Matthias Hiller, Michael Pehl, Georg Sigl

Fehlerkorrekturverfahren zur sicheren Schlüsselerzeugung mit Physical Unclonable Functions

Silizium basierte Physical Unclonable Functions (PUFs) werten Fertigungsschwankungen in elektronischen Schaltungen aus, um sicher und zuverlässig kryptographische Schlüssel in eingebetteten Systemen bereitzustellen. Die aus den Schwankungen abgeleiteten binären PUF-Antworten sind jedoch typischerweise verrauscht, enthalten also Fehler. Da der Schlüssel keine Fehler mehr enthalten darf, werden speziell für die Anforderungen von PUFs ausgelegte Fehlerkorrekturverfahren eingesetzt, um diese Fehler so zu korrigieren, dass zuverlässige Schlüssel generiert werden können.

1 Einleitung

Aktuelle Trends wie Industrie 4.0 oder das Internet der Dinge basieren zum großen Teil auf eingebetteten Systemen, die mit ihrer Umwelt interagieren und kommunizieren. Dabei werden sensible, private, und somit schützenswerte Daten verarbeitet. Zudem enthalten die Systeme oftmals Software mit wertvollem Know-how des Herstellers. Deshalb werden zunehmend kryptographische Verfahren in eingebetteten Systemen eingesetzt, die nahezu immer einen geheimen, sicher gespeicherten Schlüssel benötigen.

Dem gegenüber steht ein starker Kostendruck, aufgrund dessen konventionelle Technologien zur Realisierung sicherer Schlüsselspeicher (z. B. [1]) in den genannten Einsatzszenarien oft nicht genutzt werden können, oder sollen. Eine Lösung für den Spagat zwischen Kosteneffizienz und Sicherheit bieten Physical Unclonable Functions (PUFs):

PUFs [2, 3, 4] messen zufällige Schwankungen in den Materialeigenschaften, die bei der Fertigung unvermeidlich sind, und geben sie als PUF-Antwort aus. Silizium basierte PUFs – auf die sich diese Arbeit bezieht – sind elektronische Schaltungen, die kosteneffizient mit denselben Standardprozessen hergestellt werden, mit denen auch die übrigen Schaltungsteile auf einem Chip gefertigt werden. Dabei können die von einer PUF erzeugten binären Werte gut gegen Angriffe geschützt werden, da das Geheimnis zur Generierung eines Schlüssels nicht, wie bisher, dauerhaft im Gerät gespeichert werden muss, sondern nur bei Bedarf erzeugt wird. Ein weiterer Vorteil von PUFs ist, dass sie nicht nur auf speziell gefertigten Chips (ASICs) sondern auch auf konfigurierbaren Chips (FPGAs) – die immer öfter auch in Produkten mit geringen und mittleren Stückzahlen zum Einsatz kommen – realisiert und eingesetzt werden können.

PUFs haben jedoch den Nachteil, dass die erzeugte binär codierte PUF-Antwort, von der ein Schlüssel abgeleitet werden soll, Fehler enthalten kann. Dies ist darauf zurückzuführen, dass äußere Größen wie Temperatur, Spannung und Rauschen das Er-

Matthias Hiller

ist seit 2011 wissenschaftlicher Mitarbeiter am Lehrstuhl für Sicherheit in der Informationstechnik der Technischen Universität München.

E-Mail: matthias.hiller@tum.de

Michael Pehl

ist seit 2012 wissenschaftlicher Mitarbeiter am Lehrstuhl für Sicherheit in der Informationstechnik der Technischen Universität München.

E-Mail: m.pehl@tum.de

Georg Sigl

leitet seit 2010 den Lehrstuhl für Sicherheit in der Informationstechnik der Technischen Universität München und das Fraunhofer Institut AISEC. Zuvor verantwortete er die Entwicklung mehrerer prämierter industrieller Chipkarten Plattformen.

E-Mail: sigl@tum.de

gebnis bei der Messung der Fertigungsschwankungen beeinflussen. Die resultierenden Messfehler führen dann zu fehlerhaften Bits im abgeleiteten Geheimnis. Auch Alterungseffekte, die das Verhalten einer Schaltung mit der Zeit verändern, können zur Änderung der ursprünglichen PUF-Antwort führen.

Die Verwendung von Fehlerkorrekturverfahren ist daher unverzichtbar, um nicht nur sichere, sondern auch zuverlässige Schlüssel mithilfe von PUFs erzeugen zu können. Für eine möglichst hohe Leistungsfähigkeit der PUF-spezifischen Fehlerkorrektur wird diese in der Regel mit fehlerkorrigierenden Codes [5] verknüpft. Die unterschiedlichen Verfahren zur Fehlerkorrektur bei PUFs haben gemeinsam, dass sogenannte Helper Daten erzeugt werden, die Informationen über die PUF außerhalb der Schaltung öffentlich speichern. Mit diesen Informationen wird die zufällige PUF-Antwort auf eine regelmäßige Struktur – ein Codewort – abgebildet in der Fehler erkannt und korrigiert werden können. Dabei ist es entscheidend, dass die Helper Daten keine Informationen über den erzeugten Schlüssel enthalten.

Um dieses Ziel zu erreichen, wurden zahlreiche Fehlerkorrekturverfahren mit unterschiedlichen Eigenschaften entwickelt. Zwei neue Vertreter der beiden für PUFs verwendeten Fehlerkorrekturklassen werden in diesem Beitrag näher vorgestellt:

- **Systematic Low Leakage Coding (SLLC)** [6] nutzt die gesamte PUF-Antwort und hält dabei die Größe der Helper Daten minimal. Das Verfahren ist besonders für PUFs geeignet, bei denen nur eine kleine Anzahl von Fehlern korrigiert werden muss. SLLC hat gegenüber dem vorherrschenden Fuzzy-Extractor-Ansatz [7] den Vorteil, dass die Helper Daten keinerlei Informationen über den reproduzierten Schlüssel preisgeben.
- **Differential Sequence Coding (DSC)** [8] wählt besonders zuverlässige Bits einer PUF–Antwort aus und reduziert dadurch die notwendige Komplexität der nachfolgenden Fehlerkorrektur bei PUFs mit hoher Fehlerwahrscheinlichkeit. Anstatt wie bisher, wie z.B. in [9], feste Blockgrößen zu verwenden ermöglicht die lange differenziell codierte Sequenz eine bessere Auswahl und somit niedrigere Fehlerwahrscheinlichkeit.

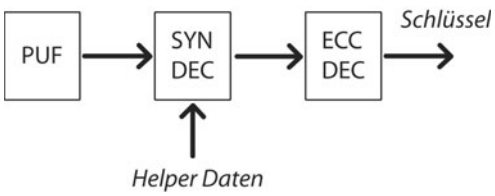
2 Bausteine zur Schlüsselerzeugung

Wie in Abbildung 1 zu sehen, enthält ein derzeitiges System zur sicheren Schlüsselerzeugung mindestens die folgenden drei Komponenten auf dem Chip, die in diesem Abschnitt genauer erläutert werden sollen:

- PUF-Schaltung
- Syndrom Decodierer
- Decodierer des fehlerkorrigierenden Codes (Englisch: Error-Correcting Code (ECC))

Das System liest die Helper Daten ein, verarbeitet PUF-Antwort und Helper Daten, und erzeugt daraus den geheimen Schlüssel. Es

Abbildung 1 | Bausteine zur Schlüsselerzeugung



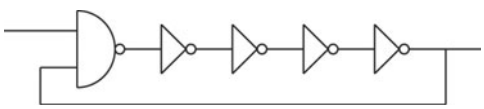
sei darauf hingewiesen, dass die Helper Daten bei der Fertigung typischerweise in einer sicheren Umgebung beim Hersteller berechnet werden und die Berechnung deshalb auch außerhalb des Chips durchgeführt werden kann.. Für diesen Schritt muss daher nicht zwingend eine Schaltung auf dem Chip vorhanden sein.

2.1 PUF-Schaltung

In den letzten Jahren wurde eine große Anzahl an PUF-Schaltungen zur Auswertung der Fertigungsschwankungen auf einem Chip und zur Erzeugung von PUF-Antworten entwickelt. Zwei dieser PUF-Schaltungen – die bereits gut erforscht sind – sind die SRAM PUF [10] und die Ringoszillator (RO) PUF [11]. Diese beiden PUF-Schaltungen werden im Folgenden kurz beschrieben:

- **SRAM PUF:** Eine SRAM-Zelle ist eine Speicherzelle eines flüchtigen Speichers. Das bedeutet, sie kann ein Bit speichern, so lange eine Versorgungsspannung anliegt. Eine SRAM-Zelle besteht im Kern aus zwei Invertern, deren Ausgang jeweils mit dem Eingang des anderen Inverters verbunden ist. Da das Ergebnis dieser Schaltung logisch nicht definiert ist, hat jede SRAM-Zelle nach dem Einschalten einen unbekannten Startzustand. Theoretisch ist dabei weder der logische Wert 0 noch der logische Wert 1 in der Zelle gespeichert. Da dies jedoch ein sehr instabiler Zustand ist, nimmt jede Zelle einen Zustand an, der einer gespeicherten logischen 0 oder 1 entspricht. Welcher der beiden Zustände angenommen wird, hängt bei einer idealen SRAM PUF-Zelle ausschließlich von Fertigungsschwankungen ab und ist nur schwer vorhersagbar. Der Wert kann daher zur Schlüsselerzeugung genutzt werden. SRAM PUFs sind sehr kleine Schaltungen, sodass viele PUF-Antwort-Bits auf einer kleinen Chipfläche erzeugt werden können.
- **Ringoszillator PUF:** Mit einer ungeraden Anzahl invertierender Elemente kann, wie in Abbildung 2 dargestellt, eine oszillierende Schaltung gebaut werden, die nie einen stabilen Zustand erreicht. Dabei wird die Verzögerung jedes einzelnen Inverters – und damit die Frequenz der Schwingung – von Fertigungsschwankungen bestimmt.

Abbildung 2 | Ringoszillator als Digitalschaltung



- Die Ringoszillator (RO) PUF vergleicht die Frequenzen von verschiedenen Oszillatoren auf einem Chip und leitet daraus die geheime PUF-Antwort ab.
- Durch die vergleichende Messung von zwei Frequenzen wird die Schaltung robuster gegen äußere Einflüsse. Schließlich beeinflussen Effekte wie Temperatur oder Alterung immer beide Oszillatoren und werden dadurch abgeschwächt. RO PUFs sind relativ zuverlässig und können gut auf FPGAs implementiert werden.

2.2 Syndrom Decodierer

In der Kanalcodierung beschreibt das Syndrom, welche Fehler in einem empfangenen Codewort vorliegen, um diese dann zu korrigieren.

Der Begriff Syndrome Coding bei PUFs ist daran angelehnt. Ziel der Syndrom Decodierung bei PUFs ist es, mit Hilfe der Helper Daten die Fehler in der PUF-Antwort zu identifizieren und zu korrigieren. Mithilfe der Helper Daten wird dazu die PUF-Antwort auf ein Codewort des nachgelagerten fehlerkorrigierenden Codes abgebildet. Wichtig dabei ist, dass die Helper Daten möglichst wenig Information über den aus der PUF-Antwort generierten Schlüssel erhalten.

Es gibt zwei vorherrschende Ansätze, um die PUF-Antwort auf ein Codewort abzubilden: Ein Teil der Verfahren, z. B. [6] [7], erzeugt Codewörter, indem die PUF-Antwort mit den Helper Daten logisch verknüpft wird. Hierbei ist der wesentlichste Unterschied zwischen den Ansätzen, wie diese Verknüpfung durchgeführt wird und – damit verbunden – wie viel Information die Helper Daten über das erstellte Geheimnis preisgeben. Enthalten die Helper Daten zu viel Information über das Geheimnis, kann eine Hashfunktion verwendet werden, die das Geheimnis komprimiert. Dadurch wird die Unsicherheit des ursprünglichen Geheimnisses auf ein kürzeres Geheimnis verdichtet, wodurch die Unsicherheit pro Bit, aber nicht die absolute Unsicherheit in dem Geheimnis steigt. Im Umkehrschluss muss dann zur Erreichung derselben Schlüssellänge, und damit desselben Sicherheitsniveaus, eine größere Anzahl Daten von der PUF erzeugt werden, was – genauso wie die Hashfunktion selbst – zu einem höheren Flächenverbrauch und damit zu einem höheren Preis pro Chip führt.

Andere Verfahren, wie z. B. [8] [9], speichern Zeiger auf einzelne Bits der PUF-Antwort die bestimmte Kriterien erfüllen. Sie werden in erster Linie verwendet, wenn Zuverlässigkeitsinformationen für die einzelnen PUF-Antworten vorliegen. Damit kann die Anzahl der Fehler, die von dem nachfolgenden ECC Decodierer korrigiert werden müssen – und damit die Komplexität der nächsten Stufe – reduziert werden, wenn zuverlässige Bits ausgewählt werden.

Wird die praxisnahe Annahme getroffen, dass aus der Position eines Bits in der PUF-Antwort kein Rückschluss auf dessen Wert gezogen werden kann, so geben zeigerbasierte Verfahren keine Informationen über das Geheimnis preis.

2.3 Decodierer des fehlerkorrigierenden Codes

Fehlerkorrigierende Codes (ECC) ermöglichen es, Fehler in einer Bitfolge zu detektieren und zu korrigieren. Für PUFs spielen typische Größen – wie decodierte Bits pro Sekunde oder Latenz bis das Decodierergebnis vorliegt – meist nur eine untergeordnete Rolle. Primär ist die Größe der Decodiererschaltung von Interesse.

In den letzten Jahren wurden vor allem seit langem bekannte Codes, wie Reed-Muller, BCH oder Faltungscodes [5], mit herkömmlicher Codeverkettung für die Implementierungen im PUF-Kontext verwendet. Sie sind oft weniger effizient als moderne Verfahren, können jedoch relativ ressourcensparend implementiert werden.

Für die Verwendung im PUF-Kontext wurden bereits verschiedene auf Größe optimierte Decodierer für die unterschiedlichen Codeklassen entwickelt. Diese zerlegen komplexe Rechenoperationen in einfachere Einzelteile, die dann seriell abgearbeitet werden. Die Implementierungen für Reed-Muller und BCH Codes in [12] und [13] verwenden optimierte Coprozessoren, die spezielle Befehle für die Decodierung des jeweiligen Codes enthalten. Dagegen wird in [14] ein doppelter Datensatz verwendet, der sehr einfach sequenzielle Berechnungen ermöglicht.

Neue Ansätze mit verallgemeinerter Codeverkettung ermöglichen es, weiterhin einfach decodierbare Codes mit kurzer Länge zu verwenden und dennoch die Effizienz der Fehlerkorrektur zu steigern [15].

3 Anforderungen an Fehlerkorrekturverfahren

Typischerweise werden die benötigte Schlüssellänge und die maximal zulässige Fehlerwahrscheinlichkeit für ein System durch eine Spezifikation vorgegeben. Durch die Kombination unterschiedlicher Algorithmen und Parameter gibt es viele verschiedene Möglichkeiten und Stellgrößen, diese Spezifikation zu erreichen. Für die Auslegung der Fehlerkorrektur ist darüber hinaus die Bitfehlerwahrscheinlichkeit am Eingang – also die Wahrscheinlichkeit für einen Fehler in einem PUF-Antwort Bit – entscheidend. Diese Fehlerwahrscheinlichkeit hängt neben dem Einsatzszenario stark vom verwendeten PUF-Typ ab. Da bei dieser Wahl oftmals auch nicht-technische – beispielsweise patentrechtliche – Erwägungen eine Rolle spielen, soll hier die Wahl des PUF-Typs und damit die Eingangsbitfehlerwahrscheinlichkeit von der Realisierung des Fehlerkorrekturverfahrens entkoppelt betrachtet werden.

Abbildung 3 zeigt drei Größen, die eine zentrale Rolle spielen, um die Effizienz der Implementierung eines Fehlerkorrekturverfahrens zu bewerten:

- **Chipfläche:** Die Anzahl der benötigten PUF-Bits und die Größe der Fehlerkorrektur bestimmen die Chipfläche der Schaltung, und damit die Kosten für das Modul.
- **Externer Speicher:** Die Helper Daten für die Fehlerkorrektur müssen extern gespeichert werden. Je nach Anwendung können diese Kosten stark variieren. Insgesamt steigen die Kosten jedoch mit zunehmender Anzahl Helper Daten Bits.
- **Laufzeit:** Die Zeit, die für den gesamten Vorgang benötigt wird, ist die letzte der drei Größen. Sie spielt vor allem dann eine Rolle, wenn andere Komponenten auf die rechtzeitige Generierung des Schlüssels angewiesen sind und eine vorherige Berechnung nicht möglich ist.

Je nach Hardware und Einsatzgebiet können die Größen unterschiedlich bewertet werden. RO PUFs auf einem FPGA benötigen z. B. eine größere Fläche, sodass es sich lohnt die Fehlerkorrektur zu verbessern, um die Anzahl der PUF-Bits zu reduzieren. Möchte man ein Modul mit einer SRAM PUF als ASIC fertigen, kann eine weniger effiziente Fehlerkorrektur durch eine größere Anzahl günstiger PUF-Bits ausgeglichen werden.

Abbildung 3 | Designgrößen für PUF Fehlerkorrektur

Die nachfolgenden Beispiele zeigen jedoch, dass es möglich ist, die Effizienz der Schlüsselerzeugung mit neuen Algorithmen zu ver-

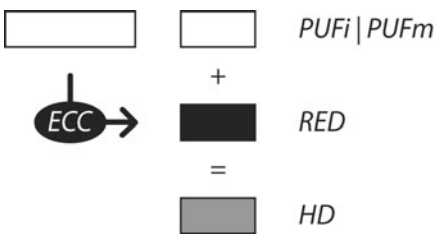
bessern. Von besonderer Wichtigkeit ist hierbei die Entwicklung geeigneter Syndrom Decodierer, die deshalb im Folgenden behandelt werden. Bei diesen liegt der Fokus in dieser Arbeit auf einer Minimierung der Helper Daten für den Syndrome Decodierer und der benötigten Anzahl PUF-Bits. Die Komplexität des Syndrom Decodierers soll hierbei möglichst gering gehalten werden.

3 Systematic Low Leakage Coding

Wie in Abschnitt 2.2 erläutert, ist die logische Verknüpfung der PUF-Antwort mit den Helper Daten zu einem gültigen Codewort ein häufig verwendeter Ansatz für die Syndrom Decodierung bei PUFs. Systematic Low Leakage Coding (SLLC) [6] soll hier als ein neuer Vertreter dieser Verfahren beispielhaft vorgestellt werden. SLLC setzt Codes mit systematischer Codierung voraus, also Codes, bei denen Informations- und Redundanzteil voneinander getrennt werden können. Entsprechend dieser Aufteilung wird nun – wie in Abbildung 4 dargestellt – bei SLLC die PUF-Antwort in zwei Teile aufgeteilt: Einen Informationsteil *PUFi* und einen Maskierungsteil *PUFm*.
Die PUF-Antwort wird während der Fertigung unter kontrollierten Umweltbedingungen gemessen. Anschließend wird aus dem Informationsteil *PUFi* – der später zur Ableitung des Schlüssels verwendet wird – die Redundanz *RED* berechnet. Um die in der Redundanz enthaltene Information über *PUFi* – die zur späteren Fehlerkorrektur benötigt wird – zu schützen, wird die Redundanz mit dem Maskierungsteil der PUF-Antwort Exklusiv-Oder verknüpft. Durch die Exklusiv-Oder Verknüpfung ist einem möglichen Angreifer der Redundanzteil nicht mehr zugänglich; er ist maskiert. Das Ergebnis kann somit öffentlich als Helper Daten *HD* gespeichert werden.

Es sei darauf hingewiesen, dass ein Angreifer nur dann noch Information über das Geheimnis aus den Helper Daten ableiten kann, wenn die Maskierung Schwächen aufweist.

Abbildung 4 | Helper-Daten-Erzeugung mit SLLC; Die Addition entspricht der SLLC Codierung

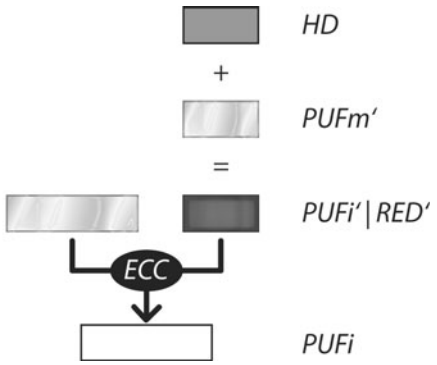


Bei der späteren Reproduktion des Schlüssels wird – wie in Abbildung 5 dargestellt – die verrauschte PUF-Antwort *PUF'* aus der PUF ausgelesen. *PUFm'* wird mit den Helper Daten *HD* wieder Exklusiv-Oder verknüpft, um daraus eine verrauschte Version *RED'* der ursprünglichen Redundanz *RED* zu berechnen. Danach kann der ECC Decodierer den ursprünglichen Informationsteil *PUFi* aus *PUFi'* und *RED'* berechnen und diesen entweder direkt als, oder zur Weiterverarbeitung zum Schlüssel ausgeben.

In [6] wurde gezeigt, dass dieses Verfahren für PUFs ohne Zuverlässigkeitsinformationen über die PUF-Antworten theoretisch optimal ist, da es die gesamte verfügbare Information in der PUF–Antwort nutzt und gleichzeitig die Größe der Helper Daten minimiert.

Für eine Beispielimplementierung des Verfahrens wurde außerdem gezeigt [6], dass mit SLLC eine erhebliche Flächenreduktion der Gesamtschaltung möglich ist. Dies ist insbesondere darauf zurückzuführen, dass bei der Implementierung – im Gegensatz zu vergleichbaren Verfahren – auf den Einsatz einer Hashfunktion verzichtet werden konnte.

Abbildung 5 | Schlüsselerzeugung mit SLLC; Die Addition entspricht der SLLC Decodierung

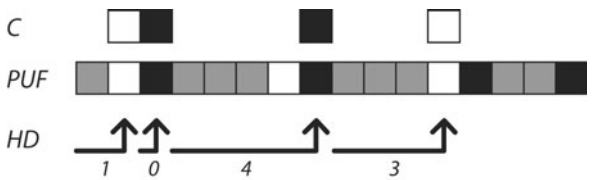


4 Differential Sequence Coding

Differential Sequence Coding (DSC) [8] ist ein Vertreter des bereits erwähnten zweiten Ansatzes, bei dem besonders zuverlässige Bits der PUF-Antwort ausgewählt und Zeiger auf die entsprechenden Positionen gespeichert werden. Dadurch können unzuverlässige Bits bei der Codierung aussortiert werden, was den Implementierungsaufwand für die Fehlerkorrektur erheblich reduziert.

Um unter Verwendung von DSC einen Schlüssel mithilfe PUFs zu speichern, wird zunächst ein zufälliges Geheimnis *Z* erzeugt. Dieses wird mit dem ECC Codierer zu einem Codewort *C* codiert. Dieses beinhaltet die geheime Information für den Schlüssel und die nötige Redundanz zur Fehlerkorrektur. Um das Geheimnis im nächsten Schritt in eine PUF einzubetten, werden nun unter kontrollierten Umweltbedingungen während der Fertigung zuverlässige PUF-Bits gesucht.

Abbildung 6 | Helper-Daten-Erzeugung mit DSC-Zeigern (weiß = 0; schwarz = 1; grau = unzuverlässig)



Wie in Abbildung 6 zu sehen ist, wird die Position der stabilen PUF-Bits als Zeiger *HD* in den Helper Daten gespeichert. Ein Angreifer kann hierbei aus der Kenntnis der Position der verwendeten PUF-Bits keinen Rückschluss auf das Codewort ziehen.

Um später beim Einsatz des Chips das Codewort und damit den Schlüssel zu reproduzieren, werden mithilfe der Zeiger *HD* die entsprechenden Bits der verrauschten PUF-Antwort *PUF'* aufgerufen, sodass sie eine verrauschte Version *C'* des Codeworts *C*

bilden. Schließlich kann der ECC Decodierer Z aus C' wiederherstellen.

Es würde viel Speicherplatz kosten, die absoluten Positionen für alle Zeiger zu speichern. Die Besonderheit bei DSC ist daher die differenzielle Speicherung des Abstands zwischen den ausgewählten Bits. D.h. es wird jeweils gespeichert, wie viele PUF-Bits zwischen dem aktuellen und dem letzten zuverlässigen Bit der PUF-Antwort liegen. Hierbei kommen kleine Abstandswerte häufiger vor, es müssen aber auch große Werte gespeichert werden können. Aufgrund der Häufigkeitsverteilung der Abstände kann die Effizienz von DSC durch Kompression der Zeiger weiter gesteigert werden [16].

Wie andere zeigerbasierte Verfahren besitzt DSC den Vorteil, dass die Fehlerwahrscheinlichkeit durch die Auswahl zuverlässiger PUF-Bits bereits reduziert wird. Im Gegensatz zu anderen Verfahren, werden bei DSC jedoch keine Blöcke in der PUF-Antwort vordefiniert, in denen jeweils eine feste Anzahl von Bits gewählt werden muss. Damit überwindet DSC den Nachteil anderer Verfahren, dass die Fehlerwahrscheinlichkeit durch den schlechtesten Block in der PUF-Antwort begrenzt ist und erlaubt eine effizientere Nutzung guter PUF-Antworten.

Dies wurde in [16] durch einen direkten Vergleich mit FPGA-Implementierungen anderer Verfahren auch praktisch gezeigt. Die DSC-Implementierung benötigte nur ein Drittel der Helper Daten und außerdem 20% weniger PUF-Antwort Bits.

5 Fazit

Fehlerkorrekturverfahren ermöglichen es, PUFs als sichere und zuverlässige Schlüsselspeicher für eingebettete Systeme zu verwenden.

Die Herausforderung auf algorithmischer Ebene liegt in den gegenläufigen Interessen, einerseits mit den Helper Daten Informationen über die PUF zu speichern, um Fehlerkorrektur zu ermöglichen, und andererseits keine Informationen über den Schlüssel durch die Helper Daten preiszugeben. Wichtige Kriterien für die Bewertung von unterschiedlichen Fehlerkorrekturansätzen für ein definiertes Szenario sind die Chipfläche, die Größe der Helper Daten und die Laufzeit.

In diesem Beitrag wurden als Beispiele zwei Repräsentanten gängiger Ansätze, nämlich SLLC und DSC, für unterschiedliche Szenarien vorgestellt. SLLC nutzt hierbei die gesamte PUF und minimiert die Größe der Helper Daten. Im Gegenzug wählt DSC besonders zuverlässige Bits der PUF-Antwort aus, um die Komplexität der Implementierung möglichst klein zu halten.

Für ein effizientes Gesamtsystem ist es wichtig, das Zusammenspiel aus PUF, Fehlerkorrektur und weiteren Komponenten durch eine geeignete Auswahl an Algorithmen und Parametern aufeinander abzustimmen.

Danksagung

Diese Arbeit wurde teilweise im Rahmen des Verbundprojekts SIBASE durchgeführt und vom BMBF unter Förderkennzeichen 01IS13020 gefördert.

Literatur

[1] H. Handschuh und E. Trichina, „Securing Flash Technology," in *FDTC*, 2007.
[2] C. Herder, M. Yu, F. Koushanfar und S. Devadas, „Physical Unclonable Functions and Applications: A Tutorial," *Proc. IEEE,* Bd. 102, Nr. 8, pp. 1126-1141, 2014.
[3] D. Merli und G. Sigl, „Physical Unclonable Functions," *DuD,* Bd. 36, Nr. 12, pp. 876-880, 2012.
[4] S. Katzenbeisser und A. Schaller, „Physical Unclonable Functions," *DuD,* Bd. 36, Nr. 12, pp. 881-885, 2012.
[5] M. Bossert, Kanalcodierung (3. Aufl.), Oldenbourg, 2013.
[6] M. Hiller, M. Yu und M. Pehl, „Systematic Low Leakage Coding for Physical Unclonable Functions," in *ASIACCS*, 2015.
[7] Y. Dodis, R. Ostrovsky, L. Reyzin und A. Smith, „Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," *SIAM J. on Computing,* Bd. 1, Nr. 97-139, p. 38, 2008.
[8] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner und G. Sigl, „Breaking through Fixed PUF Block Limitations with Differential Sequence Coding and Convolutional Codes," in *TrustED*, 2013.
[9] M. Yu und S. Devadas, „Secure and Robust Error Correction for Physical Unclonable Functions," *IEEE D&T,* Nr. 27, p. 48–65, 2010.
[10] J. Guajardo, S. S. Kumar, G. J. Schrijen und P. Tuyls, „FPGA Intrinsic PUFs and Their Use for IP Protection," in *CHES*, 2007.
[11] G. E. Suh und S. Devadas, „Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *ACM/IEEE DAC*, 2007.
[12] R. Maes, P. Tuyls und I. Verbauwhede, „Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *CHES*, 2009.
[13] A. Van Herrewege und I. Verbauwhede, „Tiny application-specific programmable processor for BCH decoding," in *IEEE SoC*, 2012.
[14] M. Hiller, L. Rodrigues Lima und G. Sigl, „Seesaw: An Area-Optimized FPGA Viterbi Decoder for PUFs," in *DSD*, 2014.
[15] S. Puchinger, S. Müelich, M. Bossert, M. Hiller und G. Sigl, „On Error Correction for Physical Unclonable Functions," in *ITG SCC*, 2015.
[16] M. Hiller und G. Sigl, „Increasing the Efficiency of Syndrome Coding for PUFs with Helper Data Compression," in *DATE*, 2014.