State-of-the-Art Kryptoverfahren für drahtlose Sensornetze – Eine Krypto-Bibliothek für MantisOS

Björn Stelte und Björn Saxe
Institut für Technische Informatik
Universität der Bundeswehr München
Neubiberg, Germany
{bjoern.stelte, bjoern.saxe}@unibw.de

Abstract: Gerade im Bereich der drahtlosen Sensornetze muss aufgrund der Ressourcen-Knappheit auf aufwendige Kryptoverfahren verzichtet werden. Dennoch ist gerade für den Einsatz von Sensornetzen in hoch-schutzbedürftigen Umgebungen eine vertrauenswürdige Nachrichtenkommunikation notwendig. Eine Verschlüsselung der Kommunikation unter den Sensorknoten wie auch zur Datensenke des Sensornetzes ist die Grundlage für eine sichere Authentifizierung und kann zur Lösung einiger bekannter Angriffe auf Sensornetze beitragen. Oftmals werden nur etablierte älterer Kryptoverfahren in Sensornetzen verwendet. In dieser Arbeit werden für den Einsatz in Sensornetzen vielversprechende neuere Verfahren untersucht und die Ergebnisse einer Untersuchung anhand einer beispielhaften Implementierung der Verfahren in MantisOS gezeigt. Die hieraus entstandene Software-Bibliothek von state-of-the-art Kryptoverfahren für MantisOS kann als Ausgangspunkt für zukünftige Sicherheitslösungen in drahtlosen Sensornetzen dienen.

1 Einleitung

Drahtlose Sensornetze bestehen aus hunderten oder tausenden von kleinen low-power Sensorknoten. Diese Knoten können mithilfe ihrer Sensorik Ereignisse in ihrer unmittelbaren Umgebung detektieren und per Radio-Module Nachrichten hierüber versenden. Die Nachrichten werden in einer Datensenke gesammelt und ausgewertet.

Viele Anwendungen im häuslichen Umfeld existieren bereits heute oder befinden sich in Planung. Die drahtlosen Sensornetze werden ebenfalls immer beliebter für Szenarien aus dem hochsicherheits Bereich, wie Objektschutz oder der Gebietsüberwachung. Auf dem Markt befindliche Sensorknoten sind jedoch bedingt durch ihre knappen Ressourcen (Rechenleistung, Speicherkapazität, Laufzeit, Batteriekapazität) nur bedingt für die genannten Szenarien aus dem hochsicherheits Bereich geeignet. Vor allem fehlen häufig effiziente Implementierungen von Kryptoverfahren, so dass in vielen Anwendungen Nachrichten nicht fälschungssicher übertragen bzw. kryptographische Signaturen keine Anwendung finden. Dieses Fehlen von geeigneten Kryptoverfahren ermöglicht potentiellen Angreifern Sensornetze mit wenig Aufwand ausser Gefecht zu setzen. So sind viele Angriffe aus der Kategorie 'man-in-the-middle' für drahtlose Sensornetze bereits bekannt, wie bspw. sniffing, data integrity, energy drain, black hole, hello flood, wormhole. Abgesehen von den in

jüngster Zeit stark beachteten Feld der Verschlüsselungverfahren mit elliptischen Kurven (ECC) hat sich auch in der klassischen Kryptographie einiges getan. Wir werden uns hier speziell mit symmetrischen Kryptoverfahren und den Hash-Verfahren auseinandersetzen.

Diese Arbeit ist wie folgt aufgebaut, in Kapitel 2 möchten wir Sicherheitsproblem und Angriffe auf die Datenübertragung des Sensornetze näher beleuchten. Eine exemplarische Implementierung von Kryptoverfahren für das Sensor-Betriebssystem MantisOS stellen wir in Kapitel 3 vor. Bekannte und neuere viel versprechende Verfahren werden kurz vorgestellt und die Verwendung der Bibliothek erläutert. In Kapitel 4 folgt der Vergleich und die Bewertung der implementierten Kryptoverfahren in Bezug auf ihrer Verwendbarkeit auf handelsüblichen Sensorknoten. Im letzten Kapitel fassen wir die Ergebnisse unserer Arbeit kurz zusammen und geben eines Ausblick auf mögliche Erweiterungen.

2 Sicherheitsproblem und Angriffe

Durch fehlenden Sicherheitsmechanismen ist die Datensicherheit gefährdet, so ist es ohne Verschlüsselung dem Angreifer möglich, den kompletten Netzverkehr abzuhören. Er kann somit Erkenntnisse über das Netz gewinnen und sie für geeignete Angriffe nutzen oder aus den Informationen neue gefälschte Informationen generieren und somit gezielt das Sensornetz zu stören.

Auch die Verfügbarkeit des Sensornetzes ist gefährdet. Ein Angreifer hat die Möglichkeit mit Denial-of-Service Angriffen, die Nutzung des ganzen oder auch von Teilen des Sensornetzes durch den Betreiber zu verhindern. [WS02] gibt einen guten Überblick über Denial-of-Service Angriffe auf drahtlose Sensornetze. So sind Denial-of-Service Angriffe prinzipielle auf jeder Schicht des Netzwerks möglich. Bezg. der Vermittlungsschicht könnte der Angreifer einzelne Sensorknoten durch manipulierte Knoten ersetzen, die eine Weiterleitung von Routing-Nachrichten unterbinden oder gezielt falsche Routing- Nachrichten verbreiten. Beispielsweise sind die Routingprotokolle, die nach Distanz-Vektor Verfahren funktionieren, anfällig für so genannte Black Hole Angriffe. In der Transportschicht gibt es hauptsächlich zwei Möglichkeiten: Flooding und Desynchronisation. Protokolle, die ihren Verbindungszustand speichern müssen, sind anfällig für Flooding-Angriffe. Die einfachste Variante ist das Senden von sehr vielen Verbindungsaufbauanforderungen an den Empfänger durch den Angreifer. Mit jeder dieser Anforderung muss ein kleiner Teil des Speichers für einen Verbindungskontext reserviert werden, auch wenn die Verbindung nie komplett aufgebaut wird. Mit genügend Anforderungen kann man so den Speicher des Empfängers überfüllen. Bei der Desynchronisation verschickt der Angreifer wiederholt Pakete mit manipulierten Sequenznummern oder vergleichbaren Kontrollschaltern an einen oder beide Kommunikationsenden. Die veränderten Kontrollinformationen veranlassen die Kommunikationspartner dazu, die vermeintlich verlogen gegangene Pakete neu zu übertragen. Mit geschicktem Timing kann der Angreifer so eine sinnvolle Kommunikation verhindern und Sender und Empfänger halten sich an zeit- und energieaufwändigen Synchronisationsmechanismen auf.

Mit Angriffen auf die Verfügbarkeit kann der Angreife eine Verhinderung und Verzögerung

von Meldungen erreichen. Da solche Angriffe zumindest auf physischer Ebene ohne grossen Aufwand durchzuführen sind, ist das Risiko für unser Szenario hoch. Die Sicherstellung der Datenintegrität und Authentizität unter den Netzwerkteilnehmern ist ebenfalls für die Gesamtsicherheit eines Netzes unerlässlich. Die reine Verschlüsselung von Nachrichten ohne Authentifizierung und Sicherstellung der Integrität hat sich schon mehrfach als verwundbar erwiesen [Bel96][BGW01][Kra01]. Ohne Integrität kann der Empfänger nicht feststellen, ob die vom Sender gesendete, ursprüngliche Nachricht verändert wurde. Schon durch Vertauschen einzelner Bits lassen sich gezielt vorhersagbare Änderungen im Klartext erzeugen [BGW01]. Ohne Authentifizierung können die Kommunikationspartner nicht sicherstellen, ob ihr Gegenüber auch derjenige ist, der er vorgibt zu sein. Dem Angreifer fällt es so deutlich leichter z.B. von ihm manipulierte Knoten in das Sensornetz einzuschleusen.

Angriffe auf die Integrität und Authentizität, die es dem Angreifer ermöglichen Nachrichten zu verfälschen, sind aufwändiger als Angriffe auf die Verfügbarkeit. Jedoch kann bei einem solchen Angriff der mögliche Schaden auch deutlich höher liegen, da der Angreifer unter Umständen die Kontrolle über das Netz erlangen und es für seine Zwecke Missbrauchen kann und dass im schlimmsten Fall dies noch nicht einmal bemerkt wird.

Die Kryptographie löst zwar nicht alle Probleme, jedoch ist der Einsatz von Kryptoverfahren unabdingbar, um Sensornetze in kritischen Infrastrukturen einsetzen zu können. So können bspw. Hash-Verfahren zur digitalen Signatur und Blockchiffren zur Nachrichtenverschlüsselung eingesetzt werden. Wir werden im nächsten Kapitel eine kryptographische Bibliothek für das Sensor-Betriebssystem MantisOS vorstellen. Kryptoverfahren aus dem Bereich der Blockchiffren und Hash-Verfahren wurden implementiert und als Bibliothek in MantisOS, einem bekannten Sensornetz-Betriebsystem, integriert.

3 Kryptographie Bibliothek für MantisOS

Wie für fast alle Arten von Rechnern, unabhängig von Größe und Einsatzgebiet, existieren auch für Sensorknoten Betriebssysteme, die für einen abstrahierten Zugriff auf Hardware der Sensorknoten und für Prozess- und Speicherverwaltung zuständig sind. Diese Arbeit beschäftigt sich mit MultimodAl system for NeTworks in In-situ wireless Sensors OS (MantisOS). MantisOS ist ein von der University of Colorado entwickeltes Betriebssystem für ein eingebettete Systeme. Es im Gegensatz zu anderen Vertretern seiner Art wie z.B. TinyOS von der University of California Los Angeles plattformübergreifend und multithreaded (TinyOS z.B. ist ereignisorientiert). Das Betriebssystem ist nach dem klassischen Schichtenmodell aufgebaut: Es besteht aus Kernel mit Scheduler, COMM-Layer, DEV-Layer und dem Netzwerk-Stack. Der Scheduler unterstützt Prioritätenvergabe und hat verschiedene Optimierungen, wie z.B. das Schlafen legen von Threads, um den Energieverbrauch zu minimieren. MantisOS enthält bis auf eine RC5 und CCITT CRC-16 Implementierung bislang keine eingebauten kryptographischen Funktionen.

3.1 Auswahl geeigneter Verfahren

Aus den Überlegungen zum vorherigen Kapitel ergeben sich zweierlei Arten von Anforderungen. Die rechen-betonten Anforderungen sind abhängig von der Art der Implementierung und den Fähigkeiten des Programmierers. Hier kommt es darauf an, dass ein Verfahren möglichst wenige, einfache Rechenoperationen durchführt, da diese massgeblich die Laufzeit und, im Falle von Sensorknoten den Energieverbrauch erheblich beeinflussen. Ein geringer Speicherverbrauch, sowohl während der Berechnung als auch für die Speicherung von eventuell benötigten Konstanten, ist ebenso wichtig. Ein Verfahren muss zur Laufzeit so wenig Speicher wie möglich verbrauchen, um für die eigentliche Anwendung genügend Speicher freizuhalten. Zugriff auf externen Speicher sollte möglichst vermieden werden, da mit jedem Zugriff der Energieverbrauch stark ansteigt. Interessant sind besonders Verfahren, deren Implementierungsaufwand gering ist oder hinsichtlich Ressourcen beschränkte Hardware wie z.B. Smart-Cards entwickelt wurden.

Des weiteren gibt es davon unabhängige kryptographische Anforderungen wie z.B. Schlüssel- und Hashlängen sowie bekannte Angriffe auf ein Verfahren und deren praktische Relevanz. Für die öffentliche und industrielle Verwendung von kryptographischen Techniken gibt es Empfehlungen von Regierungsorganisationen, so hat auch das Bundesamt für Sicherheit in der Informationstechnik (BSI) im Falle von Deutschland eine Technische Richtlinie [TR-] veröffentlicht, die Empfehlungen und Schlüssellängen für kryptographische Verfahren beinhaltet. Viele Verfahren werden im Rahmen von Auswahlverfahren für solche Empfehlungen ausführlich untersucht. Eines dieser Projekte ist das NESSIE-Projekt New European Schemes for Signatures, Integrity, and Encryption [dNP] der europäischen Kommission. Dort wurde bereits 2000 ein Aufruf zur Einreichung von kryptographischen Algorithmen gestartet, die zukünftigen Anforderungen an Sicherheit und Vorgaben an Hard- und Softwareumgebungen genügen sollten. Im Jahr 2003 wurden dann von ursprünglich 39 Algorithmen im Rahmen eines zweiphasigen Auswahlprozesses 17 Verfahren ausgewählt. Auch das amerikanische NIST National Institute of Standards and Technology benutzt Wettbewerbsverfahren zur Auswahl und Bestimmung neuer kryptographischer Standardverfahren. Momentan wird der Nachfolger des SHA Hash-Verfahrens in der Cryptographic Hash Algorithm Competition [dCHAC] bestimmt. Wir haben in unserer Bibliothek einige der bekannten und etablierten Verfahren neben neueren Verfahren aus dem CHAC Wettbewerb aufgenommen. Wir möchten hier nur einige der implementierten Verfahren kurz beschreiben, stellvertretend für die unserer Ansicht nach bedeutenden Verfahren für den Einsatz in Sensornetzen.

3.1.1 Hash-Verfahren

SHA-1 und SHA-256 ist eine vom NIST und der NSA zusammen entwickelte Hash-Funktion für den Digital Signature Standard und steht für Secure Hash Algorithm. In seiner ursprünglichen Version SHA-0 von 1994 hatte es einen Designfehler, den die 1995 erschienene Variante SHA-1 korrigierte. Die Funktion existiert in verschiedenen Varianten wie SHA-256 und SHA- 384, wobei die Zahl die Länge des von ihr berechneten Hash-Werts in Bits angibt. Ab SHA-224 spricht man auch von der SHA-2 Familie. 2005 und

2006 wurden mehrere Kollisionsangriffe gegen SHA-1 veröffentlicht [WYY05], die in naher Zukunft durchaus von praktischer Relevanz sein könnten. Seit der Veröffentlichung dieser Angriffe empfiehlt das NIST nur noch Verfahren der SHA-2 Familie zu verwenden. Auf längere Zeit gesehen sollen die Verfahren durch neue ersetzt werden und das NIST hat dazu wie schon beim Advanced Encryption Standard einen Wettbewerb ausgeschrieben. Die genaue Funktion von SHA ist im RFC 3174 beschrieben. Somit ist Verwendung von SHA-1 für nicht unbedingt sicherheitsrelevante Daten auf jeden Fall ausreichend, während SHA-256 als Mitglied der SHA-2 Familie auch für sicherheitskritische Daten verwendet werden kann.

BLAKE ist einer der aussichtsreichsten Kandidaten für SHA-3 aus der bereits erwähnten CHAC Ausschreibung. BLAKE hat bereits die erste Runde der Ausschreibung erfolgreich bestanden und ist nun mit 13 weiteren Kandidaten in der zweiten Runde. Es wurde im Auswahlprozesses ausführlich untersucht und hat bisher keine Schwächen gezeigt. Es sind insgesamt vier verschiedene Varianten verfügbar: BLAKE-28, 32, 48, 64. Die Zahl gibt dabei die Länge des Hashwert in Bytes an. Im Gegensatz zu einigen anderen SHA-3 Kandidaten besticht BLAKE mit seinem geringen Speicherbedarf und ist daher besonders für die Sensornetze von Interesse. Die Spezifikation von BLAKE haben die Autoren auf ihrer Website veröffentlicht [JPLWP09].

3.1.2 Symmetrische Verfahren

AES der Advanced Encryption Standard ist der Nachfolger von DES und wurde 2000 als neuer Standard bekannt gegeben. Vor seiner Ernennung hieß es nach den Namen seiner Entwickler Rijndael-Algorithmus. AES ist in den USA für Dokumente mit der höchsten Geheimhaltungsstufe zugelassen. Die Blockgröße ist festgelegt auf 128 Bit, mögliche Schlüssellängen sind 128, 192 und 256 Bit, die entsprechenden Verfahren heißen dann AES-128, AES-192 und AES-256. Es gibt bisher mehrere Angriffsansätze und -versuche (vgl. [BKN09]), doch keiner ist bisher von praktischer Bedeutung. Die offizielle Spezifikation von AES kann man im FIPS-197 [FIP] nachlesen. AES gehört zu den vom BSI empfohlenen Blockchiffren.

RC6 war einer der Kandidaten der AES Ausschreibung und hat es bis in die finale Runde geschafft. Er 1998 entwickelt als Nachfolger und auf der Grundlage von RC5. Wie auch RC5 hat RC6 variable Blockgrößen, Rundenzahlen (0-255) und Schlüssellängen (0-2048 Bit). Die bisher gefundenen möglichen Angriffe sind praktisch nicht durchführbar oder lassen sich durch entsprechende Wahl der Parameter verhindern [GHJ+01][BPVV99]. Auch RC6 ist patentiert, die komplette Spezifikation ist in [RRSY98] veröffentlicht.

Serpent ist von Ross Anderson, Eli Biham und Lars Knudsen ebenfalls als AES-Kandidat entwickelt worden landete auf dem zweiten Platz hinter Rijndael. Wie die meisten AES Kandidaten hat Serpent eine Blockgröße 128 Bit, aber variable Schlüssellänge von 0-256 Bit in 8 Bit Schritten. Die Rundenzahl ist höher als bei AES (10-14 vs 32) und hat somit

vermutlich eine höhere Sicherheit. Allerdings dürfte sich diese erhöhte Rundenzahl auch in der Geschwindigkeit niederschlagen. Serpent-128, -192 und -256 gehören zu den vom BSI empfohlenen Blockchiffren, eine Beschreibung zu Serpent findet sich [ABK].

3.2 Nutzung der Bibliothek

In diesem Abschnitt soll die allgemeine Vorgehensweise zur Einbindung eines kryptographischen Verfahrens in eine MantisOS Anwendung erläutert werden.

1. Header-Datei einbinden – Die Header-Dateien befinden sich im Verzeichnis SRC/-LIB/INCLUDE/CRYPTOLIB/.
2. Variablen deklarieren – für die Aufnahme des Hashwerts oder der Schlüssel bzw., wenn das Verfahren es erfordert, Kontextdatentypen.
3. Initialisierungen durchführen – nicht alle Verfahren benötigen diesen Schritt. Zur Initialisierung werden immer Schlüssel und Kontextvariable (und selten eventuell weitere Parameter, wie Rundenzahl) benötigt.
4. Berechnung des Hashwerts / Verschlüsselung – die eigentliche Berechnung erfolgt bei den Hashverfahren mit dem Aufruf der Hashfunktion. Übergeben wird die Nachricht und deren Länge in Bits. Bei den Blockchiffren wird die Verschlüsselung mit Chiffre-enc(), die Entschlüsselung Chiffre-dec() aufgerufen. Bei Verfahren, die mit Kontextvariablen arbeiten, übergibt man den Kontext und die Nachricht, ansonsten den Schlüssel und die Nachricht.
5. Freigabe von Ressourcen – ist Ver- oder Entschlüsselung erfolgt und wird ein Verfahren nicht länger benötigt, bietet es sich an den Speicherplatz, der für Kontextvariable reserviert wurde, wieder freizugeben.

4 Vergleich und Bewertung der Verfahren

Die Eingaben für die folgenden Messung (Schlüssel und Nutzdaten) bestanden aus Zufallsdaten. Alle Messungen wurden mit 100 unterschiedlichen Schlüsseln und Nutzdatensätzen durchgeführt und anschliessend stochastisch bewertet.

Die Messungen wurden mit dem Simulator Avrora (AVR Simulation and Analysis Framework Plattform) durchgeführt. Avrora wurde von der Compilers Group der University of California Los Angeles entwickelt [TT05][Web]. Mit dem Tool können Programme, die für AVR-Mikrokontroller und MICA2 Sensorknoten geschrieben wurden, simuliert und genau analysiert werden. Zusätzlich beinhaltet das Programm ein Java-Framework, mit dem der Simulator erweitert werden kann, etwa durch hinzufügen neuer Monitore. Der Simulator arbeitet auf Instruktionsebene auf den Taktzyklus genau. Die Simulation umfasst nicht nur den reinen AVR-Mikrokontroller, sondern auch externe Komponenten wie Sensoren und Netzfunktionalität (wie Chipcon CC1000 Transmitter).

4.1 Vergleich Taktzyklen

Der Avrora-Simulator kann über seinen energy-profile-Monitor die zur Abarbeitung eines Programms benötigten Taktzyklen anzeigen. Diese sind dann nach den ausgeführten Funktionen aufgeschlüsselt. Zur Bestimmung der von einem Verfahren benötigten Taktzyklen wurden die Werte der vom Verfahren verwendeten Funktionen aufaddiert. Die Werte für Ver- und Entschlüsselung bei Verfahren, die eine Initialisierungsfunktion vor ihrer Verwendung benötigen, wurden aus dem Gesamtwert für Initialisierung + Ver- bzw. Entschlüsselung ermittelt, d.h., vom Gesamtwert wurde der Wert für die Initialisierung und Freigabe wieder abgezogen.

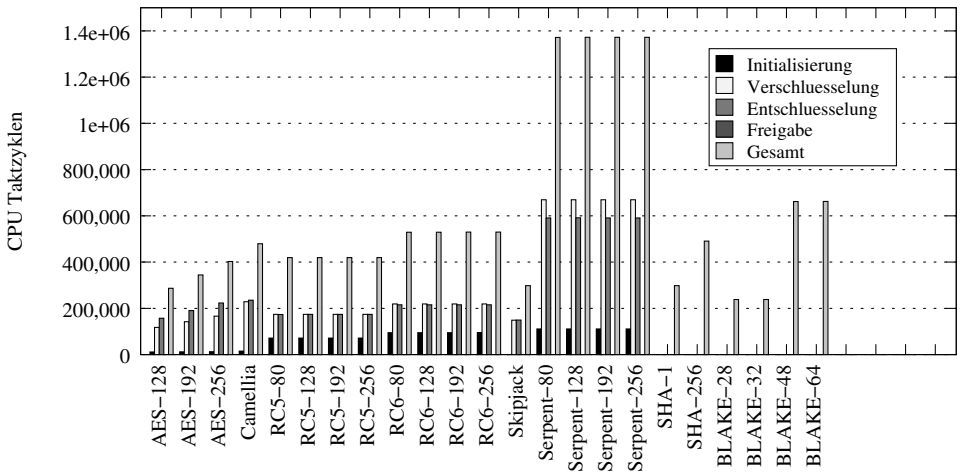

Abbildung 1: Taktzyklen Blockchiffren bzw. Berechnung des Hashwerts von 128 Byte Daten

Bei den Verfahren, die eine freie Wahl der Schlüssellänge anbieten, wirkt sich diese praktisch nicht auf die Zahl der benötigten Taktzyklen aus. Bei AES dagegen steigt sie mit der Schlüssellänge. Bei allen Verfahren bis auf AES sind auch die Werte für Ver- und Entschlüsselung entweder gleich oder minimal verschieden. Serpent benötigt mit Abstand am längsten für jeden Teilschritt, was an der hohen Anzahl der Runden (32, bei AES-128 z.B. nur 10) liegt. Lässt man die Zeiten für die eher selten ausgeführte Initialisierung weg, liegt die AES-128 Implementierung sogar auf gleicher Höhe mit dem eher einfachen und in die Jahre gekommenen Skipjack.

Die kleinen BLAKE-Funktionen (28/32) brauchen für die Berechnung eines 28 bzw. 32 Byte langen Hashwerts geraden einmal halb solange wie SHA-256, das ebenfalls einen 32 Byte lange Hashwert berechnet. Selbst SHA-1 mit einer Hashwertlänge von nur 20 Byte benötigt etwas länger als BLAKE-28/32. Die großen BLAKE-Funktionen benötigen entsprechend ihrer größeren Hashlänge noch mal deutlich länger. Die geringen Unterschiede zwischen BLAKE-28 und BLAKE-32 bzw. BLAKE-48 und BLAKE-64 ergeben sich dadurch, dass, vereinfacht gesagt, die Berechnung bei beiden gleich ist, der Hashwert aber

am Ende der Berechnung einfach abgeschnitten wird.

4.2 Vergleich Speicherverbrauch

Der Simulator bietet die Möglichkeit, alle während des Programmablaufs stattfindenden Lese- und Schreibzugriffe auf den Speicher zu protokollieren. Dieser Speichermonitor zeigt aber die Zugriffe nur in der Form “Adresse - Anzahl gelesene/geschriebene Bytes” an. Um daraus den Speicherverbrauch einzelner Funktionen abzulesen, fehlen einfach die nötigen Informationen. Auch die Ermittlung von groben Richtwerten anhand der Segmentgrößen der Data und BSS Segmente (Segmente für globale, statische Variablen) hätte bei den vorliegenden Implementierungen nicht zu brauchbaren Ergebnissen geführt (der statische Verbrauch ist fast überall 0 Bytes), da fast alle Variablen erst innerhalb von Funktionen deklariert werden. Zudem fehlt bei dieser Methode der durch Unterprogrammaufrufe wachsende Stack. Aus diesen Gründen wurde der Speichverbrauch manuell anhand des Quellcodes errechnet. Dazu wurde zunächst der Verbrauch der einzelnen Funktionen ermittelt, dann der tiefste Unterprogrammaufruf innerhalb eines abgeschlossene Schritts des jeweiligen Verfahrens (also z.B. Verschlüsseln eines Blocks) herausgesucht, um dann durch aufaddieren der Werte den maximalen Verbrauch zu bestimmen. Zu den lokalen Variablen innerhalb einer Funktionen wurden auch Rückgabewerte und die an die Funktionen übergebene Parameter mitgezählt. Auch bei dieser Methode ist die exakte Stackgröße schwierig zu ermitteln, insgesamt ist dieser Ansatz aber deutlich aussagekräftiger als nur den statischen Verbrauch der Segmente aufzuführen.

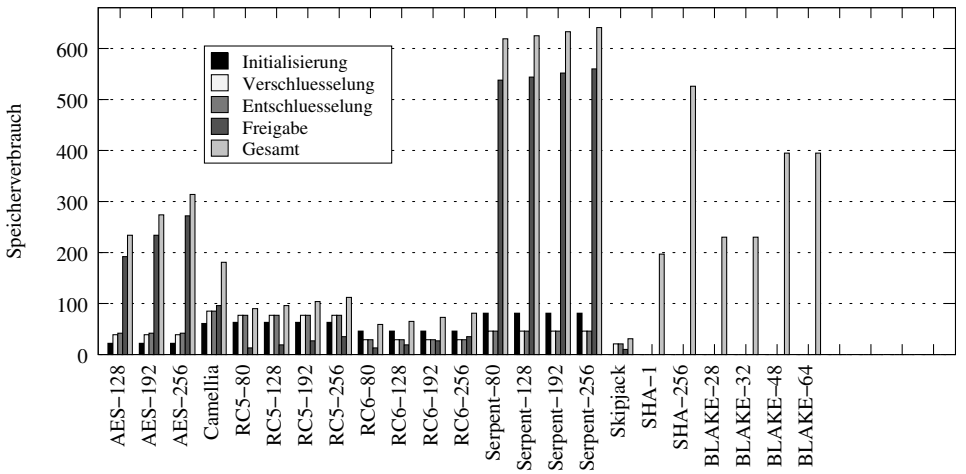

Abbildung 2: Speicherverbrauch RAM, alle Werte in Bytes

Zunächst fällt auf, dass sich bei den Verfahren, die variable Schlüssellänge unterstützen, die größere Schlüssellänge kaum im Speicherbedarf niederschlägt, nur der statische An-

teil des Schlüssels steigt zwangsläufig. RC5, RC6 befinden sich mit 100 Bytes ungefähr auf einem Niveau. AES und Serpent fallen durch einen großen statischen Anteil auf, der durch die Kontextvariablen zur Speicherung der Rundenschlüssel verursacht wird. Der Anteil von Serpent ist dabei durch die höhere Rundenzahl (32 vs 10-14) im Vergleich zu AES besonders groß. Serpent braucht somit auf einer üblichen 8-Bit Plattform mit 4 kByte RAM etwas mehr als ein Viertel des gesamten Speichers auf. Skipjack benötigt mit Abstand am wenigsten, es hat keine Kontextvariablen und auch der Verbrauch während Ver- und Entschlüsselung ist sehr niedrig.

Der Speicherverbrauch für SHA-256 und BLAKE-48/64 ist für einen sehr ressourcenschwachen Knoten wie MICA2 zu hoch. SHA-1 und die kleinen BLAKE-Funktionen liegen auf dem Niveau der meisten Blockchiffren.

4.3 Vergleich Codegröße

Die Größe des Programmcodes wurde anhand der Größe des text-Segments in Objektdateien ermittelt. Die Codegröße aller Verfahren liegt sehr platzsparend im unteren, einstelligen KByte Bereich. Nur Camellia reißt nach oben aus und stellt sich bei ohnehin vollem Flashspeicher als nicht geeignet heraus. Ähnlich sieht es bei den Hashfunktionen aus, SHA-1 ist etwas kleiner als der Rest, die BLAKE-48/64 Verfahren sind mehr als doppelt so groß.

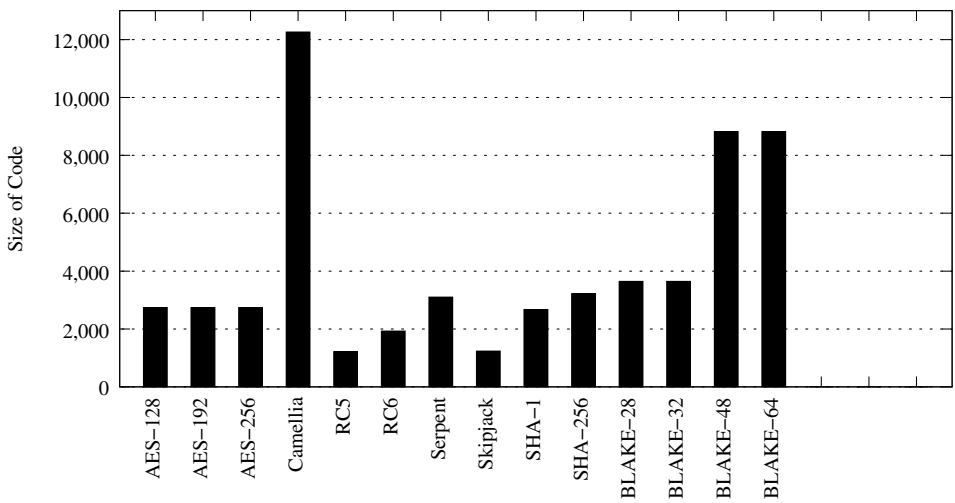

Abbildung 3: Codegröße im Flash, alle Werte in Bytes

4.4 Vergleich Flashspeicher

Zur Bestimmung der Anzahl der gelesenen Bytes aus dem Flash-Speicher wurden in den Quellcode Zähler eingefügt. Jedes mal nach dem Aufruf der speziellen Lesefunktionen (pgm_read_byte() etc.) wurde dieser entsprechend hochgezählt. Hierbei gilt es zu beachten, dass die Implementierungen von RC5, RC6, SHA-1 und SHA-256 keine Werte im Flash speichern.

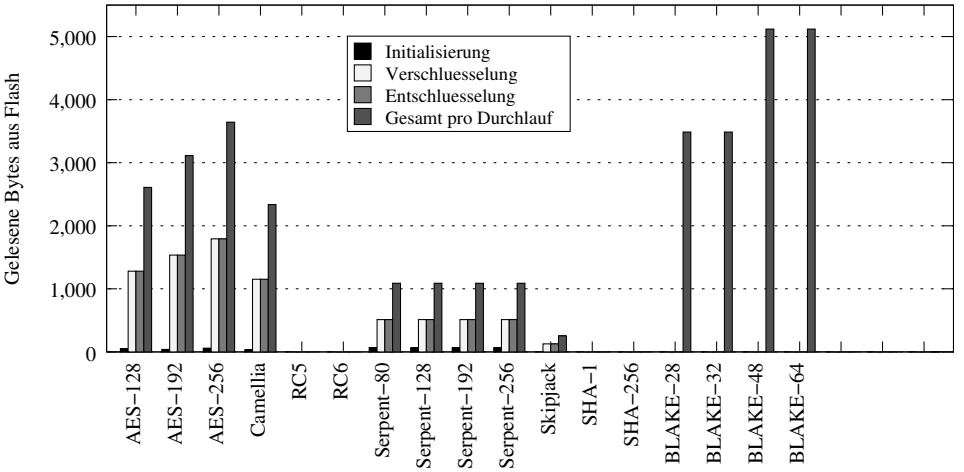

Abbildung 4: Gelesene Bytes aus dem Flash-Speicher

Die Anzahl der gelesen Bytes zur Initialisierung entspricht bei jedem Verfahren in etwa der Menge, die benötigt wird, um einen einzelnen Block zu verschlüsseln, und fällt somit nicht sonderlich ins Gewicht. Bei Serpent beeinflusst, wie auch beim RAM Verbrauch, die Schlüssellänge die Anzahl der Flashoperationen nicht, bei AES steigt die Anzahl der Flashzugriffe mit steigender Schlüssellänge. Wie beim RAM-Verbrauch, so auch bei den Flashoperationen, liest Skipjack am wenigsten aus dem Flashspeicher, abgesehen von RC5 und RC6, die überhaupt keine Werte im Flashspeicher ablegen. Bei den Hashfunktionen kommen SHA-1 und SHA-2 ganz ohne zusätzlichen Zugriffe auf den Flash aus. Sie erkaufen sich diesen Vorteil aber durch hohen Speicherverbrauch, da die für die Berechnung notwendigen Tabellen im RAM befinden.

4.5 Bewertung

In Tabelle 4.5 sind die implementierten Verfahren unter den untersuchten Gesichtspunkten gegenübergestellt. Je nach Anwendungsszenario bzw. Anforderungen, sowie zur Verfügung stehender Ressourcen, kann hierdurch ein passender Kandidat ermittelt werden.

Verfahren	Taktzyklen	Speicher RAM	Codegröße	Flash
AES-128	+	o	o	o
AES-256	o	o	o	–
Camellia	o	+	– –	o
RC5-256	o	+	++	++
RC6-256	o	++	+	++
Serpent-256	–	– –	o	+
Skipjack	+	++	++	++
SHA-1	+	++	++	++
SHA-256	o	– –	+	++
BLAKE-32	+	+	o	o
BLAKE-64	–	–	– –	–

Tabelle 1: Bewertungstabelle, Wertung von sehr gut ++ bis sehr schlecht – –, o entspricht dem Mittel

Bei der Bewertung der Verfahren haben wir versucht die Stärken und Schwächen der einzelnen Verfahren zu beachten und anhand der Kriterien Taktzyklen, Speicherverbrauch, Codelänge und Flashspeicher Beanspruchung eine objektive Empfehlung zu geben. Bzgl. der Taktzyklen bzw. der Ausführungsgeschwindigkeit ist das Verfahren AES-128 bei den Blockchiffren und die Verfahren Blake-32 und SHA-1 bei den Hash-Verfahren als besonders empfehlenswert zu nennen. Beim Punkt Speicherverbrauch führt die Empfehlung hingen zu RC6 und SHA-1, bzgl. der Codegröße ist RC5, Skipjack und SHA-1 im Vorteil. Die Verfahren RC5, RC6, Skipjack und die SHA Varianten benötigen keinen externen Flashspeicher und sind bei knappen Speichervorkommen zu verwenden.

5 Zusammenfassung und Ausblick

Zusammenfassend lässt sich folgendes sagen: Eine Implementierung von kryptographischen Verfahren ist auch für MantisOS ohne Probleme möglich und die hier verwendeten Implementierungen sind dabei auch entsprechend leistungsfähig. Für die hier untersuchten, konkreten Implementierungen kann man folgende, auf 8-Bit Plattformen anwendbare, Empfehlungen aussprechen. Auf Höheren Architekturen mit z.B. 32-Bit Prozessoren können die Ergebnisse (wegen höhere Wortbreite, mehr Befehlen, mehr Optimierungsmöglichkeiten durch den Compiler) anders ausfallen. Die AES-128 Implementierung hat sich als sehr effizientes Blockchiffre Verfahren erwiesen, nur ist der Speicherverbrauch problematisch. In diesem Fall eignet sich das langsamere aber platzsparende RC6 Verfahren. Bei den Hashverfahren geht die Empfehlung an die BLAKE-Variante BLAKE-32. Die Berechnung ist etwas schneller als bei SHA-1 mit minimal höherem Speicherverbrauch, die anderen Verfahren sind deutlich langsamer. Zieht man zusätzlich noch in Betracht, dass SHA-1 nicht mehr offiziell empfohlen wird, gibt es im Fall der hier untersuchten Bibliothek keinen Grund mehr SHA-1 noch zu verwenden. In naher Zukunft werden wir die Bibliothek um weitere bekannte Kryptoverfahren erweitern und öffentlich zugänglich ma-

chen. Durch eine erste experimentelle Portierung von TinyECC wurde die Erweiterbarkeit der Bibliothek hinsichtlich der public-key Verfahren bereits gezeigt.

Literatur

[ABK] Ross Anderson, Eli Biham und Lars Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. In *First Advanced Encryption Standard (AES) Conference*.

[Bel96] Steven M. Bellovin. Problem Areas for the IP Security Protocols. In *Proceedings of the Sixth Usenix Unix Security Symposium*, Seiten 205–214, 1996.

[BGW01] Nikita Borisov, Ian Goldberg und David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. Seiten 180–189, 2001.

[BKN09] Alex Biryukov, Dmitry Khovratovich und Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 231–249, Berlin, Heidelberg, 2009. Springer-Verlag.

[BPVV99] Johan Borst, Bart Preneel, Joos Vandewalle und Joos V. Linear Cryptanalysis of RC5 and RC6. In *Proceedings of Fast Software Encryption, Lecture Notes in Computer Science*, Seiten 16–30. Springer-Verlag, 1999.

[dCHAC] Homepage der Cryptographic Hash Algorithm Competition:. http://csrc.nist.gov/groups/ST/hash/sha-3/.

[dNP] Homepage des NESSIE-Projektes:. https://www.cosic.esat.kuleuven.be/nessie/.

[FIP] FIPS-197. Spezifikation von AES.

[GHJ+01] Henri Gilbert, Helena Handschuh, Antoine Joux, Serge Vaudenay und Gemplus Card International. A Statistical Attack on RC6, 2001.

[JPLWP09] Aumasson J.-P., Henzen L., Meier W. und Raphael C.-W. Phan. SHA-3 proposal BLAKE, 2009.

[Kra01] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, Seiten 310–331, London, UK, 2001. Springer-Verlag.

[RRSY98] Ronald L. Rivest, M. J. B. Robshaw, R. Sidney und Y. L. Yin. The RC6 Block Cipher. In *First Advanced Encryption Standard (AES) Conference*, Seite 16, 1998.

[TR-] BSI TR-02102. Kryptographische Verfahren: Empfehlungen und Schlüssellängen.

[TT05] With Precise Timing und Ben L. Titzer. Avrora: Scalable Sensor Network Simulation. In *Proc. of the 4th Intl. Conf. on Information Processing in Sensor Networks (IPSN*, Seiten 477–482, 2005.

[Web] Avrora Website:. http://compilers.cs.ucla.edu/avrora/.

[WS02] Anthony D. Wood und John A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, 2002.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu. Finding Collisions in the Full SHA-1. In *Proceedings of Crypto*, Seiten 17–36. Springer, 2005.