Diffie-Hellman, RSA, etc.

mathematische Grundlagen asymmetrischer Verschlüsselungsverfahren

Sven Moritz Hallberg
pesco@hamburg.ccc.de

SIGINT '09, 22.–24. Mai 2009

Zusammenfassung

Inzwischen sind kryptographische Techniken und Verfahren auch auf dem letzten Home-PC angekommen. Leider gilt das Thema in den meisten Köpfen als mystische "Black Art". Es soll eine allgemeinverständliche Einführung in die mathematischen Grundlagen der asymmetrischen Verschlüsselungsverfahren gegeben werden, die das Rückgrat moderner Kryptoinfrastruktur bilden. Dem Leser soll der Schlüssel gegeben werden, kryptographische Systeme, die auf diesen Primitiven aufbauen, mehr als nur oberflächlich beurteilen zu können.

Grundlagen

Vorausgesetzt werden elementare Notationen auf Schulniveau, sowie grundsätzliche Kenntnis vom Konzept der Teilbarkeit, Primzahlen sowie Modulodivision.

Die nötigen Grundlagen zu Gruppen und Restklassen werden im folgenden kurz zusammengefasst. Dabei sei angemerkt, daß das Ziel der Präsentation nicht in absoluter Rigorosität liegt, sondern einen Balanceakt zwischen mathematischer Genauigkeit und überschaubarer Kürze darstellt.

Gewisse Sätze werden insbesondere zum Thema RSA ohne Beweis angegeben und verwendet. Man sollte diese ohne Scheu annehmen und bei Interesse in der einschlägigen Literatur nachlesen.

Gruppen

Die Struktur der sogenannten *Gruppe* bildet quasi die "Bühne", auf der im folgenden die Rechnungen ablaufen werden. Es handelt sich, kurz gesagt, um die Abstraktion einer Menge von Zahlen oder anderen Objekten, mit denen sich "wie üblich" rechnen läßt.

Definition (Gruppe)**.** Eine nichtleere Menge M wird gemeinsam mit einer Verknüpfung $(\cdot) : M \times M \to M$ als Gruppe bezeichnet, wenn folgendes gilt:

- Die Verknüpfung $(\cdot)$ ist assoziativ.
- Es existiert ein *neutrales Element* $1 \in M$ mit $1 \cdot x = x$ für alle $x \in M$.
- Zu jedem $x \in M$ existiert ein *Inverses* $x^{-1} \in M$, so daß $xx^{-1} = 1$.

Es wird im folgenden vorallem um Exponentiationen gehen, die im Kontext einer Gruppe wie üblich definiert werden:

$$x^n := \prod_{i=1}^{n} x \quad , x \in M, n \in \mathbb{N}$$

Restklassen

Die Rechnungen im folgenden werden hauptsächlich mit ganzen Zahlen modulo einer bestimmten Zahl n arbeiten. Das mathematische Konstrukt hierzu sind *Restklassen*. Dabei werden alle Zahlen, die den gleichen Divisionsrest modulo n haben, zu einer "Klasse" zusammengefasst. Es gibt dann logischerweise $n - 1$ verschiedene *Restklassen modulo* n. Die Menge dieser Klassen wird im folgenden mit $\mathbb{Z}_n$ bezeichnet und in der Regel mit den Zahlen von 0 bis $n - 1$ identifiziert. Es gelten die üblichen Rechenregeln, wobei bei "Überlauf" wieder modulo n reduziert wird.

Interessant ist die Frage, wann zu einem $x \in \mathbb{Z}_n$ ein Inverses bezüglich der Multiplikation existiert. Es ist ein elementares Resultat der Zahlentheorie, daß dies genau dann der Fall ist, wenn x teilerfremd zu n ist.

RSA

Das allgegenwärtige RSA-Verfahren[6] bezieht seine Sicherheit, wie weithin bekannt, aus der Schwierigkeit, Produkte grosser Primzahlen zu faktorisieren. Wie die Primfaktorzerlegung mit RSA-Schlüsseln zusammenhängt, soll nun erklärt werden.

Den *algorithmischen* Kern von RSA bildet modulare Exponentiation. D.h. die Berechnung von Werten der Form

$$m^e \pmod{n} \quad .$$

Die Buchstaben m und e sind nicht zufällig gewählt, denn es handelt sich hier um den Verschlüsselungsteil des RSA-Verfahrens. Die Zahl m sei die zu chiffrierende Nachricht und der sogenannte *Verschlüsselungsexponent* e bildet zusammen mit dem Modulus n den public key.

Zur Entschlüsselung muss also diese Exponentiation rückgängig gemacht werden. Dazu werden wir sehen, daß modulo n ein Exponent d existiert, so daß

$$(m^e)^d = m^{ed} \equiv m \pmod{n}$$

gilt. Den nötigen "Trick" liefert der *Eulersche Satz*:

Satz (Euler)**.** *Seien $a, n \in \mathbb{N}$ teilerfremd. Bezeichne $\varphi(n)$ die Anzahl der zu n teilerfremden Zahlen kleiner/gleich n (Eulersche φ-Funktion). Dann gilt*

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad .$$

Setzen wir also einmal voraus, m sei teilerfremd zu n. Dann ergibt der Satz, daß

$$m = m \cdot 1 \equiv m \cdot m^{\varphi(n)} \quad .$$

Natürlich darf der Faktor 1 auch beliebig oft auftreten, d.h. es gilt

$$m \equiv m \cdot m^{k\varphi(n)} = m^{1+k\varphi(n)}$$

für alle $k \in \mathbb{N}$. Können wir also d und k mit

$$ed = 1 + k\varphi(n)$$

finden, so ist d geeignet als *Entschlüsselungsexponent* zu e. Obige Gleichung bedeutet nichts anderes als

$$ed \equiv 1 \pmod{\varphi(n)} \quad .$$

Mit anderen Worten: d ist das (multiplikative) Inverse zu e modulo $\varphi(n)$. Dieses läßt sich, Existenz vorausgesetzt, mittels des *erweiterten Euclidischen Algorithmus* effizient aus e und $\varphi(n)$ berechnen. Es existiert genau dann, wenn e teilerfremd zu $\varphi(n)$ ist, was bei der Erzeugung des Schlüssels sichergestellt wird.

Kennt man also neben dem Modulus n den Exponenten d wie oben bestimmt, so kann man die Nachricht m^e dechiffrieren. In der Tat bilden d und n den RSA private key. Wie bereits erläutert, reicht aber zur leichten Bestimmung von d die Kenntnis von $\varphi(n)$. Um die Sicherheit des Systems zu gewährleisten, ist es also erforderlich, daß $\varphi(n)$ nur schwer aus n berechenbar ist. Die effizienteste derzeit bekannte Methode dazu verwendet in der Tat die Primfaktorzerlegung von n. Klassischer Weise wählt man also $n = pq$ mit zwei großen Primzahlen p und q, die die Faktorisierung hinreichend erschweren. In diesem Fall berechnet sich $\varphi(n)$ wie folgt.

Lemma. *Für Primzahlen p gilt*

$$\varphi(p) = p - 1 \quad .$$

Für teilerfremde Zahlen $m, n \in \mathbb{N}$ gilt

$$\varphi(nm) = \varphi(n)\varphi(m) \quad .$$

Sei $n = pq$ mit $p, q \in \mathbb{N}$ prim. Dann gilt also:

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$$

Zuletzt bleibt noch eine Lücke in den Betrachtungen zu schließen. Wir sind oben davon ausgegangen, die Nachricht m sei teilerfremd zum Modulus n. Bei der Form $n = pq$ ist das auch sehr wahrscheinlich, aber nicht garantiert. Interessanterweise läßt sich die Annahme aber als unnötig beweisen! Dazu betrachtet man wie zuvor

$$m^{ed} = m^{1+k\varphi(n)} = m \cdot m^{k\varphi(p)\varphi(q)} \quad ,$$

jedoch vorerst nur modulo p bzw. q. Und zwar gilt analog zu oben, falls m teilerfremd zu p ist:

$$m^{ed} = m \cdot (m^{k\varphi(q)})^{\varphi(p)} \equiv m \cdot 1 \pmod{p}$$

Diese Kongruenz gilt aber auch falls m *nicht* teilerfremd zu p ist: Dann ist m ein Vielfaches von p und beide Seiten kongruent 0. Analoges gilt modulo q. Wir wissen also:

$$m^{ed} \equiv m \pmod{p}$$
$$m^{ed} \equiv m \pmod{q}$$

Hieraus folgt nach dem *Chinesischen Restsatz* unmittelbar das gesuchte Resultat:

$$m^{ed} \equiv m \pmod{pq}$$

Satz (Chinesischer Restsatz)**.** *Seien $m_1, \ldots, m_k$ paarweise teilerfremde natürliche Zahlen. Dann gilt folgende Isomorphie:*

$$\mathbb{Z}_{m_1 \cdot \ldots \cdot m_k} \simeq \mathbb{Z}_{m_1} \times \ldots \times \mathbb{Z}_{m_k}$$

Eine Anekdote zu RSA

Man stelle sich folgendes Szenario vor: Die persönliche Festplatte des Autors ist mit AES verschlüsselt. Zum Zugriff auf die Daten soll ein physischer Token nötig sein. Hierfür bietet sich aus dem Fundus des Autors eine RSA-fähige SmartCard an. Der AES-Schlüssel wird also RSA-chiffriert in einem unverschlüsselten Teil der Festplatte abgelegt. Leider beherrscht die SmartCard lediglich eine Schlüssellänge von 1024 Bit, was nicht mehr als sicher gilt. Allerdings wird zum "öffnen" der Festplatte ja nur der private key benötigt, der sicher auf der SmartCard liegt. Ein Faktorisierungsangriff ist also nicht möglich, solange der public key geheim bleibt. Bei der Implementierung des Systems stellt sich aber heraus, daß GnuPG die Arbeit mit der Karte verweigert, wenn der public key nicht vorliegt.

Frage. Ist das Verhalten von GnuPG eine behebbare technische Eigenheit oder ist der separate public key wirklich nötig?

Mit dem Wissen des vorigen Abschnitts wird deutlich, daß es tatsächlich denkbar wäre, ohne public key nicht entschlüsseln zu können:

$$\text{pub} = (e, n) \quad \text{priv} = (d, n)$$

Der Modulus n ist Teil beider Schlüssel und müsste theoretisch nicht auf der Karte gespeichert sein. Ist er es andererseits doch, so gibt es aus kryptographischer Sicht keinen Grund für das Verhalten der Software.

Schließlich findet sich im Web die Schnittstellenspezifikation der SmartCard und beantwortet die Frage: Der Modulus wird auf der Karte abgelegt. Happy patching!

Diffie-Hellman Key Exchange

Das zweite große Verfahren im Kontext von public key-Kryptographie ist a priori kein Verschlüsselungsverfahren. Das *Diffie-Hellman Schlüsselaustauschprotokoll*[1] erlaubt es Alice und Bob zunächst, abhörsicher ein shared secret zu vereinbaren, das im weiteren zur Verschlüsselung genutzt werden kann. Wie dieses System auf einfache Weise ein public key-Verfahren hervorbringt, wird weiter unten beschrieben.

Gerechnet wird beim Diffie-Hellman-Protokoll in einer endlichen Gruppe G_q der Ordnung q, wobei q eine Primzahl ist. Verwendet werden, ähnlich wie bei RSA, Exponentiationen in dieser Gruppe.

$$g^x \quad , g \in G_q, x \in \mathbb{N}$$

Im Unterschied zu RSA wird hier jedoch g ein festgelegtes Element sein, genauer gesagt ein *Erzeuger* von G_q. Das bedeutet, für jedes $g' \in G_q$ existiert ein $x \in \mathbb{N}$, so daß $g^x = g'$. Die Sicherheit des Verfahrens wird auf der Schwierigkeit beruhen, zu gegebenem g' ein solches x zu finden. Diese Aufgabe wird als *diskretes Logarithmus-Problem* (DL) bezeichnet.

Der Schlüsselaustausch verläuft wie folgt:

1. Alice wählt eine Zufallszahl a mit $0 \leq a < q$.
   Bob wählt eine Zufallszahl b mit $0 \leq b < q$.

2. Alice sendet $x := g^a$ an Bob.
   Bob sendet $y := g^b$ an Alice.

3. Alice berechnet $s_a := y^a = g^{ab}$.
   Bob berechnet $s_b := x^b = g^{ab}$.

Es kennen also am Ende beide den Wert

$$s := s_a = s_b \quad ,$$

der ihnen nun beispielsweise als geheimer Schlüssel für ein symmetrisches Verfahren dienen kann.

Man bemerke: Die Sicherheit des Systems ist dann gegeben, wenn sich g^{ab} nur schwer aus g^a und g^b berechnen läßt. Diese Aufgabe wird dementsprechend *Diffie-Hellman-Problem* (DH) genannt. Offensichtlich ist DH leicht zu lösen, wenn DL leicht zu lösen ist.

Achtung: Damit DH und DL tatsächlich schwer sind, muss G_q passend gewählt sein.

Gängig ist etwa, modulo einer Primzahl p für ein bestimmtes $g \in \mathbb{Z}_q$ einfach die Menge

$$\{\ g^x \mid 1 \leq x < p\ \}$$

zu betrachten. Diese bildet selbst eine Gruppe, deren Ordnung bei geeigneter Wahl von p und g prim ausfällt. Eine Reihe solcher Gruppen sind in [3] und [4] zur Verwendung in Internet-Protokollen definiert.

Eine ganz andere Möglichkeit bilden bestimmte Punktmengen auf sogenannten elliptischen Kurven, für die sich eine Art "Multiplikation" definieren läßt. Darauf basierende Systeme werden unter der Bezeichnung *elliptic curve cryptography* (ECC) zusammengefasst.

Ein Anekdote zu Diffie-Hellman

Man stelle sich folgendes Szenario vor: Ein Bekannter des Autors arbeitet als Entwickler von Computerspielen. Für eine Neuentwicklung soll Spielern, die via Internet gegeneinander antreten, mit kryptographischen Mitteln der Betrug erschwert werden. Die Anforderungen lauten:

- Moderate Sicherheit. Das System soll "Schummeln" nicht unmöglich machen, lediglich erschweren.
- Eigenentwicklung. Die Firmenpolitik schreibt vor, die Lösung selbst zu entwickeln.
- Geringer Programmieraufwand.
- Sehr geringer Platzaufwand pro ausgetauschtem Datenpaket.
- Geringer Rechenaufwand pro ausgetauschtem Datenpaket.

Es steht zunächst die Idee im Raum, jedes übertragene Datenpaket kryptographisch zu signieren. Allerdings wären sowohl Platz- als auch Rechenaufwand eines asymmetrischen Signaturverfahrens schon bei minimaler Sicherheit sehr hoch.

Ein einzelner DH-Schlüsselaustausch zu Beginn des Spiels hingegen kann problemlos durchgeführt werden. Mit dem so vereinbarten Schlüssel wird jedem Datenpaket eine chiffrierte Prüfsumme hinzugefügt. Das dazu eingesetzte symmetrische Verfahren erfüllt die Anforderungen an Platz- und Zeiteffizienz. Offensichtlich ist ohne Kenntnis des Schlüssels eine korrekte Angabe der Prüfsumme nicht möglich. Diese Lösung ist ein sogenanntes *message authentication code* (MAC) Verfahren [7].

Mit dem Wissen des vorangegangenen Abschnitts ist klar, daß zur Umsetzung im wesentlichen die modulare Exponentiation programmiert werden muss. Ein Standard-Algorithmus dazu ist die sogenannte *Montgomery-Multiplikation*[5].
Happy Hacking!

ElGamal-Verschlüsselung

Wie bereits erwähnt, läßt sich aus dem Diffie-Hellman-Protkoll auf einfache Weise ein public key Verschlüsselungssystem ableiten [2]. Dazu seien G_q und g wie oben vorgegeben. Ein Nutzer des Systems wählt nun einen zufälligen Exponenten a und veröffentlicht g^a.

$$\text{pub} = g^a \quad \text{priv} = a$$

Um diesem Nutzer eine Nachricht m zu schicken, wähle man den zweiten zufälligen Exponenten b und bilde $s = (g^a)^b$. Die verschlüsselte Nachricht besteht nun aus zwei Teilen:

$$\text{cmsg} = (g^b,\ m \cdot s)$$

Der Empfänger kann daraus $s = (g^b)^a$ und somit $m = m \cdot s/s$ berechnen.

Literatur

[1] Diffie, Whitfield und Martin E. Hellman: *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

[2] ElGamal, Taher: *A public key cryptosystem and a signature scheme based on discrete loga-*

rithm. In: *IEEE Transactions on Information Theory*, Band 31, Seiten 465–472, 1985.

[3] Harkins, D. und D. Carrel: *RFC 2409 – The Internet Key Exchange (IKE)*, 1998. `http://www.faqs.org/rfcs/rfc2409.html`.

[4] Kivinen, T. und M. Kojo: *RFC 3526 — More Modular Exponential (MODP) Diffie-Hellman groups*, 2003. `http://www.faqs.org/rfcs/rfc3526.html`.

[5] Montgomery, Peter L.: *Modular Multiplication Without Trial Division.* Mathematics of Computation, 44(170):519–521, 1985.

[6] R. Rivest, A. Shamir und L. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120–126, 1978.

[7] Schneier, Bruce: *Applied Cryptography.* Wiley & Sons, 1996.