Schutz eingebetteter Systeme gegen physische Angriffe

Johannes Bauer[1,2] · Felix Freiling[2]

Bosch Software Innovations GmbH[1]
johannes.bauer@bosch-si.com
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)[2]
felix.freiling@fau.de

Zusammenfassung

Im Kontext von kleinsten eingebetteten Geräten und Machine-to-Machine-Kommunikationsprotokollen innerhalb des *Internet of Things* werden zumeist Sicherheitsprotokolle auf Basis symmetrischer Kryptografie verwendet, da diese relativ einfach durch stark ressourcenbeschränkte Geräte handhabbar sind. Die dazu notwendigen geheimen kryptografischen Schlüssel werden aber typischerweise im Flash eines Microcontrollers abgelegt und sind durch physische Angriffe wie Power Analysis oder Decapping gefährdet. Die zur Abwehr derartiger Angriffe notwendigen Hardware-Sicherheitsmodule (HSM) werden üblicherweise nicht in Betracht gezogen weil sie als teuer gelten und häufig über proprietäre Schnittstellen verfügen, die nur schwer in gängige Protokolle integriert werden können. Diese Arbeit beschreibt einen effizienten und generischen Ansatz, wie man ein kostengünstiges HSM, das lediglich symmetrische Kryptografie auf Basis des SHA-256-Hashalgorithmus verwendet, zur Absicherung des weit verbreiteten und akzeptierten Sicherheitsprotokollrahmens TLS verwenden kann. Konkret zeigen wir die Integration eines symmetrischen HSM Atmel ATSHA204A in den Handshake von *datagram TLS* (DTLS).

1 Einführung

Durch die steigende Anzahl an Endgeräten im Zuge des *Internet of Things* (IoT) wachsen auch die Anforderungen an die Sicherheit der verwendeten Geräte. Zu den Anforderungen zählen heute einerseits die Verwendung von sicheren Kommunikationsprotokollen für die Ende-zu-Ende-Verschlüsselung. Andererseits wird auch eine kryptografisch eindeutige Identifizierung und Authentifizierung von Teilnehmern erwartet, um etwa kompromittierte Endknoten aus einem gesicherten Netzwerk aussperren zu können [FaHK13]. Im Kontext von kleinsten eingebetteten Geräten und *Machine-to-Machine*-Kommunikationsprotokollen basieren existierende Lösungen heute meist auf der Verwendung *symmetrischer* Kryptografie, da Algorithmen wie AES relativ einfach durch stark ressourcenbeschränkte Geräte handhabbar sind. Brachmann et al. [BKMK12] unterstreichen die Wichtigkeit guter Ende-zu-Ende Verschlüsselung und die speziellen Probleme im Hinblick auf Einbindung stark ressourcenbeschränkter Clients in solche Netzwerke.

Entscheidend für die Sicherheit der kryptografischen Algorithmen ist der Schutz eines geheimen Schlüssels, den sich Client und Server teilen. Diesen Schlüssel bezeichnet man als *pre-shared key* oder PSK. Während der Schlüssel serverseitig meist einfach abzusichern ist, wird der Schlüssel clientseitig im Normalfall im Flash-Speicher eines Microcontrollers abgelegt.

P. Schartner · N.N (Hrsg.) · D•A•CH Security 2015 · syssec (2015) 1-10.

Dies birgt allerdings das Risiko, dass ein Angreifer durch physische Angriffe wie *Power Analysis* oder das physische Öffnen des Chip-Gehäuses (sog. *Decapping*) und darauf folgendes Auslesen des Microcontrollers Kenntnis der privaten Schlüssel erlangt; beides sind Angriffe, die heute mit wenig Aufwand bei vielen existierenden Systemen durchführbar sind [Skor05]. Es ist darum bemerkenswert, dass aktuelle Arbeiten zur sicheren Speicherung von Daten in solchen verteilten Netzwerken wie etwa die von Bagci et al. [BPRR+12] explizit derartige physische Angriffe ausschließen.

Um hardwareseitige Angriffe zu verhindern werden üblicherweise Hardware-Sicherheitsmodule (HSM) eingesetzt, die wirksame Gegenmaßnahmen gegen Decapping und das Auslesen von Schlüsseln integriert haben. Die verwendeten Microchips werden vom Hersteller gehärtet, indem auf Halbleiterebene spezielle Sensoren verbaut werden, die beispielsweise ein Öffnen des Gehäuses detektieren und daraufhin sämtliches beinhaltetes Schlüsselmaterial vernichten. Auch kommen routinemäßig Maßnahmen zum Einsatz, die Power Analysis verhindern oder zumindest erschweren sollen. [Mang04]

Der Einsatz solcher HSMs gegen physische Angriffe wird auch in der Literatur empfohlen [BSPS+11], es fehlen jedoch Vorschläge für eine konkrete und vor allem kostengünstige Realisierung. Seit einiger Zeit existieren allerdings kostengünstige HSMs, die auch für den Einsatz in ressourcenbeschränkten autonomen Systemen geeignet sind (etwa das HSM Atmel ATSHA204A, Preis ca. $0.50). Allerdings verfügen diese in der Regel nur über proprietäre Schnittstellen, die eine interoperable Integration in standardisierte und offene Sicherheitsprotokolle wie TLS erschweren.

Diese Arbeit beschreibt einen effizienten und generischen Ansatz, wie man ein kostengünstiges HSM, das lediglich symmetrische Kryptografie auf Basis des SHA-256-Hashalgorithmus verwendet, zur Absicherung des weit verbreiteten und akzeptierten Sicherheitsprotokollrahmens TLS verwenden kann. Konkret zeigen wir die Integration eines symmetrischen HSM Atmel ATSHA204A in den Handshake von *datagram TLS* (DTLS).

Die Sicherheitslösungen von Bagci et al. [BPRR+12] können somit auf einfache Art und Weise auf ein Angreifermodell erweitert werden, das explizit physische Angriffe einschließt.

Der einzige Unterschied zwischen DTLS und TLS besteht darin, dass DTLS spezielle Protokolladaptierungen enthält, die es erlauben, eine unzuverlässige Transportschicht wie UDP/IP zu verwenden. Dies kommt der Verwendung auf ressourcenbeschränkten Knoten sehr entgegen, da diese keinen eigenen TCP/IP-Stack mitführen müssen. Sämtliche Betrachtungen, die wir für die eine Variante treffen gelten also analog für die jeweils andere Variante.

2 Hintergrund zu (D)TLS

Die Rechenleistung von Mikrocontrollern, die in eingebetteten Systemen zum Einsatz kommen, ist in den letzten Jahren rasant gestiegen. Dieser Leistungszuwachs führt auf Applikationsseite dazu, dass es eine Abkehr von proprietären und sicherheitstechnisch oftmals mangelhaften Protokollen gibt. Stattdessen findet eine Zuwendung zu standardisierten und offenen Protokollen statt, die bereits durch die Verwendung im Internet jahrzehntelang erprobt wurden. Ein solches Protokoll ist Transport Layer Security (TLS). [DiRe08] TLS ist weniger ein einziges Protokoll als vielmehr ein flexibler Baukasten, bei dessen Verwendung Anwender sich nicht im Detail mit kryptografischen Konstruktionen befassen müssen.

TLS definiert eine große Bandbreite an Optionen hinsichtlich der eingesetzten kryptografi-

schen Algorithmen. Alle für den Sitzungsaufbau relevanten Algorithmen sind in der Nomenklatur von TLS zu einem Text-String konkateniert, den man *cipher suite* nennt. Für stark ressourcenbeschränkte Geräte eignen sich in TLS vor allem diejenigen cipher suites, die zum Schlüsselaustausch auf vorher fixierte symmetrische Schlüssel (*pre-shared secret key*, PSK) zurückgreifen.

Im Gegensatz zu asymmetrischen Verfahren wie RSA oder ECC, bei denen Berechnungen mit modularer Langzahlarithmetik durchgeführt werden müssen, kommen bei PSK-basierenden cipher suites lediglich symmetrische Algorithmen zum Einsatz. Diese sind auch auf Knoten, die stark ressourcenbeschränkt sind, hinreichend effizient ausführbar.

Beispielsweise impliziert die cipher suite `TLS_PSK_WITH_AES_128_CCM_8` die Verwendung eines PSK, aus dem Sitzungsschlüssel abgeleitet werden, sowie die Verwendung des AES-128 Algorithmus zur Datenverschlüsselung. Dieser wird im Betriebsmodus CCM (Counter with CBC-MAC [WhHF03]) verwendet, um sowohl Verschlüsselung als auch Authentifizierung der Daten zu gewährleisten. Es ist impliziert, dass zur Ableitung der tatsächlich verwendeten symmetrischen Schlüssel durch die Verschlüsselungsfunktion AES-128 eine Schlüsselableitung mittels HMAC-SHA256 erfolgt.

Bei der praktischen Realisierung wird während der Provisionierung – also noch innerhalb der Produktionsstätte – jedem Knoten ein eigener, zufällig gewählter, PSK eingebracht. Auf Serverseite existiert eine Datenbank über alle im Feld verwendeten Knoten und deren korrespondierende PSKs. Zu Beginn der Kommunikation muss von beiden Seiten entschieden werden, welcher Schlüssel zum Einsatz kommen soll. Auf der Seite des Clients ist das im Regelfall nur ein einziger Schlüssel, der in Frage kommt. Auf Seite des Servers ist zunächst gänzlich unklar, welcher Schlüssel verwendet werden muss, um erfolgreich mit dem Client zu kommunizieren. Um dieses Problem zu lösen sieht TLS-PSK zwei spezielle Nachrichten vor: ein sogenannter *PSK Identity Hint* vom Server an den Client sowie die *PSK Identity* vom Client an den Server.

In diesen Nachrichten übermittelt jede Seite an ihr Gegenüber einen eindeutigen Bezeichner wie beispielsweise eine Seriennummer. Beide Seiten wählen dann jeweils den Schlüssel aus, den ihr Gegenüber erwartet. Typischerweise findet dabei ein einfaches Nachschlagen des kryptografischen Schlüssels aus einem assoziativen Datenfeld statt.

Wie genau das Verfahren, einen PSK aus dem gegebenen Identity Hint zu wählen, im Detail allerdings implementiert ist, ist in der Spezifikation von TLS-PSK absichtlich offen gelassen. Die Autoren schreiben deswegen kein verbindliches Vorgehen vor, weil es dutzende verschiedene Datenquellen geben könnte, aus denen die kryptografischen Schlüssel stammen könnten. So kommt beispielsweise in Frage, dass die Schlüssel direkt in den Quellcode des Programms eingebettet sind. Es ist aber auch legitim, sie aus einer Datenbank oder einer Datei zu lesen.

Um nun an ein solches System ein Hardwaresicherheitsmodul anzubinden, welches über eine proprietäre Schnittstelle verfügt, liegt es zunächst nahe, sich einfach eine eigene cipher suite zu definieren und diese dann auf Server- sowie Clientseite zu implementieren. Dies bringt aber den immensen Nachteil mit sich, dass es sich zum einen um eine nicht standardisierte Veränderung des TLS-Protokolls handelt; alle standardkonformen Bibliotheken sind streng genommen sogar dazu gezwungen, eine solche Verbindung zu terminieren. Zum anderen ist es erforderlich, dass ein Eingriff in die Struktur der TLS-Bibliothek selbst vorgenommen werden muss, um die entsprechenden Befehle zu implementieren. Dies schließt eine Verwendung von Bibliotheken, bei denen aus rechtlichen oder lizenztechnischen Gründen kein Eingriff in den Sourcecode genommen werden kann oder darf, vollständig aus.

Wir wählen daher einen anderen Ansatz: Wir verwenden eine standardisierte cipher suite und verwenden die in TLS integrierten Schlüsselableitungsfunktionen unverändert. Statt der Implementierung einer eigenen cipher suite nutzen wir die Freiheit bezüglich der *Schlüsselauswahl* des PSK aus einem vorgegeben Identity Hint, um darauf basierend ein symmetrisches HSM anzubinden. Dies wird im Detail in Abschnitt 3 beschrieben.

3 Idee und Implementierung

HSMs können symmetrische Schlüssel sicher speichern. "Sicher" bedeutet hierbei, dass die Schlüssel nach dem Speichern nicht mehr ausgelesen sondern lediglich verwendet werden können. Unser Verfahren funktioniert für alle HSMs, die einen *Message Authentication Code* auf Basis des geheimen Schlüssels und einer kryptografischen Hashfunktion berechnen können (z. B. HMAC [KrBC97]).

Die Idee unseres Verfahrens beruht darauf, den für TLS notwendigen PSK aus dem im HSM gespeicherten Schlüssel effizient abzuleiten, anstatt den PSK direkt im HSM zu speichern. Wir werden dadurch unabhängig von proprietären Schnittstellen und Datenformaten heutiger HSMs, die nicht auf ihre Verwendung in TLS-PSK zugeschnitten sind. Bei unserer Idee wird zum Provisionierungszeitpunkt ein Elterschlüssel K im HSM gespeichert, aus dem dann dynamisch zur Laufzeit der jeweils gültige PSK abgeleitet wird.

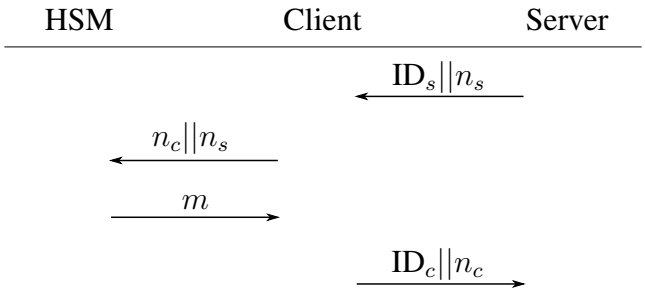

Abb. 1: Schlüsselaushandlung unter Verwendung eines symmetrischen HSM

Unser Verfahren funktioniert wie folgt (siehe auch Abb. 1): Wir integrieren eine 16 Byte lange Zufallszahl (*Nonce*) in die jeweiligen Identity Hint-Nachrichten des Clients und des Servers. Diese Nonces werden mit der ursprünglichen Identitätsinformation konkateniert.

Die erste Nachricht des Servers enthält neben dem eigentlichen PSK Identity Hint ID_s eine Nonce n_s. Der Client generiert nun selbst eine Nonce n_c und verwendet das HSM inklusive dem darin sicher gespeicherten Schlüssel K, um über die Konkatenation $n_c||n_s$ einen MAC in einem Modul-proprietären Format berechnen zu lassen. Der resultierende MAC-Wert m wird nun vom Client innerhalb seiner Implementierung von TLS direkt als PSK verwendet.

Damit der Server die Berechnung des PSK selbst nachvollziehen kann, liefert der Client dem Server in der PSK Identity Nachricht wiederum seinen eigentlichen Identity Hint sowie die seinerseits verwendete Nonce n_c. Damit kann der Server den Schlüssel K nachschlagen und die MAC-Berechnung im selben Format wie der Client durchführen. Hierfür muss er die konkreten Randbedingungen des Clients kennen (z.B. welches Modul mit welchem Verfahren konkret zum Einsatz kommt). Damit kann auch der Server den PSK berechnen und somit kann auf beiden Seiten die weitere Schlüsselableitung gemäß des TLS-Protokolls erfolgen.

4 Sicherheitsbetrachtung

Konkrete Implementierungen von TLS-Bibliotheken haben ähnliche Ansätze, wenn sie mit einer TLS-PSK cipher suite betrieben werden: Es existiert in der Regel auf Seite des Clients eine Callback-Funktion, die von der verwendeten TLS-Bibliothek während des Verbindungsaufbaus aufgerufen wird. Dieser Funktion wird der vom Server zurückgelieferte PSK Identity Hint als Parameter übergeben und es wird von ihr erwartet, dass Sie das passende symmetrische Geheimnis – eben den *pre-shared key* – zurückliefert. Für eine Anwendung ohne Hardware-Security-Modul wird hier typischerweise ein Nachschlagen in einem assoziativen Datenfeld (*dictionary*) implementiert. Eine solche Callback-Funktion ist sinnvoll und üblich, da sie eine Entkopplung des Speicherungsmechanismus für die Schlüssel gewährleistet.

Es ist wichtig zu betonen, dass für die Sicherheit des TLS-Protokolls in der Implementierung der PSK-Callback-Funktion eine Schlüsselableitung keinesfalls notwendig ist. Vielmehr wird der PSK durch die TLS-Bibliothek einem separaten Schlüsselableitungs-Mechanismus unterzogen. Um einen sichere Verbindung zu gewährleisten ist es also hinreichend, dass die PSK-Callback-Funktion die *Identitätsfunktion* auf einem vorher festgelegten, geheimen, PSK erfüllt.

Innerhalb der Sicherheitsbetrachtung muss also lediglich gewährleistet werden, dass die implementierte Lookup-Funktion aus kryptografischer Sicht besser als die Identitätsfunktion ist, um zumindest dasselbe Sicherheitsniveau zu erreichen, das auch ohne Hardware-Sicherheitsmodul gegeben wäre. Unser Verfahren verwendet den durch das Hardware-Sicherheitsmodul abgeleiteten MAC-Wert m nur als PSK zur Parametrierung von TLS; in der Folge bedeutet dies, dass die Schlüsselableitung von TLS *zusätzlich* zu unserer Schlüsselableitung verwendet wird und diese keinesfalls ersetzt.

```
32  KKKK KKKK  KKKK KKKK  KKKK KKKK  KKKK KKKK
32  SSSS SSSS  SSSS SSSS  CCCC CCCC  CCCC CCCC
 4  M M MM
11  OOOO OOOO OOO
 9  NNNN NNNN N
```

Abb. 2: Eingangsdaten für die SHA-256-Funktion beim MAC-Kommando des ATSHA204A

Das von uns verwendete Hardware-Sicherheitsmodul ATSHA204A stellt ein Kommando bereit, das `MAC` genannt wird und welches wir für unsere Implementierung im Folgenden verwenden werden. Es berechnet in einem exakt definierten Format eine SHA-256-Summe über Eingangsdaten statischer Länge. Abb. 2 zeigt diese Eingangsdaten für die vom ATSHA204A verwendete MAC-Funktion. Über die gezeigten 88 Bytes Daten (jedes Zeichen entspricht hierbei einem Byte) wird in der dargestellten Reihenfolge die SHA-256 Hashfunktion gebildet. [StTe02] Die einzelnen Komponenten sind im Detail:

- `K`: 32 Bytes langer Pre-Shared Key. Dies ist der Modulschlüssel K, der bei Produktion einmalig in das Modul eingebracht wird und auch auf Serverseite verfügbar sein muss.
- `S` und `C`: Zusammengenommen 32 Bytes an Daten, die sich jeweils in 16 Bytes aufteilen, die für die Nonce des Servers verwendet werden (`S`) und 16 darauf folgende Bytes, die für die Nonce des Clients (`C`) verwendet werden.
- `M`: 4 konstante Bytes, die den verwendeten MAC-Modus kodieren. Diese können als konstant und öffentlich betrachtet werden.

- `O`: 11 Bytes Daten im one-time programmable (OTP) Bereich des ATSHA204A. Diese Daten werden bei Produktion zufällig gewählte und zusammen mit dem Schlüssel K in das Modul eingebracht. Sie haben dieselben Schutzanforderungen wie K selbst. Auch die OTP-Daten müssen serverseitig bekannt sein, um die Schlüsselableitung erfolgreich durchführen zu können.
- `N`: 9 Bytes eindeutige Seriennummer des ATSHA204A. Hierbei handelt es sich um eine eindeutige, vom Hersteller bei der Chipfabrikation eingebrachte, Seriennummer, die den ATSHA204A eindeutig identifiziert. Die Seriennummer ist nicht notwendigerweise geheim, wird also in der Sicherheitsbetrachtung als öffentlich angesehen und muss ebenfalls auf Seite des Servers für die Schlüsselableitung bekannt sein.

Zwischen Server und Client muss also bei initialem Ausrollen der Hardware-Module das Tripel (K, O, N) geteilt sein. Dieses Tripel bildet das Geheimnis, von dem jeweils die PSK-Sitzungsschlüssel mittels der MAC-Funktion abgeleitet werden. Das Tupel (S, C) bildet die MAC-Challenge, welche in den PSK Identity Nachrichten während des TLS Handshakes ausgetauscht wird. Hierbei ist die Reihenfolge der Konkatenation derart gewählt, dass die unvertraute Server-Nonce S vor der Client-Nonce C in den Datenstrom eingebracht wird. In M werden MAC- und modulspezifische Informationen darüber kodiert, in welchem Modus das Modul betrieben wird. M kann als öffentlich und konstant angenommen werden.

4.1 Szenarien und Annahmen

Wir unterscheiden im Folgenden zwei unterschiedliche Angriffsszenarien:

1. Ein Angreifer erlangt durch reines Mitlesen oder gezielte Manipulation von S (*probing*) Kenntnis über das geheime Schlüsseltripel (K, O, N).
2. Es gelingt einem Angreifer durch geschickte Manipulation von S einen MAC-Wert m zu erzwingen, den er kennt.

Diese Angriffe hätten folgende Konsequenzen:

1. Angriff 1: Ein Angreifer, der Kenntnis über (K, O, N) erlangt, kann alle Verbindungen entschlüsseln, die in der Vergangenheit stattgefunden haben. Für alle Verbindungen, die in der Zukunft stattfinden werden, kann dieser Angreifer passiv alle Daten entschlüsseln und sie aktiv als Man-in-the-Middle beliebig modifizieren.
2. Angriff 2: Ein Angreifer, der durch Modifikation von S während des Verbindungsaufbaus einen Wert m forcieren kann, muss zunächst aktiv in Verbindungen eingreifen und kann dann während der laufenden Verbindung entweder passiv mithören oder aktiv Daten verändern.

Hierfür treffen wir folgende Annahmen:

1. Aus den im TLS-Handshake auf die PSK-Schlüsselauswahl folgenden Nachrichten ist kein Rückschluss auf den verwendeten PSK möglich. Wir betrachten die Sicherheit von TLS selbst als gegeben.
2. Der vom Client verwendete Zufallszahlengenerator (RNG) liefert Zufallszahlen mit hoher Entropie. Die statistische Verteilung der vom Client gewählten Nonces ist also ideal.
3. Die verwendete Hashfunktion SHA-256 verhält sich in Ihrer Streu- und Lawineneigenschaft ideal. Es gibt keinen algorithmischen Angriff, der die SHA-256-Funktion

grundsätzlich kompromittiert.

Die getroffenen Annahmen sind aus folgenden Gründen sinnvoll:

Annahme 1 ist die Grundvoraussetzung dafür, dass TLS-PSK überhaupt als sicheres Verfahren angesehen werden kann: Wie in Abs. 4 beschrieben kommt in der Regel als Schlüsselauswahlfunktion die Identitätsfunktion zum Einsatz. Wäre es möglich, aus den ausgetauschten Nachrichten also Rückschlüsse auf den verwendeten PSK zu machen, so könnte ein Angreifer durch Mitlesen der Pakete einer TLS-Verbindung den PSK lernen und damit die Sicherheitsziele von TLS brechen.

Annahme 2 ist plausibel, weil bei Verwendung eines Hardwaresicherheitsmoduls ATSHA204A auf Clientseite gleichzeitig ein echter Zufallszahlengenerator (TRNG) in dem HSM vorhanden ist. Dieser kann und sollte also genutzt werden und stellt dem System somit ausreichend Entropie zur Verfügung.

Annahme 3 geht davon aus, dass sich SHA-256 verhält wie eine ideale Hashfunktion mit 256 Bit Hashlänge. Gilbert et. al. widmen sich im Detail der SHA-2 Familie von Hashalgorithmen und kommen zu dem Schluss, dass alle in der Literatur beschriebenen populären Angriffe auf die SHA-2 Familie nicht anwendbar sind. [GiHa04] Sowohl Aoki et al. als auch Khovratovich et al. beschreiben Angriffe gegen SHA-256. [AGMS$^+$09, KhRS12] In beiden Fällen handelt es sich allerdings um Runden-reduzierte Varianten von SHA-256. Gegen den SHA-256 Algorithmus mit voller Rundenanzahl gibt es bisher keine Cryptanalysis, die eine Angriffskomplexität hat, welche unter der eines Brute-Force-Angriffs liegt.

4.2 Theoretische Betrachtung

Bei dem von der Hardware verwendeten Verfahren handelt es sich um die von Tsudik vorgeschlagene *envelope* Konstruktion. Dabei wird ein MAC m über eine Nachricht M berechnet, indem M mit Schlüsseln auf der linken und rechten Seite konkateniert wird und danach durch eine kryptografische Hashfunktion läuft: [Tsud92, MeVOV96]

$$m = \mathrm{H}(K_1||M||K_2)$$

Für den konkreten Fall unseres Hardware-Sicherheitsmoduls gilt hier also:

$$K_1 = K, \quad M = S||C, \quad K_2 = M||O||N$$

Envelope-Konstruktionen sind in der Literatur aufgrund ihrer Verbreitung bei beispielsweise IPSec [MeSi95] ausführlich betrachtet worden. [PrVa95, PrVa96] Preneel und Oorschot stellen bei ihrer detaillierten Betrachtung fest, dass eine Envelope-Konstruktion grundsätzlich anfällig auf einen Teile-und-Herrsche-Angriff ist, falls es möglich ist, die Länge der Nachricht durch Anhängen eines Suffixes zu verlängern. Es ist aber eine unabänderliche Hardware-Beschränkung des ATSHA204A, dass die MAC-Funktion einzig das in Abs. 4 dargestellte Format von exakt 88 Bytes Länge als Eingabe für den SHA-256-Algorithmus zulässt; insofern sind diese Angriffe in dem von uns gezeigten Szenario nicht anwendbar.

In der von uns gezeigten Konstruktion wird der aus der MAC-Funktion resultierende Wert, wie beschrieben, nur als PSK für TLS verwendet. Es folgt also direkt aus Annahme 1, dass

das Angriffsszenario 1 ausscheidet: Wenn es einem Angreifer nämlich nicht möglich ist, aus den ausgetauschten Nachrichten den PSK zu errechnen und der PSK die Ausgabe der MAC-Funktion ist, so kann der Angreifer demnach aus dem Datenverkehr m nicht rekonstruieren. Hat der Angreifer m nicht, kann er schlussendlich auch keine Rückschlüsse auf das Tripel (K, O, N) ziehen, die zur Berechnung von m führten.

Ein Angreifer kann jedoch S frei wählen; dies wird als Teil des Handshakes nämlich noch im Klartext und ohne Authentifizierung übertragen. Allerdings ist auch das Angriffsszenario 2 ausgeschlossen: Durch Annahme 2 zusammen mit 3 folgt direkt, dass sich der resultierende PSK m in Abhängigkeit von dem clientseitig eingebrachten echten Zufall jeweils lawinenartig ändert. Dies bedeutet, dass ein Angreifer kein gezieltes Datum S a priori wählen kann, da m maßgeblich vom zeitlich erst darauf folgend generierten C abhängt.

4.3 Praktische Betrachtung

Um die Sicherheit der von der verwendeten Hardware benutzten kryptografischen Konstruktion abzuschätzen werden wir nun eine Worst-Case-Betrachtung durchführen. Viele der hierbei getroffenen Annahmen sind um viele Größenordnungen konservativer als dies tatsächlich der Fall ist. Diese von uns getroffenen Annahmen sind konkret:

- Ein Angreifer hat direkten Zugang zu dem Ergebnis der MAC-Funktion m (also den PSK). Er kennt die Seriennummer des ATSHA204A und kann die komplette Challenge (S, C) frei wählen.
- Ein Teile-und-Herrsche-Angriff auf die kryptografische Konstruktion ist möglich (dies ist, wie in Abs. 4.2 gezeigt, in der Praxis nicht der Fall).
- Die Komplexität des Teile-und-Herrsche-Angriffs hängt lediglich von dem 11 Bytes langen OTP-Suffix ab; die Komplexität beträgt $2^{\frac{k}{2}}$ Nachrichten, also konkret 2^{44}.
- Es tritt keinerlei Kommunikations-Overhead auf, der ATSHA204 berechnet alle Ergebnisse in der laut Datenblatt typischen Ausführungszeit (12 ms pro MAC-Operation).

Wenn ein Angreifer nun versucht, den modulinternen Schlüssel K zu extrahieren, müsste er über 2^{44} Nachrichten mit jeweils gültigem MAC verfügen. Dann wäre ein Key-Recovery-Angriff also mit hinreichender Wahrscheinlichkeit erfolgreich. Allein die Berechnung dieser Menge an gültigen MACs mit Hilfe eines kompromittierten HSMs würde aber unter den gegebenen Bedingungen:

$$2^{44} \cdot 12\text{ms} \approx 2.11 \cdot 10^{11}\text{sec} \approx 6690\text{yr}$$

dauern. Der Flaschenhals ist also eine inhärente Hardwarebeschränkung des Moduls selbst, das grundsätzlich die MAC-Berechnung nicht schneller durchführen *kann*. In der Praxis wird die Anzahl der notwendigen Nachrichten-MAC-Paare noch um Größenordnungen höher liegen. Um die Sicherheit von eingebetteten Systemen zu gewährleisten, die so ressourcenbeschränkt sind, dass sie auf symmetrische Kryptografie angewiesen sind, wird dies also in jedem Fall ausreichend sein.

5 Zusammenfassung

Mit unserer Arbeit zeigen wir, dass eine Verwendung von Hardware-Sicherheitsmodulen auf Kleinstcomputern nicht nur möglich, sondern auch praktikabel ist. Unser Ansatz benötigt kei-

ne Veränderung an der internen Protokollstruktur von DTLS. Dadurch können auf einfache Art und Weise bestehende TLS-Implementierungen verwendet werden. Sämtlicher Code, der zur Interoperabilität notwendig ist, kann vom Benutzer in der Schlüssel-Zuordnungsfunktion implementiert werden. Die Umsetzung wird somit unabhängig von einem speziellen Hardware-Modul und dessen proprietärer Schnittstelle. Stattdessen kann es durch ein generisches Verfahren an eine Vielfalt unterschiedlicher Module angebunden werden. Da ein Einsatz solcher Module aufgrund ihres geringen Anschaffungspreises für kleinste Knoten im Internet of Things relevant ist, leistet unsere Arbeit also einen konkreten Beitrag in Richtung der Härtung dieser Endknoten gegen physische Angriffe.

Literatur

[AGMS+09] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, L. Wang: Preimages for step-reduced SHA-2. *In: Advances in Cryptology–ASIACRYPT 2009*, Springer (2009), 578–597.

[BKMK12] M. Brachmann, S. L. Keoh, O. G. Morchon, S. S. Kumar: End-to-end transport security in the IP-Based Internet of Things. *In: Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, IEEE (2012), 1–5.

[BPRR+12] I. Bagci, M. Pourmirza, S. Raza, U. Roedig, T. Voigt: Codo: Confidential data storage for wireless sensor networks. *In: Mobile Adhoc and Sensor Systems (MASS), 2012 IEEE 9th International Conference on*, IEEE (2012), 1–6.

[BSPS+11] S. Babar, A. Stango, N. Prasad, J. Sen, R. Prasad: Proposed embedded security framework for Internet of Things (IoT). *In: Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, IEEE (2011), 1–5.

[DiRe08] T. Dierks, E. Rescorla: The Transport Layer Security (TLS) Protocol Version 1.2 (RFC5246) (2008), .

[FaHK13] Z. Fan, R. J. Haines, P. Kulkarni: M2M Communications for E-Health and Smart Grid: An Industry and Standard Perspective. In: *CoRR*, abs/1306.3323 (2013), .

[GiHa04] H. Gilbert, H. Handschuh: Security analysis of SHA-256 and sisters. *In: Selected Areas in Cryptography*, Springer (2004), 175–193.

[KhRS12] D. Khovratovich, C. Rechberger, A. Savelieva: Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. *In: Fast Software Encryption*, Springer (2012), 244–263.

[KrBC97] H. Krawczyk, M. Bellare, R. Canetti: HMAC: Keyed-Hashing for Message Authentication (RFC2104) (1997), .

[Mang04] S. Mangard: Hardware countermeasures against DPA–a statistical analysis of their effectiveness. *In: Topics in Cryptology–CT-RSA 2004*, Springer (2004), 222–235.

[MeSi95] P. Metzger, W. Simpson: IP Authentication using Keyed MD5 (RFC1828) (1995), .

[MeVOV96] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone: Handbook of applied cryptography. CRC press (1996).

[PrVa95] B. Preneel, P. C. Van Oorschot: MDx-MAC and building fast MACs from hash functions. *In: Advances in Cryptology—CRYPT0'95*, Springer (1995), 1–14.

[PrVa96] B. Preneel, P. C. Van Oorschot: On the security of two MAC algorithms. *In: Advances in Cryptology—EUROCRYPT'96*, Springer (1996), 19–32.

[Skor05] S. P. Skorobogatov: Semi-invasive attacks: a new approach to hardware security analysis. Dissertation (2005).

[StTe02] N. I. of Standards, Technology: FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2. Tech. Rep., Department of Commerce (2002), .

[Tsud92] G. Tsudik: Message authentication with one-way hash functions. In: *ACM SIGCOMM Computer Communication Review*, 22, 5 (1992), 29–38.

[WhHF03] D. Whiting, R. Housley, N. Ferguson: Counter with CBC-MAC (CCM) (RFC3610) (2003), .