Effizienzverbesserungen durch schlüssel-optimierte Ver- und Entschlüsselung in Workstations

Dr. Ingrid Biehl, Techn. Universität Darmstadt, FB 20 Informatik, D-64283 Darmstadt, Germany
Prof. Dr. Jörg Keller, FernUniversität-GHS Hagen, FB Informatik, D-58084 Hagen, Germany

Kurzfassung

Arbeitsplatzsysteme verarbeiten zunehmend sehr große Datenmengen z.B. in Form digitaler Ton- und Videodaten. Sind diese vertraulich über unsichere Datenwege wie das Internet zu versenden, so müssen sie dazu verschlüsselt werden. Eine reine Software-Lösung der Verschlüsselungsaufgabe führt dabei zu einer (wenigstens zeitweise) erheblichen Belastung des jeweiligen Rechnersystems. Abhilfe können Spezialhardwarebausteine bieten, die dann allerdings nur zu diesem Zweck geeignet sind.
Wir diskutieren im vorliegenden Beitrag die Idee, statt individueller Schlüssel Hardware-Beschreibungen, die eine auf den jeweiligen Schlüssel optimierte Versionen der jeweiligen Verschlüsselungsverfahren darstellen, und programmierbare Hardware-Bausteine (FPGA) zu verwenden. Dabei betrachten wir DES, IDEA und RSA als gängige Verschlüsselungsverfahren und untersuchen, ob bei Einsatz der obigen Vorgehensweise mit Effizienzverbesserungen gegenüber der gängigen reine Software-Lösung zu rechnen ist.

1 Einleitung

Arbeitsplatzsysteme arbeiten zunehmend mit digitalen Medien wie Video und Ton. Oft sind die enthaltenen Mitteilungen vertraulich und müssen deshalb vor einer Übertragung per Internet verschlüsselt werden. Die Ver– und Entschlüsselung der dabei anfallenden Datenmengen in Realzeit überfordert oft die Leistungsfähigkeit des Prozessors des Arbeitsplatzsystems, oder führt zu nicht akzeptierbaren Leistungseinbrüchen bei anderen, gleichzeitig betriebenen Anwendungen. Deshalb wird die Realzeit-Ver– und Entschlüsselung großer Datenmengen oft durch spezielle Hardware unterstützt. Dies ist ökonomisch nicht sehr sinnvoll, da eine solche Hardware über große Zeiträume hinweg nicht benutzt wird. Außerdem ist der Anwender mit der verwendeten Hardware auch auf das jeweils implementierte Verfahren festgelegt.
Eine Lösung dieses Problems bieten programmierbare Hardwarebausteine wie FPGAs (Field Programmable Gate Arrays), in die man die zu realisierende Schaltung lädt und dann direkt nutzen kann. Ein derartiger Zusatzbaustein kann zu verschiedenen Zeiten verschiedene rechenintensive Zusatzfunktionen erfüllen, und ist auf jede davon besser zugeschnitten als ein General Purpose Prozessor. Einschubkarten mit FPGAs sind derzeit von mehreren Herstellern im Handel erhältlich. Mit gleicher Intention hat man eine ähnliche Konfiguration bei den sogenannten Custom Computing Machines gewählt. Diese bestehen vereinfacht dargestellt aus einem Mikroprozessor Core und mehreren FPGAs auf einem Chip. Damit ist die Anbindung zwischen Prozessor und FPGA sogar enger als in der von uns beschriebenen Variante mit Zusatzkarte.
Der Nachteil von FPGAs ist aber die geringere Taktrate im Vergleich zu Spezialhardware. Dies behindert ihren Einsatz zur Verschlüsselung großer Datenmengen in Realzeit.
Wir wollen im vorliegenden Beitrag zeigen, daß man diesen Nachteil beheben kann, indem man den verwendeten Verschlüsselungs-Algorithmus stets auf den gerade verwendeten Schlüssel anpaßt (im folgenden *schlüssel-optimierter Algorithmus* genannt) und damit beschleunigt. Dies ist möglich, da die Konfiguration der FPGAs immer wieder geändert werden kann. In Abschnitt 2 leiten wir aus der grundsätzlichen Idee eine Architektur her, die uns die Anpassung erlaubt. Hierbei arbeiten wir auch die Unterschiede zwischen symmetrischen, public key, und hybriden Verfahren heraus. In Abschnitt 3 untersuchen wir verschiedene populäre Verschlüsselungs-Algorithmen daraufhin, welche Beschleunigung man bei ihrer Verwendung zu erwarten hat. In Abschnitt 4 geben wir einen Ausblick über weitere Anwendungen der optimierten Verschlüsselung.

2 Optimierung von Verschlüsselungs-Algorithmen

Bei einer Verschlüsselung werden eine Nachricht x und ein Schlüssel k durch arithmetische und logische Operationen miteinander verknüpft. Ein Beispiel für solche Operationen ist die Multiplikation eines Teils der Nachricht mit einem Teil des Schlüssels. Für einen festen Schlüssel hätte man nun eine Multiplikation mit einer Konstanten, die man effizienter in Hardware implementieren kann als eine allgemeine Multiplikation.
Bei secret key Verfahren haben Sender und Empfänger den gleichen geheimen Schlüssel, der zum Ver- und Entschlüsseln dient. Statt des geheimen Schlüssels werden beide Kommunikationspartner in unserem System auf diesen Schlüssel optimierte Ver- und Entschlüsselungsverfahren verwenden.
Bei public key Verfahren sind für beide Operationen verschiedene Schlüssel notwendig. Deshalb kann der Schlüssel zum Verschlüsseln öffentlich gemacht werden. Solange der Schlüssel zum Entschlüsseln nur dem Empfänger bekannt ist, kann niemand außer diesem verschlüsselte Daten entschlüsseln, aber alle Sender, die Zugriff auf den öffentlichen Schlüssel haben, können diesem Empfänger Nachrichten verschlüsselt zukommen lassen. Diese öffentlichen Schlüssel werden in einem Schlüsselverzeichnis (key directory) zugänglich gemacht. Um nun optimierte Verschlüsselungsverfahren nutzen zu können, kann man statt eines öffentlichen Schlüssels k die Hardware-Beschreibung eines auf diesen Schlüssel optimierten Verschlüsselungs-Algorithmus im Schlüsselverzeichnis abspeichern. Entsprechend kann natürlich auch der Empfänger einen auf seinen geheimen (Entschlüsselungs-) Schlüssel optimierten Entschlüsselungs-Algorithmus verwenden.
Eine Vielzahl von Varianten sind für die Handhabung schlüssel-optimierter public key Verschlüsselungsverfahren denkbar. Eine Möglichkeit besteht darin, daß die für die jeweiligen öffentlichen Schlüssel optimierten Verschlüsselungsverfahren in öffentlichen Schlüsselverzeichnissen zugänglich sind. Da die Optimierung nur einmal, nämlich bei der Erzeugung des Schlüssels, vorgenommen werden muß, können auch rechenzeit-intensive Optimierungen zur Anwendung kommen. Eine weitere Möglichkeit ist, daß die optimierten Verfahren aus einem öffentlichen Schlüssel vom jeweiligen Sender erzeugt werden können. Hierbei bleibt das Schlüsselverzeichnis unverändert, die Optimierung darf aber nicht zuviel Zeit verbrauchen. Als dritte Möglichkeit bietet sich an, daß schlüsselabhängige Teile der Beschreibung im Verzeichnis abgelegt sind, die dann effizient zum vollständigen Verfahren ergänzt werden können (siehe Abschnitt 3.3).
Da alle gängigen public key Verschlüsselungsverfahren wesentlich zeitintensiver sind als secret key Verfahren werden beide Techniken in der Regel in sogenannten Hybridverfahren zum Verschlüsseln großer Datenmengen kombiniert. Zunächst wird ein Sitzungsschlüssel des secret key Verfahrens zufällig ausgewählt, mittels dessen die Daten dann verschlüsselt werden. Der Sitzungsschlüssel wiederum wird mittels des public key Verschlüsselungsverfahrens verschlüsselt und den verschlüsselten Daten vorangestellt. Effizienzsteigerungen kann man sich durch Verwendung eines schlüssel-optimierten public key Verschlüsselungsverfahrens und durch die Generierung und Anwendung des schlüssel-optimierten secret key Verfahrens zum Sitzungsschlüssel erhoffen. Statt des Sitzungsschlüssels kann dann auch der schlüssel-optimierte secret key Verschlüsselungsalgorithmus mit dem public key Verfahren verschlüsselt und den verschlüsselten Daten vorangestellt werden. Die dadurch bewirkte Vergrößerung der zu übertragenden Daten stellt praktisch kein nennenswerter Nachteil dar, da die Latenz von Nachrichten im Internet wegen der Paketgrößen erst ab Größen von mehr als 1kByte deutlich steigt. Alternativ dazu kann der zum Sitzungsschlüssel gehörige optimierte Verschlüsselungsalgorithmus auch vom Empfänger nach Entschlüsseln des Sitzungsschlüssels berechnet werden. Dann darf die Optimierung allerdings nicht zuviel Zeit kosten.
Für die Anwendung in der Praxis ist es notwendig, eine standardisierte, plattformunabhängige und gleichzeitig kompakte Hardwarebeschreibungssprache zu verwenden. Da oft aber nur ein Teil eines Verschlüsselungsschaltkreises beschrieben werden muß, sind Beschreibungen denkbar, die größenordnungsmäßig die Größe von öffentlichen Schlüsseln eines public key Verfahrens (z.B. RSA mit 1024 bit langen Schlüsseln) nicht wesentlich übersteigen.

3 Anwendbarkeit bei populären Verfahren

Wir wollen nun an drei Verschlüsselungs-Algorithmen untersuchen, ob durch die im vorhergehenden Abschnitt hergeleitete Anpassung eines Algorithmus an einen Schlüssel überhaupt eine Beschleunigung erwarten läßt und welchen Umfang diese haben könnte. Eine Beschreibung aller untersuchten Verfahren findet man zum Beispiel bei Schneier [3].

3.1 DES

DES ist derzeit ein de facto Standard bei symmetrischer Verschlüsselung. Allerdings hat es in jüngerer Zeit wegen seiner kleinen Schlüssellänge (64 Bit, von denen nur 56 benutzt werden) mehrere erfolgreiche Angriffe gegen DES gegeben (siehe zum Beispiel die RSA Competition, http://www.rsa.com und den EFF DES Cracker, http://www.eff.org/pub/Privacy/Crypto_misc/DESCracker) Deshalb wird DES zunehmend durch modernere Verfahren ersetzt. Eine verbreitete Alternative zum DES ist der IDEA-Algorithmus, den wir im nächsten Abschnitt untersuchen.
DES arbeitet in 16 gleichartigen Runden. In jeder Runde werden der Schlüssel und die Nachricht separat verarbeitet, aus dem Schlüssel wird ein Rundenschlüssel berechnet und mit dem aus der Nachricht in den vorangegangenen Runden berechneten Zwischenergebis durch ein bitweises exklusives Oder verknüpft. Für einen festen Schlüssel kann die Schlüsselverarbeitung entfallen, da die Rundenschlüssel jeder Runde bekannt sind. Die bitweise EXOR-Verknüpfung wird für ein Bit zur Identität, falls das betreffende Bit des Schlüssels 0 ist, ansonsten zur Negation. Die Verringerung der Schaltkreistiefe hierdurch ist allerdings unbedeutend. Viel gravierender aber wirkt die Tatsache, daß das Ergebnis der optimierten Verknüpfung weiter von der Nachricht abhängt, und also keine weiteren Optimierungen stattfinden können.

3.2 IDEA

Der IDEA-Algorithmus verwendet 128 Bit große Nachrichten und Schlüssel, die jeweils in acht 16-Bit große Blöcke zerlegt werden. In acht gleichartigen Runden werden nun solche Blöcke durch Multiplikationen, Additionen, und bitweise EXOR-Operationen verknüpft. Hierbei wird bei jeder Multiplikation und der Hälfte der Additionen ein Nachrichtenblock und ein Schlüsselblock miteinander verknüpft. Man erhält in diesen Fällen eine Multiplikation bzw. eine Addition mit einer Konstanten und somit eine Reduktion der Tiefe und der Kosten des Schaltkreises. Der IDEA-Algorithmus ist also sehr gut für die vorgeschlagene Architektur geeignet.
Um den Grad der Beschleunigung zu schätzen, beschränken wir uns auf die Multiplikationen, da diese die Tiefe des Schaltkreises dominieren. Die beiden wichtigsten Arten von Multiplizierern, die zum Einsatz kommen, sind der Array-Multiplizierer und der Wallace-Tree-Multiplizierer. Bei der Multiplikation einer Zahl a mit einer Zahl

$$b = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

bildet ein Array-Multiplizierer nacheinander die Teilsummen $a \cdot b_i$ und summiert die verschobenen Teilsummen in $n-1$ Additionen auf. Ist b eine Konstante, dann spart man alle die Additionen, bei denen $b_i = 0$ ist. Um auch bei Zahlen einen Gewinn zu erzielen, deren Binärdarstellung viele Einsen hat, benutzt man die Identität

$$\sum_{i=0}^{n-1} b_i \cdot 2^i = 2^n - 1 - \sum_{i=0}^{n-1} \overline{b_i} \cdot 2^i \,. \tag{1}$$

Hat eine Binärdarstellung also mehr als $n/2+1$ Einsen, so rechnet man gemäß der rechten Seite der Gleichung, und benötigt 2 Subtraktionen und höchstens $n/2-3$ Additionen, da das bitweise Inverse von b höchstens $n/2-2$ Einsen hat. Hat eine Binärdarstellung höchstens $n/2+1$ Einsen, so rechnet man gemäß der linken Seite der Gleichung und benötigt höchstens $n/2$ Additionen. In jedem Fall kann also die Tiefe eines Array-Multiplizierers etwa halbiert werden, wenn ein Faktor eine Konstante ist. Die zu erwartende Beschleunigung ist hier also enorm. Es gibt noch bessere Verfahren, zum Beispiel den Bernstein-Algorithmus [1], deren minimale Verbesserung allerdings nicht so einfach geschätzt werden kann.
Ein Wallace-Tree-Multiplizierer faßt die Teilergebnisse einer Multiplikation mittels eines balancierten Baumes aus Addierern zusammen. Reduziert man mittels obiger Identität (1) die Anzahl zu addierender Teilergebnisse auf etwa die Hälfte, so hat der Baum eine Addiererstufe weniger. Die zu erwartende Beschleunigung im Gatter-Modell ist also gering, da in allen Stufen des Baumes außer der Wurzel eine redundante Zahlendarstellung benutzt wird und so der letzte Addierer ungefähr die gleiche Tiefe hat wie alle anderen Stufen vorher zusammen. Bei einer 16-Bit Multiplikation gibt es im allgemeinen Fall 16 Teilergebnisse, also hat der Baum 4 Stufen, von denen die ersten drei Stufen jeweils ein Sechstel der Tiefe beitragen und die letzte die Hälfte. Das Einsparen einer Stufe reduziert also die Tiefe um ein Sechstel. Die Betrachtung im VLSI-Modell, bei dem auch Leitungslängen und –verzögerungen eine Rolle spielen, läßt aber eine größere Reduktion erwarten, wie eine ähnlich gelagerte Untersuchung über die Ersparnis beim Booth-Recoding mittels Wallace-Tree-Multiplizierer zeigt [4].

3.3 RSA

Der RSA-Algorithmus ist der bekannteste Vertreter der public key Verfahren. Jeder Teilnehmer hat einen öffentlichen Schlüssel (n, e) und einen geheimen Schlüssel d. Eine Nachricht x wird verschlüsselt durch $x^e \bmod n$. Damit der Zeitaufwand zum Verschlüsseln vertretbar gering ist, wird als öffentlicher Exponent e meist $2^{16}+1$ oder 3 verwendet. Eine verschlüsselte Nachricht y wird entschlüsselt, indem man $y^d \bmod n$ bildet. Der Rechenaufwand wird haupsächlich durch die Kosten der Exponentiation beim Entschlüsseln und in der Modulo-Bildung verursacht, speziell deshalb weil der Modulus n eine Länge von wenigstens 512 Bit hat.
Die Exponentiation kann durch die Nutzung sogenannter Addition– bzw. Division Chains beschleunigt werden [5]. Hierzu wird der Exponent in eine Summe von Termen zerlegt, so daß die Exponentiation möglichst wenig Multiplikationen erfordert. Die Bestimmung optimaler Addition– bzw. Division-Chains ist NP-hard. Deshalb werden normalerweise einfache Heuristiken mit suboptimalen Chains verwendet. Hier besteht eine Optimierung auf einen Schlüssel darin, eine sehr gute Addition-Chain für diesen Schlüssel zu finden. Da RSA ein weitverbreitetes Verfahren ist, könnte man in diesem Fall voraussetzen, daß die Beschreibung des Verschlüsselungsalgorithmus bis auf die für jeden Empfänger verschiedene Addition-Chain den Systemteilnehmern vorliegt und im Schlüsselverzeichnis nur die Beschreibung der Addition-Chain gespeichert wird. Soll eine Nachricht verschlüsselt versendet werden, ist lediglich die Addition-Chain des Empfängers aus dem Schlüsselverzeichnis zu laden und mit dem allgemeinen Verfahren auf der FPGA auszuführen.
Eine weitere Verbesserung kann dadurch erreicht werden, daß statt schneller Exponentiation mittels Addition– und Division Chains und vielfacher Modulo-Bildung auf den Zwischenergebnissen, die Argumente zunächst mittels Montgomery-Reduktion [2] umgewandelt werden und danach wiederum unter Zuhilfenahme von Additions– und Divisions-Chains und einer abschließenden Modulo-Bildung modulo des zugrundeliegenden Moduls potenziert wird.
Eine Analyse dieser beiden Optimierungen steht zur Zeit noch aus.

4 Ausblick

In dieser Arbeit stellen wir die unseres Wissens nach neue Idee vor, bei der Ver- und Entschlüsselung großer Datenmengen statt Anwendung des jeweiligen Algorithmus auf eine Nachricht mit Schlüssel-Parameter s Hardware-Beschreibungen zu benutzen, die eine auf s optimierte Version des benutzten Verschlüsselungsalgorithmus darstellen. Durch die so erzielte Beschleunigung soll eine Nutzung preisgünstiger und für viele Zwecke verwendbarer programmierbarer Hardware anstelle von Spezialhardware zur Verschlüsselung großer Datenmengen wie etwa Videodaten in Echtzeit ermöglicht werden.
Unsere weiteren Untersuchungen zielen zum einen darauf ab, experimentell mittels Xilinx FPGAs die hier vorhergesagten Beschleunigungen zu verifizieren. Zum anderen soll untersucht werden, inwieweit die hier vorgestellten Optimierungen auch bei Ver– und Entschlüsselung mittels Software nützlich sind. Ein Vorteil von heutiger Software ist die Existenz einer einheitlichen, plattformunabhängigen und kompakten Darstellung von Programmen mittels JAVA. Die Nützlichkeit ist hauptsächlich bei public key Verfahren zu erwarten, da dort mit langen Zahlendarstellungen gerechnet wird. Bei einer Software-Implementierung des IDEA-Algorithmus auf einer Plattform mit Integer-Multiplizierer ist kein Vorteil zu erwarten, da jede Multiplikation unabhängig von den Faktoren sehr schnell ausgeführt werden kann.
Interessant erscheint weiterhin die Untersuchung, inwieweit die vorgestellten Verfahren beim Einsatz kryptografischer Verfahren in Chipkarten nutzbar sind. Obwohl es derzeit nicht zur gängigen Praxis der Chipkarten-Technologie gehört, sind Applikationen denkbar, in denen der Chipkarten-Inhaber einen kryptographischen Algorithmus und einen persönlichen geheimen Schlüssel einer Applikation nach Aushändigung der Chipkarte an ihn durch einen Chipkartensystembetreiber unabhängig von diesem selbst auswählt. Da Chipkarten-Prozessoren keine allzu große Leistungsfähigkeit aufweisen, spielen derartige Ansätze aus Performancegründen derzeit kaum eine Rolle, meistens wird (selten mehr als ein) kryptographisches Verfahren im ROM einer Chipkarte fest einprogrammiert und die geheimen Schlüssel vom Kartenaussteller in die Karte eingelesen, bevor diese dem Kunden zur Verfügung gestellt wird. Aktuelle Entwicklungen wie die Java-Card oder das Mondex-System sind Beispiele eines Trends, Interpreter-Programme im ROM zu installieren und Applikations-Programme in der entsprechenden Interpreter-Sprache im Nachhinein auf der Chipkarte zu implementieren. Selbstverständlich ist dabei auf Leistungsaspekte zu achten. Es ist denkbar, daß auf den jeweiligen geheimen Schlüssel optimierte Software Lei-

stungsverbesserungen ermöglicht. Allerdings muß dabei beachtet werden, daß beim Optimierungsprozeß der Code nicht allzu sehr aufgebläht wird, um die eingeschränkte Speicherkapazität der Chipkarte nicht zu übersteigen. Der Tradeoff von Effizienzverbesserung zu höheren Speicherkosten und die Identifikation von Anwendungsszenarien aus der Praxis, in denen dieser Ansatz eine Erhöhung der Sicherheit für den Chipkartenbesitzer bewirkt, stehen daher an erster Stelle bei der Untersuchung dieser Idee.

Literatur

[1] Bernstein, R.: Multiplication by Integer Constants. Software — Practice and Experience, Vol. 16, No. 7, July 1986, pp. 641–652

[2] Menezes, A. J.; van Oorshot, P. C.; Vanstone, S. A.: Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997

[3] Schneier, B.: Applied Cryptography. 2nd Ed. New York: John Wiley & Sons, 1995

[4] Paul, W. J.; Seidel, P.-M.: On the complexity of booth recoding. Proc. 3rd Conf. on Real Numbers and Computers (RNC3), 1998

[5] Walter, C. D.: Exponentiation Using Division Chains. IEEE Transactions on Computers. Vol. 47, No. 7, July 1998, pp. 757–765