Die RWTH-Kundennummer
Eine einfache aber robuste Identitätsnummer

Christian Bischof und Guido Bunsen
Rechen- und Kommunikationszentrum der RWTH Aachen
{bischof,bunsen}@rz.rwth-aachen.de

Jürgen Müller
Lehrstuhl D für Mathematik der RWTH Aachen
juergen.mueller@math.rwth-aachen.de

Wozu eine RWTH-Kundennummer?

Wie bei anderen Organisationen, die viele Personen als Mitglieder, Angehörige oder Kunden zu verwalten haben, reicht auch bei der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen der Namen einer Person nicht aus, um ein Zeugnis auszustellen, ein Buch auszuleihen, oder eine Klausuranmeldung vornehmen zu können. Bei mehreren zehntausend Studierenden, Alumni und Mitarbeitern ist die Gefahr zu groß, dem falschen „Thomas Lehmann" oder der falschen „Julia Schröder" eine Mahnung für ausgeliehene Bücher zu schicken, oder sie mit einem vorzeitigen Diplomzeugnis für das falsche Fach zu beglücken. Eine Lösung wäre, den Namen nicht alleine zu verwenden, sondern immer nur in Kombination etwa mit der Anschrift, dem Geburtsort und dem Geburtsdatum. Allerdings sollen nicht bei jedem Verwaltungsvorgang all diese Information offengelegt werden müssen. Daher verwenden Organisationen in der Regel Kundennummern oder Mitgliedsnummern, um eine Person eindeutig identifizieren zu können.

Leider nutzen praktisch alle Organisationen verschiedene Methoden, um die Nummern zuzuweisen und deren Eindeutigkeit sicherzustellen. Das gilt nicht nur für die Vielzahl von Verwaltungen, wie etwa den Rentenversicherungen, Meldebehörden, Krankenversicherungen, Finanzämtern oder Automobilclubs, sondern bereits innerhalb einzelner Organisationen, wie etwa der RWTH: Das Studierendensekretariat vergibt Matrikelnummern; die Personalstelle vergibt Personalnummern; die RWTH-Bibliothek vergibt Nummern auf Leseausweisen; und das Rechen- und Kommunikationszentrum vergibt Benutzerkennungen. Wie zu erwarten sind diese Nummern und Kennungen unabhängig voneinander, und die meisten Personen sind mit mehrerer dieser Nummern assoziiert: Viele Studierende stehen als wissenschaftliche Hilfskräfte in einem Angestelltenverhältnis mit der RWTH, und nutzen sowohl die Bibliothek als auch die Dienste des Rechen- und Kommunikationszentrums. Im Rahmen der Einführung eines Identity Managements an der RWTH wurde nun beschlossen, mit jeder Person, die irgendwelche Dienstleistungen der RWTH in Anspruch nimmt, eine eindeutige Kennung, der sogenannten RWTH-ID, zu assoziieren, mit Hilfe derer sich aus den verschiedenen Datenbanken bei Bedarf die notwendigen Information synthetisieren lassen.

Anforderungen an die RWTH-ID

Die RWTH-ID wird also als Datenbankschlüssel verwendet werden und wird von Dienstleistern oft von Kunden erfragt werden. Deshalb sollte die RWTH-ID folgende Eigenschaften haben:

Großer Zahlenvorrat: Für die nächsten Jahrzehnte sollen ausreichend viele Kundennummern zur Verfügung stehen, ohne alte Nummern erneut verwenden zu müssen. Konservativ gehen wir von 10.000 neuen pro Jahr zu vergebenden Kundennummern aus.

Anonymität: Aus der RWTH-ID sollen keine personenbezogenen Informationen ablesbar sein. Zum Beispiel werden Matrikelnummern fortlaufend vergeben, woraus sich in etwa das Eintrittsdatum erkennen lässt.

Leichte Merkbarkeit: Die RWTH-ID sollte sich leicht merken lassen und leicht als RWTH-ID erkannt werden.

Robustheit: Einfache Übermittlungs- oder Tippfehler einer gültigen RWTH-ID sollten leicht erkannt werden, d.h. keine gültige RWTH-ID ergeben. Der Kunde kann in solch einem Fall dann aufgefordert werden, seine RWTH-ID neu einzugeben.

Zunächst ist zu entscheiden, aus welchem Symbolen die RWTH-ID gebildet werden soll: Beschränkt man sich auf Ziffern, so benötigt man 6 Ziffern für eine Million verschiedener Kundennummern, und zusätzliche Ziffern für Prüfnummern. Bildet man die Kundennummer jedoch aus den 26 Buchstaben des Alphabets unter Einbeziehung von Groß- und Kleinschreibung und der zehn Ziffern, so hat man 62 Symbole zur Verfügung. Damit kann man schon mit vier Symbolen 2^{64} = 14.776.336, also fast 15 Millionen, Kundennummern bilden.

Für die RWTH-ID wurde entschieden, unter Einbeziehung von Ziffern und Großbuchstaben 32 verschiedene alphanumerische Symbole zu verwenden, die in Tabelle 1 aufgeführt sind. Kleinbuchstaben und Symbole, die leicht verwechselt werden, finden keine Verwendung. Zum Beispiel unterscheiden sich die Symbole I und l nur wenig von 1, das Symbol O nur wenig von 0, und das Symbol V nur wenig von U, und deshalb werden I, J, O und V nicht verwendet. Damit kann man mit vier Symbolen 32^4 = 1.048.576 Kundennummern bilden, mit fünf Symbolen bereits 32^5 = 33 554 432, aus unserer Sicht eine komfortable Anzahl. Kommt noch ein Prüfsymbol dazu, so erhält man eine RWTH-ID, die aus sechs dieser Symbole besteht.

Um den Wiedererkennungswert der RWTH-ID zu steigern, vereinbaren wir eine einheitliche Schreibweise, die immer verwendet wird, wenn eine RWTH-ID auf Bescheinigungen, Anträgen, Ausweiskarten oder Internetseiten ausgegeben wird, und eine konstante Länge und die Gruppierung der einzelnen Symbole. Man kennt solche Verfahren etwa von Kreditkarten,

auf denen die Kontonummer in Vierergruppen dargestellt wird. Im Fall der RWTH-ID werden immer genau sechs Symbole verwendet, wobei zwei Gruppen von je drei Zeichen durch einen Bindestrich voneinander getrennt werden: etwa SL8-BRX.

Wenn wir eine RWTH-ID zufällig wählen, haben wir so also die ersten drei Anforderungen erfüllt. Was die Robustheit der RWTH-ID angeht, so sind die bei weitem häufigsten Tippfehler bei Tastatureingaben einzelne falsche Symbole oder das Vertauschen zweier benachbarter verschiedener Symbole (sogenannte „Dreher"); eine Statistik dazu findet man etwa in [4, Ch.1.2]. Um dies zu erreichen, nutzen wir die Struktur unseres Problems und die Möglichkeiten der Kodierungstheorie.

Kodierung der RWTH-ID und Polynome

Zur Fehlererkennung wird bei der RWTH-ID folgender Weg beschritten: Zunächst werden die verwendeten Symbole in Folgen von fünf Binärziffern, also 0 oder 1, übersetzt (siehe Tabelle 1).

00000	0	01000	8	10000	G	11000	R
00001	1	01001	9	10001	H	11001	S
00010	2	01010	A	10010	K	11010	T
00011	3	01011	B	10011	L	11011	U
00100	4	01100	C	10100	M	11100	W
00101	5	01101	D	10101	N	11101	X
00110	6	01110	E	10110	P	11110	Y
00111	7	01111	F	10111	Q	11111	Z

Tabelle 1: Kodierung der für die RWTH-ID verwendeten Zeichen

Da es gerade $2^5 = 32$ solcher Binärfolgen gibt, erklärt dies auch die Anzahl der ausgewählten Symbole. Eine RWTH-ID entspricht also einer Binärfolge der Länge 30, die alphanumerischen Symbole werden nur zur Eingabe und zur Ausgabe benutzt. Für SL8-BRX erhält man etwa

$$[11001 \mid 10011 \mid 01000 \mid 01011 \mid 11000 \mid 11101],$$

wobei wir hier zur besseren Lesbarkeit Trennstriche eingefügt haben.

Mit der Binärfolge $[a_0, a_1, \ldots, a_d]$ der Länge $d+1$ kann man ein Polynom in der Unbestimmten x assoziieren, nämlich

$$f := a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_dx^d.$$

Wählt man die obere Summationsgrenze d so klein wie möglich, was also $a_d = 1$ impliziert, so heißt d der Grad von f ; wir schreiben Grad $(f) = d$. Für Polynome gibt es Regeln für die üblichen Grundrechenarten, siehe etwa [3].

Die für uns wichtigste Eigenschaft von Polynomen ist, dass sie analog zu den ganzen Zahlen eine Division mit Rest erlauben: Sind f und $g \neq 0$ Polynome, so gibt es Polynome q und f mit Grad$(f) <$ Grad(g) und $f = q \cdot g + r$. Hat f bei Division durch g den Rest $r=0$, so heißt g ein Teiler von f. Ist das Polynom 1 der einzige gemeinsame Teiler von f und g ist, so heißen f und g teilerfremd. Es ist leicht zu sehen, dass f genau dann teilerfremd zu den Polynomen $x^j; j>0$ ist, wenn f teilerfremd zu x ist, also genau dann wenn $a_0 \neq 0$.

Einer RWTH-ID entspricht also ein Polynom vom Grad kleiner 30 (wenn die letzten Ziffern Nullen sind). Da wir Prüfinformation einfließen lassen wollen, gibt es ***zulässige*** und ***unzulässige Polynome***, also solche, die zu einer gültigen RWTH-ID gehören, und solche, die dies nicht tun. Dazu wählen wir ein geeignetes sogenanntes Erzeugerpolynom g mit Grad$(g) = k$, wobei $0 < k < 30$ ist, und lassen genau diejenigen Polynome f mit Grad$(f) < 30$ zu, die bei Division durch g den Rest $r=0$ haben. Wie in [1] gezeigt, gibt es dann genau 2^{30-k} zulässige Polynome. Wenn wir also ein geeignetes Polynom g vom Grad $k=5$ finden können, haben wir bei der 30-bit langen RWTH-ID dann immer noch über 33 Mio. zulässige Zeichenkombinationen.

Tippfehler und Dreher aus Sicht der Kodierungstheorie

Bei der Bestimmung eines geeigneten Erzeugerpolynoms nutzt man die Tatsache aus, dass Fehler bei der Eingabe der RWTH-ID aufgrund der Kodierung der Eingabezeichen in bestimmten Fenstern des Bitmusters auftauchen. Damit ist gemeint, dass ein Tippfehler maximal 5 Bits in Positionen 1-5, 6-10, 11-16 etc. verändern kann, aber nicht 5 Bits in Positionen 3-7, da hier eine Buchstabengrenze überschritten wird.

Aufgrund der Eigenschaften der Polynomdivision kann man dann folgende Forderungen an das Erzeugerpolynom g herleiten, siehe [1]:

1. Die Erkennung eines Tippfehlers ist dann gegeben, wenn Grad$(g) \geq 5$ und g teilerfremd zu x ist.
2. Ein Dreher, also das Vertauschen zweier nebeneinander liegender Zeichen, wird erkannt, wenn zudem g teilerfremd zu $1+x^5$ ist.

Wir sehen, also, dass wir k nicht kleiner als 5 wählen dürfen wenn wir Tippfehler erkennen wollen. Andererseits wollen wir k möglichst klein wählen um möglichst viele zulässige Polynome zu erhalten. Also wählen wir $k=5$.

Mit einem Computeralgebra-System, z.B. GAP [2], kann man 8 Polynome vom Grad 5 finden, die die obigen Bedingungen erfüllen. Diese Polynome sind unter dem Aspekt der Fehlererkennung gleich gut. Für die RWTH-ID wählen wir das bei [5] erwähnte, sogenannte USB-5 Polynom

$$g = 1 + x^2 + x^5$$

Generierung und Prüfung von RWTH-IDs

Wir zeigen nun an einem Beispiel, wie die Generierung und Verifikation von RWTH-IDs mittels des vorgestellten Verfahrens umgesetzt wird.

Generierung

Man wählt zunächst zufällig eine Folge von fünf Symbolen aus Tabelle 1, also etwa L8BRX, entsprechend der Bitfolge

[10011 | 01000 | 01011 | 11000 | 11101]

und dem Polynom

$$f_1 = 1 + x^3 + x^4 + x^6 + x^{11} + x^{13} + x^{14} + x^{15} + x^{16} + x^{20} + x^{21} + x^{22} + x^{24}:$$

Wir bilden $x^5 \cdot f_1$, dies entspricht der obigen Bitfolge, aber mit 5 zusätzlichen Nullbits am Anfang. Division von $x^5 \cdot f_1$ durch g ergibt den Rest $f = 1 + x + x^4$. Daraus folgt dann, dass $f = f_2 + x^5 \cdot f_1$ bei Division durch g den Rest 0 hat.

Zum Polynom f_2 gehört die Binärfolge [11001], entsprechend dem Symbol S. Zum Polynom f gehört die Verkettung der Binärfolgen von f_2 und f_1, entsprechend der Ziffernfolge SL8BRX, die wir in unserer RWTH-Schreibweise als SL8-BRX darstellen.

Verifikation:

Zur Prüfung von SL8-BRX entsprechend dem Polynom f, betrachtet man formal die Verkettung von [00000] mit der Binärfolge zu f, also

[00000 | 11001 | 10011 | 01000 | 01011 | 11000 | 11101].

Das zugehörige Polynom $x^5 \cdot f$ hat in der Tat bei Division durch g den Rest 0. Ein nicht zulässiges Polynom würde einen Rest ungleich 0 ergeben. Aufgrund des Designs g von wären wir sicher, so Tippfehler in einer Ziffer oder Dreher erkennen zu können, an welcher Stelle in der Zeichenfolge sie auftreten, können wir aber nicht feststellen.

Algorithmische Implementierung:

Sowohl für die Generierung als auch die Verifikation wird also genau derselbe Algorithmus benutzt, nämlich für ein gegebenes Polynom wird der Rest von bei Division durch $g = 1 + x^2 + x^5$ berechnet. Diese Operation lässt sich mit einem sogenannten „rückgekoppelten Schieberegister" sehr einfach ausführen. Der entsprechende Algorithmus ist in Abbildung 1 gezeigt und liefert bei der Eingabe einer einem Polynom f entsprechenden Bitfolge „bitseq" den Rest bei Division durch das der Bitfolge „genseq" entsprechende Polynom g.

```
/*_ fseq: Binärfolge der Länge d _*/
/*_ gseq: Binärfolge der Länge k _*/

poldiv(fseq,d,gseq,k)
      local rem = (gseq and 0); /* Binärfolge der Länge k_*/
      for (i = 0; i < d; i++) {
            if (rem[k-1] exor fseq[d-1]) {
                  rem >>= 1;
                  rem = (rem exor gseq);
            } else {
            rem >>= 1;
        };
        fseq >>= 1;
        }return rem;
}
```

Abbildung 1: Algorithmus zur Generierung bzw. Verifikation einer RWTH-ID

Die Operationen and und exor sind die üblichen booleschen Operationen auf Bitfolgen, die Operation „>>=" bedeutet einen Rechts-Shift der Bitfolge, wobei links mit einem Nullbit aufgefüllt wird.

Erweiterbarkeit

Sollte sich der Zeichenvorrat mit 6 Ziffern irgendwann dem Ende zuneigen, so lässt sich der Code leicht auf beliebig viele Ziffern bei gleicher Robustheit erweitern, da mit SL8BRX auch SL8BRX000…. eine zulässige Zeichenfolge ist. Damit geht ein leichter Verlust von Anonymität einher, da es offensichtlich wird, dass man unter den ersten Millionen Nutzern war, aber das ist aus unserer Sicht zu verschmerzen.

Abschließende Bemerkungen:

In vielen Organisationen wird eine Kundennummer benötigt, die einfach zu merken, anonym, und gegenüber Fehlern robust sein soll. Mithilfe eines geeignet gewählten Zeichenvorrates konstruierten wir aufbauend auf Polynomdivision einen Algorithmus, der es erlaubt, mit 6 Ziffern über 33 Mio. Zeichenketten zu konstruieren, für die wir Tippfehler in einer Ziffer oder die Vertauschung von zwei Ziffern mit einem sehr einfachen Algorithmus sofort erkennen können.

Damit wird die Nutzungsfreundlichkeit dieser Ziffernfolge erheblich gesteigert, da man noch vor der Nutzung die Validität im Hinblick auf die gängigsten Fehler sicherstellen kann. Durch Verlängerung der Zeichenkette können leicht in einer rückwärtskompatiblen Art und Weise auch größere Zahlenräume erschlossen werden.

Insgesamt wurde so unter Zuhilfenahme der mathematischen Kodierungstheorie ein algorithmisch sehr einfacher Ansatz gefunden, um ein reales Problem im Umfeld des Identitätsmanagements in einer für die Kunden und Betreiber eines solchen Systems gleichermaßen komfortablen Art und Weise zu lösen.

Literatur:

[1] G. Bunsen, J. M ler, Die RWTH Kundennummer, Preprint 2006, http://www.math.rwth-aachen.de/~Juergen.Mueller/

[2] The GAP Group: GAP-4.4 | Groups, Algorithms, and Programming, 2006, http://www.gap-system.org.

[3] J. von zur Gathen, J. Gerhard: Modern computer algebra, second edition, Cambridge University Press, 2003.

[4] D. Jungnickel: Codierungstheorie, Spektrum Verlag, 1995.

[5] P. Koopman, T. Chakravarty: Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks, Preprint: The International Conference on Dependable Systems and Networks, DSN-2004.