Sensorknoten - Sicherheit auf Schicht 2

Tobias Niedl
Betreuer: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: niedl@in.tum.de

KURZFASSUNG

Die sichere Kommunikation in Sensornetzen war und ist Gegenstand einiger Forschungsarbeiten. Diese Arbeit stellt die Systeme SPINS, TinySec und TinyPK vor, die in den letzten Jahren entwickelt wurden und Sicherheit auf Schicht 2 des OSI-Modells bieten. Alle Systeme arbeiten auf vergleichbaren Hardware-Plattformen und nutzen oder erweitern das Open-Source-Betriebssystem TinyOS für Sensor-Knoten. Das Hauptproblem für alle drei vorgestellten Systeme ist dabei die geringe Rechen- und Speicherkapazität der Sensorknoten.
Während SPINS und TinySec symmetrische Kryptographie-Verfahren verwenden, geht TinyPK einen Schritt weiter und versucht asymmetrische Verfahren zu implementieren, die pauschal mehr Ressourcen benötigen.
SPINS war einer der ersten Ansätze sichere Kommunikation in Sensornetzen zu ermöglichen. TinySec nutzte einige Erkenntnisse aus SPINS und kann heute offiziell zusammen mit TinyOS aus dem Internet heruntergeladen und verwendet werden. TinyPK zeigt, dass asymmetrische Kryptographie — zumindest teilweise — auf Sensorknoten implementiert und verwendet werden kann.

Schlüsselworte

Sensor Network Security, Link Layer Security, SPINS, TinySec, TinyPK

1. EINLEITUNG

Als Sensornetz wird ein Netzwerk von drahtlos kommunizierenden Rechnern (sog. Knoten) verstanden. Die Knoten sind batterie- bzw. akkubetrieben, verfügen über nur geringe Rechen- und Speicherkapazitäten und sind entsprechend günstig in der Produktion. Die Knoten können je nach Produktionsart für viele verschiedene Aufgaben verwendet werden, beispielsweise zur Überwachung von Umwelt-Werten. Die erfassten Messwerte können drahtlos zwischen den Knoten ausgetauscht oder an eine Basisstation, die als Gateway fungiert, geleitet werden. Aufgrund der Energieversorgung mittels Batterien sind die zur Verfügung stehenden Ressourcen stark begrenzt. Die Taktrate der Prozessoren liegt im einstelligen Megaherz-Bereich. Der verfügbare Speicher (Flash, RAM, EEPROM) liegt zwischen einigen hundert Byte und einigen Kilobyte. Aufgrund der begrenzten Ressourcen können herkömmliche Algorithmen und Protokolle, wie sie beispielsweise in IPsec [7] oder TLS [4] verwendet werden, nicht zum Schutz von Sensornetzen genutzt werden (vgl. [6]). Entsprechend erfolgte die Entwicklung der hier vorgestellten Systeme SPINS, TinySec und TinyPK immer unter dem Gesichtspunkt der mangelnden Rechenkapazitäten und dem begrenzen Energievorrat [10, 6, 13].

Für Sensorknoten existiert ein Open-Source-Betriebssystem namens TinyOS [8]. Unter TinyOS sind keine kryptographischen Funktionen implementiert um die Datenübertragung zwischen den Knoten zu sichern. Die vorgestellten Systeme erweitern oder basieren jedoch auf TinyOS und implementieren Funktionen zur Sicherung der Kommunikation auf dem sog. Link-Layer, also Schicht 2 nach dem OSI-Referenzmodell [3].

Da die Kommunikation zwischen Knoten unter TinyOS standardmäßig im Klartext erfolgt, sind Angriffe gegen ein Sensornetz sehr einfach. Ein Angreifer kann die übertragenen Pakete mitlesen, abfangen, modifiziert weiterleiten und eigene Pakete ins Netz senden. Die vorgestellten Systeme versuchen jeweils einige dieser Angriffe zu erkennen bzw. zu unterbinden.

Das Ziel von TinySec und SPINS ist es, eine sichere Kommunikation zwischen den Knoten (TinySec) bzw. einem Knoten und einer der Basisstation (SPINS) zu ermöglichen. TinyPK widmet sich der Problemstellung, ein nicht zum Sensornetz gehörendes Gerät, die „External Party“ (EP), zu authentisieren und ihr sicher einen gemeinsamen Schlüssel zu übermitteln, damit sie an der Kommunikation im Sensornetz teilnehmen kann.

In Abschnitt 2 werden zunächst einige wichtige Begriffe im Zusammenhang mit sicherer Kommunikation erläutert. Abschnitt 3 stellt SPINS vor, das aus den Protokollen SNEP und µTESLA besteht. Abschnitt 4 erläutert das neuere TinySec-System und Abschnitt 5 stellt TinyPK vor. In Abschnitt 6 folgt ein Vergleich der genannten Systeme und in Abschnitt 7 werden einige Angriffe aufgeführt, die auch bei Verwendung der vorgestellten Systeme möglich sind.

2. SICHERHEIT

2.1 Schutzziele

Für eine sichere Kommunikation gilt es, bestimmte Schutzziele zu erreichen. Diese lauten: Geheimhaltung, Integrität, Authentizität und Frische, sowie Verfügbarkeit, Verbindlichkeit und Anonymität (vgl. [5]). Die vorgestellten Systeme erreichen jedoch bei weitem nicht alle davon. Entsprechend werden im folgenden nur die Schutzziele näher erläutert, die wenigstens eines der Systeme auch erreicht.

2.1.1 Geheimhaltung (Confidentiality)

Die Geheimhaltung dient dazu, den tatsächlichen Inhalt einer Nachricht (den Klartext) gegenüber unautorisierten Parteien zu verschleiern. Dies wird meist durch eine Verschlüsselung der Nachricht erreicht. Idealerweise wird der selbe Klartext P in verschiedenen Nachrichten zu verschiedenen Kryptotexten C und C' verschlüsselt. Diese Eigenschaft wird als *semantic security* bezeichnet. Dadurch wird verhindert, dass ein Angreifer aus vielen abgehörten, verschlüsselten Nachrichten etwas über den Inhalt lernen kann. D.h. ein Angreifer soll nur mit einer 50%-igen Wahrscheinlichkeit erraten können, ob ein Bit des Kryptotextes für eine 0 oder eine 1 im Klartext steht [6].

2.1.2 Integrität (Integretiy)

Der Schutz der Integrität bedeutet, dass der Empfänger einer Nachricht erkennen kann, ob die Nachricht während der Übermittlung modifiziert wurde.

2.1.3 Authentizität (Authentication)

Authentizität verstärkt das Schutzziel der Integrität. Beim Empfang einer authentisierten Nachricht ist der Empfänger in der Lage zu prüfen, ob die Nachricht modifiziert wurde (Integrität) und ob die Nachricht wirklich vom angegebenen Absender stammt.

2.1.4 Frische (Freshness)

Die Zusicherung der Frische einer Nachricht verhindert Replay-Angriffe gegen Netzwerk-Knoten. Beispielsweise wäre es einem Angreifer möglich ein verschlüsseltes und authentisiertes Pakete abzuhören und später erneut ins Netz einzuspielen (injection). Durch die Verwendung von *Nonces* (*Number used once*, relativ großen Zufallszahlen), die in den Paketen mit übertragen werden, werden solche Replay-Angriffe verhindert.

2.2 Symmetrische und asymmetrische Kryptographie

Um die jeweiligen Schutzziele zu erreichen, verwenden die Systeme SPINS, TinySec und TinyPK zwei verschiedene Ansätze. SPINS und TinySec nutzen symmetrische, TinyPK asymmetrische Kryptographie. Auf die mathematischen Hintergründe soll an dieser Stelle nicht näher eingegangen werden. Stattdessen sollen nur die Komponenten erläutert werden, die in den Systemen verwendet werden.

2.2.1 Symmetrische Kryptographie

Symmetrische Kryptographie ist im Gegensatz zur asymmetrischen Kryptographie schneller und benötigt weniger Ressourcen bzw. kürzere Schlüssel. Schlüssellängen über 100 Bit gelten hier noch als sicher [2]. Allerdings ist es bei symmetrischen Verfahren nötig, dass zwei Partner über den selben Schlüssel K verfügen, um kommunizieren zu können. Daraus resultiert ein großes Problem und ein großer Nachteil der symmetrischen Kryptographie: Das Problem der Schlüssel-Verteilung. Es muss sichergestellt werden, dass zwei Partner den Schlüssel auf „sicherem Weg“ bekommen, also so, dass ein Angreifer ihn nicht abfangen bzw. mitlesen kann.

In der symmetrischen Kryptographie wird das Schutzziel der Authentizität mit Hilfe sog. Message Authentication Codes (MACs) erreicht. MACs werden durch kryptographische Hashfunktionen unter Einbeziehung des gemeinsamen Schlüssels erzeugt. Eine kryptographische Hashfunktion h bildet eine beliebig lange Bitfolge B auf eine Bitfolge S fester Länge ab: $S = h(B)$. Dabei werden gleiche Bitfolgen zu gleichen Hashwerten S (siehe auch [5]). Ein MAC ist so nichts weiter als ein Hashwert einer Hashfunktion h bei dem als Argument neben der Bitfolge B noch ein geheimer Schlüssel K verwendet wurde, also $MAC = h(B|K)$. Wenn Sender und Empfänger den Schlüssel K kennen, so kann der Empfänger die Korrektheit der empfangenen Nachricht prüfen, indem er selbst den MAC-Wert der Nachricht berechnet und mit dem im Paket enthaltenen MAC-Wert vergleicht.

SPINS und TinySec nutzen, wie bereits erwähnt, jeweils symmetrische Kryptographie-Verfahren.

2.2.2 Asymmetrische Kryptographie

Asymmetrische Kryptographie ist deutlich rechenintensiver und benötigt längere Schlüssel. Heute gelten 1024 Bit noch als sicher [2]. Jeder Kommunikationsteilnehmer besitzt ein Schlüsselpaar; einen privaten Schlüssel K_{Priv} und einen öffentlichen Schlüssel K_{Pub}. Der private Schlüssel muss geheim bleiben, wohingegen der öffentliche Schlüssel offen im Netz verbreitet werden darf. Wenn zwei Partner A und B kommunizieren möchten, werden vier Schlüssel benötigt: $K_{A,Pub}$, $K_{A,Priv}$, $K_{B,Pub}$ und $K_{B,Priv}$. D.h. für eine gesicherte Kommunikation wird kein gemeinsamer Schlüssel verwendet. Dadurch gibt es bei asymmetrischen Verfahren das Problem der sicheren Schlüsselverteilung nicht. Im weiteren ist es durch die Verwendung dieses Verfahrens auch möglich, das Schutzziel der Verbindlichkeit zu erreichen. D.h. ein Empfänger kann sich sicher sein, dass eine Nachricht wirklich vom angegebenen Absender stammt. Dieses Ziel wird von den vorgestellten Systemen jedoch nicht weiter verfolgt.

Die genannten vier Schlüssel werden für eine gesicherte Kommunikation wie folgt verwendet (vgl. [1, 5]):

- A möchte B eine verschlüsselte Nachricht senden
  - A verschlüsselt die Nachricht mit dem öffentlichen Schlüssel von B $K_{B,Pub}$
  - B empfängt die Nachricht und kann diese *nur* mit seinem privaten Schlüssel $K_{B,Priv}$ entschlüsseln
- A möchte seine Nachricht an B signiert versenden. Durch eine Signatur wird das Schutzziel der Authentizität erreicht
  - A berechnet einen Hashwert S der Nachricht und verschlüsselt diesen mit seinem privaten Schlüssel $K_{A,Priv}$. Dies entspricht der Signatur der Nachricht: $Sig(h; K_{A,Priv})$
  - B empfängt die Nachricht und berechnet ebenfalls den Hashwert h'
  - B „entschlüsselt“ die Signatur mit dem öffentlichen Schlüssel von A $K_{A,Pub}$ und vergleicht $h = h'$? Bei Übereinstimmung wurde die Nachricht nicht modifiziert und stammt von A
- Für die umgekehrte Kommunikation von B nach A erfolgt das Vorgehen analog

TinyPK nutzt mit RSA [11] ist ein bekanntes und weit verbreitetes asymmetrsiches Kryptographie-Verfahren.

3. SPINS – SECURITY PROTOCOLS FOR SENSOR NETWORKS

SPINS wurde 2001 an der „University of California“ entwickelt [10]. Das Ziel von SPINS ist es, eine „viele-zu-einer“-Kommunikation zu ermöglichen. Im Zentrum steht eine Basisstation, die auch als Gateway fungiert und über mehr Ressourcen als ein gewöhnlicher Knoten verfügt. SPINS bietet zwei Protokolle an; SNEP und µTESLA. SNEP bietet Schutz für die bidirektionale Kommunikation zwischen zwei Knoten durch Verschlüsselung, Authentisierung, Integrität und Frische. µTESLA ist eine Abänderung des TESLA-Protokolls [9] und dient dazu authentisierte, aber unverschlüsselte Broadcast-Nachrichten im Sensornetz zu verteilen.

3.1 SNEP – Sensor Network Encryption Protocol

Die Verschlüsselung erfolgt mit dem RC5-Algorithmus [12] im Counter-Mode (CTR). Dies hat mehrere Gründe:

- RC5 ist ein symmetrisches Verfahren. Es ist dadurch schneller und benötigt kürzere Schlüssel als asymmetrische Verfahren [12]
- RC5 verwendet keine S-Boxen für Permutationen und benötigt dadurch weniger Speicher [12]
- Der erzeugte Kryptotext hat die selbe Größe wie der zugehörige Klartext [10], was aufgrund der geringen Bandbreite im Netzwerk vorteilhaft ist

Der Counter-Mode (CTR) funktioniert wie folgt: In mehreren Durchläufen wird aus einem Zähler i und dem symmetrischen Schlüssel K ein temporärer Schlüssel K_i erzeugt. Insgesamt wird ein sog. Schlüsselstrom $K_0, K_1, \ldots, K_n$ generiert. Der Schlüsselstrom wird dann mit dem Klartext mittels XOR verknüpft, wodurch der Kryptotext entsteht (siehe Abb. 1).
Durch Verwendung des CTR wird die sog. *semantic security* Eigenschaft erreicht. D.h. gleiche Klartexte P werden zu verschiedenen Kryptotexten C und C' verschlüsselt.
Um Paket-Overhead einzusparen wird der Zähler-Wert nicht mit jeder Nachricht übertragen, sondern implizit von den kommunizierenden Knoten gespeichert und bei jedem Paket einer Kommunikation zwischen zwei Partnern erhöht.

Die Authentizität, Integrität und Frische von Nachrichten wird mittels MACs erreicht. In die MAC-Berechnung gehen neben der Nachricht und dem gemeinsamen Schlüssel von Sender und Empfänger auch der Zähler-Wert und ggf. eine *Nonce* ein.

3.2 µTESLA

µTESLA ist eine Abwandlung des TESLA-Protokolls [9] und dient dem authentisierten Versenden von Broadcasts. Die Nachrichten-Authentisierung erfolgt dabei wie bei SNEP durch MACs, die in der Nachricht mit übertragen werden.

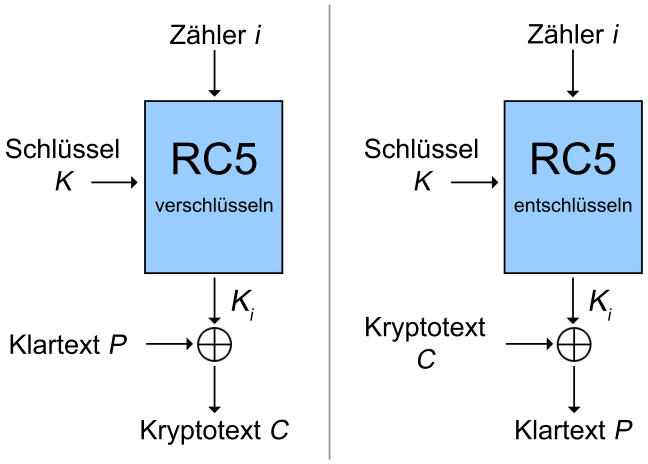

Abbildung 1: Ver- und Entschlüsselung mit RC5 im CTR-Mode

Wie bereits erläutert, geht in den MAC-Wert einer Nachricht ein geheimer Schlüssel ein. Würde für jede Broadcast-Nachricht der selbe Schlüssel verwendet, so könnte ein Angreifer jedoch leicht gültige Broadcasts versenden, sobald er im Besitz des Schlüssels wäre (was z.B. durch einen physikalischen Zugriff auf einen Knoten erfolgen könnte). Zur Lösung dieses Problems arbeitet µTESLA wie auch TESLA mit einer *key chain*.

Der Sender von Broadcast-Nachrichten erzeugt zunächst einen Zufalls-Bytestrom K_n. Anschließend wird darauf eine Hashfunktion h angewandt, womit ein anderer Pseudo-Zufallswert K_{n-1} erzeugt wird: $K_{n-1} = h(K_n)$ Durch wiederholtes anwenden der Hashfunktion auf den jeweils zuvor erzeugten Wert wird in jedem Durchlauf ein neuer Bytestrom erzeugt: $K_{n-1}, \ldots, K_2, K_1, K_0$. Diese Folge von Byteströmen entspricht der *key chain* und die Einträge K_i entsprechen den Schlüsseln, die in die spätere MAC-Berechnung der Broadcasts eingehen. Diese erzeugten Schlüssel sind zunächst nur dem Sender bekannt.

Nun wird die Zeit in Epochen $0, 1, 2 \ldots, n-1, n$ eingeteilt. Jeder Epoche i wird dabei ein Schlüssel K_i zugeteilt. Schickt der Sender in Epoche i eine Nachricht, so berechnet er den zugehörigen MAC mit dem Schlüssel K_i. Die Schlüssel der *key chain* werden also in der Reihenfolge n bis 0 erzeugt und in der Reihenfolge 0 bis n verwendet (siehe Abb. 2).

Empfängt ein Knoten eine Broadcast-Nachricht, so ist er zunächst nicht in der Lage die Nachricht zu authentisieren, da er den Schlüssel (der aktuellen Epoche) nicht kennt. Er muss die Nachricht daher zwischenspeichern. Nach Ablauf der Epoche i, in der die Nachricht gesendet wurde, teilt der Sender den Empfängern den Schlüssel K_i der letzten Epoche mit. Durch Erhalt des Schlüssels kann der Empfänger dann die gespeicherte Nachricht authentisieren. In einer Epoche können auch mehrere Nachrichten versendet werden. Der Empfänger muss dann alle Nachrichten speichern, bis der zugehörige Schlüssel gesendet wird. Aufgrund der knappen Ressourcen der Knoten darf eine Epoche nicht zu lange dauern, da ein Empfänger sonst nicht alle Nachrichten puffern kann. Außerdem müssen Sender und Empfänger

über „schwach synchronisierte“ Uhren verfügen [10].

Die Sicherheit dieses Verfahrens ist durch die Einweg-Eigenschaft der Hashfunktion gegeben: Der Sender berechnet den MAC in Periode i mit K_i. Wenn ein Angreifer gültige MAC-Werte berechnen wollte, so müsste er aus dem Schlüssel der vorherigen Epoche K_{i-1} den aktuellen Schlüssel K_i berechnen: $K_i = h^{-1}(K_{i-1})$. Das ist aber gerade die Umkehrung einer kryptographischen Hashfunktion, die nicht möglich ist. Abbildung 2 stellt die Erzeugung und Verwendung der *key chain* nochmals graphisch dar.

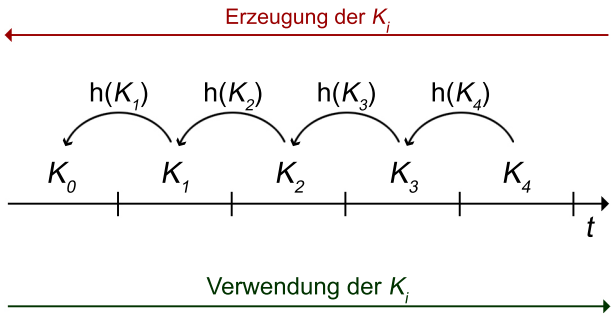

Abbildung 2: Erzeugung und Verwendung der K_i in der *key chain*

Wegen der knappen Ressourcen der Knoten können diese nach [10] selbst keine *key chain* berechnen und speichern. Diese Funktion bleibt der Basisstation vorbehalten. Muss ein Knoten einen Broadcast senden, so beschreibt [10] dazu zwei Möglichkeiten:

1. Der Knoten sendet die Broadcast-Nachricht mittels SNEP verschlüsselt an die Basisstation, welche dann einen Broadcast erzeugt
2. Der Knoten sendet den Broadcast. Die zugehörige *key chain* wird aber von der Basisstation erzeugt und verteilt. Vor dem Senden eines Broadcasts teilt die Basisstation dem Broadcast-Knoten den aktuellen Schlüssel via SNEP mit

4. TINYSEC

Wie in [6] kritisiert, wurde SPINS weder komplett spezifiziert noch vollständig implementiert. Außerdem ist SPINS wegen der Verwaltung von Zählern für jede Kommunikation zwischen Paaren von Knoten relativ komplex.
TinySec [6] erweitert ebenfalls TinyOS um kryptographische Funktionen. Es ist vollständig implementiert und kann zusammen mit TinyOS im Internet heruntergeladen werden. TinySec wurde 2004 an der „UC Berkeley“ vorgestellt. Das Ziel von TinySec ist es, eine sichere Kommunikation zwischen zwei beliebigen Knoten im Sensornetz zu ermöglichen. Eine Basisstation gibt es nicht. TinySec bietet drei mögliche Kommunikationsprotokolle an:

1. Klartext: unsichere Kommunikation mittels Nachrichten wie von TinyOS definiert (Kompatibilität zu TinyOS)
2. TinySec-Auth (authenticated): Nachrichten sind unverschlüsselt, aber authentisiert
3. TinySec-AE (authenticated encryption): Nachrichten sind verschlüsselt und authentisiert

4.1 TinySec-Auth

Eine Nachricht besteht aus einem 5 Byte Header, 0-29 Byte Nutzdaten, sowie einem 4 Byte Trailer. Der Header besteht aus drei Feldern: Zieladresse (Dest), Nachrichten-Typ (AM) und Nachrichtenlänge (Len). Der Trailer enthält den 4 Byte langen MAC. In den MAC geht neben den Nutzdaten auch der Header ein (siehe Abb. 3).

4.2 TinySec-AE

Die Nachricht besteht hier aus einem 8 Byte Header, ebenfalls 0-29 Byte Nutzdaten und ebenfalls einem 4 Byte Trailer für den MAC. Der Header enthält hier fünf Felder: Empfänger-Adresse (Dest), Nachrichten-Typ (AM), Länge (Len), Sender-Adresse (Src) und Counter (Ctr). Der Nutzdatenteil der Nachricht ist verschlüsselt (siehe Abb. 3).

4.3 Verschlüsselung

TinySec nutzt den Skipjack Algorithmus im Cipherblock-Chaining Mode (CBC). Im Gegensatz zu RC5, der in SNEP verwendet wird, ist Skipjack nicht patentiert (vgl. [6]).

Der CBC-Modus funktioniert wie folgt: Der Klartext wird in Blöcke gleicher Größe aufgeteilt: $P_0, P_1, \ldots, P_n$. Ein Block P_i wird unter Verwendung des Schlüssels K und eines zuvor erzeugten Kryptotextblocks C_{i-1} zu einem Kryptotextblock C_i verschlüsselt (siehe Abb. 4).
Bei der Verschlüsselung des ersten Blocks P_0 gibt es noch keinen Kryptotextblock, der in die Verschlüsselung eingehen könnte. Daher wird ein sog. Initialisation Vector (IV) verwendet.

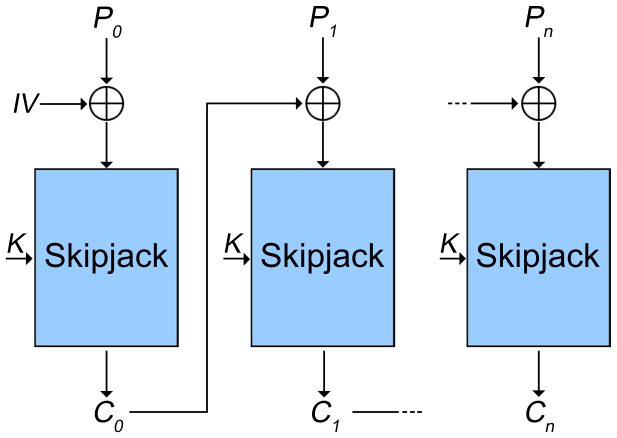

Abbildung 4: Verschlüsselung im CBC-Modus mit Skipjack

Der IV ist nicht geheim, sollte aber möglichst zufällig und einmalig sein. Außerdem sollte sich der IV nicht wiederholen, da es dann möglich ist, dass gleiche Klartexte zu gleichen Kryptotexten verschlüsselt werden.
Damit der Empfänger einer Nachricht diese entschlüsseln

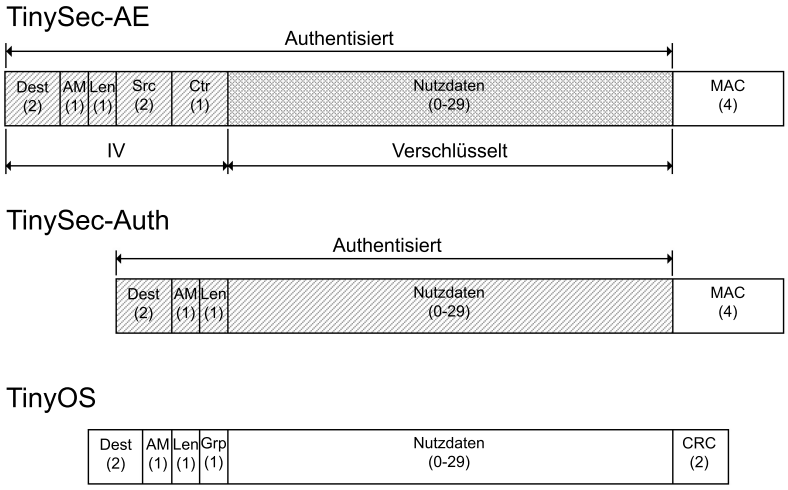

Abbildung 3: Nachrichten-Formate für TinySec-AE, TinySec-Auth und TinyOS [6]

kann, muss er den IV kennen. Da der IV aber möglichst groß sein sollte (um eine 2-malige Verwendung zu vermeiden), ergibt sich das Problem, dass viele zusätzliche Bytes für jedes Paket übertragen werden müssen. In TinySec-AE wird das Problem gelöst, indem der IV wie folgt zusammengesetzt wird: IV=Dest|AM|Len|Src|Ctr. Diese Werte entsprechen genau den Header-Feldern von TinySec-AE Nachrichten: Ziel-Adresse, Nachrichten-Typ, Länge, Quell-Adresse, Counter.
TinySec-AE ist hier effizient, da es für den IV die Felder verwendet, die ohnehin in jedem Nachrichten-Header enthalten sein müssen (außer Counter). Counter dient als fortlaufende Nummer dazu, eindeutige IVs zu erzeugen und ist zwei Byte groß. Es müssen für den IV also nur zwei zusätzliche Byte übertragen werden, obwohl der IV insgesamt 8 Byte lang ist.

4.4 Authentisierung

Für die Authentisierung von Nachrichten wird auch unter TinySec ein MAC verwendet. Zur Berechnung des MAC wird jedoch nicht extra eine kryptographische Hashfunktion wie MD5 oder SHA-1 verwendet, sondern der bereits implementierte Verschlüsselungs-Algorithmus Skipjack im CBC-Modus. Der Vorteil dieser Vorgehensweise wird durch die Code-Einsparung begründet [6].

Die Generierung von MACs durch eine CBC-Verschlüsselung funktioniert prinzipiell wie folgt: Die Nachricht P wird wie bei der Verschlüsselung in Teilblöcke $P_0, P_1, \ldots, P_n$ zerlegt. Die Blöcke werden nacheinander zu den C_i verschlüsselt. Die Kryptotexte C_0 bis C_{n-1} werden abschließend verworfen. C_n stellt den MAC-Wert der Nachricht P dar.

Da für Verschlüsselung und Authentisierung aus Sicherheitsgründen zwei verschiedene Schlüssel verwendet werden (jeder Knoten ist im Besitz dieser zwei Schlüssel), kann der C_n Block der durch die Verschlüsselung entsteht, nicht als MAC-Wert verwendet werden. Entsprechend muss der Verschlüsselungs-Algorithmus bei Tiny-AE-Nachrichten zwei mal durchlaufen werden, was relativ viel Energie benötigt.

5. TINYPK

Die bereits vorgestellten Systeme SPINS und TinySec benutzen symmetrische Kryptographie um eine gesicherte Kommunikation zu ermöglichen. TinyPK implementiert mit RSA [11] ein bekanntes asymmetrisches Kryptographie-Verfahren und wurde 2004 von „BBN Technologies“ vorgestellt [13]. TinyPK verfolgt drei Ziele:

1. Ein außenstehendes Gerät, die *External Party* (EP), die zunächst nicht zum Sensornetz gehört, zu authentisieren
2. Der EP sicher den symmetrischen Schlüssel K zu übermitteln, der im Sensornetz zur gesicherten Kommunikation genutzt wird (z.B. von TinySec)
3. Einen Knoten aus dem Netz gegenüber der EP zu authentisieren

Auf den Knoten sind nur die Public-Key-Operationen *Verschlüsselung* und *Prüfen einer Signatur* implementiert, da diese im Vergleich zu den Private-Key-Operationen *Entschlüsseln* und *Erzeugen einer Signatur* relativ wenig Ressourcen benötigen [13].

5.1 Verwendete Techniken

TinyPK verwendet zwei Techniken, die in Netzen mit leistungsstärkeren Clients ebenfalls weit verbreitet sind und die im folgenden kurz erläutert werden sollen.

5.1.1 Certification Authority (CA)

Eine CA ist eine Instanz, der ein Client bzw. Nutzer vertraut. Eine CA bürgt mittels Zertifikat dafür, dass ein öffentlicher Schlüssel zu einer bestimmten Partei gehört, beispielhaft: „Ich, die CA XYZ, versichere, dass $K_{A,Pub}$ dem Benutzer A gehört.“ Diese Zusicherung wird in Form von Zertifikaten, wie sie auch in SSL/TLS verwendet werden, ausgedrückt. Ein Zertifikat ist im wesentlichen eine Datenstruktur, die den Namen einer Partei, deren Public-Key und

eine Signatur enthält. Die Signatur ist von der CA ausgestellt. Mit Hilfe des Public-Key der CA kann die Signatur und damit die Echtheit des Zertifikates überprüft werden.

5.1.2 Diffie-Hellman Verfahren (DH)

Das Diffie-Hellman Verfahren dient dazu, einen gemeinsamen Schlüssel zwischen zwei Teilnehmern A und B zu etablieren, ohne dass der Schlüssel dabei übertragen wird. Dies ist vor allem dann nötig, wenn der Schlüssel über einen potentiell (kryptographisch) unsicheren Kanal ausgehandelt werden sollen. Das Verfahren ist weit verbreitet und gut dokumentiert, wie [5] entnommen werden kann.

5.2 Authentisierung einer External Party

Es werden folgende Annahmen und Voraussetzungen gemacht (vgl. [13]):

- Die Sensorknoten kommunizieren untereinander gesichert mittels symmetrischer Kryptographie-Verfahren, z.B. mit TinySec. Damit kennen alle, oder zumindest einige, einen gemeinsamen symmetrischen Schlüssel K
- Die Knoten kennen eine Certification Authority (CA), der sie vertrauen. Die Knoten sind im Besitz des öffentlichen Schlüssels der CA $K_{CA,Pub}$
- Die External Party kann zu Beginn nicht mit dem Netz kommunizieren, da sie den Schlüssel K nicht kennt
- Die EP kann — im Gegensatz zu den Knoten — auch Private-Key Operationen durchführen, da sie über mehr Rechenleistung verfügt
- Die EP besitzt eine Signatur $Sig(K_{EP,Pub}; K_{CA,Priv})$ ihres öffentlichen Schlüssels $K_{EP,Pub}$, die von der CA ausgestellt, (d.h. von der CA mit $K_{CA,Priv}$ signiert, wurde

Die Authentisierung der EP läuft wie folgt ab: Die EP sendet zwei Nachrichten:

1. Die Signatur ihres Public Key $Sig(K_{EP,Pub}; K_{CA,Priv})$
2. Eine Signatur $Sig((Nonce, Checksum); K_{EP,Priv})$ über eine *Nonce* und eine Prüfsumme von $K_{EP,Pub}$. Diese Signatur wurde nicht von der CA, sondern von der EP erstellt

Der Knoten „entschlüsselt“ die erste Signatur mit dem öffentlichen Schlüssel der CA $K_{CA,Pub}$. Anschließend ist der Knoten im Besitz des öffentlichen Schlüssels der EP $K_{EP,Pub}$. Mit diesem kann der Knoten nun die zweite Signatur „entschlüsseln“ und erhält so die *Nonce* und die Prüfsumme. Die *Nonce* dient dazu eine Replay-Attacke zu erkennen, womit das Schutzziel der Frische erreicht wird. Verläuft die Prüfung positiv, ist die EP authentisiert.

5.3 Übermittlung des Sensornetzschlüssels

Der Knoten sendet nun eine Antwortnachricht: $Enc((Nonce, K); K_{EP,Pub})$. Diese enthält den symmetrischen Schlüssel K und die empfangene *Nonce*. Die Nachricht ist mit dem öffentlichen Schlüssel der EP $K_{EP,Pub}$ verschlüsselt.

5.4 Authentisierung eines Knoten

Da die Knoten keine Private-Key Operationen und somit auch keine Signaturberechnungen durchführen können, besitzt jeder Knoten einige statische Informationen, die zur Authentisierung verwendet werden:

- Einen ID-Text, der in bestimmter Formatierung unter anderem folgende Informationen enthält: Serien-Nummer, Konstruktionsdatum, Initial-Daten
- Einen statischen, öffentlichen Diffie-Hellman-Wert $DH_{Node,Pub}$
- Eine Signatur des Diffie-Hellman-Wertes, die von der CA ausgestellt wurde: $Sig(DH_{Node,Pub}; K_{CA,Priv})$

Die EP sendet nun erneut eine Nachricht an den Knoten, diesmal jedoch nicht im Klartext, sondern verschlüsselt im TinySec-AE-Format: $Enc(Nonce, DH_{EP,Pub}; K)$. Die Nachricht enthält eine *Nonce* und einen temporären DH-Wert $DH_{EP,Pub}$. Der Knoten empfängt die Nachricht und berechnet aus den DH-Werten einen neuen Schlüssel K_{DH}. Anschließend berechnet er einen MAC-Wert aus der empfangenen *Nonce* und dem neu erstellten Schlüssel K_{DH}: $MAC(Nonce; K_{DH})$. Der MAC-Wert wird zusammen mit dem signierten statischen DH-Wert des Knotens $Sig(DH_{Node,Pub}; K_{CA,Priv})$, sowie dem ID-Text zurück gesandt.

Die EP empfängt die Antwort und ist damit im Besitz des öffentlichen DH-Schlüssels des Knotens $DH_{Node,Pub}$. Damit kann die EP ebenfalls den Schlüssel K_{DH} berechnen und mit diesem den empfangenen MAC-Wert prüfen. Verläuft die Prüfung positiv, wird der ID-Text des Knoten zusammen mit dem erzeugten DH-Schlüssel K_{DH} in einer Datenbank gespeichert.

Sendet zukünftig ein Sensornetz-Knoten Daten an die EP, muss er für jede Nachricht einen MAC berechnen und mitsenden, in den der vereinbarte DH-Schlüssel K_{DH} eingeht. Die EP prüft dann die MAC anhand des DH-Schlüssels. Ist der MAC gültig, ist die Nachricht bzw. der Absender authentifiziert und die Nachricht wird akzeptiert. Abbildung 5 stellt den Protokollablauf nochmals graphisch dar.

6. VERGLEICH UND BEWERTUNG

Die vorgestellten Systeme SPINS, TinySec und TinyPK verwenden bzw. erweitern das Open-Source-Betriebssystem TinyOS, das an sich keine Sicherheitsmechanismen für die Übertragung bietet. Alle drei Systeme verfolgen verschiedene Ziele. SPINS und TinySec implementieren dazu die effizienteren symmetrischen Kryptographie-Verfahren, während TinyPK asymmetrische Verfahren umsetzt.

SPINS war einer der ersten Versuche um die Kommunikation in Sensornetzen zu schützen. Der Schwerpunkt liegt in der sicheren Kommunikation zwischen einem Knoten und einer Basisstation (SNEP), bzw. in der gesicherten Übertragung von Broadcasts (µTESLA). TinySec wurde später entwickelt. Der Schwerpunkt liegt hier in der gesicherten Kommunikation zwischen beliebigen Knoten. Das Konzept einer Basisstation bzw. von Broadcast-Nachrichten fehlt. In

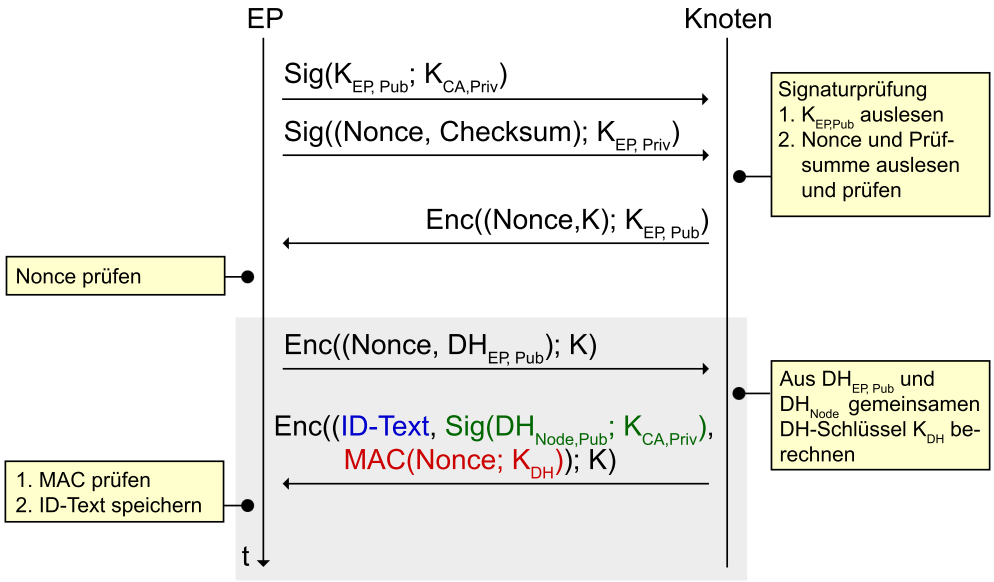

Abbildung 5: Protokollablauf der TinyPK Authentisierungen

SPINS besitzt jeder Knoten einen eigenen Schlüssel, welcher der Basisstation bekannt ist. In TinySec teilen sich alle Knoten eines Netzes einen gemeinsamen Schlüssel.

SPINS und TinySec verwenden jeweils einen Zähler, der in die Verschlüsselung bzw. den MAC eingeht. SPINS verwaltet diese Zähler lokal auf jedem Knoten. D.h. für jede Kommunikation, die ein Knoten mit einem anderen Knoten, bzw. der Basisstation, betreibt, muss er einen Zähler-Wert speichern und verwalten (stateful Protokoll). Es sei jedoch angemerkt, dass es die Grundidee von SPINS ist, eine gesicherte Kommunikation von Knoten zu Basisstation bzw. umgekehrt zu ermöglichen. Es ist zwar auch möglich, dass zwei Knoten direkt miteinander „sprechen“, das ist jedoch nicht der Schwerpunkt des Konzepts.
Anders hingegen in TinySec. Hier soll jeder Knoten mit jedem anderen Knoten sicher kommunizieren können. Ein Knoten speichert den Zähler für den Nachrichten-Schutz daher nicht lokal, sondern überträgt ihn im Header jeder Nachricht, wodurch der Aufwand zur Verwaltung der Zähler-Werte entfällt. Dadurch wird allerdings für jedes Paket ein größerer Overhead übertragen, was mehr Energie benötigt und somit die Laufzeit der Knoten verkürzt.
Die Authentisierung bzw. der Integritätsschutz von Nachrichten erfolgt in SPINS wie in TinySec analog. Beide Systeme verwenden MACs die in jeder Nachricht mit übertragen werden.
SPINS bietet gegenüber TinySec mit µTESLA die Möglichkeit authentisierte Broadcast-Nachrichten zu versenden. Diese Funktion ist in TinySec nicht implementiert.

Für die „Authentisierung“ wird in TinySec ein MAC-Wert genutzt, der durch einen Schlüssel berechnet wird, der im Netz allgemein bekannt ist. Dadurch kann ein Empfänger sicherstellen, dass die Nachricht während der Übertragung nicht verändert wurde. Ein Empfänger kann aber nicht nachvollziehen, ob die Nachricht wirklich von einer vertrauenswürdigen Quelle stammt. Jeder Knoten, der im Besitz des Schlüssels ist, kann „authentisierte“ Nachrichten versenden. TinySec erreicht daher zwar das Schutzziel der Integrität, das der echten Authentizität aber nicht.
Im weiteren wird in TinySec-Auth Nachrichten kein Wert übertragen, mit dem ein Replay-Angriff erkannt werden könnte. Entsprechend wird mit TinySec-Auth das Schutzziel der Frische nicht erreicht.

Sowohl SPINS als auch TinySec verwenden ausschließlich symmetrische Kryptographie mit der Begründung asymmetrische Verfahren benötigen zu viele Ressourcen [10, 6]. Daher wird in beiden Systemen ein gemeinsamer Schlüssel für die Kommunikation benötigt. In TinySec teilen sich alle Knoten den selben Schlüssel. In SPINS teilt sich jeder Knoten einen Schlüssel mit der Basisstation. Es gibt in keinem der beiden Systeme die Möglichkeit Schlüssel automatisch zu verteilen. Das stellt einen großen Nachteil dar, da ein Schlüssel nur für eine begrenzte Zeit verwendet werden sollte. Im Idealfall wird für jede neue Kommunikation zwischen zwei Partnern ein neuer Schlüssel verwendet. Da die Systeme SPINS und TinySec keine automatische Schlüsselverteilung vorsehen, müsste ein neuer Schlüssel durch physikalischen Zugriff auf den oder die jeweiligen Knoten verteilt werden. Dieses Vorgehen ist aufwendig und wird folglich nicht oft durchgeführt werden.

Das Primäre Ziel von TinyPK ist nicht die sichere Kommunikation zwischen Knoten zu ermöglichen, sondern eine Authentisierung zwischen einer External Party (EP), die nicht zum Sensor-Netz gehört, und einem Sensornetz durchzuführen. Nach erfolgreicher Authentisierung wird der symmetrische Schlüssel K, der vom Sensornetz für die sichere Kom-

munikation verwendet wird, zur EP übertragen. TinyPK ist also als Erweiterung für andere Systeme wie TinySec gedacht. TinyPK nutzt eingeschränkt asymmetrische Kryptographie, benötigt zur Vertrauensbildung jedoch eine Certification Authority (CA). Die External Party muss — ähnlich wie die Basisstation in SPINS — über mehr Rechenleistung als ein gewöhnlicher Knoten verfügen, da sie auch in der Lage sein muss, Private-Key-Operationen durchzuführen.

7. ANGRIFFSMÖGLICHKEITEN

Alle vorgestellten Systeme arbeiten mit Funk-Kommunikation. Entsprechend sind alle Systeme durch gezielte Störung der jeweiligen Frequenzen angreifbar (DoS-Angriff). Schutz vor dieser Angriffsart wird durch die vorgestellten Systeme explizit nicht adressiert.

In TinySec teilen sich alle Knoten einen gemeinsamen Schlüssel zur sicheren Kommunikation. Das Vertrauen der Knoten untereinander basiert lediglich auf dem Besitz des Schlüssels. Damit hängt die Sicherheit des Systems stark vom physikalischen Schutz der einzelnen Knoten ab. Schafft es ein Angreifer den Schlüssel aus einem Knoten auszulesen, ist das gesamte System ausgehebelt und wirkungslos.

In SPINS tritt dieses Problem zunächst nicht so drastisch auf, da jeder Knoten einen eigenen Schlüssel besitzt, den außer ihm nur die Basisstation kennt. Erlangt ein Angreifer physikalischen Zugriff auf einen Knoten um so den Schlüssel auszulesen, ist zunächst nicht die Kommunikation des gesamten Netzes gefährdet. Allerdings ist ein Angreifer in der Lage die Kommunikation zwischen Basisstation und dem kompromittierten Knoten zu stören bzw. abzuhören. Im Weiteren ist es einem Angreifer auch möglich, sich als Knoten gegenüber der Basisstation auszugeben (spoofing). Auf diesem Weg kann ein Angreifer dann nicht nur gefälschte Nachrichten an die Basisstation senden, sondern über diese auch Broadcast-Nachrichten an das gesamte System verteilen.
Gelingt es einem Angreifer die Basisstation zu kompromittieren, so ist es prinzipiell auch möglich die Schlüssel aller Knoten auszulesen. Damit wäre das System wirkungslos.

Ein weiterer Angriff auf SPINS-Knoten ist möglich, wenn sich ein Angreifer als Basisstation ausgibt und mittels µTESLA Broadcast-Nachrichten sendet. Knoten müssen diese Nachrichten solange zwischenspeichern, bis sie einen Schlüssel erhalten haben, um die Nachrichten nachträglich zu authentisieren. Ein Angreifer kann hier viele Nachrichten senden, ohne je einen Schlüssel zu verteilen. Die Knoten werden so mit Paketen geflutet (flood-Angriff).

8. ZUSAMMENFASSUNG

Sichere Kommunikation in Sensor-Netzen ist trotz geringer Bandbreite und geringen Ressourcen möglich. Die vorgestellten Systeme SPINS, TinySec und TinyPK verfolgen unterschiedliche Ziele und erreichen diese mit unterschiedlichen Methoden. SPINS geht davon aus, dass im Sensornetz viele Knoten zu einer Basisstation „sprechen“, bzw. umgekehrt. TinySec kennt keine zentrale Basisstation. Hier sollen alle Knoten untereinander sicher kommunizieren können. SPINS und TinySec verwenden ähnliche Lösungsansätze mit symmetrischen Kryptographie-Verfahren. TinyPK implementiert Teile von Public-Key Verfahren auf Knoten, benötigt jedoch immer einen Kommunikationspartner (EP), der über mehr Rechenleistung als ein gewöhnlicher Knoten verfügt. Das Ziel von TinyPK ist die sichere Authentisierung einer EP gegenüber dem Sensornetz und umgekehrt. TinyPK kann daher beispielsweise als Erweiterung zu TinySec verwendet werden.

9. LITERATUR

[1] *M 3.23 Einführung in kryptographische Grundbegriffe*, `https://www.bsi.bund.de/` Bundesamt für Sicherheit in der Informationstechnik (BSI), 2006.

[2] *G 4.35 Unsichere kryptographische Algorithmen*, `https://www.bsi.bund.de/` Bundesamt für Sicherheit in der Informationstechnik (BSI), 2009.

[3] H. Zimmermann. OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol - Version 1.2. RFC 5246, IETF, August 2008.

[5] C. Eckert. *IT-Sicherheit*. Oldenbourg Wissensch.Vlg, 2006.

[6] C. Karlof, N. Sastry, and D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks In *SenSys*, Baltimore, Maryland, USA, November 2004.

[7] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, IETF, November 1998.

[8] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*. Springer Berlin / Heidelberg, 2005.

[9] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *CryptoBytes*, 5(2):2–13, 2002.

[10] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Mobile Computing and Networking*, Rome, Italy, 2001.

[11] R. Rivest, A. Shamir, and L. Adleman. A Method for obtaining Digital Signatures and Public Key Cryptosystems. *Communication of the ACM*, 21(2):120–126, 1978.

[12] R. L. Rivest. *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science (LNCS)*, chapter The RC5 Encryption Algorithm, pages 86–96. Springer Berlin / Heidelberg, 1995.

[13] R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *SASN*, Washigton, DC, USA, October 2004.