NutriSafe

Sicherheit in der Lebensmittelproduktion und -logistik
durch die Distributed-Ledger-Technologie

Giesecke+Devrient

NutriSafe Toolkit

– DLT-Sicherheitsbausteine –

Erhöhte DLT Sicherheit gegen Seitenkanalangriffe zweiter Ordnung mittels verbessertem Maskenwechsel

Dr. Lars Hoffmann

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Bundesministerium Landwirtschaft, Regionen und Tourismus

SIFO.de

KIRAS Sicherheitsforschung

FFG Forschung wirkt.

Dieses Dokument ist Bestandteil im **NutriSafe Toolkit**:

nutrisafe.de/toolkit

In einer Kooperation zwischen Deutschland und Österreich forschen Universitäten, Unternehmen und Behörden daran, die Lebensmittelproduktion sowie deren Logistik unter Nutzung von Distributed-Ledger-Technologie sicherer zu machen.

Das Projekt NutriSafe wird auf Österreichischer Seite innerhalb des Sicherheitsforschungs-Förderprogramms KIRAS durch das Bundesministerium für Landwirtschaft, Regionen und Tourismus (BMLRT) gefördert (Projektnummer: 867015). Auf Deutscher Seite wird das Projekt innerhalb des Programms Forschung für die zivile Sicherheit vom Bundesministerium für Bildung und Forschung (BMBF) gefördert (FKZ 13N15070 bis 13N15076).

nutrisafe.de | nutrisafe.at

Erhöhte DLT Sicherheit gegen Seitenkanalangriffe zweiter Ordnung mittels verbessertem Maskenwechsel

Dr. Lars Hoffmann[1]

[1] Giesecke+Devrient Mobile Security GmbH

München 2021

Giesecke+Devrient Mobile Security GmbH

Prinzregentenstr. 159

81677 München

Inhalt

1 Einleitung / Einordnung in den Kontext der Distributed-Leger-Technologie

Im Hyper-Ledger-Framework besteht die Möglichkeit ein PKCS#11 kompatibles Device, wie z.B. eine Smartcard als Hardware-Wallet, einzubinden. Eine solche Hardware-Wallet bietet die Möglichkeit Schlüssel sicher zu verwahren und anzuwenden, um zum Beispiel NutriSafe Transaktionen zu signieren. Nur der berechtigte Besitzer soll die Hardware-Wallet verwenden können. Einem Angreifer, der in den Besitz einer solchen Hardware-Wallet erlangt, soll es nahezu unmöglich sein diese Hardware-Wallet zu verwenden oder Informationen über die in der Hardware-Wallet gespeicherten Schlüssel zu erlangen. Um dies zu erreichen, muß sich der Benutzer gegenüber der Hardware-Wallet und diese gegenüber dem Hyper-Ledger-Hintergrundsystem authentifizieren. Ein klassisches Verfahren ist hierbei die Benutzerauthentifizierung mittels einer PIN („personal identification number“). Die Authentifizierung der Hardware-Wallet kann mit Hilfe eines ECDH („ellliptic-curve Diffie-Hellman“) unterstützendes Verfahrens ausgeführt werden. Der dazu benötigte Schlüssel kann auf der Smartcard generiert und sicher verwahrt werden. Mit Hilfe des ECDH Verfahrens kann die Smartcard und das DLT-Hintergrundsystem ein gemeinsames Geheimnis aushandeln. Das Geheimnis wird dann benutzt, um mit Hilfe sogenannter Schlüsselableitungsfunktionen („key derivation function“, abgekürzt KDF) Sitzungsschlüssel für die Verschlüsselung und Datenintegrität der nachfolgenden Kommunikation herzuleiten. Ähnliche Verfahren werden bei Wallets und der Ableitung von privaten Schlüsseln verwendet. Die verwendeten Schlüsselableitungsfunktionen können hashbasiert (nach PKCS#5) oder, falls zur Kommunikation ein TLS-Protokoll verwendet wird, HMAC basiert, sogenannte HKDFs, sein. Da ein Angreifer nahezu keine Information über den Wert des Geheimnisses und der verwendeten Schlüssel erhalten darf, müssen die Implementierungen auf der Karte gegen Seitenkanalangriffe abgesichert werden.

Bei einem Seitenkanalangriff nutzt der Angreifer zusätzliche Informationen, welche er beim Betreiben des Devices zu erlangen versucht. Als Informationsquellen, auch Seitenkanäle genannt, dienen dabei zum Beispiel das Zeitverhalten, der Stromverbrauch oder das elektromagnetische Abstrahlverhalten. Die ersten Seitenkanalangriffe [KJJ99] gehen auf eine Veröffentlichung von Paul Kocher et al. zurück. Er hat erkannt, daß der Stromverbrauch einer Chipkarte von den Daten abhängt, die verarbeitet werden. Der Stromverbrauch ist dabei proportional zu dem Spannungsabfall an einem Vorwiderstand, der in Reihe mit der Chipkarte angeschlossen wird. Während des Ablaufs der Software auf der Chipkarte kann der Spannungsabfall am Widerstand mit Hilfe eines Speicheroszilloskopes aufgenommen werden. In [KJJ99] werden die auf den Stromverbrauch als Seitenkanal basierenden Angriffe in „simple power analysis“ (SPA) und „differential power analysis“ (DPA) unterteilt. Bei einer erfolgreichen SPA reicht eine einzelne Aufnahme einer Stromkurve, welche den Stromverbrauch während der Abarbeitung der Chipkartensoftware zeigt, aus, um Informationen über einzelne Bits zu erlangen. Bei einer DPA wird eine statistische Analyse vieler Stromkurven vorgenommen, um Teile des verwendeten Schlüssels Stück für Stück zu bestimmen. Für jede aufgenommene Stromkurve und für jeden Teil eines Schlüssels werden für alle möglichen Werte dieses Schlüsselteils Zwischenergebnisse der Implementierung berechnet, die nur von diesem Schlüsselteil abhängen. Jeder mögliche Wert eines Schlüsselteils wird als Schlüsselhypothese bezeichnet. Mit Hilfe der Stromkurven kann nun überprüft werden, ob zu einem Zeitpunkt der

Aufnahme die Zwischenergebnisse, die mit Hilfe einer Schlüsselhypothese berechnet wurden, auch tatsächlich im Softwareablauf auf der Chipkarte verarbeitet werden. Ist dies der Fall, so zeigt sich zu diesem Zeitpunkt eine statistische Auffälligkeit, die sich bei Erhöhung der Anzahl der verwendeten Stromkurven immer stärker abzeichnet.

Da anstelle der Stromkurve jede beliebige andere Seitenkanalkurve verwendet werden kann, die eine Datenabhängigkeit der ausgeführten Implementierung widerspiegelt, sprechen wir im folgenden nur noch von Seitenkanalangriffen und Seitenkanalkurven. Dennoch werden auch hier die Abkürzungen SPA und DPA verwendet.

Werden bei einer Analyse die Daten mehrere Zeitpunkte einer Seitenkanalkurve verknüpft ausgewertet, um eine Schlüsselhypothese zu validieren, so spricht man von Seitenkanalangriffen höherer Ordnung [JPS05, PRB09]. Insbesondere spricht man von Seitenkanalangriffen zweiter Ordnung, wenn zwei Zeitpunkte verwendet werden.

2002 stellten S. Chari, J. R. Rao, P. Rohatgi [CRR02] die Templateanalyse vor, die als eine Erweiterung einer SPA gesehen werden kann. Hier benötigt der Angreifer ein dem anzugreifenden Device (Smartcard, Hardware-Wallet) im Idealfall identisches Device, bei dem er ein Zwischenergebnis in der Implementierung auf jeden möglichen Wert setzen kann. Dadurch ist es ihm möglich Schablonen, sogenannte Templates, aus den aufgenommenen Seitenkanalkurven zu erzeugen. Alle Seitenkanalkurven, die mit Hilfe des Zieldevices erzeugt werden können, werden nun mit den Schablonen abgeglichen und der Grad der Übereinstimmung bestimmt. Hierdurch ergibt sich für das gesuchte Zwischenergebnis auf dem Zieldevice (Smartcard, Hardware-Wallet) eine Wahrscheinlichkeitsverteilung der angenommenen Werte.

Templateanalysen fallen somit unter den Oberbegriff des „supervised learnings“. Es hat sich gezeigt, daß Methoden, wie zum Beispiel „maschine learning“ oder „deep learning“, die in anderen Anwendungen zum „supervised learning“ verwendet wurden, sich auch auf das Problem der Seitenkanalanalyse übertragen lassen [HGM+11, HZ12, PSB+18]. Insbesondere hat sich gezeigt, daß, falls durch eine Templateanalyse Teilinformation über den gesuchten Schlüssel erlangt werden kann, die Analyse häufig durch ein geeignetes Neuronales Netz verbessert werden konnte.

Zur Vertiefung der Theorie der Seitenkanalanalysen sei auf [MOP07] verwiesen.

Implementierungen kryptographischer Algorithmen, die auf Hardware-Wallets verwendet werden, sollten gegen Seitenkanalangriffe geschützt werden. Man unterscheidet bei der Implementierung, je nach Maßnahme, zwischen Verstecken und Maskieren.

Beim Verstecken wird versucht die Information, welche über einen Seitenkanal an den Angreifer preisgegeben wird, zeitlich oder örtlich zu verrauschen, oder durch Überlagerung durch andere Signale abzuschwächen. Diese Maßnahmen machen es einem Angreifer schwieriger die benötigten Informationen aufzusammeln. Sie sind aber allein meist nicht ausreichend einen erfolgreichen Angriff zu verhindern.

Beim Maskieren werden die abgearbeiteten Daten zur Laufzeit mit einem oder mehreren dem Angreifer unbekannten Wert/-en mathematisch überlagert, so daß die maskierten Daten stochastisch unabhängig zu den unmaskierten Daten sind. Maskierte Daten und die zugehörigen Masken werden in der englischen Literatur auch als „shares“ bezeichnet.

Die Maskierungen müssen derart gestaltet sein, daß sich der unmaskierte Wert aus dem maskierten Daten und den Masken wieder berechnen läßt.

Je nach den in der Implementierung vorgegebenen Rechenschritten sind unterschiedliche Maskierungsarten von Vorteil, da sie sich unterschiedlich leicht oder schwer umsetzen lassen.

So sind für binäre Operationen, wie Exklusiv-Oder-Verknüpfung (abgekürzt xor-Verknüpfung; im vorliegenden Dokument auch durch das Operatorsymbol $\oplus$ bezeichnet), oder Shift- oder Rotationsoperationen boolesche Maskierungen am besten geeignet. Aus einem geheim zu haltenden Wert d wird also eine maskierte Repräsentation x mit $x = d \oplus s$ abgeleitet, wobei s die zufällig gewählte Maske ist. Wie erwähnt wird diese Art der Maskierungsregel als boolesche Maskierungsregel bezeichnet, und die verwendeten Masken als boolesche Masken.

Für arithmetische Operationen, wie die Addition oder Subtraktion Modulo einer Zweierpotenz, stellen arithmetische Masken die natürliche Wahl dar. Statt des geheim zu haltenden Wertes d wird also z.B. eine maskierte Repräsentation x mit einer der Maskierungsregeln $x = d + r \bmod 2^n$ oder $x = d - r \bmod 2^n$ oder $x = r - d \bmod 2^n$ oder $x = -(d + r) \bmod 2^n$ verarbeitet, wobei n die Bitbreite ist, mit der die Berechnungen ausgeführt werden. Im vorliegenden Dokument werden solche Maskierungsregeln auch als additive Maskierungsregeln bezeichnet, und zwar unabhängig davon, ob bei der Maskierung eine Addition oder eine Subtraktion ausgeführt wird. Entsprechend werden die verwendeten Masken als additive Masken bezeichnet.

Bei fehlerfreier Implementierung verhindern die gerade genannten booleschen und additiven Maskierungsregeln Seitenkanalangriffe erster Ordnung. Gegen Seitenkanalangriffe zweiter Ordnung bieten diese Regeln jedoch keinen Schutz, weil z.B. bei der booleschen Maskierungsregel ein Angreifer, der die Werte x und s ausspäht, daraus den geheim zu haltenden Wert d ermitteln kann. Durch Erweiterung der Formeln durch jeweils eine weitere Maske kann der Schutz gegen Seitenkanalangriffe zweiter Ordnung erzielt werden. So ist z.B. eine boolesche Maskierung mit zwei zufällig gewählten Masken s_1, s_2 und der Maskierungsregel $x = d \oplus s_1 \oplus s_2$ gegen Seitenkanalangriffe zweiter Ordnung sicher. Dasselbe gilt für eine additive Maskierung mit zwei zufällig gewählten Masken r_1, r_2 und einer Maskierungsregel der Form $x = \pm d \pm r_1 \pm r_2 \bmod 2^n$, also beispielsweise $x = d + r_1 + r_2 \bmod 2^n$. Die Information von d wird also jeweils auf drei „shares“ aufgeteilt.

Kommen bei einer Implementierung, wie zum Beispiel bei den zuvor genannten hash- oder HMAC basierten Schlüsselableitungsfunktionen, sowohl binäre als auch arithmetische Operationen vor, so muß das eine Maskierungsschema in das andere überführt werden können, ohne daß dabei die Daten zu einem beliebigen Zeitpunkt unmaskiert vorliegen. Man spricht hier von einem Maskenwechsel oder Maskierungsübergang.

Im folgenden wird die erste effiziente Implementierung für einem Maskenwechsel einer arithmetischen Maskierung zu einer booleschen Maskierung vorgestellt, die gegen Seitenkanalangriffe zweiter Ordnung geschützt ist.

2 Stand der Technik

In [G01] sind Verfahren zum Maskierungsübergang beschrieben, die gegen Seitenkanalangriffe erster Ordnung geschützt sind. Diese Verfahren sind auch in der Offenlegungsschrift WO 02/065692 A1 offenbart. Bei einem Übergang von einer additiven zu einer booleschen Maskierung wird das gewünschte Resultat Bit für Bit gemäß einer rekursiven Formel berechnet.

In [B10, D12, NP10] werden weitere Verfahren zum Übergang zwischen unterschiedlichen Maskierungen, die gegen Seitenkanalangriffe erster Ordnung geschützt sind, vorgestellt, unter anderem zum Übergang von einer additiven zu einer booleschen Maskierung. Bei manchen der Verfahren wird jeweils mindestens eine vorab berechnete Tabelle mit einer Vielzahl von Einträgen herangezogen.

Eine Sicherung gegen Seitenkanalangriffe erster Ordnung stellt bereits einen wichtigen Vorteil dar. Es wird jedoch in jüngster Zeit zunehmend wichtiger, auch Schutz gegen Seitenkanalangriffe zweiter Ordnung zu bieten. Dies ist z.B. eine Forderung gewisser Sicherheitsstufen der *Common Criteria for Information Technology Security Evaluation* (Allgemeine Kriterien für die Bewertung der Sicherheit von Informationstechnologie) gemäß der Norm ISO/IEC 15408, und auch bei EMVCo-Evaluationen. Die genannten Sicherheitsstufen beziehungsweise Evaluierungen sind aktuell noch nicht auf den Bereich der DLT übertragen. Es fehlt bei DLT- oder Wallet-Anwendungen aktuell ein einheitlicher einzuhaltender Sicherheitsstandard. Weil aber die Werte, die mittels einer DLT verwaltet werden, eine hohe Attraktivität für Angreifer besitzen, müssen diese anhand vergleichbarerer hoher Sicherheitsstandards, bereits normierten Verfahren, gesichert werden.

Es wäre daher wünschenswert, ein Maskierungsübergangsverfahren bereitzustellen, das einen im Vergleich zu bekannten Verfahren gesteigerten Schutz gegen Seitenkanalangriffe aufweist. Ferner wäre es wünschenswert, wenn sich das Verfahren trotz der hohen gebotenen Sicherheit effizient implementieren ließe.

2018 stellte Luk Bettale et al. [BCZ18] ein Verfahren zum Übergang einer booleschen in eine arithmetische Maskierung vor, das gegen Seitenkanalangriffe höherer Ordnung resistent ist.

3 Abgesichertes Verfahren für einen Maskenwechsel von einer arithmetischen zu einer booleschen Maskierung

Ausgehend von dem in [B10] vorgestellten Formeln für einen Maskierungsübergang von einer booleschen zu einer additiven Maskierung wird im folgenden die erste Implementierung für einem Maskenwechsel einer arithmetischen Maskierung zu einer booleschen Maskierung vorgestellt, die gegen Seitenkanalangriffe zweiter Ordnung geschützt ist. Dies schließt die Lücke für eine effiziente und gegen Seitenkanalangriffe zweiter Ordnung resistente Implementierung von Hashfunktionen oder anderen kryptographischen Algorithmen, bei denen nicht nur eine Richtung eines Maskenwechsels erforderlich ist.

3.1 Vorüberlegungen

Für die nun zu beschreibenden Beziehungen sei eine maskierte Repräsentation x eines geheim zu haltenden Basiswertes d mit einer booleschen Maske s gegeben, so dass also $x = d \oplus s$ gilt. Berechnet werden soll eine additive Maske r, mit der die maskierte Repräsentation x den geheim zu haltenden Basiswert d darstellt, also für die $x = d + r \bmod 2^n$ bei gegebener Berechnungsbitbreite n gilt. Um die Rechnung gegen einen Seitenkanalangriff erster Ordnung zu sichern, wird ein zufälliger Verschleierungswert z eingeführt. Die arithmetische Maske r ergibt sich dann durch die folgende, im folgenden als "XOR2ADD-Formel" bezeichnete Beziehung:

$$r = ((d \oplus s \oplus z) - (d \oplus z)) \oplus s \oplus ((s \oplus z) - z)$$

Diese Beziehung ist in leicht abgewandelter Form bereits aus [B10] bekannt. Die Korrektheit der XOR2ADD-Formel ergibt sich aus einer Betrachtung der vermöge $F(x, s) := (x - (x \oplus s)) \bmod 2^n$ definierten Funktion. Für die oben genannte maskierte Repräsentation x des geheim zu haltenden Basiswertes d und die oben genannte boolesche Maske s stellt der Wert $F(x, s)$ die gesuchte additive Maske r dar, weil $d + F(x, s) = d + (x - (x \oplus s)) = d + x - d = x$ gilt. Für die Abbildung $F(\_, s)$ gilt: $F(x \oplus y, s) = F(x, s) \oplus F(y, s) \oplus F(0, s)$. Somit ergibt sich:

$$\begin{aligned} r &= F(x, s) \\ &= F((x \oplus z) \oplus (z \oplus s) \oplus s, s) \\ &= F(x \oplus z, s) \oplus F(z \oplus s, s) \oplus F(s, s) \\ &= ((x \oplus z) - (x \oplus z \oplus s)) \oplus ((z \oplus s) - z) \oplus s \end{aligned}$$

Die XOR2ADD-Formel folgt, indem in der gerade abgeleiteten Beziehung für r der Wert x durch $d \oplus s$ ersetzt wird.

Die gerade genannte XOR2ADD-Formel lässt sich dadurch erweitern, dass nicht nur ein zufälliger Verschleierungswert z, sondern zwei Verschleierungswerte z_1 und z_2 herangezogen werden, um die Berechnung auch gegen Seitenkanalangriffe zweiter Ordnung zu schützen. Unter Verwendung von z_1, z_2 ergibt sich die arithmetische Maske r dann durch die folgende Beziehung, die hier als "erweiterte XOR2ADD-Formel" bezeichnet wird:

$$r = ((d \oplus s \oplus z_1 \oplus z_2) - (d \oplus z_1 \oplus z_2)) \oplus ((s \oplus z_1) - z_1) \oplus ((s \oplus z_2) - z_2)$$

3.2 Mathematische Herleitung des Maskenwechsels

In dem in Abbildung 1 gezeigten Ausführungsbeispiel des Maskierungsübergangsverfahrens 10 wird hierzu zunächst in Schritt 24 die erste boolesche Maske s_1 zufällig gewählt und dann in einem Verfahren, das den wiederholt ausgeführten Abschnitt 22 aufweist, die zweite boolesche Maske s_2 geeignet bestimmt. Hierzu werden vier Beziehungen (A), (B), (C) und (D) ausgewertet, die im folgenden beschrieben und erläutert werden. Gemäß den XOR2ADD-Formeln werden in diesen Beziehungen zufällig gewählte Verschleierungswerte z_1, z_2, z_3, z_4 und z_5 verwendet.

Die erste Beziehung (A) berechnet eine additive Maske r, die der zufällig gewählten booleschen Maske s_1 entspricht, also für die $(d \oplus s_1) \oplus s_2 = (d + r) \oplus s_2 = x$ gilt. Gemäß der erweiterten XOR2ADD-Formel ließe sich eine solche additive Maske r durch die folgende Beziehung (A0) bestimmen:

$$r = (((d \oplus s_1) \oplus z_1 \oplus z_2) - (d \oplus z_1 \oplus z_2)) \oplus ((s_1 \oplus z_1) - z_1) \oplus ((s_1 \oplus z_2) - z_2)$$

Es ist jedoch leicht zu validieren, dass der Wert r nicht unmittelbar berechnet werden darf, weil die Kenntnis von r zusammen mit der Kenntnis von s_1 Rückschlüsse auf den geheim zu haltenden Wert d zulassen würde. Eine solche Berechnung wäre somit gegen einen Seitenkanalangriff zweiter Ordnung anfällig. Es wird daher nicht der Wert r berechnet, sondern ein abgewandelter Wert r_{1a}, der im folgenden als "erster additiver Wert" bezeichnet wird. Der erste additive Wert r_{1a} ist gegenüber dem Wert r mit einem als booleschen Maske dienenden Verschleierungswert a, der in Schritt 26 zufällig gewählt wird, maskiert. Somit gilt $r_{1a} = r \oplus a$, und durch Einsetzen in die obige Beziehung (A0) ergibt sich die folgende Beziehung (A1):

$$r_{1a} = (((d \oplus s_1) \oplus z_1 \oplus z_2) - (d \oplus z_1 \oplus z_2)) \oplus ((s_1 \oplus z_1) - z_1) \oplus ((s_1 \oplus z_2) - z_2) \oplus a$$

Der geheim zu haltende Wert d steht für die Berechnung natürlich nicht zur Verfügung. Weil $x = d \oplus s_1 \oplus s_2$ und somit $d = x \oplus s_1 \oplus s_2$ gelten, lässt sich die Beziehung (A1) auch als die folgende Beziehung (A) schreiben:

$$r_{1a} = ((x \oplus s_2 \oplus z_1 \oplus z_2) - (x \oplus s_1 \oplus s_2 \oplus z_1 \oplus z_2)) \oplus ((s_1 \oplus z_1) - z_1) \oplus ((s_1 \oplus z_2) - z_2) \oplus a$$

Diese Beziehung (A) wird in dem ersten Maskierungsübergang 16 ausgewertet. Da in der Beziehung (A) die zu berechnende boolesche Maske s_2 auftaucht, ist der erste Maskierungsübergang 16 Teil des wiederholt ausgeführten Abschnitts 22 und wird, verzahnt mit dem zweiten Maskierungsübergang 18 sowie weiteren Berechnungen, mit zunehmend längeren Bitsequenzen ausgeführt. Dies ist in Abbildung 1 durch den strichpunktierten Pfeil 30 und die weiteren strichpunktierten Pfeile angedeutet.

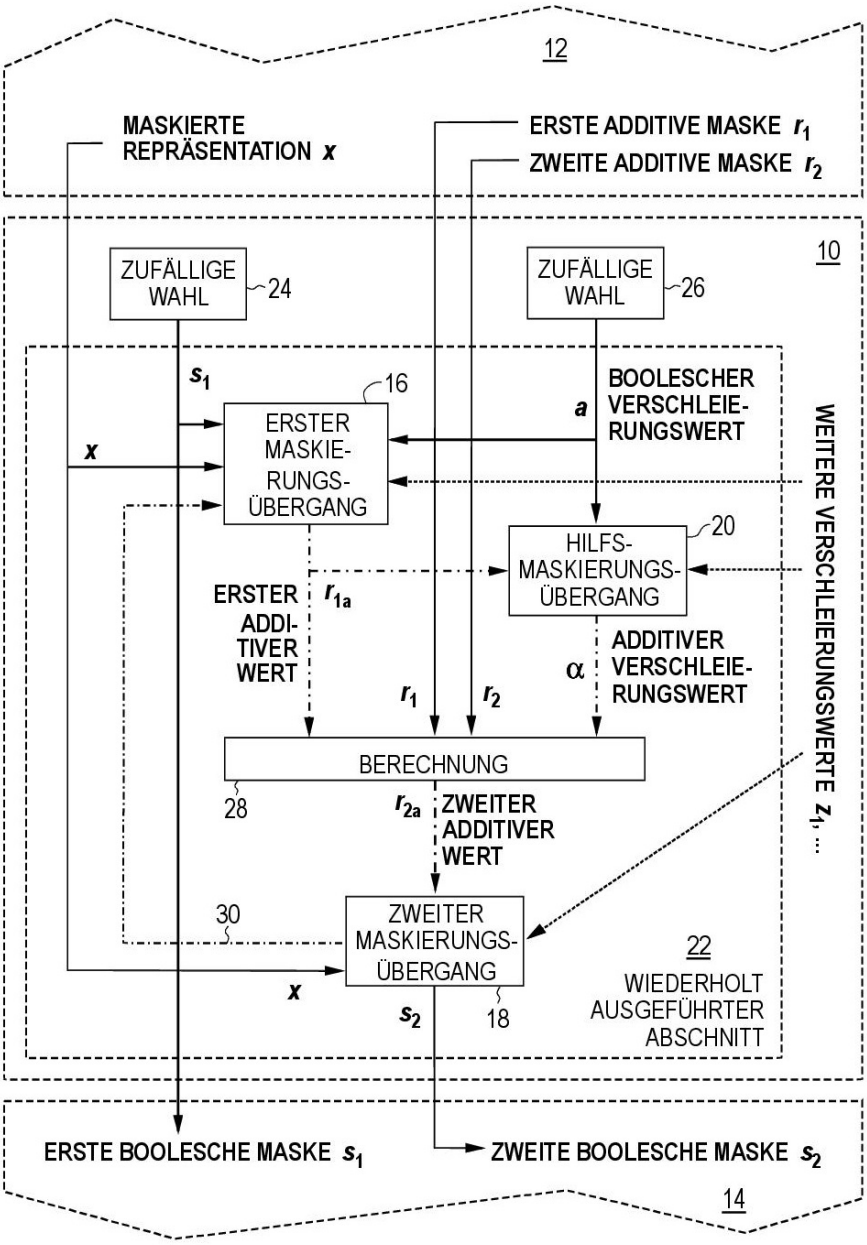

Abbildung 1: Diagramm zur Veranschaulichung des vorgestellten gegen Seitenkanalangriffe zweiter Ordnung abgesicherten Maskenwechsels von einer additiven zu einer booleschen Maskierung

Um im weiteren Verfahrensverlauf mit additiven Operationen arbeiten zu können, wird der boolesche Verschleierungswert *a* in dem Hilfs-Maskierungsübergang 20 in einen

entsprechenden additiven Verschleierungswert α umgewandelt. Aus der für die Maske α gewünschten Beziehung $r \oplus a = r + \alpha$ ergibt sich vermöge der einfachen XOR2ADD-Formel die folgende Beziehung (B0):

$$\alpha = (((r \oplus a) \oplus z_3) - (r \oplus z_3)) \oplus a \oplus ((a \oplus z_3) - z_3)$$

Da der Wert r nicht zur Verfügung steht, wird er in der tatsächlich durchgeführten Berechnung durch den ersten additiven Wert r_{1a} ersetzt, der ja durch $r_{1a} = r \oplus a$ definiert wurde. Indem $r = r_{1a} \oplus a$ in die obige Beziehung (B0) eingesetzt wird, ergibt sich die folgende Beziehung (B), die in Schritt 20 ausgewertet wird:

$$\alpha = ((r_{1a} \oplus z_3) - (r_{1a} \oplus a \oplus z_3)) \oplus a \oplus ((a \oplus z_3) - z_3)$$

Die maskierte Repräsentation x ist mit den zwei gegebenen additiven Masken r_1, r_2 maskiert, so dass $x = d + r_1 + r_2$ gilt. Das hier beschriebene Maskierungsübergangsverfahren 10 geht von der Idee aus, die additive Maskierung neu zu "verteilen", nämlich in den Wert r, der seinerseits der bereits in Schritt 24 zufällig gewählten Maske s_1 entspricht, und einen zweiten additiven Wert r_{2a}, aus dem dann die zu ermittelnde Maske s_2 berechnet werden kann. Mit anderen Worten soll die additive Maskierung nicht als $r_1 + r_2$, sondern in der Form $r + r_{2a}$ für ein geeignetes r_{2a} ausgedrückt werden, so dass $r_1 + r_2 = r + r_{2a}$ gilt. Somit ergibt sich die folgende Beziehung (C0) für r_{2a}:

$$r_{2a} = r_2 - (r - r_1).$$

Da wiederum der Wert r nicht zur Verfügung steht, wird $r = r_{1a} - a$ in die obige Beziehung (C0) eingesetzt. Dadurch ergibt sich die folgende Beziehung (C), die in Schritt 28 ausgewertet wird.

$$\begin{aligned} r_{2a} &= r_2 - (r_{1a} - \alpha - r_1) \\ &= r_2 - (r_{1a} - r_1) + \alpha \end{aligned}$$

Der zweite additive Wert r_{2a} stellt in gewisser Weise die "passende Ergänzung" zu der zufällig gewählten ersten booleschen Maske s_1 dar, die dann auch der gesuchten booleschen Maske s_2 entspricht. Dies lässt sich durch die folgende Gleichungskette veranschaulichen:

$$x = d + r_1 + r_2 = (d + r) + r_{2a} = (d + r) \oplus s_2 = (d \oplus s_1) \oplus s_2$$

In dem zweiten Maskierungsübergang 18 wird der zweite additive Wert r_{2a} in eine entsprechende boolesche Maske, nämlich die gesuchte zweite boolesche Maske s_2, umgewandelt. Aus der erweiterten XOR2ADD-Formel ergibt sich hierfür die folgende Beziehung (D0):

$$r_{2a} = (((d+r) \oplus s_2 \oplus z_4 \oplus z_5) - ((d+r) \oplus z_4 \oplus z_5)) \oplus ((s_2 \oplus z_4) - z_4) \oplus ((s_2 \oplus z_5) - z_5)$$

Der geheim zu haltende Wert d und der Wert r stehen nicht zur Verfügung, aber durch Einsetzung von $x = (d + r) \oplus s_2$ erhält man aus (D0) die folgende Beziehung (D1):

$$r_{2a} = ((x \oplus z_4 \oplus z_5) - ((x \oplus s_2) \oplus z_4 \oplus z_5)) \oplus ((s_2 \oplus z_4) - z_4) \oplus ((s_2 \oplus z_5) - z_5)$$

Da der Wert r_{2a} bekannt ist (bzw. in der Operation 28 schrittweise berechnet wird), wird die Beziehung (D1) teilweise nach s_2 aufgelöst, so dass sich die folgende Beziehung (D) ergibt:

$$s_2 = ((((x \oplus z_4 \oplus z_5) - ((x \oplus s_2) \oplus z_4 \oplus z_5)) \oplus ((s_2 \oplus z_4) - z_4) \oplus r_{2a}) + z_5) \oplus z_5$$

In dem wiederholt ausgeführten Abschnitt 22 werden die Beziehungen (A), (B), (C) und (D) iterativ, zunächst für das niederwertigste Bit und dann für je ein weiteres Bit, berechnet. Eine solche iterative Ausführung ist erforderlich, weil sich die Gleichung (D) nicht nach dem gesuchten Wert s_2 auflösen läßt.

Durch die Ausführung des wiederholt ausgeführten Abschnitts 22 zunächst für das niederwertigste Bit und dann für immer genau ein weiteres Bit werden sowohl die Korrektheit des Endergebnisses als auch die Resistenz gegenüber Seitenkanalangriffen zweiter Ordnung sichergestellt.

Insgesamt werden somit die beiden booleschen Masken s_1 und s_2 ermittelt, die – zusammen mit dem unveränderten Wert x – das Ergebnis des Maskierungsübergangsverfahrens bilden.

Ferner werden im oben beschriebenen Ausführungsbeispiel neben dem booleschen Verschleierungswert a bzw. dem entsprechenden additiven Verschleierungswert α fünf weitere Verschleierungswerte z_1, z_2, z_3, z_4 und z_5 eingesetzt. In einer einfachen aber nicht maximal effizienten Implementierung können für diese Werte voneinander unabhängige Zufallszahlen verwendet werden.

3.3 Optimierte, effiziente Implementierung des Maskenwechsels

Im folgenden wird eine optimierte schrittweise Implementierung des oben beschriebenen Ausführungsbeispiels des Maskierungsübergangsverfahrens 10 dargestellt, bei der die Anzahl der voneinander unabhängigen Zufallszahlen für die Maskierungswerte z_1, z_2, ... auf drei reduziert wurde. Diese Implementierung geht von den Eingangswerten `x`, `r1` und `r2` aus,

wobei r1 und r2 additive Masken sind und x eine maskierte Repräsentation eines geheim zu haltenden Basiswertes d ist. Somit gilt x = d + r1 + r2. Es werden dann die folgenden Verfahrensschritte ausgeführt, wobei "bitlength(x)" die Anzahl der Bits ("Bitlänge") des Registers oder Speicherbereichs angibt, in dem sich der Wert x befindet:

Wähle zufällige Werte z1, z2, z3, a und s1			(Schritt X1)
A	←	x ⊕ z1	(Schritt X2)
B	←	x ⊕ z2	(Schritt X3)
C	←	B ⊕ z1	(Schritt X4)
z13	←	z1 ⊕ z3	(Schritt X5)
s1z2	←	s1 ⊕ z2	(Schritt X6)
s1z3	←	s1 ⊕ z3	(Schritt X7)
s12	←	s1z2 - z2	(Schritt X8)
s13	←	s1z3 - z3	(Schritt X9)
ar	←	r1 ⊕ a	(Schritt X10)
s13ar	←	s13 ⊕ ar	(Schritt X11)
s123ar	←	s13ar ⊕ s12	(Schritt X12)
b1	←	ar - r1	(Schritt X13)
b1a	←	b1 ⊕ a	(Schritt X14)
i	←	1	(Schritt X15)
s1z2r1	←	(s1z2 ⊕ r1) mod (1<<i)	(Schritt X16)
s2z2	←	(r2 ⊕ s1z2r1) mod (1<<i)	(Schritt X17)
Solange (i ≠ bitlength(x)) führe aus:			(Schritt X18)
i	←	i + 1	(Schritt X18a)
a1	←	(A ⊕ s2z2) mod (1<<i)	(Schritt X18b)
a2	←	(a1 ⊕ z13) mod (1<<i)	(Schritt X18c)
a3	←	(a2 ⊕ s1) mod (1<<i)	(Schritt X18d)
a4	←	(a2 - a3) mod (1<<i)	(Schritt X18e)
r1ar	←	(a4 ⊕ s123ar) mod (1<<i)	(Schritt X18f)
r1a	←	(r1ar ⊕ r1) mod (1<<i)	(Schritt X18g)
r1r	←	(r1a ⊕ ar) mod (1<<i)	(Schritt X18h)
b2	←	(r1ar - r1r) mod (1<<i)	(Schritt X18i)

```
alpha ← (b1a ⊕ b2) mod (1<<i)        (Schritt X18j)
tmp   ← (r1a - r1) mod (1<<i)        (Schritt X18k)
r2tmp ← (r2 - tmp) mod (1<<i)        (Schritt X18l)
r2a   ← (r2tmp + alpha) mod (1<<i)   (Schritt X18m)
c1    ← (C - a1) mod (1<<i)          (Schritt X18n)
c2    ← (a1 ⊕ B) mod (1<<i)          (Schritt X18o)
c3    ← (c2 - z1) mod (1<<i)         (Schritt X18p)
c4    ← (c1 ⊕ r2a) mod (1<<i)        (Schritt X18q)
c5    ← (c4 ⊕ c3) mod (1<<i)         (Schritt X18r)
s2z2  ← (c5 + z2) mod (1<<i)         (Schritt X18s)
```

Als Ergebnis erhält man die beiden booleschen Masken `s1z2` und `s2z2`. Es gilt:

$$x = d + r1 + r2 = d \oplus s1z2 \oplus s2z2$$

In der obigen Implementierung wird der erste Maskierungsübergang 16 in Schritt X18g mit der Berechnung von `r1a` abgeschlossen; der Hilfs-Maskierungsübergang 20 wird in Schritt X18j mit der Berechnung von alpha abgeschlossen; die Operation 28 wird in Schritt X18m mit der Berechnung von r2a abgeschlossen; und der zweite Maskierungsübergang 18 wird in Schritt X18s mit der Berechnung von s2z2 abgeschlossen. In der Implementierung wurden diverse Optimierungen vorgenommen. Insbesondere wurden Berechnungen möglichst weitgehend aus der Schleife herausgezogen. Die Anzahl der weiteren Verschleierungswerte wurde auf drei, nämlich z1, z2 und z3, reduziert.

Die Berechnung innerhalb der Schleife erfolgt für jeweils die i niederwertigsten Bits, wobei i mit dem Wert 2 beginnt und bei jedem Schleifendurchlauf um 1 erhöht wird. Dies wird in der obigen Implementierung durch Operationen der Form y mod (1<<i) dargestellt, wobei 1<<i den Wert 2^i darstellt und daher *y* mod (1<<i) den *i* niederwertigsten Bits von *y* (mit auf Null gesetzten höherwertigen Bits) entspricht.

Die gerade beschriebene Implementierung benötigt 18 * bitlength(x) – 3 elementare Operationen. Sie ist für beliebige Bitlängen von Daten anwendbar. Durch numerische Simulation für alle Eingangswerte bis zu einer bestimmten vorgegebenen Bitlänge wurde nachgewiesen, dass die Berechnung zum gewünschten Ergebnis konvergiert und gegen Seitenkanalangriffe zweiter Ordnung resistent ist.

Die Kontrolle der Sicherheit der Implementierung muß jedoch auf der eingesetzten Hardware erfolgen, da die in der Simulation geforderte Unabhängigkeit der einzelnen Rechenschritte auf einer Hardware nicht unbedingt gegeben sein muß.

4 Zusammenfassung

Wenn in dem Hyper-Ledger-Framework ein PKCS#11 kompatibles Sicherheitsdevice wie eine Smartcard eingesetzt wird, so ergibt sich die Notwendigkeit, daß zum Schutz der auf der Karte gespeicherten und eingesetzten Schlüssel die Implementierungen gegen Seitenkanalangriffe geschützt werden müssen. Dies bedeutet, daß auch die für die Schlüsselableitungen eingesetzten Hashfunktionen sicher zu implementieren sind. Hierbei ergibt sich die Schwierigkeit, daß unterschiedliche Operatoren hintereinander ausgeführt werden müssen, die eine unterschiedliche Art der Maskierung zum Schutz der Daten benötigen. Insbesondere sind Maskenwechsel von einer booleschen zu einer arithmetischen Maskierung und umgekehrt notwendig. In der veröffentlichten Literatur finden sich jedoch für den Maskenwechsel von einer arithmetischen zu einer booleschen Maskierung nur Verfahren, die gegen Seitenkanalangriffe erster Ordnung abgesichert sind.

In dieser Veröffentlichung wurde das erste Verfahren für einen Maskenwechsel einer arithmetischen Maskierung zu einer booleschen Maskierung vorgestellt, das gegen Seitenkanalangriffe zweiter Ordnung geschützt ist. Dies schließt die Lücke für eine effiziente und gegen Seitenkanalangriffe zweiter Ordnung resistente Implementierung von Hashfunktionen oder anderen kryptographischen Algorithmen, die im Hyper-Ledger-Framework eingesetzt werden.

Literatur

[B10] EP 1 596 527 B1, S. Bauer, “Switching from Boolean to arithmetric masking”, 2010

[BCZ18] L. Bettale, J.-S. Coron, R. Zeitoun, “Improved High-Order Conversion From Boolean to Arithmetic Masking”, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 22-45, 10.46586/tches.v2018.i2.22-45, 2018

[CRR02] S. Chari, J. R. Rao, P. Rohatgi, “Template attacks”, in Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2002, Springer Verlag 2002, LNCS XXX, pp. XXX-XXX

[D12] B. Debraize, „Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking”, in Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2012, Springer Verlag 2012, LNCS 7428, pp. 107 – 121

[G01] L. Goubin, „A Sound Method for Switching between Boolean and Arithmetic Masking“, in Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2001, Springer Verlag 2001, LNCS 2162, pp. 3-15

[HGM+11] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, “Machine learning in side-channel analysis: a first study”, in J. Cryptographic Engineering, 1(4): pp. 293–302, 2011.

[HZ12] A. Heuser and M. Zohner, „Intelligent machine homicide - breaking cryptographic devices using support vector machines”, in W. Schindler and S. A. Huss, editors, Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings, volume 7275 of Lecture Notes in Computer Science, pp. 249–264. Springer, 2012.

[JPS05] M. Joye, P. Paillier, B. Schoenmakers, “On Second-Order Differential Power Analysis”, in Cryptographic Hardware and Embedded Systems – CHES 2005, Springer, 2005, pp. 293-308.

[KJJ99] P. Kocher, J. Jaffe and B. Jun, “Differential Power Analysis“, in Proceedings of Advanced Cryptology – CRYPTO´99, Springer, 1999, pp. 388-397.

[MOP07] S. Mangard, E. Oswald, T. Popp, “Power Analysis Attacks – Revealing the Secrets of Smart Cards“, Springer, 2007.

[NP10] EP 1 664 979 B1, O. Neisse, J. Pulkus, “Transition between masked representations of a value during cryptographic calculations”, 2010

[PRB09] E. Prouff, M. Rivain, R. Bevan, “Statistical Analysis of Second Order DPA”, in IEEE Transactions on Computers, vol 58, num 6, pp 799-811, June 2009.

[PSB+18] E. Prouff and R. Strullu and R. Benadjila and E. Cagli and C. Dumas, “Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database”, Cryptology ePrint Archive, Report 2018/053, https://eprint.iacr.org/2018/053, 2018