Sicherheitsaspekte und Vergleich der Blockchiffren LED und TEA

Michael Appel[1], Christof Pauer[1] und Alexander Wiesmaier[1,2,3]

[1] TU Darmstadt
[2] AGT International
[3] Hochschule Darmstadt

Zusammenfassung. In dieser Arbeit werden verschiedene Verschlüsselungsalgorithmen, die im Internet of Things (IoT) speziell auf mobilen und embedded Systemen Verwendung finden untersucht. Hierfür werden die Blockverschlüsselungsverfahren TEA und LED vorgestellt und hinsichtlich der Sicherheit untersucht. Anschließend werden die Verschlüsselungsverfahren gegenübergestellt und die Vor- und Nachteile der einzelnen Verfahren herausgearbeitet. Hierbei wird der Schwerpunkt auf Performance, sowie auf die Sicherheit der Algorithmen gelegt.

Schlüsselwörter: Intenret der Dinge (IoT); leichtgewichtige Blockchiffren; LED; TEA

1 Einleitung

1.1 Allgemeines

Durch den technischen Fortschritt hält das Internet immer stärkeren Einzug in unser alltägliches Leben. Die immer kleiner und günstiger werdenden elektronischen Steuerungs- und Kommunikationskomponenten wurden, insbesondere in den letzten Jahren, immer stärker in "Dinge"des alltäglichen Lebens eingebaut. Dieser Trend wird mit dem Überbegriff Ïnternet of Things"(IoT) bezeichnet. Typische Anwendungsfelder sind z.B. Hausautomatisierung, Sicherheitstechnik im Privat oder Geschäftsumfeld, sowie der unterstützende Einsatz in der Industrie [30]. Durch die sehr hohe Preissensitivität in dem Umfeld wird ein großer Fokus auf die Effizienz der verwendeten Programme und Algorithmen gelegt. Benötigt ein optimierter Algorithmus z.B. nur die Hälfte an Rechenzeit und Speicher, ist es möglich entsprechend günstigere Hardware einzusetzen. Hochgerechnet auf die produzierte Stückzahl kann hierdurch ein beachtlicher Betrag eingespart werden bzw. die IoT Technik in entsprechend günstigere "Dingeëingebaut werden.
Durch die immer stärker werdende Integration der Technik in den alltäglichen Lebensablauf und damit zwangsläufig auch in den hochpersönlichen Lebensbereich, steigt auch der Anspruch an Vertraulichkeit und Sicherheit der erhobenen Daten und an die vernetzten Geräte. Vor diesem Hintergrund wurden besonders effiziente Verschlüsselungsalgorithmen entwickelt, die teilweise speziell an

die verwendete Hardware angepasst sind. In dieser Arbeit werden zwei Blockverschlüsselungsverfahren, die sich für den Einsatz auf schwächerer Hardware eignen vorgestellt, miteinander verglichen und anhand von vorher eingeführten Angriffsmethoden auf bekannte Schwachstellen hin beleuchtet. Weiter wird die Performance der Algorithmen dargelegt und mit einem Standardverfahren verglichen.

Der Rest dieses Papers ist wie folgt aufgebaut: Kapitel 2 gibt einen kurzen Überblick über weitere Ausarbeitungen die sich aus ähnlichen Gesichtspunkten mit den Themen dieses Papers beschäftigen. In Kapitel 3 werden einige Angriffsverfahren auf Blockverschlüsselungen vorgestellt und erläutert. In den Kapiteln 4 und 5 werden die Blockverschlüsselungsverfahren TEA und LED vorgestellt und deren Besonderheiten beleuchtet. Diese Verschlüsselungsverfahren wurden für die Analyse in diesem Paper gewählt, weil sie besonders leichtgewichtig und performant sind, aber auch in der Praxis eingesetzt werden und dort in geknackt wurden (TEA, siehe Punkt 5.6). LED zeichnet sich durch eine besondere Eignung für die Implementierung in kleinster Hardware aus und ist daher besonders im RFID Bereich interessant. Dabei werden mittels der vorher eingeführten Angriffsverfahren auch eventuelle Schwachstellen der Verfahren dargelegt. In Kapitel 6 werden die Blockverschlüsselungsverfahren aus Performance und Effizienz Gesichtspunkten miteinander, sowie mit AES verglichen. In Kapitel 7 wird ein kurzes Fazit gegeben. Zur besseren Einordnung sind in diesem Paper auch Vergleichswerte des Blockverschlüsselungsverfahrens KATAN enthalten.

1.2 Mathematische Notationen

In diesem Paper werden die folgenden mathematischen Operatoren verwendet:

Integer Addition: Die Addition zweier integer Zahlen modulo 2^n ist notiert als $x \boxplus y$. Wobei $x, y \in \mathbb{Z}_{2^n}$ Der Wert von n ergibt sich aus dem Kontext.

Exklusiv-oder (xor): $x \oplus y$

Bitweiser Shift: Die logische Verschiebung (shift) von x um y Bits ist notiert als $x << y$ (nach links). Die logische Verschiebung von x um y Bits nach Rechts ist notiert als $x >> y$.

2 Related Work

Aufgrund des Bekanntheitsgrades der Algorithmen TEA und LED existieren zahlreiche Paper die sich ebenfalls mit den Algorithmen aus verschiedenen Gesichtspunkten beschäftigen. Zuerst zu nennen sind dabei die Paper die sich im Allgemeinen mit den Verschlüsselungsalgorithmen beschäftigen. Sie stellen jeweils einen der Algorithmen vor und gehen jeweils auf bestimmte Eigenschaften der Algorithmen in verschiedenen Detailgraden ein. Für den Tiny Encryption Algorithm (TEA) lassen sich hierbei die folgenden Paper exemplarisch

anführen: "TEA, a Tiny Encryption Algorithm"[24], "Tiny Encryption Algorithm (TEA)"[2] sowie "The Tiny Encryption Algorithm (TEA)"[29]
Die zweite relevante Gruppe von sich mit der Thematik beschäftigenden Ausarbeitungen, bilden die Paper welche einen besonderen Fokus auf die Leichtgewichtigkeit und damit die Eignung der Algorithmen in besonders kleiner, rechenschwacher und billiger Hardware beschäftigt. Hier sind im speziellen die Paper "Design and Implementation of Low Power Hardware Encryption for Low Cost Secure RFID Using TEA"[14], sowie "Hardware Implementation of a TEA-Based Lightweight Encryption for RFID Security"[15] zu nennen.
Die letzte Gruppe bilden die Paper, welche die Algorithmen unter Sicherheitsgesichtspunkten betrachen und Schwachstellen sowie mögliche Angriffe beschreiben. Hierzu gehören u.a. die folgenden: "Related-key rectangle attack on 36 rounds of the XTEA block cipher"[18] sowie "Meet-in-the-Middle Attacks on Reduced-Round XTEA"[22].

3 Angriffe

In diesem Abschnitt werden verschiedene Angriffstypen auf Blockchiffren kurz beschrieben. Die Angriffe spielen für die Chiffren, die in diesem Paper behandelt werden jeweils eine mehr oder weniger große Rolle und werden in den Abschnitten bezüglich der Sicherheit der Chiffren wieder aufgegriffen.

Brute-force. Bei einer brute-force Attacke werden systematisch alle möglichen Schlüssel auf den Chiffretext angewandt. Dabei wird angenommen, dass der Angreifer keine Vorkenntnisse über Schlüssel hat, die wahrscheinlicher als andere Schlüssel sind. Im Mittel muss der Chiffretext mit der Hälfte aller möglichen Schlüssel entschlüsselt werden. Darüber hinaus muss der Klartext als solcher erkannt werden.
Lineare Kryptoanalyse. Dieser Angriff setzt einen known-plaintext Angreifer voraus, d.h. der Angreifer kennt zu bestimmten Klartexten den jeweiligen Chiffretext. Die Idee des Angriffs ist es, lineare Gleichungen für Teile bzw. einzelne Operationen der Chiffre zu finden. Über die Gleichungen wird versucht Klartext-, Chiffretext- und Schlüsselbits in ein Verhältnis zu setzen, um Schlüssel- bzw. Klartextbits mit einer größeren Wahrscheinlichkeit als $\frac{1}{2}$ zu bestimmen. Ein wichtiger Perfomancefaktor dieser Angriffe ist es, wieviele Klartext-Chiffretext Paare benötigt werden.
Algebraische Kryptoanalyse. Diese Art der Kryptoanalyse hat das gleiche Ziel wie die lineare Kryptoanalyse, allerdings beschränkt man sich nicht auf lineare Gleichungen sondern Polynomgleichungen beliebigen Grades. Eine besondere Eigenschaft im Zusammenhang mit algebraischer Kryptoanalyse ist, dass oft keine genauen Komplexitäten angegeben werden können und stattdessen auf andere Metriken zurückgegriffen werden muss, wie z.B. die Laufzeit auf einer bestimmten Testumgebung.
Differenzielle Kryptoanalyse. Die differenzielle Kryptoanalyse ist ein chosen-plaintext Angriff, d.h. der Angreifer kann von ihm gewählte Klartexte verschlüsseln. Um die Attacke durchzuführen, werden Paare $(\Delta X, \Delta Y)$ verglichen,

wobei ΔX und ΔY jeweils die Differenz (z.B. XOR) zweier Werte darstellen, z.B. die Differenz von jeweils zwei Ein- und Ausgaben des Algorithmus. Ziel dieser Analyse ist es bestimmte Schlüssel als wahrscheinlicher einzustufen.
Related-key Angriffe. Bei einem related-key Angriff wird angenommen, dass der Angreifer nicht nur den Chiffretext unter Verwendung des ursprünglichen Schlüssels K kennt, sondern auch die Verschlüsselung mit Schlüsseln K', die von K abgeleitet sind, d.h. $K' = f(K)$.
Meet-in-the-middle Angriffe (MITM). MITM Attacken setzen mindestens ein bekanntes Klartext-Chiffretext Paar voraus (known- bzw. chosen-plaintext). Der Angreifer versucht in der ersten Phase Schlüssel zu filtern, d.h. der Schlüsselraum wird eingegrenzt. Im zweiten Schritt wird mittels brute-force oder einer anderen Attacke der richtige Schlüssel gesucht. Nähere Details zu der Angriffstechnik finden sich z.B. in Takanori Isobe und Kyoji Shibutani[13].
Seitenkanalattacken. Eine Seitenkanalattacke greift nicht den Algorithmus als solchen an, sondern die physische Implementierung. Hierbei versucht der Angreifer z.B. von der Dauer oder dem Stromverbrauch bestimmter Operationen auf geheime Informationen zu schließen. Außerdem kann der Angreifer versuchen gezielt Bits zu kippen, z.B. durch Manipulation der anliegenden Spannung.

4 LED

Light Encryption Device (LED) ist eine symmetrische Blockchiffre, die von Guo et al. [11] in 2011 veröffentlicht wurde. Die Chiffre ist leichtgewichtig und kann effizient in Hardware implementiert werden. Aufgrund dieser Eigenschaften schlagen die Autoren vor, dass LED grundsätzlich zur Ver- bzw. Entschlüsslung im IoT Bereich geeignet ist. Als konkreter Anwendungsfall wird die sichere Speicherung bzw. Übertragung von RFID-Tags genannt.

4.1 Verschlüsselung

LED operiert auf einer Blockgröße von 64 Bit. Die Schlüssellänge beträgt 64 (LED-64) bzw. 128 Bit (LED-128). Grundsätzlich sind auch Schlüssellängen zwischen 64 und 128 Bit möglich, z.B. 80 Bits. In diesem Fall werden die restlichen Bits bis 128 mit dem Präfix des Schlüssels aufgefüllt (padding). Als `Zustand` bezeichnen wir die aktuellen Bytes des zu verschlüsselnden Blocks. Der Zustand und Schlüssel werden als (4×4) Matrix notiert. Die Matrixeinträge des Zustands und Schlüssels sind jeweils 4 Bit Blöcke (Nibble) und stellen Elemente des Körpers $\mathbb{F}_{2^4}$ dar. Bei einer Schlüssellänge von 128 Bit gibt es entsprechend 2 Matrizen mit jeweils 64 Bit. Eine Runde setzt sich aus 4 Teilschritten zusammen: `AddConstants`, `SubCells`, `ShiftRows` und `MixColumnsSerial`. Die Anzahl der Runden beträgt 32 (LED-64) bzw. 48 (LED-128). Im Folgenden wird jeder Teilschritt einer Runde beschrieben.

AddConstants. Zu Beginn jeder Runde wird die Operation `AddConstants` durchgeführt. Hierzu wird der Zustand mit der folgenden Matrix mittels XOR

bitweise addiert:

$$\begin{pmatrix} 0\ (rc_5||rc_4||rc_3)\ 0\ 0 \\ 1\ (rc_2||rc_1||rc_0)\ 0\ 0 \\ 2\ (rc_5||rc_4||rc_3)\ 0\ 0 \\ 3\ (rc_2||rc_1||rc_0)\ 0\ 0 \end{pmatrix}$$

Der Bitvektor $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ wird vor der ersten Runde mit 0 initialisiert. Vor der Addition wird jeweils die Matrix der letzten Runde übernommen, der Bitvektor wird um eine Position nach links verschoben und rc_0 wird auf den Wert $rc_5 \oplus rc_4 \oplus 1$ gesetzt.

SubCells. In diesem Schritt wird jeder Nibble des Zustands durch einen anderen ersetzt (Sbox). Die Sbox wurde von der Blockchiffre `PRESENT` [3] übernommen.

Tabelle 1: `PRESENT` Sbox.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

ShiftRows. Für $i = 1, 2, 3, 4$ wird jeweils die i-te Zeile zyklisch um $i - 1$ Positionen nach links verschoben.

MixColumnsSerial. Jeder Spaltenvektor v des Zustands wird mit $M \cdot v$ ersetzt. Um die Körperelemente zu multiplizieren wird das Polynom $X^4 + X + 1$ verwendet.

$$M = \begin{pmatrix} 4 & 2 & 1 & 1 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix}$$

Nach jedem Schritt (4 Runden) und zu Beginn der Verschlüsselung wird außerdem noch die Operation `addRoundKey` durchgeführt. Dazu wird der Schlüssel mittels XOR auf den Zustand addiert. Im Falle von LED-128 wechseln sich die beiden Schlüssel nach jedem Schritt ab. Die Bezeichnung „round key“ ist an dieser Stelle irreführend, da der Schlüssel immer gleich bleibt. Eine Übersicht des Verschlüsselungsprozesses ist in Fig. 1 dargestellt.

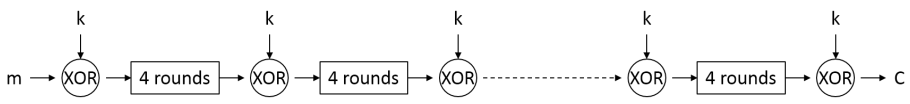

Abb. 1: LED Verschlüsselung

4.2 Sicherheit

LED ist bezüglich der Rundenoperationen sehr ähnlich zu AES. Die Rundenanzahl ist mit 32 (LED-64) bzw. 48 (LED-128) vergleichsweise hoch angesetzt. AES-128 nutzt 10 Runden. Allerdings muss darauf hingewiesen werden, dass AES nach jeder Runde den Rundenschlüssel addiert. Bei LED wird diese Operation durch AddConstants ersetzt. Somit entspricht ein Schritt (4 Runden) in LED, 4 Runden single-key AES. Dieser Vergleich wird gezogen, weil die Sicherheit für den Fall 4 Runden single-key AES bereits genauer erforscht wurde[21][6].

Im Folgenden wird eine Übersicht der Angriffe gegeben, die auf LED[4] existieren. Vincent Grosso et al.[10] untersuchen algebraische Attacken auf LED. Evaluiert wird die benötigte Zeit für die Schlüsselwiederherstellung bei unterschiedlicher Anzahl bekannter Schlüsselbits. Bei voller Rundenanzahl kann der Schlüssel in ca. 5 Minuten[5] wiederhergestellt werden, wenn 16 von 64 Schlüsselbits unbekannt sind. Ergebnisse für den Fall, dass mehr als 16 Schlüsselbits unbekannt sind werden nicht geliefert. Weiterhin wird festgestellt, dass der Angriff nur einen Vorteil gegenüber einer brute-force Attacke bringt, wenn mindestens 9 Schlüsselbits unbekannt sind.

In den Arbeiten von Xinjie Zhao et al.[32] und Philipp Jovanovic et al.[16] werden algebraische Angriffe mit einer Seitenkanalattacke kombiniert. Beide Angriffe setzen vorraus, dass zweimal der gleiche Klartext mit dem selben Schlüssel verschlüsselt werden kann. Bei der zweiten Verschlüsselung wird versucht in der 30. Runde nach der Operation SubCells ein Fehler im ersten Eintrag des Zustands zu produzieren, sodass an dieser Stelle ein zufälliger Wert steht. Auf Basis der Differenz der Ausgaben und den inversen Rundenoperationen wird ein Gleichungssystem gebildet, mit dem sich der Schlüsselraum auf $2^{19} \sim 2^{25}$ bzw. $2^6 \sim 2^{17}$ reduzieren lässt[16][32].

Takanori Isobe und Kyoji Shibutani[13] untersuchen MITM Attacken auf leichtgewichtige Chiffren. Es handelt sich dabei um einen chosen-plaintext Angriff mit dem Ziel den Schlüssel wiederherzustellen. Im Fall `LED-64` werden 8 Runden angegriffen, wobei 2^8 Klartext-Chiffretext Paare und 2^{56} Verschlüsselungen benötigt werden. Bezüglich `LED-128` werden 16 Runden bei einer Zeitkomplexität von 2^{112} und einer Datenkomplexität von 2^{16} erreicht.

Mendel et al.[19] beschreiben differenzielle Angriffe im single- und related-key Kontext. Ziel des Angriffes ist die Schlüsselwiederherstellung. Im related-key Angriff schaffen die Autoren es, LED in 16 Runden bei $2^{62.7}$ Verschlüsselungen anzugreifen. Allerdings hat der Angriff mit $2^{62.7}$ auch eine hohe Daten- und Speicherkomplexität. In dem Paper wird außerdem ein Angriff auf `LED-128` aufgezeigt, der unter der Voraussetzung, dass K_0 erraten wird (Datenkomplexität 2^{64}), den gesamten Schlüssel $K_0||K_1$ bei 2^{96} Verschlüsselungen wiederherstellen kann.

Dass in 2015 veröffentliche Paper von Ivica Nikolic et al.[21] greift mit einer Kombination aus MITM und differenzieller Kryptoanalyse die bisher meisten

[4] Die angegebenen Werte beziehen sich auf `LED-64`, wenn nicht anders angegeben.
[5] Eine Aussage über die verwendete Testumgebung wird nicht getroffen.

Runden (ohne Seitenkanal) mit 20 (`LED-64`) bzw. 40 (`LED-128`) an. Im Gegensatz zu den bisher genannten Angriffen, zielt dieser nicht auf die Schlüsselwiederherstellung ab. Stattdessen kann der Angreifer bei einem erfolgreichen Angriff eine zufällige Permutation von den 2^k möglichen Permutationen des verwendeten Schlüssels unterscheiden (distinguishing attack). Dieser Angriff ist zwar weniger bedrohlich als eine Schlüsselwiederherstellung, allerdings können auf dieser Grundlage mit geringerer Komplexität weiterführende Angriffe durchgeführt werden. Die Ergebnisse sehen wie folgt aus. Im Fall `LED-64` werden 20 Runden bei einer Zeitkomplexität von $2^{60.2}$ und einer Datenkomplexität von $2^{61.5}$ angegriffen. Bei `LED-128` sind 10 Runden mit $2^{60.3}$ Verschlüsselungen und einer Speicherkomplexität von 2^{60} betroffen.

Eine Zusammenfassung der Angriffe ist in Tabelle 2[21] dargestellt. Neben den Komplexitäten, werden die Angriffsvoraussetzungen (single-key (SK), related-key (RK), chosen-key (CK)), die Anzahl der attackierten Runden und die Auswirkung des Angriffs (key-recovery (KR), distinguisher (D)) gelistet.

Tabelle 2: Angriffe auf LED[21]

Chiffre	Attacke	Typ	Runden	Zeit	Daten	Speicher	Ref
`LED-64` (32 Runden)	MITM (SK)	KR	8	2^{56}	2^{8}	2^{11}	[13]
	Linear/Differenziell (CK/RK)	KR	16	$2^{62.7}$	$2^{62.7}$	$2^{62.7}$	[19]
	Linear/Differenziell (CK)	D	15	2^{16}	–	2^{16}	[11]
	MITM/Differenziell (CK)	D	16	$2^{33.5}$	–	2^{32}	[21]
	MITM/Differenziell (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[21]
`LED-128` (48 Runden)	MITM (SK)	KR	16	2^{112}	2^{16}	2^{19}	[13]
	Linear/Differenziell(CK/RK)	KR	24	2^{96}	2^{64}	2^{32}	[19]
	Linear/Differenziell(CK/RK)	D	27	2^{16}	2^{16}	2^{32}	[11]
	MITM/Differenziell(CK)	D	32	$2^{33.5}$	–	2^{32}	[21]
	MITM/Differenziell(CK)	D	40	$2^{60.3}$	–	2^{60}	[21]

5 Tea Familie

5.1 Einführung TEA

Der Tiny Encryption Algorithm (TEA) wurde mit dem Ziel entwickelt einen performanten und mathematisch nicht zu komplizierten Verschlüsselungsalgorithmus zu entwerfen, der sich im Besonderen auch für die Verwendung auf leistungsschwächeren Kleinstrechnern im Internet of Things (IoT) Umfeld eignet. Die Algorithmen der TEA-Familie sind Varianten einer Feistelchiffre und damit Blockchiffren. TEA verschlüsselt 64 Bit große Blöcke, die direkt weiter in zwei 32 Bit Blöcke gesplittet werden. Der von dem klassischen TEA Algorithmus verwendete Schlüssel hat eine Länge von 128 Bit. Tea ist ein rundenbasiertes Verschlüsselungsverfahren. Die Anzahl der verwendeten Runden ist variabel, es

werden jedoch 32 Tea Zyklen empfohlen. Bedingt durch den symmetrischen Aufbau des Verschlüsselungsalgorithmus (siehe Punkt 5.3) entspricht ein Zyklus in TEA zwei Feistelrunden. [2]

Der Algorithmus übertrifft die Performance von DES (siehe Punkt 5.7) und kann in alle Programmiersprachen implementiert werde. Für viele gängige Programmiersprachen existieren bereits Referenzimplementierungen die mit wenig Aufwand verwendet werden können. Mit einer Stärke von 32 Zyklen ist die getestete Implementierung 60% schneller als die Referenzimplementierung mit 56-Bit DES. Die Verschlüsselungsstärke von TEA kann durch die Erhöhung der Verschlüsselungszyklen weiter gesteigert werden. [24]

5.2 Die goldene Zahl

Um Angriffen entgegenzuwirken die versuchen die Symmetrie der Verschlüsselungsrunden auszunutzen, ist es bei einigen Verschlüsselungsverfahren üblich in jeder Runde ein unterschiedliches Vielfaches der so genannten goldenen Zahl mit einzurechnen. Dies bewirkt, das es keine Bits gibt, die sich in mehreren aufeinander folgenden Verschlüsselungsrunden nicht ändern. Die klassische goldene Zahl ist definiert als: [25]

$$\frac{1+\sqrt{5}}{2} \tag{1}$$

In TEA wird zu diesem Zweck eine von der goldenen Zahl abgeleitete Konstante verwendet: [24]:

$$(\sqrt{5}-1)2^{31} \tag{2}$$

Diese Konstante initialisiert die Variable delta und entspricht gerundet auf eine ganze Zahl: [24]:

$$delta = 2654435769_{10} = 9E3779B9_{16} \tag{3}$$

Die mathematische Definition der Konstanten soll dem Verdacht entgegenwirken, dass es sich nicht um eine zufällig gewählte Zahl handelt, sondern um eine bewusste Schwächung des Algorithmus zum Einbau eines Backdoors. Aus diesem Grund ist es in der Kryptographie an diese Stelle üblich keine willkürlich gewählten Zufallszahlen hart zu codieren, sondern diese durch einfache mathematische Operationen nachvollziehbar zu generieren.[25]

5.3 Verschlüsselungsalgorithmus

Wie bereits unter 5.1 beschrieben, handelt es sich bei Tea um ein Blockverschlüsselungsverfahren, mit dem ausschließlich Blöcke von 64 Bit verschlüsselt werden können. Zu Verschlüsselung wird der 64 Bit große Block in zwei 32 Bit Blöcke geteilt. Ein Block wird mit L (links) der andere mit R (rechts) bezeichnet. Die Blöcke werden nach jeder Verschlüsselungsrunde vertauscht. Der 128 Bit lange Schlüssel wird in vier Unterschlüssel aufgeteilt und mit K[0-3] bezeichnet. Die ersten 32 Bit entsprechen dabei dem Schlüssel K[0], die zweiten 32 Bit

dem Schlüssel K[1] usw. Die Verschlüsselungsschritte sind in Abbildung 2 dargestellt und werden unter Punkt 5.5 in formaler weiße eingeführt. Weitere Details können in der angegebenen Quelle nachgelesen werden. [2]

5.4 Charakteristika der TEA Verschlüsselungsroutine

Der TEA Verschlüsselungsalgorithmus besitzt die folgenden Charakteristika:

- Wie üblich bei einer Festeilchiffre hat jede Runde i zwei Eingaben, Left[i] und Right[i], ab der zweiten Runde ist es jeweils die Ausgabe der gegenüberliegenden Seite der Vorrunde. Initialisiert werden die Variablen jeweils mit einer Hälfte des zu verschlüsselnden Blocks.
- Der in jeder Runde verwendete Schlüssel K[i] ist ein Teil des 128 Bit langen Schlüssels K.
- Die Konstante delta wird mit einer von der s.g. „goldenen Zahl“ abgeleiteten Konstante initialisiert und stellt sicher, dass die Teilschlüssel unterschiedliche Chiffren erzeugen und keine kryptographisch relevante Signifikanz aufweisen (siehe Punkt 5.2).
- Im Verschlüsselungsalgorithmus werden keine Zufallszahlen verwendet, d.h. identischer Text bei gleichen Schlüssel führt zur identischen Chiffre

5.5 Formale Definition der Verschlüsselung

Zur Verdeutlichung der Funktionsweise sind nachfolgend die Verschlüsselungsfunktionen der TEA Blockchiffre aufgelistet. In Abbildung 2 sind die Verschlüsselungschritte grafisch dargestellt. Wie unter 5.3 eingeführt, wird mit K[0-3] der jeweilige Teil des verwendeten Schlüssels bezeichnet und mit Right[i] bzw. Left[i] der jeweilige Teil des aktuell zu verschlüsselnden Blocks. Delta[i] ist eine Abwandlung der unter 5.2 eingeführte goldene Zahl. Wie dort beschrieben ist delta wie folgt definiert:

$$delta = (\sqrt{5} - 1)2^{31} = 2654435769_{10} = 9E3779B9_{16} \tag{4}$$

Eine TEA Runde besteht aus zwei Feistelrunden, aus diesem Grund erhöht sich in jeder TEA Runde die Laufvariable i um den Wert 2. Die Variable i bezeichnet dabei die Feistelrunden. In einer TEA Runde werden die folgenden Operationen ausgeführt:

$$delta[i] = (i + 1)/2 * delta \tag{5}$$

$$Left[i + 1] = Right[i] \tag{6}$$

$$Right[i + 1] = Left[i] \boxplus F(Right[i], K[0, 1], delta[i]) \tag{7}$$

$$Left[i + 2] = Right[i + 1] \tag{8}$$

$$Right[i+2] = Left[i+1] \boxplus F(Right[i+1], K[2,3], delta[i]) \tag{9}$$

$$F(M, K[j,k], delta[i]) = ((M << 4) \boxplus K[j]) \oplus (M \boxplus delta[i]) \oplus ((M >> 5) \boxplus K[k]) \tag{10}$$

Gleichung 6 und 7 bilden den ersten Schritt einer TEA Runde (die erste Feistelrunde). Die Gleichungen 8 sowie 9 schließen die TEA Runde dann ab (die zweite Feistelrunde). F() bezeichnet die so genannte Rundenfunktion (Gleichung 10), welche die wesentlichen Schritte der kryptographischen Operation enthält. Die Funktion ist immer identisch, wird jedoch jede Feistelrunde mit verschiedenen Parametern aufgerufen.

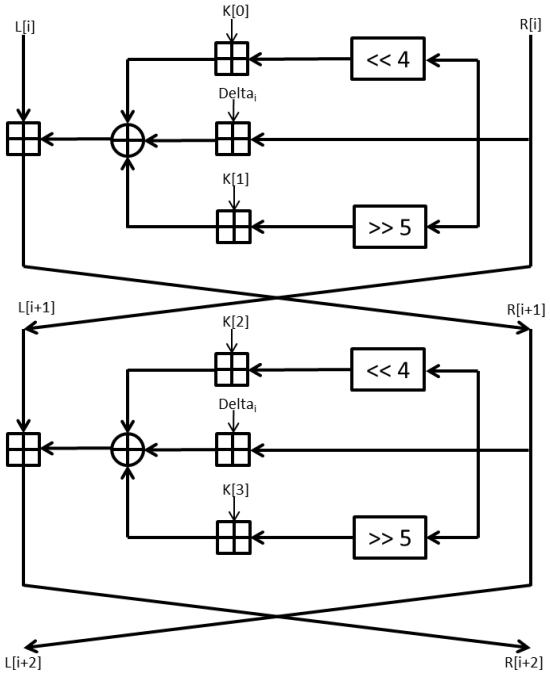

Abb. 2: Verschlüsselungsschritte TEA, Quelle: Frei nach [27]

5.6 Schwachstellen von Tea

TEA wurde bereits in einigen Kryptoanalysen behandelt und auf Schwachstellen untersucht. Die aktuell bekannten Schwachstellen der ursprünglichen TEA Implementierung sind:

- Hash Kollisionen, bei der Verwendung als Hash Funktion: TEA wurde nicht entwickelt um als Hash Funktion verwendet zu werden und erfüllt nicht die für kryptographische Hashfunktionen zentrale Bedingung der preimage resistance. Dies bedeutet es ist mit vergleichsweise wenig Aufwand möglich, zu einem gegebenen Hashwert Y, einen Eingabewert X zu finden, dessen Hashwert nach Anwendung der Hashfunktion ebenfalls auf Y Abbildet. Somit lassen sich gezielt Kollisionen zu einem gegebenen Hashwert errechnen. [29] [26]
- Key cracking: Anfälligkeit für das rohe Durchprobieren aller möglichen Schlüssel (Brute force). In Verbindung mit einem known plaintext-ciphertext Paar lassen sich die notwendigen Iterationen stark reduzieren und damit die Effizienz des Angriffs deutlich steigern.[29]
- Identische Schlüssel (Equivalent Keys): Aufgrund einer konstruktionellen Schwachstelle im TEA Verschlüsselungsalgorithmus, hat jeder Schlüssel zum entschlüsseln einer Chiffre, weitere drei äquivalente Schlüssel mit denen sich die Chiffre ebenfalls entschlüsseln lässt. Dies bedeutet der effektive Schlüsselraum eines 128 Bit langen Schlüssels wird auf 126 Bit reduziert. [29]

Nachfolgend wird die equivalent Key Schwachstelle in dem TEA Algorithmus exemplarisch in Anlehnung an [1] detaillierter beschrieben und die mathematischen Hintergründe hergeleitet. Erzeugen in einem Verschlüsselungssystem zwei verschiedene Schlüssel (K und K') mit dem identischen Klartext die identische Chiffre werden die Schlüssel K und K' als Äquivalent bezeichnet. Dies lässt sich durch folgende Gleichung ausdrücken:

$$E_K(T) = E_{K'}(T) \tag{11}$$

Im Umkehrschluss bedeutet dies, dass sich eine mit dem Schlüssel K verschlüsselte Chiffre auch mit dem Schlüssel K' dechiffrieren lässt:

$$D_{K'}(E_K(T)) = T \tag{12}$$

Bei einem guten Verschlüsselungssystem ist der Anspruch, dass es keine äquivalenten Schlüssel gibt. Die Anzahl der Äquivalenzklassen der Chiffren beträgt in diesem Fall 2^k wobei k die Schlüssellänge in Bit ist. Im Fall von TEA sollte es bei einem 128 Bit langen Schlüssel demzufolge $2^{128} \approx 3,4 * 10^{38}$ verschiedene Äquivalenzklassen geben. Untersuchungen haben jedoch ergeben, dass es in TEA nur 2^{126} verschiedene Äquivalenzklassen bei einer Schlüssellänge von 128 Bit gibt. Dies bedeutet, es gibt in diesem Fall nur 2^{126} sich von einander unterschiedenen Chiffren von einem gegebenen Text T. Damit gibt es für jede Chiffre vier zueinander äquivalente Schlüssel $K_0 ... K_3$ mit denen sich jeweils der Klartext T wieder dechiffrieren lässt. Veranschaulichen lässt sich dies an dem folgenden Beispiel:

$$\forall a, b \in \mathbb{Z}_{2^{32}} \tag{13}$$

$$2^{31} \boxplus 2^{31} = 0 \tag{14}$$

$$a \boxplus 2^{31} = a \boxplus 80000000_h \tag{15}$$

Dies bedeutet:

$$a \boxplus b = (a \oplus 80000000_h) \boxplus (b \oplus 80000000_h) \tag{16}$$

Auf diese Art lässt sich auf die Rundenfunktion von TEA manipulieren:

$$F(M, K[j,k], delta[i]) = F(M, (K[j] \oplus 80000000_h, K[k] \oplus 80000000_h, delta[i])) \tag{17}$$

Dies bedeutet, das jeder 128 Bit Schlüssel $K_0 ... K_3$ drei äquivalente Schlüssel in der Form von

$$(K[0], K[1], K[2] \oplus 80000000_h, K[3] \oplus 80000000_h \tag{18}$$

$$(K[0] \oplus 80000000_h, K[1] \oplus 80000000_h, K[2], K[3]) \tag{19}$$

$$(K[0] \oplus 80000000_h, K[1] \oplus 80000000_h, K[2] \oplus 80000000_h, K[3] \oplus 80000000_h) \tag{20}$$

besitzt.
Damit bietet ein 128 Bit langer Schlüssel mit der TEA Verschlüsselung nur einen Schlüsselraum von 126 Bit und damit die Sicherheit wie sie von einem 126 Bit langen Schlüssel zu erwarten wäre. [1]

5.7 Performance

Wie unter Punkt 5.1 beschrieben, war es ein zentrales Ziel bei der Entwicklung von TEA eine hohe Performanz zu erreichen. In einem exemplarischen Test in einer Java Umgebung war TEA 18 mal schneller als die von Java bereitgestellte DES Implementierung. Weitere Tests haben ergeben, dass TEA (128 Bit, 32 Iterationen) 60% schneller ist als 56-Bit DES und 4 mal schneller ist als 168-Bit 3DES. Da die nachfolgend beschriebenen Block Versionen Block TEA und XX-TEA von den Entwicklern explizit für größere Datenblöcke empfohlen werden, ist hierbei mit einer nochmaligen Performancesteigerung zu rechnen. [29]

5.8 Weiterentwicklung XTEA (eXtended TEA)

Der XTEA (eXtended TEA) Algorithmus ist eine Weiterentwicklung von TEA und korrigiert u.a. die unter Punkt 5.6 beschriebene Schwachstelle der äquivalenten Schlüssel. Wie TEA arbeitet XTEA mit 64-bit Blöcken und einem 128-bit langen Schlüssel. Empfohlen werden auch für XTEA 64 Verschlüsselungsrunden.

[18] Die Verbesserungen gegenüber TEA wurden durch eine komplexere Schlüsselverwaltung, sowie eine Veränderung der Shift, XOR und Additionsoperatoren erreicht.[28] Nach dem aktuellen Stand der Forschung kann jedoch auch XTEA nicht als ein Verschlüsselungsverfahren ohne Schwachstellen angesehen werden. Es existieren Beschreibungen von erfolgreichen Angriffen gegen XTEA durch Ausnutzung einer Related-key Schwachstelle[18]. Hierbei wurde der XTEA Algorithmus durch eine teilweise deutliche Reduzierung der Verschlüsselungszyklen jedoch bewusst geschwächt. Es ist jedoch davon auszugehen, dass bei entsprechend größerem Aufwand die Angriffe auch bei den empfohlenen 64 Verschlüsselungsrunden anwendbar ist. Diese Angriffe sind unter anderem in [17], [18] und [22] beschrieben.

5.9 Modifikation Block TEA

Block TEA wurde gleichzeitig mit XTEA veröffentlicht und unterscheidet sich technisch nur gering von XTEA. Im Gegensatz zu XTEA benötigt Block TEA keine feste Blockgröße, sondern kann mit Blöcken beliebiger Größe arbeiten. Dies bedeutet, dass Block TEA keinen Operationsmodus (mode of operation) zur Gewährleistung von Vertraulichkeit und Authentizität benötigt. Block TEA wird direkt auf die gesamte Nachricht angewandt. Intern wird dabei die Rundenfunktion (vgl. 5.3) iterativ und zyklisch auf die Gesamte Nachricht angewandt. Die verwendete Rundenfunktion ist identisch zu XTEA. Damit ist Block TEA auch von den selben Schwachstellen wie XTEA betroffen. [22] [28]

5.10 Weiterentwicklung Corrected Block TEA (auch bezeichnet als XXTEA)

Corrected Block TEA bzw. XXTEA ist eine 1998 veröffentlichte Weiterentwicklung von Block TEA. Wie Block TEA besitzt es keine feste Blockgröße und kann auf die gesamte Nachricht angewandt werden. Ziel der Entwicklung von XXTEA war es die bekannten Schwachstellen von Block TEA zu korrigieren. Hierfür wurden einige Veränderungen in der Rundenfunktion vorgenommen. Die Referenzimplementierung von XXTEA Correction to xtea ist verfügbar unter [23]. Auch für XXTEA existieren bereits dokumentierte erfolgreiche Angriffe auf den Algorithmus. Das Paper Cryptanalysis of XXTEA [31] beschreibt einen erfolgreichen chosen plaintext Angriff mit 2^{59} plain- chiphertext Paaren.

6 Vergleich der Algorithmen

Um Software- und Hardwareimplementierungen der Chiffren TEA und LED zu vergleichen, wird in diesem Teil zusätzlich das Kryptosystem KATAN[4] (bzw. KTANTAN) aufgeführt. KATAN unterscheidet sich hinsichtlich der funktionsweise stark von LED, jedoch sind für Softwareimplementierungen Werte in der

gleichen Größenordnung zu erwarten. Da uns für LED keine Softwareimplementierungen bekannt sind, die man mit einer entsprechenden AES bzw. TEA Implementierung vergleichen könnte, wird an dieser Stelle KATAN verwendet. Entsprechend haben wir keine repräsentative Hardwareimplementierung von TEA gefunden und verwenden stattdessen auch hier die Chiffre KATAN (bzw. KTANTAN).

Die Blockchiffren LED und KATAN sind für Hardware optimiert, während TEA und der Referenzalgorithmus AES eher für die Implementierung in Software konzipiert wurden. Bei der Entwicklung von KATAN und LED wurde darauf geachtet, dass sie effizient in Hardware, d.h. mit möglichst wenig Hardwarebausteinen realisiert werden können. Durch den Fokus auf die Hardwareimplementierung haben die Algorithmen KATAN und LED eine entsprechend schlechtere Effizienz in der Softwareimplementierung. In Tabelle 3 sind Softwareimplementierungen der Algorithmen AES, TEA und KATAN dargestellt. Auffallend ist der deutlich höhere (Faktor 10) Energieverbrauch von KATAN gegenüber AES und TEA. Der Energieverbrauch korreliert auch mit der Anzahl der zum Ver- und Entschlüsseln benötigten CPU Zyklen. Insgesamt schneidet TEA schlechter als AES ab, d.h. die Anzahl der benötigten CPU Zyklen, der Energieverbrauch und der Durchsatz sind bei AES deutlich besser als bei TEA. Bei den Werten in der Tabelle ist zu beachten, dass die Blockgröße von AES doppelt so groß wie die von TEA bzw. LED und KATAN-64 ist und damit die Werte in den Spalten (CPU-)Zyklen und Energieverbrauch entsprechend normalisiert werden müssen. Auffällig ist der Wert von 0 Byte RAM bei der TEA-Implementierung von Thomas Eisenbarth et al.[8]. Leider lässt sich der Quelle keine Begründung für den Wert entnehmen. Wir vermuten, dass ausschließlich mit CPU Registern gearbeitet wurde und diese nicht als Speicher (RAM) gewertet wurden.

Tabelle 3: Vergleich der Softwareimplementierungen (bei 4MHz)

Chiffre	**Block-größe**	**Schlüssel-größe**	**Code Größe**	**RAM**	**Zyklen**	**Zyklen**	**Durch-satz**	**Energie-verbrauch**
	[bits]	[bits]	[bytes]	[bytes]	[enc+key]	[dec+key]	[Kbps]	[μJ]
AES [7]	128	128	1659	33	4557	7015	-	19.2
AES [8]	128	128	2606	224	6637	7429	77.1	-
TEA [7]	64	128	648	24	7408	7539	-	30.3
TEA [8]	64	128	1140	0	6271	6299	40.8	-
KATAN[7]	64	80	338	18	72063	88525	-	289.2

In Tabelle 4 werden Hardwareimplementierungen von KATAN, AES und LED verglichen. AES benötigt eine deutlich größere Anzahl an Gattern, hat aber auch einen Geschwindigkeitsvorteil gegenüber den Vergleichsalgorithmen. LED und KATAN verbrauchen jeweils eine vergleichbare Anzahl an Gattern. Die kompakteste Implementierung ist mit KTANTAN32 möglich. KATAN hat einen deutlichen Geschwindigkeitsvorteil gegenüber LED, dieser Vorteil vergrößert sich

nochmal bei steigender Blockgröße von KATAN. Ein Vorteil von LED ist, dass die Schlüssellänge flexibel ist und an das gewünschte Sicherheitsniveau angepasst werden kann.

Tabelle 4: Vergleich der Hardwareimplementierungen [11] [4]

Chiffre	Schlüssel-größe	Block-größe	Zyklen pro Block	Durchsatz bei 100 KHz (Kbps)	Struktur-größe	Fläche GE
flexible Schlüssel						
AES-128 [12]	128	128		0.08	0.13	3100
AES-128 [9]	128	128	1032	12.4	0.35	3400
AES-128 [20]	128	128	226	56.6	0.13	2400
KATAN32	80	32		12.5	0.13	802
KATAN48	80	48		18.8	0.13	927
KATAN64	80	64	255	25.1	0.13	1054
LED-64	64	64	1248	5.1	0.18	966
LED-80	80	64	1872	3.4	0.18	1040
LED-96	96	64	1872	3.4	0.18	1116
LED-128	128	64	1872	3.4	0.18	1265
feste Schlüssel						
KTANTAN32	80	32		12.5	0.13	462
KTANTAN48	80	48		18.8	0.13	588
KTANTAN64	80	64	255	25.1	0.13	688
LED-64	64	64	1280	5.13	0.18	688
LED-80	80	64	1872	3.4	0.18	695
LED-96	96	64	1872	3.42	0.18	700
LED-128	128	64	1872	3.42	0.18	726

In Tabelle 5 sind verschiedene Angriffe auf TEA und LED zusammengefasst. Dabei werden einerseits die Angriffe aufgeführt, die die bisher meisten Runden angreifen (abgesehen von Seitenkanalattacken) und andererseits wird eine MITM Attacke genannt, da dieser Angriff sowohl auf LED als auch auf XTEA durchgeführt wurde[13]. Bei XTEA entspricht der MITM-Angriff dem Angriff mit den bisher meist attackierten Runden.

Tabelle 5: Angriffe auf (X)TEA und LED im Vergleich

Chiffre	Attacke	Typ	Runden	Zeit	Daten	Speicher	Ref
`LED-64`	MITM (SK)	KR	8	2^{56}	2^{8}	2^{11}	[13]
(32 Runden)	MITM/Differenziell (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[21]
`LED-128`	MITM (SK)	KR	16	2^{112}	2^{16}	2^{19}	[13]
(48 Runden)	MITM/Differenziell (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[21]
`XTEA` (64 Runden)	MITM (SK)	KR	29	2^{124}	2^{45}	2^{4}	[13]
`TEA` (64 Runden)	Linear	KR	23	$2^{119.64}$	2^{64}	–	[5]

7 Fazit

Abschließend lässt sich sagen, dass es sehr wichtig ist vor welchem Hintergrund ein Algorithmus entwickelt wurde. Die Fokussierung auf die Hardwareimplementierung von LED und KATAN schlägt sich in den Performancedaten sehr deutlich nieder. Für die TEA Familie als Softwarealgorithmus finden sich keinerlei Daten bezüglich einer Hardwareimplementierung. Durch die offensichtliche Diskrepanz in den Anforderungen lässt sich die Ungeeignetheit jedoch schlussfolgern.

Jeder der untersuchten Algorithmen hat spezifische Vor- und Nachteile. So ist z.B. die Implementierung von KTANTAN32 sehr kompakt und LED lässt sich im Sicherheitsniveau anpassen. Jedoch sollte der erste Schritt bei der Wahl eines Verschlüsselungsalgorithmus immer die Selektion auf Algorithmen die für die eigene Umgebung entwickelt wurden sein. Alle untersuchen Algorithmen besitzen spezifische Schwachstellen für die bereits spezielle Angriffe existieren. Jedoch wurden in jedem der beschriebenen Angriffe die Algorithmen bewusst geschwächt oder andere spezielle Voraussetzungen (physikalischer Zugriff auf die Hardware an speziellen Stellen) durch die der Angriff erst möglich wird. Häufig lassen sich diese Faktoren durch die Art der Implementierung ausschließen. In diesem Fall spricht nichts gegen eine weitere Verwendung der untersuchten Algorithmen. Es sollte jedoch immer die neueste Version des jeweiligen Algorithmus verwendet werden, da mit jeder neuen Version häufig Schwachstellen behoben wurden bzw. die Effizienz gesteigert wurde. Dies gilt, wie das Beispiel AES zeigt, teilweise auch für neue Implementierungen bestehender Algorithmen.

Es ist wichtig und sinnvoll für verschiedene Anwendungsfälle verschiedene Verschlüsselungsalgorithmen zu entwickeln und zu verwenden. Nach heutigem Stand der Technik gibt es keinen universalen Algorithmus der für jeden Anwendungsfall uneingeschränkt zu empfehlen ist.

Literatur

1. V. R. ANDEM. A cryptanalysis of the tiny encryption algorithm. `http://www.tayloredge.com/reference/Mathematics/VRAndem.pdf`.
2. B. Andrews. Tiny encryption algorithm (tea) cryptography 4005.705.01 graduate team acd final report. `http://www.cs.rit.edu/~sdc3737/final.pdf`.
3. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 450–466, Berlin, Heidelberg, 2007. Springer-Verlag.
4. C. Cannière, O. Dunkelman, and M. Knežević. Katan and ktantan – a family of small and efficient hardware-oriented block ciphers. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 272–288, Berlin, Heidelberg, 2009. Springer-Verlag.
5. J. Chen, M. Wang, and B. Preneel. Impossible differential cryptanalysis of the lightweight block ciphers tea, xtea and hight. In *AFRICACRYPT'12*, pages 117–137, 2012.
6. P. Derbez, P. Fouque, and J. Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 371–387, 2013.
7. T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F.-X. Standaert, and L. van Oldeneel tot Oldenzeel. Compact implementation and performance evaluation of block ciphers in attiny devices. In *Proceedings of the 5th International Conference on Cryptology in Africa*, AFRICACRYPT'12, pages 172–187, Berlin, Heidelberg, 2012. Springer-Verlag.
8. T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Des. Test*, 24(6):522–533, Nov. 2007.
9. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. Aes implementation on a grain of sand. *IEE Proceedings - Information Security*, 152:13–20(7), October 2005.
10. V. Grosso, C. Boura, B. Gérard, and F.-X. Standaert. A note on the empirical evaluation of security margins against algebraic attacks (with application to low cost-ciphers led and piccolo). In *33rd WIC Symposium on Information Theory in the Benelux*, 2012.
11. J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The led block cipher. In *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'11, pages 326–341, Berlin, Heidelberg, 2011. Springer-Verlag.
12. P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, DSD '06, pages 577–583, Washington, DC, USA, 2006. IEEE Computer Society.
13. T. Isobe and K. Shibutani. Security analysis of the lightweight block ciphers xtea, led and piccolo. In *Proceedings of the 17th Australasian Conference on Information Security and Privacy*, ACISP'12, pages 71–86, Berlin, Heidelberg, 2012. Springer-Verlag.

14. P. Israsena. Design and implementation of low power hardware encryption for low cost secure rfid using tea.
15. P. Israsena and S. Wongnamkum. Hardware implementation of a tea-based light-weight encryption for rfid security.
16. P. Jovanovic, M. Kreuzer, I. Polian, and U. Passau. An algebraic fault attack on the led block cipher.
17. Y. Ko, S. Hong, W. Lee, S. Lee, and J.-S. Kang. Related key differential attacks on 27 rounds of xtea and full-round gost. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer Berlin Heidelberg, 2004.
18. J. Lu. Related-key rectangle attack on 36 rounds of the xtea block cipher. *Int. J. Inf. Secur.*, 8(1):1–11, Jan. 2009.
19. F. Mendel, V. Rijmen, D. Toz, and K. Varici. Differential analysis of the led block cipher. *IACR Cryptology ePrint Archive*, 2012:544, 2012. informal publication.
20. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, pages 69–88, Berlin, Heidelberg, 2011. Springer-Verlag.
21. I. Nikolic, L. Wang, and S. Wu. Cryptanalysis of round-reduced LED. *IACR Cryptology ePrint Archive*, 2015:429, 2015.
22. G. Sekar. Meet-in-the-middle attacks on reduced-round xtea. `http://www.cosic.esat.kuleuven.be/publications/article-1505.pdf`.
23. D. J. Wheeler. Correction to xtea. `http://www.movable-type.co.uk/scripts/xxtea.pdf`.
24. D. J. Wheeler. Tea, atin yencryption algorithm. `http://www.cix.co.uk/~klockstone/tea.pdf`.
25. Wikipedia. Nothing up my sleeve number. `https://en.wikipedia.org/wiki/Nothing_up_my_sleeve_number/`. visited 28-July-2015.
26. Wikipedia. Kryptologische hashfunktion. `https://de.wikipedia.org/wiki/Kryptologische_Hashfunktion`, 2015. visited 21-November-2015.
27. Wikipedia. Tiny encryption algorithm. `https://de.wikipedia.org/wiki/Tiny_Encryption_Algorithm`, 2015. visited 28-Aug-2015.
28. Wikipedia. Xtea. `https://en.wikipedia.org/w/index.php?title=XTEA&oldid=660786817`, 2015. visited 15-July-2015.
29. D. Williams. The tiny encryption algorithm (tea). `http://derekwilliams.us/docs/CPSC-6128-TEA-Encryption.pdf`.
30. G. Wirtschaftslexikon. Internet der dinge. `http://wirtschaftslexikon.gabler.de/Definition/internet-der-dinge.html?referenceKeywordName=Internet+of+Things`, 2015. visited 28-July-2015.
31. E. Yarrkov. Cryptanalysis of xxtea. `http://eprint.iacr.org/2010/254.pdf`.
32. X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, and K. Ji. Algebraic differential fault attacks on led using a single fault injection, 2012.