Konzept meist inherent unterstützen.³ Grundsätzlich spricht nichts dagegen und oftmals ist der Weg über Scriptsprachen ein schneller und einfacher Weg ans Ziel. Das Grafikprogramm GIMP geht genau diesen Weg und ist damit sehr erfolgreich. Es gibt aber gute Gründe warum dieser Weg nicht immer der gewünschte ist.

Zum einen muß die Hostanwendung (also die Anwendung die das Plugin lädt) das Script interpretieren – also einen Interpreter für die gewählte Scriptsprache mitbringen. Das kann je nach Komplexität der Scriptsprache recht aufwendig sein (den Trivialfall, daß die Hostanwendung selber ein Script ist, einmal ausgenommen). Zum anderen sind Scripte letztlich Textdateien die meist in unkompielter Form verarbeitet werden. Das hat den Vorteil leichter Änderbarkeit aber auch den Nachteil, daß es jederzeit ändern kann⁴ und die Interpretation von Code insbesondere bei mathematischen Berechnungen oft nachteilig ist (längere Laufzeit). Zudem kann es Nachteile bei der Anbindung/Nutzung von Fremdbibliotheken geben, die sich bei Verwendung von nativen Plugins so nicht stellen.

Wenn man sich entschieden hat keine Scriptsprachenlösung zu wählen und andere Lösungsmöglichkeiten wie Java, Corba oder .Net auch nicht gefallen, landet man bei Widget-Toolkits. Kommt dann noch die Frage der Plattformunabhängigkeit ins Spiel bleibt nicht mehr viel übrig und wxWidgets taucht am Horizont auf.

Soviel zur Motivation warum native Plugins mit wxWidgets einen Blick wert sind, und nun mal etwas Butter zu den Fischen.

wxWidgets-Plugins - Grundlagen

wxWidgets realisiert Plugins als shared objects. Diese haben je nach Plattform unterschiedliche Ausprägungen und sind unter Unixsystem meist durch die Endung .so zu erkennen und z.B. unter Windows als DLLs realisiert. Gemein ist beiden Varianten das sie zur Laufzeit

²Zumindest für diejenigen Plattformen, die ein shared library Konzept unterstützen

³Es gibt auch noch andere Möglichkeiten z. B. ala Corba aber für die Motivation sollen Scriptsprachen genügen.

⁴will man ja vielleicht nicht

des Programms per dlopen geladen oder entladen werden können.

Die Hauptarbeit leistet also bereits das Betriebssystem und dem Programmierer bleibt an sich nicht viel zu tun, außer darauf zu achten daß die Symbole, die durch den Plugincode definiert werden auch korrekt exportiert werden, um von Linker beim dynamischen Laden sauber auflösbar zu sein. Nun für z. B. klassisches C ist das relativ einfach und funktioniert sehr gut (wie man z. B. an der libc unter Unix sieht, die nahezu jedes Unix-Programm dynamisch lädt).

wxWidgets ist in seiner nativen Form in C++ implementiert, das macht die Sache etwas schwieriger, da alle exportierten Symbole in dem C++-Plugin schick verziert sind (sogeanntes Name(de)mangling). Leider gibt es für die daraus resultierende Darstellung der Symbole keinen Standard, so daß jeder Compilerbauer seinen eigenen Weg beschreitet und man – etwas lachs gesprochen – in der Hostanwendung für jede mögliche Compilervariante/-version, die zur Erzeugung der Plugins verwendet wurde die Symbole gesondert extrahieren müßte. Das ist reichlich mühsam und nicht zu empfehlen. Wie bekommt man nun aber für ein C++-Objekt (z. B. eine Klasse) das Symbol heraus?

Eine Variante das Problem des Mangeling zu umschiffen ist, in jedes Plugin eine statische Instanz der Klasse einzubringen und für diese Variable das Mangeling abzuschalten. In der Host-Anwendung kann dann auf diese statisch exportierte Variable normal zugegriffen werden um z. B. Methoden der Klasse aufzurufen. Wie das ganz grob funktioniert ist im folgenden Beispielcode dargestellt, der zunächst noch kein wxWidget verwendet.

Wir beginnen mit einem Beispielplugin, daß lediglich eine beliebige Form – in unserem Fall einen Kreis auf der Konsole darstellt. Das allgemeine Interface für ein solches Plugin ist im folgenden dargestellt. Es deklariert eine vollständig virtuelle Methode, die jedes Shape-Plugin implementieren muß. Aus Bequemlichkeitsgründen wird auch noch definiert, daß es extern (in der Hostanwendung) ein map Objekt geben soll in das wir ein konkretes Plugin automatisch eintragen können, sobald es geladen wird.

```
#ifndef __SHAPE_H
#define __SHAPE_H

#include <map>
#include <string>
using namespace std;

// base class for all shapes
class shape
{
public:
    virtual void draw() = 0;
};

// typedef to make easy factory setup
typedef shape *maker_t();

// our global factory
```

```
extern map<string, maker_t *, less<string> > factory;
#endif // __SHAPE_H
```

Nachdem wir nun das Interface für das Plugin kennen, folgt ein konkretes Plugin. Dieses kann einen Kreis darstellen – also etwas allgemeiner eine Grafik.

```
#ifndef __CIRCLE_H
#define __CIRCLE_H

using namespace std;
#include "shape.hh"

class circle : public shape
{
public:
    void draw();
};
#endif // __CIRCLE_H
```

Jetzt folgt die Implementierung:

```
#include <iostream>
#include "circle.hh"

void circle::draw()
{
    // simple ascii circle
    cout << "\n";
    cout << "    ****\n";
    cout << "   *      *\n";
    cout << "  *        *\n";
    cout << " *          *\n";
    cout << "*            *\n";
    cout << " *          *\n";
    cout << "  *        *\n";
    cout << "   *      *\n";
    cout << "    ****\n";
    cout << "\n";
}

extern "C"
{
    shape *maker()
    {
        return new circle;
    }

    class proxy
    {
    public:
        proxy()
        {
            // register the maker with the factory
            factory["circle"] = maker;
        }
    };

    // our one instance of the proxy
    proxy p;
}
```

Wie man erkennt, deklariert und implementiert es eine Klasse, die die eigentliche Arbeit macht und den Kreis anzeigt. Dazu leitet sie von dem allgemeinen Plugininterface ab und muß folgerichtig auch die vollständig virtuelle Methode draw implementieren. Soweit hat es noch nicht viel mit Plugins zu tun. Interessant ist der zweite Abschnitt, in dem als extern "C", also ohne Namemangling eine statische Methode deklariert wird, die ein neues Objekt der Circle-Klasse anlegt und eine Klasse proxy deklariert wird. Diese Klasse trägt über ihren Konstruktor lediglich in die extern definierte map (Wir erinnern uns, es war oben vereinbart, daß es soetwas in der Hostanwendung

geben soll.) unter dem gewählten Bezeichner circle eine Instanz unserer konkreten Pluginklasse ein. Damit das beim Laden des Plugins ganz automatisch passiert, wird dann auch noch ein statisches Objekt⁵ der Klasse proxy erzeugt. Damit erreiche ich, daß durch einfaches Laden des Plugins bereits die ganze Arbeit erledigt ist.

Und nun fehlt uns noch die Hostanwendung dazu:

```
#include <iostream>
#include <map>
#include <list>
#include <vector>
#include <string>
#include <dlfcn.h>
#include <stdio.h>
#include <unistd.h>
#include "shape.hh"

using namespace std;

// size of buffer for reading in directory entries
static unsigned int BUF_SIZE = 1024;

// our global factory for making shapes
map<string, maker_t *, less<string> > factory;

int main(int argc, char **argv)
{
    // handle to read directory
    FILE *dl;
    // command string to get dynamic lib names
    char *command_str = "ls *.so";
    // input buffer for lib names
    char in_buf[BUF_SIZE];
    // list to hold handles for dynamic libs
    list<void *> dl_list;
    list<void *>::iterator itr;
    // vector of shape types used to build menu
    vector<string> shape_names;
    // list of shape objects we create
    list<shape *> shape_list;
    list<shape *>::iterator sitr;
    map<string, maker_t *, less<string> >::iterator fitr;
    // get the names of all the dynamic libs
    // (.so files) in the current dir
    dl = popen(command_str, "r");
    if(!dl)
    {
        perror("popen");
        exit(-1);
    }

    void *dlib;
    char name[1024];
    while(fgets(in_buf, BUF_SIZE, dl))
    {
        // trim off the whitespace
        char *ws = strpbrk(in_buf, " \t\n");
        if(ws) *ws = '\0';
        // append ./ to the front of the lib name
        sprintf(name, "./%s", in_buf);
        dlib = dlopen(name, RTLD_NOW);
        if(dlib == NULL)
        {
            cerr << dlerror() << endl;
            exit(-1);
        }
        // add the handle to our list
        dl_list.insert(dl_list.end(), dlib);
    }

    int i = 0;
    // create an array of the shape names
    for(fitr=factory.begin(); fitr!=factory.end(); fitr++)
    {
        shape_names.insert(shape_names.end(), fitr->first);
        i++;
    }
}
```

⁵statische Objekte werden beim Laden des Plugins automatisch angelegt

```

int choice;
// create a menu of shapes
while(1)
{
    i = 1;
    for(fitr=factory.begin(); fitr!=factory.end(); fitr++)
    {
        cout << i << " - Create " << fitr->first << endl;
        i++;
    }
    cout << i << " - Draw created shapes\n";
    i++;
    cout << i << " - Exit\n";
    cout << "> ";
    cin >> choice;
    if(choice == 1)
    {
        // destroy any shapes we created
        for(sitr=shape_list.begin();
           sitr!=shape_list.end();sitr++)
        {
            delete *sitr;
        }
        // close all the dynamic libs we opened
        for(itr=dl_list.begin();
           itr!=dl_list.end(); itr++)
        {
            dlclose(*itr);
        }
        exit(1);
    }
    if(choice == i - 1)
    {
        // draw the shapes
        for(sitr=shape_list.begin();
           sitr!=shape_list.end();sitr++)
        {
            (*sitr)->draw();
        }
    }
    if(choice > 0 && choice < i - 1)
    {
        // add the appropriate shape to the shape list
        shape_list.insert(shape_list.end(),
                           factory[shape_names[choice-1]]());
    }
}
}
}

```

Bei Plattformunabhängigkeit ist dieser Weg aber dennoch etwas mühsam, da es je nach Plattform meist kleine Unterschiede gibt, die es zu beachten gilt. Unter wxWidgets wird der Entwickler aber auch dabei unterstützt, so dass die Art der Plattform nahezu *ignoriert* werden kann. Dazu wird über einige Hilfsklassen und Präprozessordirektiven die mühselige Handarbeit zum Laden und Erzeugen der Pluginobjekte gekapselt und die Plattfordetails versteckt. Das Prinzip ist im wesentlichen das gleiche, aber in der Implementierung sieht die Arbeit mit Plugins unter wxWidgets entsprechend etwas anders aus. Das nachfolgende Beispiel ist ein sehr simples Beispiel – dabei ändert das Plugin lediglich den Titeltext des Fensters der Hostanwendung.

Wir beginnen wieder mit der Pluginschnittstelle:

```

#ifndef __PLUGIN_H__
#define __PLUGIN_H__

#include <wx/dynload.h>
#include <wx/dynlib.h>

/**
 * Since this file is included both from inside a plugin and
 * inside main app, we need to use WXEXPORT or WXIMPORT. When
 * including from plugin, be sure to define WXUSINGDLL and

```

```

 * WXMaking_MY_DLL beforehand, but when including from main
 * application, define WXUSINGDLL and WXUSING_MY_DLL.
 */
#ifdef WXUSINGDLL && (defined(WXMaking_MY_DLL) || \
defined(WXUSING_MY_DLL))
    #if defined(WXMaking_MY_DLL)
        #define WXP_EXPORTED_FROM_APP_TO_PLUGIN WXEXPORT
    #elif defined(WXUSING_MY_DLL)
        #define WXP_EXPORTED_FROM_APP_TO_PLUGIN WXIMPORT
    #endif
#else
    #define WXP_EXPORTED_FROM_APP_TO_PLUGIN
#endif

#define WxDll_Entry_Function() \
extern "C" WXEXPORT const wxClassInfo *wxGetClassFirst(); \
const wxClassInfo *wxGetClassFirst() { \
    return wxClassInfo::GetFirst(); \
}

```

```

class WXP_EXPORTED_FROM_APP_TO_PLUGIN Plugin: public wxObject
{
    DECLARE_ABSTRACT_CLASS(Plugin)
public:
    virtual void OnEntry(wxWindow *mainframe) = 0;
};

IMPLEMENT_ABSTRACT_CLASS(Plugin, wxObject)

#endif

```

Das Plugin deklariert wieder eine vollständig virtuelle Methode – diesmal `OnEntry` genannt. Ein Header für ein Beispielplugin ist im Folgenden angegeben:

```

#ifndef __MINIMAL_PLUGIN_H__
#define __MINIMAL_PLUGIN_H__

#ifdef __GNUG__
    #pragma interface "minimal_plugin.h"
#endif
#include <wx/wx.h>
#include <wx/dynlib.h>

/* These must be defined before including plugin.h
   to notify we'r the plugin */
#define WXMaking_MY_DLL
#define WXUSINGDLL

#include "plugin.h"

class Minimal_Plugin : public Plugin
{
    DECLARE_DYNAMIC_CLASS(Minimal_Plugin)
public:
    Minimal_Plugin()
    {
        printf("[Plugin ] Constructor\n");
    }

    virtual ~Minimal_Plugin()
    {
        printf("[Plugin ] Destructor\n");
    }

    void OnEntry(wxWindow *mainframe);
};

#endif

```

Und hier wieder die Implementierung:

```

#ifdef __GNUG__
    #pragma implementation "minimal_plugin.h"
#endif

#include "minimal_plugin.h"

IMPLEMENT_DYNAMIC_CLASS(Minimal_Plugin, Plugin)
WxDll_Entry_Function();

```

```
void Minimal_Plugin::OnEntry(wxWindow *mainframe)
{
    mainframe->SetTitle(
        wxString::Format(
            wxT("%s by %s - Title changed by plugin!"),
            wxTheApp->GetAppName().c_str(),
            wxTheApp->GetVendorName().c_str()
        )
    );
}
```

Für das Plugin braucht man jetzt wieder eine passende Hostanwendung. Auch dafür wird jetzt der Header und dann die Implementierung angegeben:

```
#ifndef __MINIMAL_H__
#define __MINIMAL_H__

#ifdef __GNUG__
#pragma interface "minimal.h"
#endif

#include "wx/wx.h"

/* These must be defined before including plugin.h to notify
 * we'r the main app
 */
#define WXUSINGDLL
#define WXUSING_MY_DLL

#include "plugin.h"
#include <wx/dynlib.h>

class Minimal : public wxApp
{
public:
    virtual bool OnInit(); // Application startup
    wxFrame *mainframe; // Global for accessing from plugins
private:
    void LoadPlugin(); // Plugin loader function
};

DECLARE_APP(Minimal)
// Forward declaration for wxGetApp() to work

#endif

#ifdef __GNUG__
#pragma implementation "minimal.h"
#endif

#include "minimal.h"

IMPLEMENT_APP(Minimal)

/**
 * Application startup. Display a sample frame and call
 * LoadPlugin().
 */
bool Minimal::OnInit()
{
    /* Application/Vendor names */
    SetAppName(wxT("Minimal"));
    SetVendorName(wxT("Samples"));

    /* Some dummy frame */
    mainframe = new wxFrame(NULL, -1,
        wxT("Minimal plugin sample"));
    mainframe->Show(true);

    printf("[Main app] Before plugin loading...\n");

    /* Call plugin loading function */
    LoadPlugin();

    printf("[Main app] After plugin loading...\n");

    return true;
}
```

```
/**
 * Loads minimal_plugin library
 * and calls OnEntry() function in it.
 */
void Minimal::LoadPlugin()
{
    /* Load minimal_plugin dynamic library */
    wxPluginLibrary *plglib =
        wxPluginManager::LoadLibrary(wxT("./minimal_plugin"));

    /* If loading fails, abort here. */
    if (!plglib)
    {
        // No need to assert, wx will do that few lines above
        return;
    }

    /* Create a dynamic object from
     * the plugin library by the class name
     */
    Plugin *plg = (Plugin *) wxCreateDynamicObject(
        wxT("Minimal_Plugin"));

    /* Assert if dynamic object creation
     * fails for some reason
     */
    wxASSERT_MSG(plg, \
        wxT("Unable to create dynamic object from lib."));

    /* Call OnEntry() function in the lib and
     * pass main frame pointer to it
     */
    plg->OnEntry(mainframe);

    /* Lets say we'r done with the plugin, delete it now. */
    delete plg;

    /* And don't forget to unload the library also */
    wxPluginManager::UnloadLibrary(wxT("./minimal_plugin"));
}
```

Und nun noch das passende Makefile für die Hostanwendung und nachfolgend für das Plugin.

```
WXCONFIG = wx-config

# Debug compilation flags
DEBUG_FLAGS = -pipe -l. -c -Wall -g -gdb

# Don't modify these - automatically
# generated from WXCONFIG
CXX = $(shell $(WXCONFIG) --cxx)
CXX_FLAGS = $(shell $(WXCONFIG) --cxxflags)
LIBS = $(shell $(WXCONFIG) --libs)

# Final binary name
PROGRAM = minimal

#####
# Objects list. #
#####
# If new files are added to source tree,
# add them here also.
#
OBJECTS = minimal.o

# File suffixes
.SUFFIXES: .o .cpp

#####
# Compilation #
#####

# CPP Files
.cpp.o :
    $(CXX) $(CPP_FLAGS) $(DEBUG_FLAGS) $(CXX_FLAGS) -o $@ $<

# Linking
minimal: $(OBJECTS)
    $(CXX) -o $(PROGRAM) $(OBJECTS) $(LIBS)

# All builds program with default settings.
all: $(PROGRAM)
```

```

# Cleanup command
clean:
    rm -f $(OBJECTS) $(PROGRAM)

WXCONFIG = wx-config

# Debug compilation flags
DEBUG_FLAGS = -pipe -I. -c -Wall -g -gdb

# Don't modify these - automatically
# generated from WXCONFIG
CXX = $(CCACHE) $(shell $(WXCONFIG) --cxx)
CXX_FLAGS = $(shell $(WXCONFIG) --cxxflags)
LIBS = $(shell $(WXCONFIG) --libs) -shared

# Final binary name
PROGRAM = minimal_plugin

# List of objects to build
OBJECTS = minimal_plugin.o

# File suffixes
.SUFFIXES: .o .cpp

# CPP Files
.cpp.o :
    $(CXX) $(DEBUG_FLAGS) $(CXX_FLAGS) -o $@ $<

# Linking
minimal_plugin: $(OBJECTS)
    $(CXX) -o $(PROGRAM).so $(OBJECTS) $(LIBS)

# All builds program with default settings.
all: $(PROGRAM)

# Cleanup command
clean:
    rm -f $(OBJECTS) $(PROGRAM)

```

Wie zu erkennen ist, spart man sich die Details, wie genau ein Plugin auf dem entsprechenden System geladen wird. Ob es eine DLL ist oder ein `.so` Objekt klärt `wxWidgets` und auch so erledigt `wxWidgets` das konkrete Laden und Instanzieren der Plugin-Klasse im Hintergrund. Die Verwendung von Plugins (ähnlich wie auch von Bibliotheken) gestattet es relativ einfach und schnell Änderungen an Teilen des Programms durchzuführen. Wenn z. B. ein konkreter Algorithmus als Plugin implementiert ist, kann dieser (sofern man an der Schnittstelle nichts ändert) auch zur Laufzeit getauscht werden. Bei größeren Projekten spart man sich das Linken der Gesamtanwendung gegen das geänderte Modul, da die Änderung unmittelbar zur Laufzeit realisiert werden kann (entladen und neuladen). Ein weiterer Vorteil ist, daß die Verwendung von Plugins recht strikte Anforderungen (*design by contract*) an die Schnittstelle legt, denn diese sollte man nicht ständig ändern. Dadurch wird der Programmierer weit mehr als im üblichen Sinne angehalten, strukturiert zu programmieren und nicht, weil es eben gerade bequemer ist die Schnittstelle abzuändern, oder weitere Modulabhängigkeiten zu verursachen. Ein etwas umfangreicheres praktisches Beispiel ist das OCR-System *Mole* [2], dieses realisiert alle wichtigen Teilalgorithmen als Plugins, so daß insbesondere auch das Experimentieren sehr vereinfacht wird.

Zusammenfassung

`wxWidget-Plugins` sind eine recht einfache Technik Programme zu strukturieren und dynamisch zur Laufzeit um neue Funktionen zu erweitern oder unbenötigte Funktionalität zu entfernen. Sie sind einfach in bestehende `wxWidget` Projekte zu integrieren und gestatten die Erstellung von plattformunabhängigen Pluginsystemen. Der Benutzer erhält die Möglichkeit vorhandene Programme (die eine `wxWidget-Pluginschnittstelle` besitzen) auf einfache Art seinen Bedürfnissen anzupassen und um neue (den Schnittstellen entsprechende) Funktionalität zu erweitern. Der Programmierer hat nur einen geringen Mehraufwand, erhält aber ein höheres Maß an natürlicher Strukturierung bei gleichzeitiger flexibler Erweiterung der geschriebenen Software. Auch die Wiederverwendung von einzelnen Softwareteilen wird auf dem Niveau von `shared objects` effizient und plattformunabhängig einsetzbar. Einfach mal ausprobieren und Spaß haben.

Referenzen

- [1] <http://www.wxwindows.org/>
- [2] <http://gaos.org/cgi-bin/cvsweb/mole/?cvsroot=schleif>

▼ Kryptographische Skatrunde

HEIKO STAMER: Die freie Bibliothek `libTMCG` und eine Referenzimplementierung für Skat

Dieser Artikel beschreibt die Grundlagen und die technische Umsetzung eines mentalen Kartenspiels. Kryptographische Protokolle sichern dabei die Fairness zwischen den Teilnehmern. Nach einer kurzen Einführung lernen wir zwei Möglichkeiten der kryptographischen Kartenkodierung kennen und vergleichen sie hinsichtlich ihrer Komplexität. Danach folgt eine ausführliche Behandlung der freien Bibliothek `libTMCG`, welche uns einfache Datenstrukturen und die notwendigen Protokolle für sichere mentale Kartenspiele zur Verfügung stellt. Schließlich wird als Anwendung eine Implementierung des Skatspiels diskutiert.

Sichere Mehrpersonenspiele erfordern in Netzwerken häufig einen zentralen Server oder Schiedsrichter, welcher für die faire Verteilung des Spielmaterials sorgt und die Einhaltung der jeweils geltenden Regeln überwacht. Die Teilnehmer vertrauen hierbei praktisch auf dessen Unabhängigkeit, die jedoch nicht immer gegeben ist.¹ Zudem ist ein solcher Schiedsrichter in manchen Situationen einfach nicht verfügbar. Deshalb gibt es seit vielen Jahren kryptographische Protokolle, welche ohne zentralen Vertrauenspunkt (Trusted Third Party) die Fairness (in gewissen Grenzen) sicherstellen können. Deren Funktionsweise beruht auf sogenannten Zero-Knowledge Beweisen², welche – grob gesprochen –

¹Jeder rational handelnde Eigentümer eines Online-Spielkasinos möchte sein Geschäft natürlich profitabel und risikolos betreiben.

²Literaturempfehlung: Die Autoren der Arbeit *How to Explain Zero-Knowledge Protocols to Your Children* [QGB89] erklären das dahinterstehende Konzept sehr unterhaltsam und theoriearm.

bestimmte Eigenschaften eines Objektes zeigen können, ohne daß hierbei Wissen³ über den Gegenstand selbst offengelegt wird. Diese Beweise sind probabilistischer Natur, d. h. sie haben eine gewisse Fehler- bzw. Betrugswahrscheinlichkeit $p \leq 1/2$, die aber durch sequentielle Wiederholung schnell in den vernachlässigbaren Bereich gerückt werden kann. Ein erstaunliches Resultat auf diesem Gebiet ist die Tatsache [GMW87], daß für alle Sprachen $L \in \mathcal{NP}$ die Zugehörigkeit beliebiger Elemente zu L ohne deren Preisgabe beweisbar ist, falls man die Existenz sicherer Verschlüsselungsfunktionen voraussetzt.

Das mächtige Konzept der *Zero-Knowledge Beweise* hat vielfältige Anwendungsmöglichkeiten: Identifizierungsprotokolle, Gruppensignaturschemata (z. B. Direct Anonymous Attestation [BCC04, 2] im TPM-Standard v1.2 der Trusted Computing Group), verifizierbare Verschlüsselung und Geheimnisteilung (VSS), elektronische Wahlen, digitales Geld, faire Online-Auktionen und sichere Mehrparteien-Berechnung. Zur letzten Kategorie zählt auch die Gewährleistung der Fairness bei virtuellen Kartenspielen, mit welcher wir uns in diesem Artikel näher beschäftigen wollen. Dem Konzept sind natürlich auch Grenzen gesetzt: Verlust oder mutwillige Weitergabe vertraulicher Informationen durch den Betrachter bzw. Eigentümer kann kein Protokoll verhindern!

Einleitung

Prinzipiell können *mentale Kartenspiele* als Interaktion zwischen (mindestens zwei) konkurrierenden Parteien angesehen werden, wobei keinerlei physikalisches Spielmaterial (also Spielkarten) zum Einsatz kommt. Wer auf diese seltsam erscheinende Idee kam, ist nicht eindeutig: In einer der ersten Arbeiten [RSA79] zu diesem Thema, wird die Fragestellung „Is it possible to play a fair game of Mental Poker?“ auf Robert W. Floyd zurückgeführt. Allerdings soll Werner Heisenberg in seinen Memoiren erwähnt haben, daß sich bereits Niels Bohr während eines langweiligen Ski-Urlaubs mit mentalen Kartenspielen befaßte, aber zu keinem brauchbaren Ergebnis kam. [1]

Das einfachste Beispiel eines mentalen Spiels wurde 1981 in [Blu81] betrachtet: Zwei Parteien möchten einen Münzwurf übers Telefon durchführen. Beide wollen das Ergebnis zu ihren Gunsten beeinflussen und es gibt keine vertrauenswürdige dritte Person. Die von Manuel Blum vorgestellte Lösung verwendet erstmals eine Kodierung

³Man beachte den Bedeutungsunterschied der Wörter „Wissen“ und „Information“, wobei wir für letzteres die Definition aus der klassischen Informationstheorie heranziehen. Beispiel: Es sei die Zahl $2^{2^{2^3}} - 1$ gegeben. Ihre Primfaktoren stellen keine zusätzliche Information dar, weil die Zerlegung eindeutig ist und alle Faktoren deshalb implizit in der Zahl enthalten sind. Jedoch kann ihre Kenntnis einen Wissensvorsprung bedeuten, weil andere Instanzen aufgrund der Größe nicht in der Lage sind, die Faktorisierung sofort zu bestimmen.

auf Basis quadratischer Reste. Aufgrund ihrer homomorphen Eigenschaft wurde diese Kodierung dann lange Zeit als *state-of-the-art* in vielen Protokollen eingesetzt.

Kurz darauf stellten Shafi Goldwasser und Silvio Micali eine Variante [GM82] für das Kartenspiel „Poker“ vor, welche das Offenlegen aller Karten nach dem Spielende erfordert, um stattgefundenen Betrugsversuche zu entdecken. Gerade aber für Poker ist diese Lösung unbefriedigend, weil hierbei die Strategie der Spieler ersichtlich wird. Weitere Nachteile sind die enorme Komplexität und die ineffiziente Verwendung von Primzahlen.

Die erste vollständige Lösung für Poker wurde 1987 von Claude Crépeau [Cré87] veröffentlicht. Die Idee basiert ebenfalls auf der Kodierung mit quadratischen Resten, nutzt aber das ANDOS-Schema (*all-or-nothing disclosure of secrets*) [BCR87]. Insbesondere kann damit erreicht werden, daß keine verdeckte Karte bei Spielende geöffnet werden muß und so die Spielstrategie der Teilnehmer geheim bleibt. Allerdings hatte auch diese Lösung noch Nachteile: die sequentielle Abhängigkeit einzelner Teilprotokolle, die Voraussetzung eines eindeutigen Kartentyps und das Fehlen wichtiger Karten- und Stapeloperationen.

Schließlich hat Christian Schindelhauer [Sch98] diesen Ansatz 1998 aufgegriffen und an vielen Stellen erweitert bzw. verallgemeinert. Sein Werkzeugkasten bietet deshalb erstmals die Möglichkeit, fast jedes beliebige Kartenspiel in sicherer Art und Weise elektronisch umzusetzen, ohne dabei auf die Verfügbarkeit eines vertrauenswürdigen Vermittlers angewiesen zu sein. Natürlich kann auch diese Lösung nicht verhindern, daß unehrliche Spieler ihre (eigentlich verdeckt zu haltenden) Karten insgeheim einer Koalition preisgeben und daraus Vorteile ziehen. Es wird lediglich garantiert, daß die privaten Karten jedes Spielers nicht *gegen seinen Willen* von anderen Teilnehmern gesehen oder verändert werden können.

Zwei aktuelle Vorschläge für Kartenkodierungen stammen von Adam Barnett und Nigel Smart [BSm03]. Die Autoren verallgemeinern zuerst das Modell der Maskierung virtueller Spielkarten, indem sie die abstrakte kryptographische Primitive VTMF (*Verifiable k -out-of- k Threshold Masking Function*) einführen. Dann werden die Karten- und Stapeloperationen aus [Sch98] im Kontext dieser Notation beschrieben. Schließlich stellen Barnett und Smart noch zwei konkrete Implementierungen solcher VTMFs vor. Beide Ansätze basieren nicht mehr auf quadratischen Resten und ermöglichen deshalb eine sehr effiziente Kodierung der Spielkarten.

Kryptographische Grundlagen

Zuerst ein paar allgemeine Definitionen, stellenweise entnommen aus WIKIPEDIA [14, 15, 16]: In der Gruppentheorie ist eine *zyklische Gruppe* eine Gruppe, die von einem einzelnen Element erzeugt wird. Sie besteht aus allen Potenzen dieses Erzeugers g , d. h. $\langle g \rangle := \{g^n \mid n \in \mathbb{Z}\}$.

Eine beliebige Gruppe G ist also zyklisch, wenn sie ein Element g enthält (*Erzeuger der Gruppe*), so daß jedes Element aus G eine Potenz von g ist. Zyklische Gruppen sind die einfachsten Gruppen und können vollständig klassifiziert werden: Für jede natürliche Zahl n gibt es eine zyklische Gruppe C_n mit genau n Elementen, und es gibt die unendliche zyklische Gruppe, die additive Gruppe der ganzen Zahlen \mathbb{Z} . Jede andere zyklische Gruppe ist zu einer dieser Gruppen isomorph. Ist n eine natürliche Zahl, dann faßt man ganze Zahlen mit gleichem Rest bei Division durch n zu sogenannten *Restklassen* zusammen. Die Restklassen bilden zusammen den *Restklassenring*, der mit $\mathbb{Z}/n\mathbb{Z}$ oder \mathbb{Z}_n bezeichnet wird (sprich \mathbb{Z} modulo n). Eine Restklasse $a + n\mathbb{Z}$ mit $\text{ggT}(a, n) = 1$ heißt *prime Restklasse modulo n* . Die Gruppe der primen Restklassen modulo n heißt *prime Restklassengruppe modulo n* und wird mit $(\mathbb{Z}/n\mathbb{Z})^*$ bzw. \mathbb{Z}_n^* bezeichnet. Weil diese Struktur eine endliche abelsche Gruppe bezüglich der Multiplikation bildet, spielt sie in der Kryptographie eine bedeutende Rolle. Die *Eulersche Phi-Funktion* ist eine zahlentheoretische Funktion. Sie ordnet jeder natürlichen Zahl n die Anzahl der Zahlen $a \in \mathbb{N}, a \leq n$ zu, die zu n teilerfremd sind (also $|\{a \in [1, n] \mid \text{ggT}(a, n) = 1\}|$). Sie ist benannt nach Leonhard Euler und wird mit dem griechischen Buchstaben φ (Phi) bezeichnet. Ihr Wert entspricht der Gruppenordnung von \mathbb{Z}_n^* , d. h. $\varphi(n) = |\mathbb{Z}_n^*|$. Betrachten wir vorerst die Menge \mathbb{P} der Primzahlen: Da jede Primzahl p nur durch 1 und sich selbst teilbar ist, bleibt sie auch zu den Zahlen 1 bis $p - 1$ teilerfremd, daher gilt $\varphi(p) = p - 1$. Die Phi-Funktion ist multiplikativ für zwei Zahlen $m, n \in \mathbb{N}$ sofern diese keinen gemeinsamen Teiler außer 1 haben, d. h. $\varphi(mn) = \varphi(m)\varphi(n)$ falls $\text{ggT}(m, n) = 1$. Die Berechnung von φ für zusammengesetzte n ergibt sich aus dieser Multiplikativität: Hat unsere Zahl die kanonische Darstellung $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k}$ mit $e_i \geq 1, p_i \in \mathbb{P}$ für $i = 1, \dots, k$, dann gilt $\varphi(n) = (p_1 - 1)p_1^{e_1 - 1} \cdot (p_2 - 1)p_2^{e_2 - 1} \cdot \dots \cdot (p_k - 1)p_k^{e_k - 1}$. Die Menge der *quadratischen Reste modulo n* definiert man als $\mathbb{QR}_n := \{a \in \mathbb{Z}_n^* \mid \exists b \in \mathbb{Z}_n : b^2 \equiv a \pmod{n}\}$. Sie ist eine Untergruppe von \mathbb{Z}_n^* . Analog bezeichnet $\mathbb{NQR}_n := \mathbb{Z}_n^* \setminus \mathbb{QR}_n$ die Menge der *quadratischen Nichtreste modulo n* , welche jedoch nicht multiplikativ abgeschlossen ist. Das *Legendre-Symbol* für eine ungerade Primzahl p und ein $a \in \mathbb{Z}_p^*$ sei wie folgt definiert:

$$\left(\frac{a}{p}\right) := \begin{cases} +1 & a \in \mathbb{QR}_p \\ -1 & \text{sonst} \end{cases}$$

Das *Jacobi-Legendre-Symbol* ist die Verallgemeinerung für zusammengesetzte Zahlen $n \in \mathbb{N}$ und ein $a \in \mathbb{Z}_n^*$:

$$\left(\frac{a}{n}\right) := \begin{cases} \left(\frac{a}{p}\right) & \text{falls } n \text{ Primzahl} \\ \left(\frac{a}{p}\right) \cdot \left(\frac{a}{m}\right) & \text{falls } n = p \cdot m \text{ und } p \text{ Primzahl} \end{cases}$$

Weiterhin definieren wir (analog zu [Sch98]) die Mengen $\mathbb{Z}_n^\circ := \{a \in \mathbb{Z}_n^* \mid \left(\frac{a}{n}\right) = 1\}$ und $\mathbb{NQR}_n^\circ := \mathbb{Z}_n^\circ \cap \mathbb{NQR}_n$ (Pseudoquadrate modulo n). Falls nun n das Produkt

von genau zwei verschiedenen Primzahlpotenzen ist,⁴ dann hat die Gruppe \mathbb{QR}_n die Ordnung $|\mathbb{Z}_n^*|/4$ und auch \mathbb{NQR}_n° enthält genau $|\mathbb{Z}_n^*|/4$ Elemente. Diese Eigenschaft ermöglicht die gleichverteilte Kodierung eines Bit als Zahl $z \in \mathbb{Z}_n^\circ$, so daß das Bit genau dann gesetzt ist, wenn z in \mathbb{QR}_n liegt.

Effiziente kryptographische Protokolle verwenden oft „schwierige“ Probleme aus der Mathematik zur Umsetzung ihrer Sicherheitsziele, z. B. das Zerlegen großer natürlicher Zahlen in ihre Primfaktoren oder die Bestimmung des diskreten Logarithmus in endlichen Gruppen.⁵ Für solche *kryptographischen Annahmen* existiert kein strenger Beweis im Sinne der Komplexitätstheorie. Dennoch geht man von der praktischen Schwierigkeit⁶ folgender Probleme aus, falls die jeweiligen Sicherheitsparameter entsprechend groß gewählt sind:

Primfaktorisierung (FAKTOR) Darunter wollen wir das Problem verstehen, zu einer gegebenen positiven ganzen Zahl n ihre paarweise verschiedenen Primfaktoren p_1, p_2, \dots, p_k zu finden, so daß $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k}$ und $e_i \geq 1$ gilt. Es existiert zwar ein Algorithmus der in Polynomialzeit feststellen kann, ob eine gegebene Zahl prim ist, allerdings scheint die Zerlegung in Faktoren weitaus schwieriger. Zur Zeit haben die schnellsten Algorithmen ein subexponentielles Laufzeitverhalten in der Größe von n , so daß die Zerlegung einer 300-stelligen Zahl (bestehend aus genau zwei, vergleichbar großen Primfaktoren) momentan noch unmöglich erscheint. Aber der technische Fortschritt bleibt nie stehen . . .

Quadratisches Restproblem (QRP) Für eine Zahl $n \in \mathbb{N}$ und ein beliebiges $a \in \mathbb{Z}_n^*$ soll mit Wahrscheinlichkeit größer $1/2$ entschieden werden, ob a ein quadratischer Rest modulo n ist. Falls man die Primfaktorisierung von n kennt oder es sich sogar um eine Primzahl handelt, wird das Problem einfach. In diese Richtung existiert nämlich eine Reduktion $\text{QRP} \leq_P \text{FAKTOR}$, die in Polynomialzeit durchgeführt werden kann.

Quadratwurzeln (QWURZEL) Zu gegebenen $n \in \mathbb{N}$ und $a \in \mathbb{QR}_n$ soll ein $x \in \mathbb{Z}_n^*$ bestimmt werden, so daß $x^2 \equiv a \pmod{n}$ gilt. Falls n eine Primzahl ist, gibt es einen Polynomialzeitalgorithmus, der die gesuchte Quadratwurzel x berechnet. Für ein zusammengesetztes n kann man zeigen, daß diese Fragestellung äquivalent zur Primfaktorisierung ist, d. h. es existieren Reduktionen in beide Richtungen $\text{QWURZEL} \equiv_P \text{FAKTOR}$.

Diskreter Logarithmus (DLP) Sei g ein Erzeuger der endlichen zyklischen Gruppe $\langle g \rangle$. Zu einem gegebenen Wert $y \in \langle g \rangle$ soll die kleinste positive ganze Zahl x (diskreter Logarithmus zur Basis g) bestimmt werden, so

⁴ $n = p_1^{e_1} \cdot p_2^{e_2}$ für ungerade Primzahlen $p_1 \neq p_2$ und $e_i \geq 1$

⁵Zumindest glaubt man, daß solche Fragen äußerst schwierig sind, weil bis heute kein schneller Algorithmus für ihre Lösung gefunden wurde. Natürlich kann ein falscher Glaube auch aufs „Glatteis“ führen, wie sich z. B. vor etwa zwei Jahren bei der positiven Beantwortung der lange untersuchten Fragestellung $\text{PRIMES} \in_P \mathcal{P}$ gezeigt hat. Zwar waren in diesem Fall die Auswirkungen für die Kryptographie eher gering, jedoch sollte das nicht dazu verleiten, immer weitergehende Annahmen zutreffen.

⁶„Alienhardware“ (Zitat von Rüdiger Weis) ausgenommen

daß $y = g^x$ gilt. In Gruppen mit enorm vielen Elementen (große Gruppenordnung) kann zwar g^x „effizient“ berechnet werden, die Umkehroperation scheint jedoch sehr schwierig zu sein.

Computational Diffie-Hellman (CDH) Zu gegebenen Werten $g^a \in \langle g \rangle, g^b \in \langle g \rangle$ (die Exponenten a, b sind dabei unbekannt) soll ein $z \in \langle g \rangle$ gefunden werden, so daß $z = g^{ab}$ gilt. Dieses Problem wird leicht, wenn man diskrete Logarithmen in $\langle g \rangle$ berechnen kann, d. h. $\text{CDH} \leq_{\mathcal{P}} \text{DLP}$. Für die Umkehrung ist eine solche Polynomialzeitreduktion nicht bekannt.

Decisional Diffie-Hellman (DDH) Diese stärkere Annahme besagt, daß es bei Kenntnis von $g^a \in \langle g \rangle$ und $g^b \in \langle g \rangle$ (die Exponenten a, b sind wieder unbekannt) schwierig ist, den Wert $g^{ab} \in \langle g \rangle$ von einem zufälligen Element $g^c \in \langle g \rangle$ zu unterscheiden, sofern a, b, c zufällig und uniform im Intervall $[1, |\langle g \rangle|]$ gewählt sind. Mit anderen Worten: Die Verteilungen $\{g^a, g^b, g^{ab}\}$ und $\{g^a, g^b, g^c\}$ sind effizient nicht unterscheidbar. Wie man sofort sieht, kann eine Polynomialzeitreduktion $\text{DDH} \leq_{\mathcal{P}} \text{CDH}$ zum vorherigen Problem konstruiert werden. Doch wie verhält sich die Fragestellung in anderer Richtung? Es gibt erstaunlicherweise einige Gruppen, in denen DDH trivial ist, aber das CDH-Problem noch schwer zu sein scheint. Das sind z. B. Gruppen, deren Ordnung durch kleine Primzahlen teilbar ist, also insbesondere \mathbb{Z}_p^* . Ergo muß bei der Auswahl einer DDH-schweren Gruppe mit besonderer Vorsicht vorgegangen werden. Dan Boneh [Bon98] nennt einige Familien von Gruppen, die hierfür in Frage kommen.

Random-Oracle Modell (ROM [FS86, BR93, 3]) Ein *Zufallsorakel* ist eine Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}^n$ (für ein festes n), wobei der Funktionswert $f(x)$ für jede Eingabe $x \in \{0, 1\}^*$ *echt zufällig* ist. Die Funktion ist öffentlich, d. h. man stellt keinerlei Anforderungen hinsichtlich der Vertraulichkeit. Insbesondere ist es also akzeptabel, wenn jedermann (auch ein potentieller Gegner) auf beliebige Werte von f zugreifen kann. Leider sind solche Orakel rein theoretische Konstruktionen und können nicht implementiert werden. Das dahinterstehende Konzept ist allerdings äußerst nützlich, weil es viele wünschenswerte Eigenschaften⁷ besitzt und deshalb vereinfachte Sicherheitsbeweise ermöglicht. Approximativ kann f durch eine kryptographische Hashfunktion⁸ implementiert werden. Es gibt viele Verfahren deren Sicherheit im *Random-Oracle Modell* bewiesen werden kann, also unter der Annahme, daß sich die verwendete Hashfunktion wie ein Zufallsorakel verhält.

Sichere mentale Kartenspiele

Wir wollen nun die konkrete Realisierung sicherer mentaler Kartenspiele betrachten. Allgemein werden folgende Anforderungen [Sch98, Gut00] gestellt:

⁷Beispielsweise erhält man durch Auswertung an beliebigen Stellen keine Information über den Funktionswert an noch nicht ausgewerteten Stellen. Weiterhin ist ein Zufallsorakel natürlich *kollisionsresistent*, d. h. es ist (bei großen n) nicht effizient möglich, zwei Eingaben $x, y \in \{0, 1\}^*$ zu finden, für die $f(x) = f(y)$ gilt.

⁸z. B. SHA-1, RIPEMD-160 [5], ...

1. Es sollen beliebig viele Mitspieler, Karten und Kartentypen möglich sein.
2. Die Datenstruktur für die Spielkarten erlaubt eine eindeutige Identifizierung jeder Karte (Rückseite). Der Typ (Vorderseite) einer verdeckten Karte bleibt jedoch verborgen, sofern er nicht durch eine gewollte Operation ersichtlich wird.
3. Stapel dienen als Behälter für Karten.
4. Die Karten- und Stapeloperationen garantieren ein breites Einsatzgebiet für diverse Kartenspiele. Beispielsweise sollen für Karten *das Erzeugen*, *das Verdecken (Maskieren)*, *das Offenlegen*, *das vom Stapel nehmen*, *das in den Stapel (geheim) einfügen*, *das zufällige Erzeugen* und für Stapel *das Erzeugen*, *das Vereinigen*, *das Teilen*, *das Mischen*, *das Abheben*, *das Vergleichen* zur Verfügung stehen.
5. Ein Teilnehmer kann den Typ einer von allen Mitspielern verdeckten Karte nur erfahren, falls auch alle damit einverstanden sind.
6. Keine Koalition von Teilnehmern kann den Typ privater Karten gegen den Willen eines einzelnen Spielers (Inhaber) bestimmen.⁹
7. Die Spielstrategie soll geheim bleiben, d. h. es ist nicht notwendig, verdeckt oder privat gebliebene Karten bei Spielende zu öffnen.
8. Der Berechnungs- und Kommunikationsaufwand sollte sich in vernünftigen Grenzen bewegen.

Unser *Angriffsmodell* sei wie folgt fixiert: Alle Spieler verhalten sich protokollkonform¹⁰ (*semi-honest*), sind aber u. U. neugierig (bzgl. der Geheimnisse anderer Teilnehmer) oder arbeiten in Koalition zusammen, um sich einen Vorteil im Spiel zu verschaffen. Dieses Szenario wird oft auch als *honest-but-curious* Modell bezeichnet.

Die im nächsten Abschnitt behandelte Bibliothek `libTMCG` basiert hauptsächlich auf Christian Schindelhauers *Toolbox for Mental Card Games*. Details der Zero-Knowledge Beweise können dem Technischen Report [Sch98] der Universität Lübeck entnommen werden. Wir gehen hier nur kurz auf Abweichungen bzw. Ergänzungen [BSm03] zur Kartenkodierung ein.

Modifikationen: Der Korrektheitsbeweis des öffentlichen Schlüssels ist durch die reduzierte Version¹¹ des Verfahrens von Gennaro, Micciancio und Rabin [GMR98] realisiert. Um

⁹triviale Schlußfolgerungen sind ausgenommen

¹⁰Wir stellen also keine stärkeren Schutzziele, wie *Robustheit* oder *Verfügbarkeit*, an die Implementierung, da diese Forderungen in einer asynchronen Umgebung (Netzwerk) kaum erfüllbar sind.

¹¹nur Stufe 1 (Square Free) und Stufe 2 (Prime Power Product)

deren Voraussetzungen zu erfüllen, mußte auch die Schlüsselstruktur geringfügig angepaßt werden: Wir verwenden sogenannte *sichere Primzahlen*¹² p_i und q_i , die wesentlich dünner in \mathbb{N} verteilt sind, aber bessere Sicherheitseigenschaften haben. Weiterhin erfüllen p_i, q_i die Kongruenzen

$$p_i \equiv 3 \pmod{4}, \quad q_i \equiv 3 \pmod{4},$$

um eine schnelle Berechnung von Quadratwurzeln modulo $p_i \cdot q_i$ zu ermöglichen und

$$p_i \not\equiv 1 \pmod{8}, \quad p_i \not\equiv q_i \pmod{8},$$

um den Voraussetzungen von [GMR98] zu genügen. Dadurch wird natürlich die Schlüsselerzeugung langsamer, aber das Protokoll insgesamt robuster. Außerdem wurde die Verifikation des Schlüssels¹³ mit Hilfe der *Fiat-Shamir Heuristik* [FS86, BR93] in einen nicht-interaktiven Zero-Knowledge Beweis [BFM88, BSG91] umgewandelt. Hier setzen wir also neben der kryptographischen Annahme, daß FAKTOR schwer ist, auch noch voraus, daß sich die Hashfunktion¹⁴ g [BR93] wie ein Zufallsorakel (ROM) verhält.

Die erzeugten Schlüssel dienen (neben ihrer Verwendung in den Protokollen der Toolbox [Sch98]) parallel für asymmetrische Kryptographie.¹⁵ Dabei wird das Verfahren von Rabin [Rab79, 4] sowohl für Signatur (PRab [BR96]) als auch für Verschlüsselung (SAEP [Bon01]) eingesetzt.

Kartenkodierungen: Jeder Spieler i besitzt einen Schlüssel bestehend aus geheimen (p_i, q_i) und öffentlichen (m_i, y_i) Teil. Hierbei ist $m_i = p_i \cdot q_i$ das Produkt zweier großer Primzahlen (z. B. 1024 Bit) und $y_i \in \text{NQR}_{m_i}^\circ$ ist zufällig gewählt. Eine verdeckte Karte in einem Spiel mit k Teilnehmern und M verschiedenen Kartentypen¹⁶ kodiert in $w = \lceil \log_2 M \rceil$ Bits wird durch die Matrix

$$Z = \begin{pmatrix} z_{1,1} & \cdots & z_{1,w} \\ \vdots & \ddots & \vdots \\ z_{k,1} & \cdots & z_{k,w} \end{pmatrix}$$

beschrieben, wobei die $z_{i,j} \in \mathbb{Z}_{m_i}^\circ$ sind. Die Matrixelemente enthalten jeweils ein Bit

$$b_{i,j} = \begin{cases} 0 & z_{i,j} \in \text{QR}_{m_i} \\ 1 & \text{sonst} \end{cases}$$

verteilte Information über den Typ τ der Karte. Dieser Wert $\tau \in [0, M-1]$ kann durch die Formel

$$\tau = \sum_{j=1}^w 2^{j-1} \cdot \bigoplus_{i=1}^k b_{i,j}$$

¹²Eine Primzahl p heißt *sicher*, wenn auch $(p-1)/2$ prim ist.

¹³Sowohl der Beweis, daß m_i ein Produkt von genau zwei Primfaktoren ist, als auch die Überprüfung der Bedingung $y \in \text{NQR}_{m_i}^\circ$.

¹⁴siehe libTMCG/mpz_shash.cc für deren Implementierung

¹⁵Diese Designentscheidung ist diskussionswürdig: Prinzipiell sollten kryptographische Schlüssel nur für den vorgesehenen Zweck benutzt werden. Ein paralleler Einsatz für andere Aufgaben kann unerwünschte Seiteneffekte haben, welche die Sicherheitsmaßnahmen des eigentlichen Hauptanwendungsgebiets unterlaufen.

¹⁶Beim Skatspiel haben wir z. B. $M = 32$ unterschiedliche Typen.

berechnet werden, sofern alle Teilnehmer zustimmen und die Werte $b_{i,j}$ aus ihrer Zeile der Matrix offenlegen.¹⁷ Unter der Annahme, daß die Bestimmung der Quadratresteigenschaft (QRP) modulo m_i schwer ist, stellt diese Kodierung eine für unsere Zwecke sichere Datenstruktur dar. Mit ihrer Hilfe können Protokolle [Sch98] konstruiert werden, die folgende Karten- und Stapeloperationen ermöglichen:

- ✓ Erzeugung einer offenen oder verdeckten Karte mit festem Typ $\tau \in [0, M-1]$
- ✓ Umwandlung (Maskierung) einer offenen in eine verdeckte Karte
- ✓ Erzeugung einer verdeckten Karte mit zufälligem Typ τ (uniform aus $[0, M-1]$)
- ✓ Aufnehmen und Offenlegen einer verdeckten Karte
- ✓ Stapeln von offenen oder verdeckten Karten
- ✓ Mischen¹⁸ oder Abheben¹⁹ eines Stapels
- ✗ Mengenvergleiche (Inklusion, Schnitt) von verdeckten mit offenen Stapeln
- ✗ Geheimes Einfügen einzelner Karten in einen verdeckten Stapel
- ✗ Zeitnahe Regelkontrolle direkt beim Ausspiel

Die Markierung gibt jeweils an, ob dieser Punkt bereits in der Bibliothek libTMCG implementiert ist (✓) oder noch nicht (✗). Zur Verifikation der Korrektheit jeder Operation verwendet die Toolbox effiziente Zero-Knowledge Beweise mit maximaler Kommunikationskomplexität der Ordnung $\mathcal{O}(k^2)$. Die Betrugswahrscheinlichkeit²⁰ kann bei t -maliger Wiederholung der Beweise auf $p \leq 2^{-t}$ beschränkt werden. Durch Variation dieses Parameters kann man einen Kompromiß zwischen Sicherheit, Laufzeit und Übertragungsvolumen erreichen.

Dennoch ist die Datenstruktur aus [Sch98] sehr exzessiv im Speicherplatzbedarf und folglich auch im Übertragungsvolumen. Das trifft insbesondere auf Spiele mit großer Teilnehmer- oder Kartentypenzahl zu, weil sich die Matrixkodierung linear in k und logarithmisch in M verhält.

Zur Verbesserung dieser Situation wurde kürzlich eine VTMF-Implementierung [BSm03] vorgestellt. Ihre Sicherheit beruht auf der kryptographischen Annahme, daß die Maskierung (ähnlich dem ElGamal-Verfahren) in einer zyklischen Gruppe $G := \langle g \rangle$ stattfindet, wo das DDH-Problem schwer ist. Für G wird vorerst die in [Bon98] erwähnte Gruppe der quadratischen Reste modulo p verwendet, wobei p hier eine sichere Primzahl sein muß. Weiterhin haben wir bei der Gruppenkonstruktion aus Effizienzgründen [vOW96, RS00] $p \equiv 7 \pmod{8}$ gewählt, so daß $g = 2$ ein Erzeuger von QR_p ist. Dadurch kann bei der modularen Exponentiation erhebliche Rechenzeit gespart werden. Zur eindeutigen Kodierung der Kartentypen dienen uns die M kleinsten Quadratreste modulo p beginnend bei 1. Weil diese Zahlen äußerst kurz sind, ist eine Benutzung eingeschränkter Exponenten [vOW96] zwecks Laufzeitverbesserung u. U. als kritisch [BJN00] anzusehen,

¹⁷Die Korrektheit wird mit Zero-Knowledge Beweisen gezeigt.

¹⁸beliebige Permutation

¹⁹zyklische Verschiebung

²⁰in unserem Angriffsmodell (*honest-but-curious*) und unter den getroffenen kryptographischen Annahmen

	Kartenkodierung nach		
	[Sch98]	[BSm03]	[BSm03]
kryptograph. Annahme	QRP	DDH	EC-DDH
Bitkomplexität für eine Karte (allgemein)	$k \cdot \lceil \log_2 M \rceil \cdot \ell$	2ℓ	2ℓ
Bitkomplexität für eine Karte beim Skatspiel ($k = 3, M = 32, \ell \in \{1024, 160\}$)	15360	2048	322
libTMCG Unterstützung	✓	✓	✗

Tabelle 1: Vergleich der Kartenkodierungen

weshalb momentan zufällige Werte aus dem gesamten Restklassenring $\mathbb{Z}_{(p-1)/2}$ zum Einsatz kommen. Der Korrektheitsbeweis der Maskierung erfolgt über einen nicht-interaktiven *Zero-Knowledge Proof of Knowledge*, welcher in diesem Fall die Gleichheit zweier diskreter Logarithmen zu unterschiedlichen Basen zeigt. Die in der Originalarbeit [BSm03] angegebene Konstruktion nach Chaum und Pedersen [CP92] wurde durch eine effizientere Variante [CS97] ersetzt. Die Vermeidung der Interaktivität bedingt auch hier die zusätzliche Voraussetzung, daß sich die Hashfunktion g [BR93] wie ein Zufallsorakel (ROM) verhält.

Ein Vorteil der resultierenden Datenstruktur ist ihre Unabhängigkeit von k und M (siehe Vergleich in Tabelle 1). Trotzdem hat in beiden Kodierungsschemata auch der Sicherheitsparameter ℓ (zugrundeliegende kryptographische Annahme) einen großen Einfluß. Durch Kryptographie über elliptischen Kurven (ECC)²¹ besteht auch hier noch erhebliches Potential zur Reduzierung des Übertragungsvolumens.

Bibliothek libTMCG

Die freie²² C++-Bibliothek libTMCG implementiert die Konzepte der vorangegangenen Abschnitte und stellt damit eine ideale Plattform für sichere mentale Kartenspiele dar. Sie ist momentan noch kein eigenständiges Projekt, wird aber demnächst aus SecureSkat [12] ausgegliedert.

Unsere Bibliothek hängt von zwei freien Bibliotheken der GNU-Familie ab, d. h. sie nutzt deren Funktionalität:

libgmp [6] stellt uns alle notwendigen Operationen für die Arithmetik mit beliebig langen Zahlen bereit. Auch komplizierte Funktionen, wie das Jacobi-Legendre-Symbol, sind vorhanden und zudem effizient umgesetzt. Wir benötigen eine Version ≥ 4.1 .

libgcrypt [7] bietet uns kryptographische Primitiven, von denen wir insbesondere die Hashfunktion RIPEMD-160 [5] und einige Routinen zur Erzeugung des sicheren Pseudozufalls verwenden. Diese Bibliothek benötigen wir in einer Version $\geq 1.2.0$.

²¹Beispielsweise könnte G als Gruppe der Punkte einer elliptischen Kurve über dem Körper \mathbb{F}_p gewählt werden, sofern die Gruppenordnung selbst wieder prim ist. Das DDH-Problem scheint dann ebenfalls schwer zu sein. [Bon98]

²²GNU General Public License, Version 2, June 1991

Beide Softwareprojekte wurden für viele Rechnerarchitekturen optimiert und sind unter diversen Betriebssystemen lauffähig. Sie stehen damit der Portierung auf andere Plattformen nicht entgegen.

Hauptbestandteil der Kartenspielbibliothek libTMCG sind die Klassen SchindelhauerTMCG [Sch98] und BarnettSmartVTMF_dlog [BSm03], welche die Protokolle der erwähnten Arbeiten implementieren. Außerdem gibt es folgende vier wichtige Datenstrukturen:

TMCG_Card repräsentiert eine Spielkarte gemäß der Kodierung von Schindelhauer [Sch98].

TMCG_CardSecret stellt das zugehörige Geheimnis dar, welches zum Verdecken (Maskieren) einer Karte in diesem Schema notwendig ist.

VTMF_Card repräsentiert eine Spielkarte in der verbesserten Kodierung von Barnett und Smart [BSm03].

VTMF_CardSecret stellt das zugehörige Geheimnis dar, welches zum Maskieren in diesem Schema notwendig ist.

Die Datentypen für Stapel sind generisch (über Klassentemplates) umgesetzt, so daß sie für beide Karten- bzw. Geheimnisarten funktionieren:

TMCG_OpenStack<CardType> repräsentiert einen Stapel (offener) Karten der Datenstruktur CardType. Dieser Container enthält Paare (pair<size_t, CardType>) bei denen die erste Komponente den Typ der korrespondierenden Karte (zweite Komponente) darstellt.

TMCG_Stack<CardType> repräsentiert einen Stapel (verdeckter) Karten der Datenstruktur CardType. Dieser Container enthält die Karten und nur implizit deren Typ.

TMCG_StackSecret<CardType> repräsentiert ein Stapelgeheimnis, welches bei einigen Stapeloperationen (z. B. Mischen, Abheben) notwendig ist. Dieser Container enthält Paare (pair<size_t, CardSecretType>) bei denen die erste Komponente den Permutationsindex des korrespondierenden Geheimnis (zweite Komponente) angibt.

Weiterhin gibt es noch drei Schlüsselstrukturen, die allerdings nur für die ursprüngliche Kodierung [Sch98] oder asymmetrische Kryptographie (Verschlüsselung oder Signatur nach dem Rabin-Verfahren) benötigt werden:

TMCG_SecretKey repräsentiert einen geheimen Schlüssel. Der persistente Inhalt besteht aus einem Namen, einer Email-Adresse, dem Tupel (p_i, q_i, m_i, y_i) , einem nicht-interaktiven Zero-Knowledge Beweis, daß $m_i = p_i \cdot q_i \wedge y_i \in \text{NQR}_{m_i}^o$ gilt, und der Signatur für diese Daten.

TMCG_PublicKey steht für einen öffentlichen Schlüssel. Der Inhalt ist wie beim geheimen Schlüssel aufgebaut, mit einer Ausnahme: Die Primfaktoren p_i, q_i sind natürlich nicht im Tupel enthalten.

TMCG_PublicKeyRing repräsentiert ein dynamisches Feld (Container) der öffentlichen Schlüssel aller Mitspieler, d. h. $((\dots, m_1, y_1), (\dots, m_2, y_2), \dots, (\dots, m_k, y_k))$.

Die Verwendung der Klassen und Datenstrukturen werden wir nun an einem kleinen Beispiel (siehe Quelltexte ab den Seiten 19 und 25) illustrieren:

Bob und seine Ex-Freundin Alice wollen das bekannte Kartenspiel *Schwarzer Peter*²³ spielen. Der Gewinner soll den gemeinsam gekauften japanischen Kleinwagen der Marke *Reisschüssel* erhalten – es ist also davon auszugehen, daß beide das Spiel (heimlich) zu ihren Gunsten beeinflussen möchten. Alice und Bob wohnen mittlerweile hunderte Kilometer entfernt und können nur über das weltweite Datennetz kommunizieren. Außerdem ist keine Person bekannt, die beiden hinreichend vertrauenswürdig erscheint, in dieser delikaten Angelegenheit als Vermittler zu agieren. Zum Glück haben sie in einem Artikel der Zeitschrift „Offene Systeme“ von einer freien Bibliothek gelesen, die genau für solche Aufgaben entwickelt wurde.

Unser Beispiel setzt die effizientere Kartenkodierung nach [BSm03] ein, d. h. wir verwenden die Strukturen `VTMF_Card`, `VTMF_CardSecret` und die Klasse `BarnettSmartVTMF_dlog`. Alle notwendigen Datentypen stehen durch Einbinden der Datei `libTMCG.hh` im globalen Namensraum bereit. Die Kommunikation der beiden Programme erfolgt der Einfachheit halber über die Standardein- bzw. -ausgabe. Alice und Bob könnten die entsprechenden Datenströme umlenken und mit `netcat` oder `ssh` weiterleiten.

Zuerst werden die beiden Klasseninstanzen durch Konstruktoraufruf erzeugt (Zeile 10 und 13). Dabei sind verschiedene Argumente von Bedeutung:

t ist der Sicherheitsparameter, welcher die Betrugswahrscheinlichkeit p nach oben beschränkt, d. h. $p \leq 2^{-t}$.

k ist die Anzahl der Mitspieler.

w ist die Anzahl der Bits, die zur Kodierung der M verschiedenen Kartentypen notwendig ist, d. h. $w = \lceil \log_2 M \rceil$.

Am Aufruf des `BarnettSmartVTMF_dlog`-Konstruktors erkennt man, daß Alice die Gruppenparameter für G (DDH-schwer) erzeugt und später (Zeile 21) an Bob sendet. Da diese Parameter in beiden Programmen geprüft werden, stellt ein solches Vorgehen kein Sicherheitsproblem dar. In der Praxis könnten wir sogar öffentlich verfügbare Parameterwerte verwenden, sofern sie unseren Optimierungs- und Kodierungsbedingungen²⁴ genügen.

Der weitere Ablauf ist im Quelltext ausführlich dokumentiert und sollte deshalb leicht verständlich sein.

Referenzimplementierung SecureSkat

Im Frühjahr 2002 begann ich mit der Arbeit an diesem Projekt. Anfangs fand die Entwicklung unter dem Namen *OpenSkat* statt, später erfolgte dann die Umbenennung in *SecureSkat* (Details: siehe graue Box) und der Transfer

²³Der Beweis, daß dieses Spiel bei gleichverteilten Mischen immer terminiert, bleibt dem Leser als Übungsaufgabe überlassen.

²⁴Beispielsweise muß $g = 2$ ein Erzeuger von \mathbb{QR}_p sein, wobei p eine sichere Primzahl ist.

nach GNU/Savannah [12]. Der gesamte Quelltext (inklusive `libTMCG`) umfaßt mittlerweile ca. 15 100 Zeilen C++/C. Die letzte stabile Version unter altem Namen war *OpenSkat* 2.2, welches auch eine graphische Spieloberfläche auf Grundlage von *XSkat* [10] enthielt. Im neuen Projekt mußte sie leider entfernt werden.²⁵

OpenSkat versus SecureSkat

Vor langer, langer Zeit trug dieses Projekt mal den Namen "OpenSkat". Im Rahmen eines Umzugs in die GNU-Savanne wurde von den dortigen Parkwächtern aber eine Namensänderung gewünscht.

In the other hand can you look for another name to your project?

Do you have any suggestions? 'FreeSkat' sounds a little bit strange and may be in some sense misleading.

This is because we supports projects of the Free Software movement, not projects of the Open Source movement.

I understand your concerns. However, in this case the prefix 'open' should be considered more from a cryptographical rather than from philosophical point of view. Nevertheless, if the community has a better name suggestion ...

We are careful about ethical issues and insist on producing software that is not dependent on proprietary software.

While Open Source as defined by its founders means something pretty close to Free Software, it's frequently misunderstood.

Trotz aller Erklärungsversuche konnte der alte Name nicht gerettet werden und ward somit auf immer verloren. Schon drohte neues Unheil am Horizont:

You have to remove the 'xskat' dependence, because the license of this project is a GPL-Incompatible, if you remove this dependence we can accept the project in Savannah.

I don't have suggestions for a new name, but you can try with FreeSkat or even with a name without the 'Skat' word on it.

As you explain it, the 'open' word should not be considered from the philosophical point of view, but if we do not make the difference anyone will see your project thinking that has more relation with the Open Source movement rather than the Free Software movement.

To be continued in a free world ...

SecureSkat: Anonymer CVS Zugriff

```
export CVS_RSH="ssh"
cvs -z3 -d:ext:anoncvs@savannah.gnu.org:\
    /cvsroot/secskatskat co secskatskat
```

Prinzipiell ist *SecureSkat* eine sichere, dezentrale Implementierung des Kartenspiels *Skat* [9], wobei sicher in diesem Kontext bedeutet, daß die Betrugswahrscheinlichkeit bei Karten- und Stapeloperationen durch einen Sicherheitsparameter t begrenzt werden kann. Dazu bedient

²⁵Hinweis: Das Programm `openSkat_gui` (z. B. im Archiv `openSkat-2.2.tar.gz` enthalten) kann auch mit *SecureSkat* benutzt werden, denn alle Schnittstellen sind kompatibel.

		OpenSkat ≤ 1.9	SecureSkat [12]
Sicherheitsparameter t	Betrugswahrscheinlichkeit $p \leq 2^{-t}$	Übertragungsvolumen pro Spiel und Spieler mit $\ell = 1024$ und Schema [Sch98]	Übertragungsvolumen pro Spiel und Spieler mit $\ell = 1024$ und Schema [BSm03]
2	$\leq 0,25$	≈ 3 MB	≈ 1 MB
4	$\leq 0,0625$	≈ 5 MB	$\approx 1,6$ MB
8	$\leq 0,00390625$	≈ 10 MB	$\approx 2,8$ MB
16	$\leq 0,00001526$	≈ 20 MB	≈ 5 MB

Tabelle 2: Übertragungsvolumen versus Sicherheit

sich die Software der Konzepte aus [Sch98, BSm03] und benutzt mittlerweile direkt die Bibliothek libTMCG. Die Wahl von t hat erheblichen Einfluß auf Übertragungsvolumen und Laufzeit (siehe Tabelle 2).

Spielaushandlung und Protokollsteuerung erfolgen über den Kanal #openSkat, der auf einem beliebigen IRC-Server [8] (z. B. gaos.org) angesiedelt sein kann. Für die Beweise werden jedoch zu Beginn der Sitzung eigene Verbindungen²⁶ zwischen den Teilnehmern aufgebaut.²⁷ Eine genaue Beschreibung der Installation und Benutzung von SecureSkat liefert die enthaltene Datei README [12].

TODO: Noch nicht (bzw. nur teilweise) fertiggestellt sind

- Internationalisierung mit GNU gettext [13],
- Dokumentation und Fehlerbehandlung,
- freie graphische Spieloberfläche.

Helfende Mitarbeit, sei es durch Testen der Software, Melden von Fehlern, oder eigene Verbesserungen am Quelltext²⁸, ist jederzeit willkommen. Auch die sichere Umsetzung anderer Kartenspiele mit Hilfe der Bibliothek libTMCG wäre ein sehr wünschenswertes Ziel.

Zum Abschluß noch eine grundsätzliche Bemerkung: Die in diesem Artikel vorgestellte Software trägt zwar den Namen SecureSkat, ist aber keineswegs *perfekt sicher*. Jedes hinreichend komplexe (von Menschen geschaffene) Objekt enthält notwendigerweise mehr oder weniger Fehler. Im Quelltext von OpenSkat, dem Vorgänger von SecureSkat, fand ich bei bisherigen Kontrollen sogar besonders schwere Fehler. Beispielsweise erlaubte eine fehlende Festlegung in einem Zero-Knowledge Beweis die triviale Faktorisierung des Moduls m_i mittels Euklidischen Algorithmus. Daraus resultierte natürlich die totale Unsicherheit des Programms bezüglich seiner eigentlichen Aufgabe – dem sicheren mentalen Kartenspiel. Weitere, hoffentlich nicht ganz so kritische, Fehler sind wahrscheinlich enthalten und können nur durch gewissenhafte Begutachtung gefunden werden. Trotz aller Versuche, *vollständige Sicherheit* kann und wird es nie geben!

²⁶authentifiziert, verschlüsselt und komprimiert

²⁷Voraussetzung: keine paranoid konfigurierten Feuerwände

²⁸Große Teile von SecureSkat wurden schnell (d. h. fast immer unelegant) entwickelt und bedürfen dringend einer Verbesserung.

Zusammenfassung

In diesem Artikel wurden die Grundlagen sicherer mentaler Kartenspiele besprochen und eine freie Bibliothek zu ihrer Implementierung vorgestellt.

Referenzen

- [Sch98] Christian Schindelbauer: *Toolbox for Mental Card Games*, Technical Report A-98-14A, Universität Lübeck, 1998
- [BSm03] Adam Barnett, Nigel P. Smart: *Mental Poker Revisited*, In K.G. Paterson (Ed.): *Cryptography and Coding 2003*, LNCS 2898, pp. 370–383, 2003
- [Bon98] Dan Boneh: *The Decision Diffie-Hellman Problem*, In *Proceedings of the Third Algorithmic Number Theory Symposium*, LNCS 1423, pp. 48–63, 1998
- [GMR98] Rosario Gennaro, Daniele Micciancio, Tal Rabin: *An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products*, In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pp. 67–72, 1998
- [BR93] Mihir Bellare, Phillip Rogaway: *Random oracles are practical: A paradigm for designing efficient protocols*, In *Proceedings First Annual ACM Conference on Computer and Communications Security*, pp. 62–73, 1993
- [FS86] Amos Fiat, Adi Shamir: *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*, In *Advances in Cryptology: CRYPTO '86 Proceedings*, LNCS 263, pp. 186–194, 1987
- [BFM88] Manuel Blum, Paul Feldman, Silvio Micali: *Non-interactive zero-knowledge and its applications*, In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pp. 103–112, 1988
- [BSG91] Manuel Blum, Alfredo De Santis, Silvio Micali, Giuseppe Persiano: *Non-Interactive Zero Knowledge*, *SIAM Journal on Scientific Computing*, Volume 20(6), pp. 1084–1118, 1991
- [Rab79] Michael O. Rabin: *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*, Technical Report MIT-LCS-TR-212, MIT Laboratory for Computer Science, 1979
- [BR96] Mihir Bellare, Phillip Rogaway: *The Exact Security of Digital Signatures—How to Sign with*

- RSA and Rabin*, In *Advances in Cryptology: EUROCRYPT '96 Proceedings*, LNCS 1070, pp. 399–416, 1996
- [Bon01] Dan Boneh: *Simplified OAEP for the RSA and Rabin Functions*, In *Advances in Cryptology: CRYPTO '2001 Proceedings*, LNCS 2139, pp. 275–291, 2001
- [vOW96] Paul C. van Oorschot and Michael J. Wiener: *On Diffie-Hellman Key Agreement with Short Exponents*, In *Advances in Cryptology: EUROCRYPT '96 Proceedings*, LNCS 1070, pp. 332–343, 1996
- [RS00] Jean-François Raymond, Anton Stiglic: *Security Issues in the Diffie-Hellman Key Agreement Protocol*, ZKS Technical Report (draft), 2000
- [BJN00] Dan Boneh, Antoine Joux, Phong Q. Nguyen: *Why Textbook ElGamal and RSA Encryption are Insecure*, In *Proceedings of ASIACRYPT '2000*, LNCS 1976, pp. 30–43, 2001
- [CP92] David Chaum, Torben P. Pedersen: *Wallet Databases with Observers*, In *Advances in Cryptology: CRYPTO '92 Proceedings*, LNCS 740, pp. 89–105, 1992
- [CS97] Jan Camenisch, Markus Stadler: *Proof Systems for General Statements about Discrete Logarithms*, Technical Report, 1997
- [Gut00] Walter Guttman: *Kartenspielen übers Telefon*, Hauptseminar Sichere Telekommunikation WS 1999/2000, Universität Ulm, 2000
- [RSA79] Adi Shamir, Ronald L. Rivest, Leonard M. Adleman: *Mental Poker*, Technical Report MIT-LCS-TM-125, MIT Laboratory for Computer Science, 1979
- [Blu81] Manuel Blum: *Coin Flipping by Telephone: A protocol for solving impossible problems*, In *CRYPTO '81 Proceedings*, pp. 11–15, 1981; and *Proceedings of the 24th IEEE Computer Conference*, pp. 133–137, 1982
- [GM82] Shafi Goldwasser, Silvio Micali: *Probabilistic Encryption and How to Play Mental Poker Hiding All Partial Information*, In *Proceedings of the 14th Annual ACM Symposium on the Theory of Computing*, pp. 365–377, 1982
- [Cré87] Claude Crépeau: *A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face*, In *Advances in Cryptology: CRYPTO '86 Proceedings*, LNCS 263, pp. 239–247, 1987
- [BCR87] Gilles Brassard, Claude Crépeau, Jean-M. Robert: *All-or-nothing disclosure of secrets*, In *Advances in Cryptology: CRYPTO '86 Proceedings*, LNCS 263, pp. 234–238, 1987
- [GMW87] Oded Goldreich, Silvio Micali, Avi Wigderson: *How to Prove All NP-Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design*, In *Advances in Cryptology: CRYPTO '86 Proceedings*, LNCS 263, pp. 171–185, 1987
- [QGB89] Jean-J. Quisquater, Louis Guillou, Tom Berson: *How to Explain Zero-Knowledge Protocols to Your Children*, In *Advances in Cryptology: CRYPTO '89 Proceedings*, LNCS 435, pp. 628–631, 1989
- [BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen: *Direct Anonymous Attestation*, In *Proceedings of 11th ACM Conference on Computer and Communications Security*, 2004
- [1] <http://catless.ncl.ac.uk/Risks/17.21.html#subj9>
- [2] <http://www.zurich.ibm.com/security/daa/>
- [3] <http://www.crypto.ethz.ch/teaching/lectures/Krypto04/ROM.pdf>
- [4] <http://www.kisa.or.kr/technology/sub1/Rabin.htm>
- [5] <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>
- [6] GNU Multiple Precision Arithmetic Library, the fastest bignum library on the planet!, <http://www.swox.com/gmp/>
- [7] GNU Crypto Library, <http://directory.fsf.org/security/libgrypt.html>
- [8] RFC 1459: Internet Relay Chat Protocol, <http://www.ietf.org/rfc/rfc1459.txt?number=1459>
- [9] Offizielle Skatregeln: <http://www.dskv.de/>
- [10] XSkat: <http://www.xskat.de/>
- [11] <http://freshmeat.net/projects/openskat/>
- [12] SecureSkat und libTMCG: <http://savannah.nongnu.org/projects/secureskat>
- [13] <http://www.gnu.org/software/gettext/>
- [14] http://de.wikipedia.org/wiki/Zyklische_Gruppe
- [15] <http://de.wikipedia.org/wiki/Restklassenring>
- [16] http://de.wikipedia.org/wiki/Eulersche_%CF%86-Funktion

Quelltext SchwarzPeter_Alice.cc

```

1 // libTMCG
  #include <libTMCG.hh>

  int main
5   (
    {
      // Initialisieren der "Toolbox for Mental Card Games"
      // -----
      size_t t = 16, k = 2, w = 4; // p_Betrug <= 2^{-16}, 2 Spieler, 13 Typen
10     SchindelhauerTMCG *tmcg = new SchindelhauerTMCG(t, k, w);

      // Initialisieren, Erzeugen ...
      BarnettSmartVTMF_dlog *vtmf = new BarnettSmartVTMF_dlog();
      // ... und Überprüfen der Gruppe G
15     if (!vtmf->CheckGroup())
        {
          std::cerr << ">> Gruppe G fehlerhaft erzeugt" << std::endl;
          return -1;
        }
20     // Senden der Gruppenparameter an Bob
      vtmf->PublishGroup(std::cout);
      // Erzeugen und Senden des (öffentlichen) Schlüssels
      vtmf->KeyGenerationProtocol_GenerateKey();
      vtmf->KeyGenerationProtocol_PublishKey(std::cout);
25     // Einfügen von Bob's öffentlichen Schlüssel
      if (!vtmf->KeyGenerationProtocol_UpdateKey(std::cin))
        {
          std::cerr << ">> Schlüsselbeweis falsch" << std::endl;
          return -1;
30     }

      // Blatt mit 25 Karten erstellen (12 Paare und ein "Schwarzer Peter")
      // -----
      TMCG_OpenStack<VTMF_Card> Anfangsstapel;
35     for (size_t i = 0; i < 13; i++)
        {
          for (size_t j = 0; j < 2; (i != 0) ? j++ : j = 2)
            {
              VTMF_Card c;
40              tmcg->TMCG_CreateOpenCard(c, vtmf, i); // Karte mit Typ i erzeugen,
              Anfangsstapel.push(i, c); // und auf den offenen Stapel legen.
            }
          }

45     // Anfangsstapel mischen: Alice zuerst, dann Bob.
      // -----
      std::cerr << "Anfangsstapel mischen ..." << std::endl;
      TMCG_Stack<VTMF_Card> Mischstapel, Mischstapel_Alice, Mischstapel_Bob;
      TMCG_StackSecret<VTMF_CardSecret> Mischgeheimnis;
50     Mischstapel.push(Anfangsstapel); // offenen in allgemeinen Stapel umwandeln
      tmcg->TMCG_CreateStackSecret(Mischgeheimnis, false, // volle Permutation
          Mischstapel.size(), vtmf);

```

```

// ... Alice
55  tmcg->TMCG_MixStack(Mischstapel, Mischstapel_Alice,
    Mischgeheimnis, vtmf); // Maskieren
std::cout << Mischstapel_Alice << std::endl; // Stapel an Bob senden
tmcg->TMCG_ProofStackEquality(Mischstapel, Mischstapel_Alice,
    Mischgeheimnis, false, vtmf, std::cin, std::cout); // Korrektheit beweisen
60
// ... Bob
char *tmp = new char[TMCG_MAX_STACK_CHARS];
std::cin.getline(tmp, TMCG_MAX_STACK_CHARS);
if (!Mischstapel_Bob.import(tmp))
65  {
    std::cerr << ">< Stapelformat falsch (Trollversuch?)" << std::endl;
    return -1;
}
delete tmp;
70  if (!tmcg->TMCG_VerifyStackEquality(Mischstapel_Alice,
    Mischstapel_Bob, false, vtmf, std::cin, std::cout)) // Beweis verifizieren
    {
        std::cerr << ">< Stapelbeweis falsch (Betrugsversuch?)" << std::endl;
        return -1;
75  }

// Stapel teilen: Alice erhält die Karten 1 bis 13 und Bob den Rest.
// -----
std::cerr << "Stapel teilen ..." << std::endl;
80  TMCG_Stack<VTMF_Card> Kartenstapel_Alice, Kartenstapel_Bob;
for (size_t i = 0; i < 13 ; i++)
    Kartenstapel_Alice.push(Mischstapel_Bob[i]);
for (size_t i = 13; i < 25 ; i++)
    Kartenstapel_Bob.push(Mischstapel_Bob[i]);
85

// Das eigentliche Spiel läuft in einer (Endlos-)Schleife.
// -----
while (1)
{
90    // Handkarten privat aufdecken: Alice zuerst, dann Bob.
    // -----
    // Die drei Operationen SelfCardSecret, VerifyCardSecret und
    // TypeOfCard müssen in *genau* dieser Reihenfolge aufgerufen werden!
    TMCG_OpenStack<VTMF_Card> Handkarten;
95    size_t Typ = 0;

    // ... Alice
    std::cerr << "Meine Karten: ";
    for (size_t i = 0; i < Kartenstapel_Alice.size(); i++)
100    {
        tmcg->TMCG_SelfCardSecret(Kartenstapel_Alice[i], vtmf);
        if (!tmcg->TMCG_VerifyCardSecret(Kartenstapel_Alice[i], vtmf,
            std::cin, std::cout))
        {
105            std::cerr << ">< Öffnungsbeweis falsch (Betrugsversuch?)" <<
                std::endl;
            return -1;
        }
        Typ = tmcg->TMCG_TypeOfCard(Kartenstapel_Alice[i], vtmf);

```

```

110         Handkarten.push(Typ, Kartenstapel_Alice[i]);
           std::cerr << Typ << " ";
       }
       std::cerr << std::endl;

115     // ... Bob
       for (size_t i = 0; i < Kartenstapel_Bob.size(); i++)
       {
           tmcg->TMCG_ProofCardSecret(Kartenstapel_Bob[i], vtmf,
120             std::cin, std::cout);
       }

       // Ein Paar offen ablegen (sofern möglich): Bob zuerst, dann Alice.
       // -----
       TMCG_OpenStack<VTMF_Card> Paarstapel;
125     size_t Paare = 0, LetzterTyp = 0;

       // ... Bob
       std::cin >> Paare;
       std::cin.ignore(1, '\n'); // Newline verwerfen
130     if (Paare > 1)
       {
           std::cerr << ">< Unerlaubte Paaranzahl (Trollversuch?)" << std::endl;
           return -1;
       }
135     if (Paare)
       {
           std::cerr << "Gegner legt ab: ";
           for (size_t i = 0; i < 2; i++)
           {
140             VTMF_Card c;
               char *tmp = new char[TMCG_MAX_CARD_CHARS];
               std::cin.getline(tmp, TMCG_MAX_CARD_CHARS);
               if (!c.import(tmp))
               {
145                 std::cerr << ">< Kartenformat falsch (Trollversuch?)" <<
                           std::endl;
                           return -1;
               }
               delete tmp;
150             // Prüfen, ob Karte im Stapel ist
               if (!Kartenstapel_Bob.find(c))
               {
                   std::cerr << ">< Karte nicht vorhanden (Betrugsversuch?)" <<
155                     std::endl;
                   return -1;
               }
               // Karte aufdecken und Korrektheit prüfen
               tmcg->TMCG_SelfCardSecret(c, vtmf);
               if (!tmcg->TMCG_VerifyCardSecret(c, vtmf, std::cin, std::cout))
160             {
                   std::cerr << ">< Öffnungsbeweis falsch (Betrugsversuch?)" <<
                           std::endl;
                           return -1;
               }
165             Typ = tmcg->TMCG_TypeOfCard(c, vtmf);
               // Prüfen, ob wirklich ein Paar vorliegt

```

```

        if (i % 2)
        {
            if (LetzterTyp != Typ)
170         {
                std::cerr << ">< Kein Paar (Trollversuch?)" << std::endl;
                return -1;
            }
            else
175         {
                std::cerr << Typ << " ";
            }
        }
        else
            LetzterTyp = Typ;
        // Karte entfernen
180     Kartenstapel_Bob.remove(c);
    }
    std::cerr << std::endl;
}

185 // .. Alice
// Paare suchen
for (size_t i = 0; i < Handkarten.size(); i++)
{
    for (size_t j = 0; j < Handkarten.size(); j++)
190     {
        // Paar gefunden?
        if ((i < j) &&
            (Handkarten[i].first == Handkarten[j].first))
        {
195         Paarstapel.push(Handkarten[i]);
            Paarstapel.push(Handkarten[j]);
            i = Handkarten.size();           // keine weiteren Paare suchen
            break;
        }
    }
200 }
}
// Anzahl senden, Paare aufdecken und entfernen
std::cout << (Paarstapel.size() / 2) << std::endl;
if (Paarstapel.size())
205 {
    std::cerr << "Ich lege ab: ";
    for (size_t i = 0; i < Paarstapel.size(); i++)
    {
        std::cout << Paarstapel[i].second << std::endl;
210     tmcg->TMCG_ProofCardSecret(Paarstapel[i].second, vtmf,
        std::cin, std::cout);
        Kartenstapel_Alice.remove(Paarstapel[i].second);
        Handkarten.remove(Paarstapel[i].first);
        if (i % 2)
215         std::cerr << Paarstapel[i].first << " ";
    }
    std::cerr << std::endl;
}

220 // Aufräumen und Abbruchkriterium prüfen
// -----
Paarstapel.clear();
Mischstapel_Alice.clear(), Mischstapel_Bob.clear();

```

```

225     if (Kartenstapel_Alice.size() == 0)
        {
            std::cerr << ">> Glück gehabt!" << std::endl;
            break;
        }
230     if (Kartenstapel_Bob.size() == 0)
        {
            std::cerr << ">> Zonk! ('Schwarzer Peter')" << std::endl;
            break;
        }

235     // Eine Karte beim Gegner ziehen: Der mit weniger Karten zieht.
    // Nach dem Ziehen wird neu gemischt und die Korrektheit gezeigt.
    // -----
    size_t WelchePosition;
    std::vector<bool> Ablauf;
240    VTMF_Card c;

    if (Kartenstapel_Alice.size() > Kartenstapel_Bob.size())
        Ablauf.push_back(false);
    else
245        Ablauf.push_back(true);

    for (size_t i = 0; i < Ablauf.size(); i++)
    {
250        if (Ablauf[i])
            {
                // Alice zieht ...
                WelchePosition = mpz_srandom() % Kartenstapel_Bob.size();
                std::cout << WelchePosition << std::endl;
                c = Kartenstapel_Bob[WelchePosition]; // Karte holen, ...
255                Kartenstapel_Bob.remove(c);         // entfernen, ...
                tmcg->TMCG_SelfCardSecret(c, vtmf);    // aufdecken, ...
                if (!tmcg->TMCG_VerifyCardSecret(c, vtmf, std::cin, std::cout))
                    {
260                        std::cerr << ">> Öffnungsbeweis falsch (Betrugsversuch?)" <<
                            std::endl;
                        return -1;
                    }
                Typ = tmcg->TMCG_TypeOfCard(c, vtmf);
                std::cerr << "Ich ziehe die Karte (von Position " <<
265                WelchePosition << "): " << Typ;
                if (Typ)
                    std::cerr << std::endl;
                else
                    std::cerr << " (Zonk!)" << std::endl;
270                Kartenstapel_Alice.push(c); // ... und auf Alices Stapel legen.
                // ... und mischt neu.
                tmcg->TMCG_CreateStackSecret(Mischgeheimnis, false,
                    Kartenstapel_Alice.size(), vtmf);
                tmcg->TMCG_MixStack(Kartenstapel_Alice, Mischstapel_Alice,
275                Mischgeheimnis, vtmf); // Maskieren, ...
                std::cout << Mischstapel_Alice << std::endl; // senden, ...
                tmcg->TMCG_ProofStackEquality(Kartenstapel_Alice, Mischstapel_Alice,
                    Mischgeheimnis, false, vtmf, std::cin, std::cout); // beweisen.
                Kartenstapel_Alice = Mischstapel_Alice;
280            }
        }
    }

```

```
else
{
    // Bob zieht ...
    std::cin >> WelchePosition;
285    std::cin.ignore(1, '\n'); // Newline verwerfen
    if (WelchePosition >= Kartenstapel_Alice.size())
    {
        std::cerr << ">< Falscher Index (Trollversuch?)" << std::endl;
        return -1;
290    }
    c = Kartenstapel_Alice[WelchePosition]; // Karte holen,
    Kartenstapel_Alice.remove(c);          // entfernen,
    tmcg->TMCG_ProofCardSecret(c, vtmf, std::cin, std::cout); // aufdecken,
    Kartenstapel_Bob.push(c);              // ... und auf Bobs Stapel legen.
295    // ... und mischt neu.
    char *tmp = new char[TMCG_MAX_STACK_CHARS];
    std::cin.getline(tmp, TMCG_MAX_STACK_CHARS);
    if (!Mischstapel_Bob.import(tmp))
    {
300        std::cerr << ">< Stapelformat falsch (Trollversuch?)" << std::endl;
        return -1;
    }
    delete tmp;
    if (!tmcg->TMCG_VerifyStackEquality(Kartenstapel_Bob,
305        Mischstapel_Bob, false, vtmf, std::cin, std::cout)) // verifizieren
    {
        std::cerr << ">< Stapelbeweis falsch (Betrugsversuch?)" << std::endl;
        return -1;
    }
310    Kartenstapel_Bob = Mischstapel_Bob;
}
}
}

315 // Aufräumen
    delete vtmf, delete tmcg;
}
```

Quelltext SchwarzerPeter_Bob.cc

```

1 // libTMCG
  #include <libTMCG.hh>

  int main
5   (
  {
    // Initalisieren der "Toolbox for Mental Card Games"
    // -----
    size_t t = 16, k = 2, w = 4; // p_Betrug <= 2^{-16}, 2 Spieler, 13 Typen
10   SchindelhauerTMCG *tmcg = new SchindelhauerTMCG(t, k, w);

    // Initalisieren und Überprüfen der Gruppe G (von Alice erzeugt)
    BarnettSmartVTMF_dlog *vtmf = new BarnettSmartVTMF_dlog(std::cin);
    if (!vtmf->CheckGroup())
15   {
        std::cerr << ">> Gruppe G fehlerhaft erzeugt" << std::endl;
        return -1;
    }

    // Erzeugen und Senden des eigenen VTMF-Schlüssels
    vtmf->KeyGenerationProtocol_GenerateKey();
    vtmf->KeyGenerationProtocol_PublishKey(std::cout);
    // Einfügen von Alice's Schlüssel
    if (!vtmf->KeyGenerationProtocol_UpdateKey(std::cin))
25   {
        std::cerr << ">> Schlüsselbeweis falsch" << std::endl;
        return -1;
    }

    // Blatt mit 25 Karten erstellen (12 Paare und ein "Schwarzer Peter")
    // -----
    TMCG_OpenStack<VTMF_Card> Anfangsstapel;
    for (size_t i = 0; i < 13; i++)
    {
35       for (size_t j = 0; j < 2; (i != 0) ? j++ : j = 2)
        {
            VTMF_Card c;
            tmcg->TMCG_CreateOpenCard(c, vtmf, i); // Karte mit Typ i erzeugen,
            Anfangsstapel.push(i, c); // und auf den offenen Stapel legen.
40         }
    }

    // Anfangsstapel mischen: Alice zuerst, dann Bob.
    // -----
45   std::cerr << "Anfangsstapel mischen ..." << std::endl;
    TMCG_Stack<VTMF_Card> Mischstapel, Mischstapel_Alice, Mischstapel_Bob;
    TMCG_StackSecret<VTMF_CardSecret> Mischgeheimnis;
    Mischstapel.push(Anfangsstapel); // offenen in allgemeinen Stapel umwandeln
    tmcg->TMCG_CreateStackSecret(Mischgeheimnis, false, // volle Permutation
50     Mischstapel.size(), vtmf);

    // ... Alice

```

```

char *tmp = new char[TMCG_MAX_STACK_CHARS];
std::cin.getline(tmp, TMCG_MAX_STACK_CHARS);
55  if (!Mischstapel_Alice.import(tmp))
    {
        std::cerr << ">< Stapelformat falsch (Trollversuch?)" << std::endl;
        return -1;
    }
60  delete tmp;
    if (!tmcg->TMCG_VerifyStackEquality(Mischstapel, Mischstapel_Alice,
        false, vtmf, std::cin, std::cout))
    {
65      std::cerr << ">< Stapelbeweis falsch (Betrugsversuch?)" << std::endl;
        return -1;
    }
    // ... Bob
    tmcg->TMCG_MixStack(Mischstapel_Alice, Mischstapel_Bob,
        Mischgeheimnis, vtmf); // Maskieren
70  std::cout << Mischstapel_Bob << std::endl; // Stapel an Alice senden
    tmcg->TMCG_ProofStackEquality(Mischstapel_Alice, Mischstapel_Bob,
        Mischgeheimnis, false, vtmf, std::cin, std::cout); // Korrektheit beweisen

    // Stapel teilen: Alice erhalt die Karten 1 bis 13 und Bob den Rest.
75  // -----
    std::cerr << "Stapel teilen ..." << std::endl;
    TMCG_Stack<VTMF_Card> Kartenstapel_Alice, Kartenstapel_Bob;
    for (size_t i = 0; i < 13 ; i++)
        Kartenstapel_Alice.push(Mischstapel_Bob[i]);
80  for (size_t i = 13; i < 25 ; i++)
        Kartenstapel_Bob.push(Mischstapel_Bob[i]);

    // Das eigentliche Spiel lauft in einer (Endlos-)Schleife.
    // -----
85  while (1)
    {
        // Handkarten privat aufdecken: Alice zuerst, dann Bob.
        // -----
        // Die drei Operationen SelfCardSecret, VerifyCardSecret und
90  // TypeOfCard mussen in *genau* dieser Reihenfolge aufgerufen werden!
        TMCG_OpenStack<VTMF_Card> Handkarten;
        size_t Typ = 0;

        // ... Alice
95  for (size_t i = 0; i < Kartenstapel_Alice.size(); i++)
        {
            tmcg->TMCG_ProofCardSecret(Kartenstapel_Alice[i], vtmf,
                std::cin, std::cout);
        }
100

        // ... Bob
        std::cerr << "Meine Karten: ";
        for (size_t i = 0; i < Kartenstapel_Bob.size(); i++)
        {
105            tmcg->TMCG_SelfCardSecret(Kartenstapel_Bob[i], vtmf);
            if (!tmcg->TMCG_VerifyCardSecret(Kartenstapel_Bob[i], vtmf,
                std::cin, std::cout))
            {
                std::cerr << ">< offnungsbeweis falsch (Betrugsversuch?)" <<

```

```

110         std::endl;
           return -1;
       }
       Typ = tmcg->TMCG_TypeOfCard(Kartenstapel_Bob[i], vtmf);
       Handkarten.push(Typ, Kartenstapel_Bob[i]);
115       std::cerr << Typ << " ";
   }
   std::cerr << std::endl;

   // Ein Paar offen ablegen (sofern möglich): Bob zuerst, dann Alice.
120   // -----
   TMCG_OpenStack<VTMF_Card> Paarstapel;
   size_t Paare = 0, LetzterTyp = 0;

   // ... Bob
125   // Paare suchen
   for (size_t i = 0; i < Handkarten.size(); i++)
   {
       for (size_t j = 0; j < Handkarten.size(); j++)
       {
130           // Paar gefunden?
           if ((i < j) &&
               (Handkarten[i].first == Handkarten[j].first))
           {
135               Paarstapel.push(Handkarten[i]);
               Paarstapel.push(Handkarten[j]);
               i = Handkarten.size();           // keine weiteren Paare suchen
               break;
           }
       }
   }
140   // Anzahl senden, Paare aufdecken und entfernen
   std::cout << (Paarstapel.size() / 2) << std::endl;
   if (Paarstapel.size())
   {
145       std::cerr << "Ich lege ab: ";
       for (size_t i = 0; i < Paarstapel.size(); i++)
       {
           std::cout << Paarstapel[i].second << std::endl;
           tmcg->TMCG_ProofCardSecret(Paarstapel[i].second, vtmf,
150               std::cin, std::cout);
           Kartenstapel_Bob.remove(Paarstapel[i].second);
           Handkarten.remove(Paarstapel[i].first);
           if (i % 2)
               std::cerr << Paarstapel[i].first << " ";
155       }
       std::cerr << std::endl;
   }

   // ... Alice
160   std::cin >> Paare;
   std::cin.ignore(1, '\n'); // Newline verwerfen
   if (Paare > 1)
   {
165       std::cerr << ">< Unerlaubte Paaranzahl (Trollversuch?)" << std::endl;
       return -1;
   }
}

```

```

if (Paare)
{
    std::cerr << "Gegner legt ab: ";
170   for (size_t i = 0; i < 2; i++)
    {
        VTMF_Card c;
        char *tmp = new char[TMCG_MAX_CARD_CHARS];
        std::cin.getline(tmp, TMCG_MAX_CARD_CHARS);
175   if (!c.import(tmp))
        {
            std::cerr << ">< Kartenformat falsch (Trollversuch?)" <<
                std::endl;
            return -1;
180   }
        delete tmp;
        // Prüfen, ob Karte im Stapel ist
        if (!Kartenstapel_Alice.find(c))
        {
185   std::cerr << ">< Karte nicht vorhanden (Betrugsversuch?)" <<
                std::endl;
            return -1;
        }
        // Karte aufdecken und Korrektheit prüfen
190   tmcg->TMCG_SelfCardSecret(c, vtmf);
        if (!tmcg->TMCG_VerifyCardSecret(c, vtmf, std::cin, std::cout))
        {
            std::cerr << ">< Öffnungsbeweis falsch (Betrugsversuch?)" <<
                std::endl;
195   return -1;
        }
        Typ = tmcg->TMCG_TypeOfCard(c, vtmf);
        // Prüfen, ob wirklich ein Paar vorliegt
        if (i % 2)
200   {
            if (LetzterTyp != Typ)
            {
                std::cerr << ">< Kein Paar (Trollversuch?)" << std::endl;
                return -1;
205   }
            else
                std::cerr << Typ << " ";
        }
        else
210   LetzterTyp = Typ;
        // Karte entfernen
        Kartenstapel_Alice.remove(c);
    }
    std::cerr << std::endl;
215 }

// Aufräumen und Abbruchkriterium prüfen
// -----
Paarstapel.clear();
220 Mischstapel_Alice.clear(), Mischstapel_Bob.clear();
if (Kartenstapel_Alice.size() == 0)
{
    std::cerr << ">< Zonk! ('Schwarzer Peter')" << std::endl;
}

```

```

    break;
225     }
    if (Kartenstapel_Bob.size() == 0)
    {
        std::cerr << ">> Glück gehabt!" << std::endl;
        break;
230     }

    // Eine Karte beim Gegner ziehen: Der mit weniger Karten zieht.
    // Nach dem Ziehen wird neu gemischt und die Korrektheit gezeigt.
    // -----
235     size_t WelchePosition;
    std::vector<bool> Ablauf;
    VTMF_Card c;

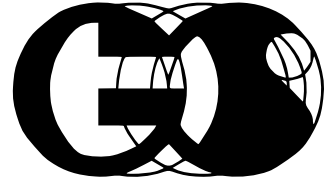
    if (Kartenstapel_Alice.size() > Kartenstapel_Bob.size())
240         Ablauf.push_back(true);
    else
        Ablauf.push_back(false);

    for (size_t i = 0; i < Ablauf.size(); i++)
245     {
        if (Ablauf[i])
        {
            // Bob zieht ...
            WelchePosition = mpz_srandom() % Kartenstapel_Alice.size();
250             std::cout << WelchePosition << std::endl;
            c = Kartenstapel_Alice[WelchePosition]; // Karte holen, ...
            Kartenstapel_Alice.remove(c);          // entfernen, ...
            tmcg->TMCG_SelfCardSecret(c, vtmf);     // aufdecken, ...
            if (!tmcg->TMCG_VerifyCardSecret(c, vtmf, std::cin, std::cout))
255             {
                std::cerr << ">> Öffnungsbeweis falsch (Betrugsversuch?)" <<
                    std::endl;
                return -1;
            }
            Typ = tmcg->TMCG_TypeOfCard(c, vtmf);
260             std::cerr << "Ich ziehe die Karte (von Position " <<
                WelchePosition << "): " << Typ;
            if (Typ)
                std::cerr << std::endl;
265             else
                std::cerr << " (Zonk!)" << std::endl;
            Kartenstapel_Bob.push(c); // ... und auf Bobs Stapel legen.
            // ... und mischt neu.
            tmcg->TMCG_CreateStackSecret(Mischgeheimnis, false,
270             Kartenstapel_Bob.size(), vtmf);
            tmcg->TMCG_MixStack(Kartenstapel_Bob, Mischstapel_Bob,
                Mischgeheimnis, vtmf); // Maskieren, ...
            std::cout << Mischstapel_Bob << std::endl; // senden, ...
            tmcg->TMCG_ProofStackEquality(Kartenstapel_Bob, Mischstapel_Bob,
275             Mischgeheimnis, false, vtmf, std::cin, std::cout); // beweisen.
            Kartenstapel_Bob = Mischstapel_Bob;
        }
        else
        {
280             // Alice zieht ...

```

```
std::cin >> WelchePosition;
std::cin.ignore(1, '\n'); // Newline verwerfen
if (WelchePosition >= Kartenstapel_Bob.size())
{
285     std::cerr << ">> Falscher Index (Trollversuch?)" << std::endl;
        return -1;
}
c = Kartenstapel_Bob[WelchePosition]; // Karte holen, ...
Kartenstapel_Bob.remove(c); // entfernen, ...
290 tmcg->TMCG_ProofCardSecret(c, vtmf, std::cin, std::cout); // aufdecken,
Kartenstapel_Alice.push(c); // ... und auf Alices Stapel legen.
// ... und mischt neu.
char *tmp = new char[TMCG_MAX_STACK_CHARS];
std::cin.getline(tmp, TMCG_MAX_STACK_CHARS);
295 if (!Mischstapel_Alice.import(tmp))
{
    std::cerr << ">> Stapelformat falsch (Trollversuch?)" << std::endl;
    return -1;
}
300 delete tmp;
if (!tmcg->TMCG_VerifyStackEquality(Kartenstapel_Alice,
    Mischstapel_Alice, false, vtmf, std::cin, std::cout)) // verifizieren
{
    std::cerr << ">> Stapelbeweis falsch (Betrugsversuch?)" << std::endl;
305     return -1;
}
Kartenstapel_Alice = Mischstapel_Alice;
}
}
310 }

// Aufräumen
delete vtmf, delete tmcg;
}
```



Mitgliedsantrag GAOS e. V.

Ja! Ich will Mitglied bei GAOS – Gesellschaft für die Anwendung Offener Systeme e. V. werden, denn ich möchte damit beispielsweise die Idee der *Freien Software* aktiv unterstützen und verbreiten.

Name:^a

Vorname:

volljährig:^b ja, nein

Schüler, Student oder Auszubildender:^c ja, nein

Straße:

PLZ:^d

Ort:

E-Mail:

Telefon:

PGP/ GnuPG:^e

1. Gruppe:

2. Gruppe:

Typ: Keyserver:

Mit der Beantragung der Mitgliedschaft erkenne ich die Satzung und Finanzordnung des Vereins GAOS an. Ich weiß, daß ich die Mitgliedschaft jederzeit schriftlich kündigen kann, wobei der Halbjahresbeitrag auch anteilig nicht rückerstattet wird. Mir ist bekannt, daß ich meine Mitgliedsbeiträge in der Höhe von 15,36 € (für 6 Monate für Schüler, Studenten und Auszubildende) bzw. 30,72 € (für 6 Monate für sonstige Mitglieder) rechtzeitig auf das Konto

Kontoinhaber: Gesellschaft für die Anwendung Offener Systeme e.V.

Kontonummer: 1 100 523 436

Bankleitzahl: 860 555 92

Kreditinstitut: Stadt- und Kreissparkasse Leipzig

Verwendungszweck: Vorname und Name des Mitglieds

zu überweisen habe. Ich weiß, daß GAOS meine Daten vertraulich behandelt und nicht Dritten zugänglich macht. Mir ist klar, daß ich mich jederzeit unter <http://gaos.org/> bzw. durch die Mitgliederversammlung über unseren Verein informieren kann.

Datum

Unterschrift

^abei Institutionen/ Unternehmen nur der Name

^bbitte zutreffendes ankreuzen, ggf. Unterschrift des Erziehungsberechtigten

^cggf. Beleg beifügen

^dPostleitzahl

^eFingerprint, Typ und Keyserver für den öffentlichen Schlüssel, falls vorhanden

▼ Über uns

Was Sie schon immer über den GAOS e.V. wissen wollten, sich aber nie zu fragen trauten.

GAOS, die Gesellschaft für die Anwendung offener Systeme, ist ein eingetragener Verein, der im Dezember 1999 von Mitgliedern der *Leipziger Linux User Group (Lug-L)* gegründet wurde. Zunächst sollte GAOS nur als Träger der Veranstaltungen dieser Benutzergruppe auftreten, um mehr Rechtssicherheit für die vom Idealismus getragenen Aktionen zu schaffen: Installationsparties und Vortragsreihe sollten rechtlich als das anerkannt werden, was sie sind, nämlich als gemeinnützige Tätigkeit.

Dennoch beschränkt sich GAOS nicht auf Linux. Im Gegenteil, Zielsetzung gemäß unserer Satzung ist die Förderung *Offener Systeme*, sei es Hardware (F-CPU, OS-Car) oder Software (GNU, Linux, BSD-Unix, etc.), und das heißt in erster Linie, die gemeinsame Idee von Freizügigkeit und offenem Austausch hinter diesen Systemen zu verbreiten.

Am besten fassen diese Gemeinsamkeiten die *Debian Free Software Guidelines (DFSG)* [1] zusammen: Offene Systeme liegen in einer Form vor, die dazu geeignet ist, sie zu verändern, zu verbessern oder einfach nur zu verstehen, und es gibt keine Lizenz, die verhindern könnte, genau diese Rechte, nämlich auf das Aneignen und Weitergeben von Wissen, auch auszuüben. Darunter fallen insbesondere die meisten OpenSource-Projekte¹ und alle freien Projekte im Sinne der *Free Software Foundation*.

Referenzen

[1] http://www.de.debian.org/social_contract#guidelines

Autorenverzeichnis

V. i. S. d. P. sind folgende Personen/Autoren:

▶	Holger Klawitter info@klawitter.de Patentrecht und Landeigentum	5
▶	Frank-Michael Schleif schleif@informatik.uni-leipzig.de Plugins mit wxWidgets	5–10
▶	Heiko Stamer stamer@gaos.org Kryptographische Skatrunde	10–32
▶	Sven Türpe sven@gaos.org Dieser Satz enthält drei Fähler.	3–5

Impressum

Offene Systeme

ISSN: 1619-0114

▼ Herausgeber

GAOS e.V., c/o Marcus Bautze
Rosenowstraße 71, 04357 Leipzig

eMail: gaos@gaos.org
WWW: <http://gaos.org/>

▼ T_EXnischer Editor

Heiko Stamer, stamer@gaos.org
76F73011 329D27DB 8D7C3F97 4F584EB8 FB2BE14F

▼ Copyright & Lizenz

Alle Artikel dieser Zeitschrift unterliegen der *GNU Free Documentation License*. Die genauen Lizenzbestimmungen finden Sie unter <http://www.fsf.org/copyleft/fdl.html>. Waren- und Firmennamen werden ohne Gewährleistung ihrer freien Verwendbarkeit benutzt und sind möglicherweise geschützt.

▼ Danksagung

Diese Zeitschrift entstand unter Verwendung *Freier Software*, u. a. Debian GNU/Linux, T_EX, L^AT_EX und METAFONT. Unser besonderer Dank gilt deshalb allen beteiligten Entwicklern.

Autoren gesucht!

Wir sind jederzeit an längeren Artikeln oder kurzen Ankündigungen zu den Themenbereichen

- freie Hard- oder Software,
- offene Standards, Technologien und Projekte,
- Wissens-Allmende und ähnliche Konzepte

interessiert. Ihre Texte (vorzugsweise L^AT_EX-Format) können Sie an den Herausgeber senden. Bitte beachten Sie, daß wir keine Honorare zahlen und nur Beiträge veröffentlichen, welche von den Autoren unter *GNU Free Documentation License* gestellt werden.

¹Vorsicht, der Begriff wird sehr inflationär verwendet.