

Technische Universität Darmstadt

Fachbereich Informatik
Fachgebiet Theoretische Informatik

Prof. Dr. Johannes Buchmann



Masterarbeit

GMSS Signatur Generation für RFID Tags

Bearbeiter: Musab Haj Stifi
Prüfer: Prof. Dr. Johannes Buchmann
Betreuer: Dipl.-Math. Erik Dahmen
Abgabe: 15.08.2007

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Masterarbeit beigetragen haben.

Bedanken möchte ich mich besonders bei Herrn Prof. Dr. Johannes Buchmann und Herrn Erik Dahmen für die hervorragende Betreuung meiner Masterarbeit und für die zahlreichen wissenschaftlichen Ratschläge, welche stets zur Verbesserung der Arbeit beigetragen haben.

Und nicht zuletzt bei meinen Eltern für ihre Liebe, ihr Vertrauen und den bedingungslosen Rückhalt, den ich immer wieder bei ihnen finden konnte. Vielen Danke dafür!

Erklärung

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbständig, und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet.

Darmstadt, den 15.08. 2007

.....

Musab Haj Stifi

Abstrakt

Dieser Artikel stellt einen Antrag für einen Authentifikation Algorithmus für *Radio Frequency Identification (RFID)*-Tags dar. *RFID*-Tags sind die Mikrochips, die einen kleinen Mikroprozessor und einen Speicher EEPROM mit maximum 4 Kbyte haben, die an den Produkten angebracht werden, um sie kontaktlos während der Produktion oder gebräuchlich über Hochfrequenz zu kennzeichnen. Kryptographie Authentifikation ist notwendig, um die eingebrannten Waren vor Fälschung zu schützen. Vorhandenes digital Signatur Schema wie (*RSA*), (*DSA*) und (*ECDSA*) werden in andere Applikationen verwendet. Die Sicherheit dieser Verfahren beruht auf der Schwierigkeit des *Faktors large composite integers* und *computing discrete logarithms*. Man hofft, dass diese Berechnungsprobleme in Zukunft gelöst werden können. Bei der Implementation des *RSA*-Algorithmus muss beachtet werden, dass zum Teil große Anforderungen an die Hardware gestellt werden, da das Finden von großen Primzahlen sehr komplex ist und der Euklidische Algorithmus zur Bestimmung des $\text{ggT}()$ und das Potenzieren von $\text{mod } n$ notwendig ist. Um diesen Algorithmus auch auf schwächere Hardware wie ein *RFID*-Tag zu implementieren, werden zum Beispiel die Schlüssel extern erzeugt und dann eingespielt und/oder nur eine digitale Prüfsumme signiert.

Um der Notwendigkeit einer schnellen, niedrigen Berechnung, einer rationellen Speicherung sowie der Authentifikation Lösung für *RFID*-Tags gerecht zu werden, wird ein neues Schema für Authentisierung in dieser Masterarbeit vorgeschlagen. Die begrenzte Rechnerleistung und die niedrige Speicherkapazität, die der *RFID*-Tag hat, ermöglichen es nicht, den Niedrigleistungsanforderungen und der schnellen Berechnung gerecht zu werden. Ein neues digital Signatur Schema, One-Time Signatur Schema (*OTSS*) verwendet. Das neue vorgeschlagene digital Signatur Schema (*GMSS*) "Merkle Signatur Schema" ist eine interessante Alternative zu gut eingerichteten Signatur Schemata wie (*RSA*), (*DSA*), und (*ECDSA*). Es zielt auf schnelle Signatur und Verifizierung bei niedrigem Zeitaufwand und geringer Speicherkapazität. Die Sicherheit von (*GMSS*) beruht auf der kryptographisch sicheren *One-Way Funktion*, die entworfen ist, um die folgenden Eigenschaften zu haben:

Es ist rechnerisch unmöglich, zwei unterschiedlichen Nachrichten zu finden, die die gleiche Nachricht digest produzieren. *One-Way Funktionen* sind einfacher auf schwächerer Hardware wie *RFID*-Tags einzuführen und sind gewöhnlich rechnerisch leistungsfähiger als (*RSA*) und (*ECDSA*).

Inhaltsverzeichnis

1	Einführung	1
1.1	Motivation	1
1.2	Entwicklungsgeschichte	2
1.3	Gliederung der Arbeit	2
2	RFID-System	3
2.1	Komponenten eines RFID-Systems	3
2.2	Funktionsweise	3
2.2.1	Energieversorgung	4
2.2.2	Sendefrequenz	5
2.2.3	Speicherstruktur und Leistungsfähigkeit	6
2.3	Datenübertragung, Packet und Frame formats-Overhead	8
2.3.1	ISO/IEC 18000 Standard	8
2.3.2	IEEE 802.3 Protokoll für RFID-Tags	9
2.3.3	IEEE 802.11 Protokoll für RFID-Tags	10
2.4	Anwendung RFID-System	11
2.4.1	Logistik	12
3	Kryptographische Maßnahmen für RFID-System	13
3.1	Einleitung	13
3.2	Kryptographische Theorie	13
3.2.1	Ziele der Kryptographie	13
3.3	Das Konzept der digitalen Signaturen	14
3.4	Authentikationsverfahren	15
3.4.1	Symmetrische Authentikationsverfahren	15
3.4.2	Asymmetrische Authentikationsverfahren	16
3.5	Sicherheit von RFID-Systemen	16
3.5.1	Angriffe auf RFID-Systeme	17
3.6	Sicherheitsverfahren für die RFID-Systeme	19
3.6.1	Vertraulichkeit der Informationen	19
3.6.2	Authentifizierung der Informationen des Tags	19
3.6.3	Integrität der Informationen	20
3.7	Kryptographie Implementation	20
3.8	Symmetrische Authentifizierung	20
3.8.1	Get-Challenge-Response	20
3.9	Asymmetrische Authentifizierung	22

Inhaltsverzeichnis

3.9.1	RSA Verfahren	22
3.9.2	RSA Signatur	22
3.9.3	RSA Verifikation	23
3.10	Hash-Funktion	24
3.10.1	Secure Hash Algorithm SHA-1	25
4	MSS Merkle Signatur Schema	27
4.1	One-Time Signatur Schema OTSS	27
4.1.1	Lamport One-Time Signatur Schema	28
4.1.2	Winternitz One-Time Signatur Schema	30
4.2	Merkle Signatur Schema	32
4.2.1	MSS Schlüsselpaare Generation	33
4.2.2	MSS Signatur Generation	34
4.2.3	MSS Signatur Verifikation	35
4.3	GMSS Merkle Signatur Schema	35
4.3.1	GMSS Schlüsselpaare Generation	40
4.3.2	GMSS Signatur Generation	40
4.3.3	GMSS Signatur Verifikation	40
5	GMSS Signatur Generation für RFID-Tags	42
5.1	Einleitung	42
5.2	Eigenschaften des RFID-Tags für GMSS	42
5.3	Optimal Parameter group	43
5.4	Auswahl der Hash-Funktion	43
5.5	Auswahl der One-Time Signatur	43
5.6	GMSS-Baum Parameter	45
5.7	Implementation GMSS Signatur auf RFID-Tag	47
5.7.1	Anforderungsspeicher EEPROM für GMSS-Baum	47
5.7.2	Anforderungsspeicher EEPROM um einen GMSS-Baum zu bilden	49
5.7.3	Anforderungsspeicher EEPROM für Signatur Generation	53
5.8	Merkle-Offline-Baum Generation	54
5.9	GMSS-Baum Parameter	65
5.10	Implementation GMSS Signatur auf RFID-Tag	68
5.10.1	Anforderungsspeicher EEPROM für GMSS-Baum	68
5.10.2	Anforderungsspeicher EEPROM um einen GMSS-Baum zu bilden	70
5.10.3	Anforderungsspeicher EEPROM für Signatur Generation	72
5.11	Berechnung der Authentifizierungspfade	75
6	Zusammenfassung	86

Abbildungsverzeichnis

2.1	Komponenten eines RFID-Systems	3
2.2	RFID-Systems	4
2.3	RFID-Frequenzbänder	5
2.4	RFID-Tag	6
2.5	RFID-Tag Leistungsfähigkeit	7
2.6	Erzeugung des Sendesignals	11
2.7	Anwendungsbereiche RFID-Technologie	12
2.8	Anwendung RFID-Sytem in Logistik	12
3.1	Signierungsvorgang	15
3.2	Asymmetrische-Authentikation	16
3.3	Angriffsarten eines RFID Systems bei der Kommunikation	17
3.4	Ablauf einer Authentifizierung zwischen Tag und Lesegerät	21
3.5	Block Diagram für RSA	23
3.6	SHA-1 mit MSS	25
4.1	Merkle-Baum	34
4.2	Merkle-Authentifizierungspfad	35
4.3	GMSS-Baum	36
4.4	Konstruktion-GMSS	37
4.5	GMSS Signatur	38
4.6	GMSS-Baum nach 2^3 Signaturen	38
4.7	GMSS-Authentifizierungspfad	39
5.1	Beziehung zwischen w, t_w , Signaturgröße, Signaturzeit	44
5.2	GMSS-Baum	45
5.3	GMSS-Baum für RFID-Tag	46
5.4	Merkle-Baum $\tau_{1,0}$ mit allen Blöcken	47
5.5	Merkle-Baum $\tau_{2,0}$ mit allen Blöcken	48
5.6	Merkle-Online-Baum $\tau_{2,0}$, Merkle-Offline-Baum $\tau_{2,1}$	50
5.7	Anwendung PRNG	51
5.8	Berechnung eines Merkle Knotens	52
5.9	Authentifizierungsbaum für d_1	55
5.10	Authentifizierungsbaum für d_2	56
5.11	Authentifizierungsbaum für d_3	57
5.12	Authentifizierungsbaum für d_4	57
5.13	Authentifizierungsbaum für d_5	58

Abbildungsverzeichnis

5.14	Authentifizierungsbaum für d_6	59
5.15	Authentifizierungsbaum für d_7	60
5.16	Authentifizierungsbaum für d_8	61
5.17	Berechnung des Merkle-Offline-Baums	62
5.18	Merkle-Offline-Baum	62
5.19	GMSS-Baum	66
5.20	GMSS-Baum für RFID-Tag	67
5.21	Merkle-Baum $\tau_{1,0}$ mit allen Blöcken	68
5.22	Merkle-Baum $\tau_{2,0}$ mit allen Blöcken	69
5.23	Merkle-Baum $\tau_{3,0}$ mit allen Blöcken	70
5.24	Merkle-Online-Baum $\tau_{2,0}$, Merkle-Offline-Baum $\tau_{2,1}$, Merkle-Online-Baum $\tau_{3,0}$, Merkle-Offline-Baum $\tau_{3,1}$	71
5.25	Authentifizierungsbaum für d_1 und die Blöcke	76
5.26	Authentifizierungsbaum für d_2 und die Blöcke	77
5.27	Authentifizierungsbaum für d_3 und die Blöcke	77
5.28	Authentifizierungsbaum für d_4 und die Blöcke	78
5.29	Anwendung der Offline-Baum	79
5.30	Erzeugung eines Offline-Baums	79
5.31	Authentifizierungsbaum für d_5 und die Blöcke	80
5.32	Authentifizierungsbaum für d_6 und die Blöcke	81
5.33	Authentifizierungsbaum für d_7 und die Blöcke	82
5.34	Authentifizierungsbaum für d_8 und die Blöcke	82
5.35	Erzeugung Merkle-Offline-Baum	83
5.36	Erzeugung Merkle-Offline-Baum	83
5.37	Erzeugung Merkle-Offline-Baum	84
5.38	Merkle-Offline-Baum	84

Tabellenverzeichnis

2.1	RFID-Alle frequenzbereich	5
2.2	Frame-Format vom Lesegerät zum RFID-Tag	8
2.3	Frame-Format vom RFID-Tag zum Lesegerät	9
2.4	IEEE 802.3 Frame Format	10
2.5	IEEE 802.11 Frame Format	10
5.1	GMSS One-Time Signatur Parameter	65
5.2	GMSS One-Time Signatur Parameter	75
6.1	GMSS One-Time Signatur Parameter	87
6.2	GMSS One-Time Signatur Parameter	88

1 Einführung

1.1 Motivation

'Auch eine Reise von tausend Meilen beginnt mit einem Schritt.'

Chinesisches Sprichwort

Die Notwendigkeit der Kennzeichnung der verschiedenen Produkte und jede Waren erhöht sich in unserer automatisierten Welt. Jedes der heutigen geschäftlichen Produkte und Ware muss auf seine Weise auf dem Weg vom Produzenten bis zum Endverbraucher gekennzeichnet werden. Die Verwendung eines *Radio Frequency Identification System RFID* ist ein guter Versuch zur automatisierten Kennzeichnung der Produkte. *RFID*-Tags sind die Mikrochips, die an die täglichen Produkte angebracht werden, um sie zu kennzeichnen. Wir verwenden ein neues Signatur Schema mit eigenen Parameter für *RFID*-Tag, das Umkehrkommunikation zwischen einem Lesegerät und einem *RFID*-Tag herstellt. Es ist notwendig, dass ein Hersteller seine Produkte vor Plagiarismus schützen möchte, oder ein Kunde sicher sein möchte, dass seine Ware von einer bestimmten Firma produziert wird. Der beste Weg, Authentisierung auf *RFID*-Tag einzuführen ist, Kryptographie Algorithmus hinzuzufügen. Momentan gibt es kein starkes Verfahren, die kryptographische Authentifikation auf *RFID*-Tag anzuwenden. Diese ist ausgeschlossen wegen der niedrigen Speicherkapazität und der Anforderung des Niedrigleistungsverbrauchs. Traditionelle Verschlüsselungsalgorithmen gelten zu umfangreich, die erfordern neue Verfahren. Das Ziel der Masterarbeit ist, die Umsetzbarkeit der GMSS Merkle Signatur Verfahren auf *RFID*-Tags für Authentifikation, Verschlüsselung und Integrität zu entwickeln, die spezifisch zu dem *RFID*-System umgearbeitet werden. Wir definieren einige Parameter und benutzen einige Hash-Funktionen wie SHA-1, so dass sie auf *RFID*-Tags verwendet werden können. Der *RFID*-Tag hat einen Mikroprozessor, der nur die kleinen Grund-Funktionen wie (\wedge , \vee , \oplus) berechnen kann, und einen kleinen EEPROM mit 4 Kbyte. Der *RFID*-Tag benutzt den EEPROM, um die Variablen des GMSS Merkle Signatur Schema und die Berechnungsoperationen zu speichern.

1.2 Entwicklungsgeschichte

Die *RFID*-Technologie ist eine automatische Identifikationstechnologie, bei der eine Information, auf einem *RFID*-Tag gespeichert wird, welcher einen Mikrochip besitzt und als elektronischer Datenspeicher dient. Die Informationen können mittels drahtloser Kommunikation, über eine Distanz von einigen Metern, von einem Lesegerät ausgelesen werden. Die Stärken von *RFID*, speziell gegenüber dem Barcode, liegen in der vollautomatischen, gleichzeitigen Erkennung mehrerer *RFID*-Tag, wobei keine Sichtverbindung zwischen Lesegerät und *RFID*-Tag nötig ist. Informationen können auf einem *RFID*-Tag mit Datenspeicher während des Einsatzes verändert werden, was bei einem Barcode nicht möglich ist.

1.3 Gliederung der Arbeit

Die Arbeit lässt sich in vier Hauptkapitel gliedern.

Kapitel 2: Das Ziel dieses Kapitels ist es, einen Überblick über die *RFID*-Technik zu liefern. Es handelt sich um die Komponenten eines *RFID*-Systems, die Funktionsweise, die Sendefrequenz, Speicherstruktur, und die Datenübertragung.

Kapitel 3: erklärt die kryptographischen Maßnahmen für *RFID*-Tag. Es handelt sich um die Authentikationsverfahren, Authentifizierung der Information des Tags.

Kapitel 4: Dieses Kapitel soll einen Überblick über das One-Time Signatur Schema geben. Es handelt sich um die Lamport One-Time Signatur, die Winternitz One-Time Signatur, und die Anwendung des GMSS Merkle-Baums.

Kapitel 5: erörtert die Frage der Umsetzbarkeit des GMSS One-Time Signatur Schema auf dem *RFID*-Tag. Hier werden eigene Parameter von der GMSS One-Time Signatur dargestellt. Das Kapitel bietet zwei unterschiedlichen Verfahren, um GMSS auf dem Tag anwenden zu können. Es handelt sich um den Anforderungsspeicher EEPROM für das GMSS One-Time Signatur Schema.

Schließlich werden in der Zusammenfassung alle wichtigen Ergebnisse dieser Masterarbeit aufgezeigt und kritisch beurteilt. ausgezeichnet.

2 RFID-System

2.1 Komponenten eines RFID-Systems

Ein typisches *RFID*-System besteht aus den folgenden drei Komponenten (siehe Abbildung 2.1). Rechner, Lesegerät und *RFID*-Tag (siehe Abbildung 2.2). Das Lesegerät ist über eine serielle Schnittstelle oder Netzwerkverbindung mit dem Rechner verbunden. Die Applikation auf dem Rechner schickt Kommandos und Daten an das Lesegerät und erhält wiederum Antwortdaten vom Lesegerät zurück. Beispiele für Kommandos sind das Auslesen der Identifikationsnummern aller *RFID*-Tags im Lesebereich, oder das Beschreiben eines *RFID*-Tags mit Daten. Die Kommandos werden dann vom Lesegerät kodiert und auf ein magnetisches elektromagnetisches Wechselfeld moduliert, das zusätzlich zu den Daten die *RFID*-Tag mit Energie versorgt. Alle *RFID*-Tags, die sich im Feld des Lesegeräts befinden, empfangen die vom Lesegerät ausgesandten Befehle und Daten und schicken ihre jeweiligen Antwortdaten an das Lesegerät zurück.

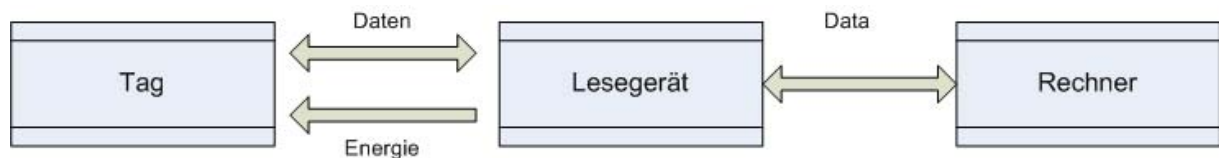


Abbildung 2.1: Komponenten eines RFID-Systems

2.2 Funktionsweise

Die Funktionsweise eines *RFID*-Systems lässt sich durch technische Eigenschaften wie Energieversorgung und Speicherstruktur des *RFID*-Tags, Sendefrequenz des Lesegeräts, und Datenübertragung zwischen Lesegerät und *RFID*-Tag und eingesetztem Zugriffsverfahren beschreiben und klassifizieren [1].

2.2.1 Energieversorgung

RFID-Tag benötigen Energie zum einen, um ihren Mikrochip zu betreiben, und zum anderen, um Daten zum Lesegerät zu senden. Dabei unterscheiden wir die folgenden drei Arten von *RFID*-Tag:

- Passive *RFID*-Tags benutzen die Energie des Feldes, das vom Lesegerät erzeugt wird, sowohl für das Betreiben des Mikrochips als auch zum Senden der Daten.
- Semi-aktive *RFID*-Tags haben eine interne Batterie, mit der sie ihren Mikrochip versorgen. Sie benutzen aber zum Senden der Daten die Energie des Feldes des Lesegeräts.
- Aktive *RFID*-Tags haben eine interne Batterie, die sie für beide Zwecke benutzen.

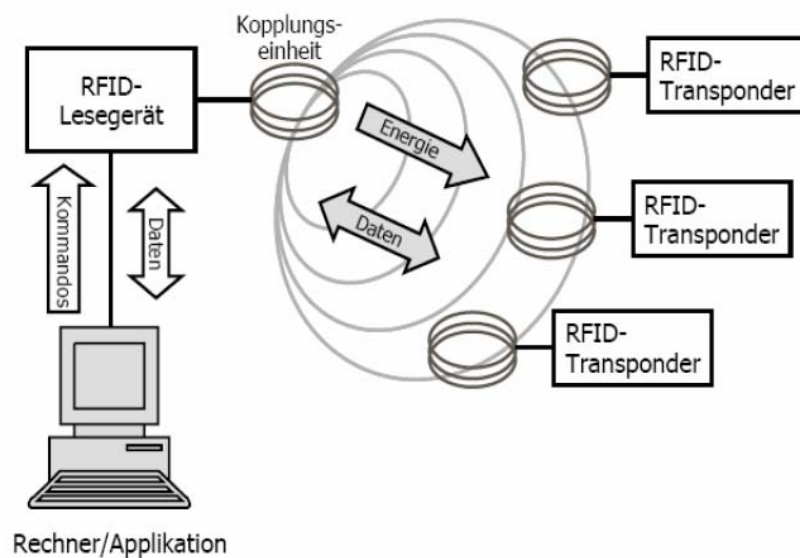


Abbildung 2.2: RFID-Systems

2.2.2 Sendefrequenz

Die Sendefrequenzen der meisten *RFID*-Systeme liegen in den lizenzfreien *ISM* Bändern (Industrial-Scientific-Medical), die für industrielle, wissenschaftliche und medizinische Anwendungen weltweit freigehalten sind. Der Frequenzbereich kommt unterhalb 135 kHz und um 900 MHz. Damit fallen die typischen Sendefrequenzen eines *RFID*-Systems in die folgenden vier Bereiche (siehe Abbildung 2.3) und (Tabelle 2.1).

1. Low Frequency (LF): 30 kHz - 300 kHz.
2. High Frequency (HF): 3 MHz - 30 MHz.
3. Ultra High Frequency (UHF): 300 MHz - 3 GHz.
4. Mikrowellen Frequency (MF): 3 GHz - 5,8 GHz.

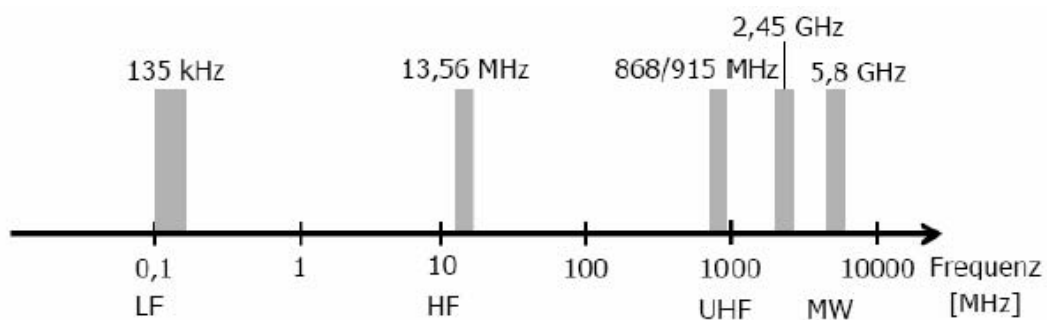


Abbildung 2.3: RFID-Frequenzbänder

RFID-Frequenz	Information
125 KHz (LF)	Passive RFID-Tags
13,56 MHz (HF)	Passive RFID-Tags
868 MHz (UHF)	Aktive und Passive RFID-Tags
2,45 GHz	Aktive RFID-Tags

Tabelle 2.1: RFID-Alle frequenzbereich

2.2.3 Speicherstruktur und Leistungsfähigkeit

Ein zentrales Unterscheidungsmerkmal von *RFID*-Systemen besteht in der jeweils zum Einsatz kommenden Speichertechnologie, wobei grundsätzlich zwischen Read-only- und Read-write-Systemen unterschieden werden kann (siehe Abbildung 2.4):

- Read-only-Tags, die nach dem Programmiervorgang beim Hersteller vom Lesegerät nur gelesen werden können, sind kostengünstiger in der Herstellung. Variable Information, die mit dem Tag assoziiert werden soll, muss in einer Datenbank des *RFID*-Systems abgelegt werden. Beim Auslesen des Tags wird diese Information anhand der *ID-Nummer* *Seriennummer* des Tags aus der Datenbank abgerufen.
- Read-write-Tags sind durch den bereitgestellten Speicher teurer in der Herstellung. Dadurch können leistungsfähige Sicherheitsmechanismen implementiert und auch variable Informationen auf dem Tag selbst neu gespeichert werden. In *RFID*-Systemen kommen die im Folgenden erläuterten ROM, RAM, EEPROM Technologien zum Einsatz.
 1. **ROM:** Ein *Read Only Memory* ist ein digitaler Festwertspeicher, in dem Daten dauerhaft und unveränderlich gespeichert werden. Die Daten werden während der Produktion fest in der Memory abgelegt und können weder elektrisch noch optisch gelöscht oder verändert werden.
 2. **RAM:** Ein *Random Access Memory* wird umgangssprachlich als Arbeitsspeicher bezeichnet. Die Haupteigenschaft eines RAM ist es, den Speicherbaustein mit Daten zu beschreiben.
 3. **EEPROM:** Ein *Electrically Erasable Programmable Read Only Memory* Das Schreiben Lesen können bis zu 10^6 bzw. 10^8 -mal wiederholt werden. Die Daten bleiben ohne Stromzufuhr bis zu zehn Jahren erhalten. Zu den typischen Anwendungen von EEPROM zählen kleine Speicherkarten im *RFID*-Tag. Der private Schlüssel und die Verschlüsselungsparameter werden hier gespeichert.

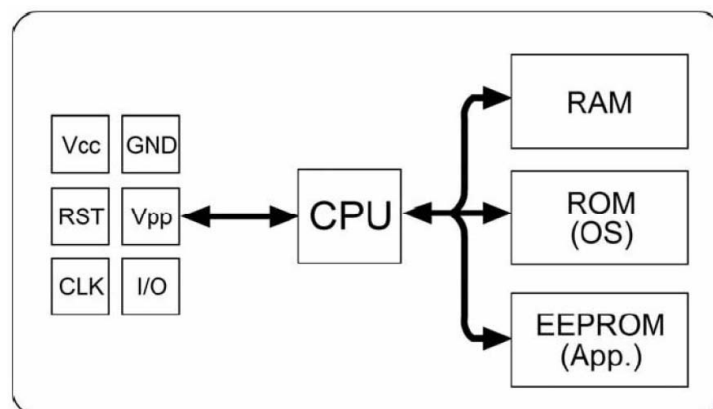


Abbildung 2.4: RFID-Tag

2 RFID-System

RFID-Systeme lassen sich entsprechend ihrer jeweiligen Leistungsmerkmale klassifizieren.

Low-End-Systeme Low-End-*RFID*-Systeme sind so genannte 1-Bit-Systeme, die bereits langjährig für einfache Überwachung genutzt werden. Sie benötigen keine integrierte Schaltung. 1-Bit-Systeme werden zur elektronischen Diebstahlsicherung im Einzelhandel genutzt. Hierfür bedarf es keines Mikroprozessors. Verschlüsselungsfunktionen werden nicht unterstützt.

Systeme mittlerer Leistungsfähigkeit Das Mittelfeld des Leistungsspektrums wird durch *RFID*-Systeme mit wieder beschreibbaren Datenspeichern (beispielsweise EEPROM bei passiven, RAM bei aktiven Tag) von wenigen Byte bis über 100 Kbyte gebildet. Systeme mittlerer Leistungsfähigkeit können sowohl mit einem Zustandsautomaten als auch mit einem Mikroprozessor ausgestattet sein. Vor unberechtigtem Auslesen werden Systeme mittlerer Leistungsfähigkeit durch Authentifizierungs- oder auch Kryptofunktionen geschützt.

High-End-Systeme Im High-End-Bereich finden sich überwiegend kontaktlose Chipkarten mit Mikroprozessor und einem Chipkarten-Betriebssystem (SmartCard OS). Die Karten verfügen über komplexere Algorithmen zur Authentifizierung und Verschlüsselung. Das obere Ende des High-End-Bereichs bilden schließlich Dual-Interface-Karten, welche mit einem kryptografischen Co-Prozessor ausgestattet sind. Die Arbeitsfrequenz liegt typischerweise bei 13,56 MHz, die Reichweite unterhalb von 15 Zentimetern. Die Datenübertragungsrate kann bis zu 848 kBit/s betragen. Mit einem kryptografischen Co-Prozessor werden asymmetrische Verschlüsselungsverfahren unterstützt (siehe Abbildung 2.5).

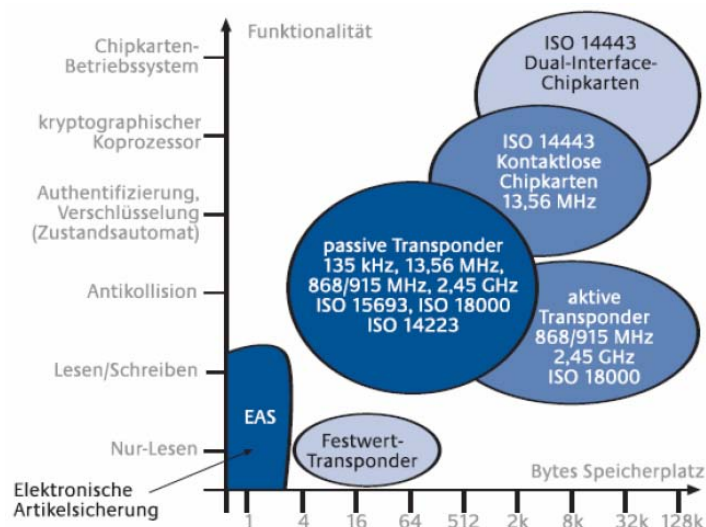


Abbildung 2.5: RFID-Tag Leistungsfähigkeit

2.3 Datenübertragung, Packet und Frame formats-Overhead

Hohe Datenübertragungsraten sind wichtig, falls eine große Datenmenge vom Speicher EEPROM des *RFID*-Tags, der alle Authentikationsdaten hat, in kürzester Zeit gelesen werden soll. Für LF und HF Systeme *ISO-Standards 15693* und *14223* liegt die Datenübertragungsrate bei 5 kbit/s. Allerdings erlauben neuartige *RFID*-Systeme im HF Bereich, die dem *ISO-Standard 18000* entsprechen, Datenraten von über 100 kbit/s. UHF Systeme des *ISO-Standards 18000 Part 6 Mode A* erreichen 50 kbit/s. In dieser Masterarbeit wird ein neues Protokoll benutzt, das für die Datenübertragung zwischen dem Lesegerät und dem *RFID*-Tag verwendet. *IEEE 802.11* liegt die Datenübertragungsrate bei 1Mbit/s. Die Datenübertragungsrate beeinflusst unter anderem auch, wie viele *RFID*-Tag pro Sekunde erkannt werden können.

2.3.1 ISO/IEC 18000 Standard

Der *ISO/IEC 18000-3 Standard* beschreibt die Kommunikation des *RFID*-Tags mit einem Lesegerät. Die Frequenz beträgt 13.56 MHz. Modulation, Antikollisionsmethoden, Protokollparameter und andere spezifische Informationen werden innen [2] dargestellt. Die Kommunikation zwischen dem Lesegerät und den *RFID*-Tag erfolgt über die Modulation. Das Protokoll definiert, wie man Anweisungen und Daten zwischen dem Lesegerät und dem *RFID*-Tag in beiden Richtungen austauscht. Jeder Befehl geht über einem Antrag vom Lesegerät zum Tag und aus einer Antwort vom Tag zum Lesegerät. Das Lesegerät sendet an das *RFID*-Tag eine Nachricht, welche dann vom Tag signiert wird und es die Authentikationsdaten bestimmt. Später werden diese an das Lesegerät zurückgesendet. Anträge und Antworten werden innerhalb eines Rahmens mit den Begrenzungen *Start of Frame SOF* und *End of Frame EOF* enthalten. Das Protokoll bestimmt die Größe der Information, die in einen Rahmen übertragen werden. Jeder Antrag und jede Antwort besteht aus einem Abschnitt wie in der (Tabelle 2.2) angezeigt wird.

SOF	Flags	0XA0	IC Mfg code	UID	Data	CRC	EOF
16 bit	8 bit	8 bit	8 bit	64 bit	128 bit	16 bit	16 bit

Tabelle 2.2: Frame-Format vom Lesegerät zum RFID-Tag

- SOF: Start Of Frame.
- Flags: Anzeigen, ob ein oder zwei Trägerfrequenzen und welche Datenrate für die Antwort verwendet werden sollte.
- 0XA0: Command Code; dieser Befehl ist ein kundenspezifischer Befehl, der durch einen Hersteller eingeführt wird.

2 RFID-System

- IC Mfg Code: IC manufacturer code.
- UID: Unique Identifier, wird verwendet, um ein spezifisches Tag zu adressieren.
- Data: ist die Größe der Nachricht.
- CRC: Cyclic Redundancy Check, wird verwendet, um errors während der Transmission zu ermitteln.
- EOF: End Of Frame.

Die Antwort des Tags ist die Information, die in einen Frame gesendet wird. Tabelle 2.3 zeigt die Teile dieses Frames an.

SOF	Flags	UID	Data	CRC	EOF
16 bit	8 bit	64 bit	128 bit	16 bit	16 bit

Tabelle 2.3: Frame-Format vom RFID-Tag zum Lesegerät

2.3.2 IEEE 802.3 Protokoll für RFID-Tags

Die Protokolle *ISO/IEC 18000*, die für die Übertragung der Authentikationsdaten verwendet werden, *Challenge-Response Protocols*. In *Challenge-Response Protocols* schickt das Lesegerät (Verifier) eine Nachricht zum *RFID*-Tag (Signer), die Größe der Nachricht ist 128 bits. Diese Nachricht wird vom *RFID*-Tag verschlüsselt, das einen privaten Schlüssel verwendet und die entschlüsselte Nachricht zu dem Lesegerät zurücksendet. Die Größe der verschlüsselten Nachricht beträgt auch 128 bits. In diesem Papier wird ein neues digital Signatur Schema für *RFID*-Tags definiert. Das Lesegerät schickt eine Nachricht mit der Größe $< 2^{16}$ zum *RFID*-Tag, das diese Nachricht signieren soll. Die Größe dieser Signatur beträgt ungefähr 25 kbits. Das Protokoll *ISO/IEC 18000* wird in diesem Schema nicht verwendet, weil die Größe des Frames mehr als 128 bit ist. Deswegen verwenden wir ein neues Protokoll, um die Daten zwischen Lesegerät und *RFID*-Tag zu übertragen [3].

Das allgemein verwendete *IEEE 802.3 MAC-Frame* für *RFID*-System wird in der (Tabelle 2.4) aufgezeigt.

- Preamble: Synchronize Kommunikation zwischen RFID-Tag und Lesegerät.
- SOF: Start Of Frame.

2 RFID-System

Preamble	SOF	Adresse1	Adresse2	Length Data	Data	CRC
2 byte	2 byte	2 or 6 byte	2 or 6 byte	1 byte	0-1500 byte	4 byte

Tabelle 2.4: IEEE 802.3 Frame Format

- Adresse1: Lesegerät Adresse.
- Adresse2: *RFID*-Tag Adresse.
- CRC: Cyclic Redundancy Check, wird verwendet, um errors während der Transmission zu ermitteln .

2.3.3 IEEE 802.11 Protokoll für RFID-Tags

Passive *RFID*-Tags Produkte erscheinen in den 125 KHz, in 13.56 MHz, in 915 MHz und in 2.45 GHz Frequenzbänder [4]. Es können höhere effektive Datenraten innerhalb der UHF Bänder erzielt werden. Dabei ist das Konzept von *IEEE 802.11* Protokoll für *RFID*-System das geeignetste [5] . Die kabellose Datenübertragungsrate wurde im Standard *IEEE 802.11* auf 1 Mbit/s oder 2 Mbit/s festgelegt. *IEEE 802.11* definiert einen neuen Frame, um die Daten zwischen dem Lesegerät und dem *RFID*-Tag übertragen zu können (siehe Tabelle 2.5).

Preamble	SOF	PLCP Header	Prüfsumme	MAC-Data	CRC
80 Bit	16 Bit	1 Mbit/s	16 Bit	0-2946 byte	32 Bit

Tabelle 2.5: IEEE 802.11 Frame Format

- Preamble: Synchronize Kommunikation zwischen RFID-Tag und Lesegerät.
- SOF: Start Of Frame.
- PLCP-Header: beinhaltet Daten.
- Prüfsumme: 16 Bit.
- MAC-Data: enthält neben den Nutzdaten Angaben zu Adressierung, Fragmentierung und Sicherheitseinstellungen
- CRC: Cyclic Redundancy Check, wird verwendet, um errors während der Transmission zu ermitteln .

Spread-Spectrum-Verfahren Daten benötigen zur Übertragung eine gewisse Bandbreite. Um Störungen im genutzten Frequenzband besser kompensieren zu können, ist es möglich die Bandbreite künstlich zu erhöhen. Es wurde eingeführt, um sich gegen gezielte Störungen schützen zu können. Um die Bandbreite von Daten zu steigern, werden sie mit einem Signal mit höherer Bandbreite moduliert (siehe Abbildung 2.6). Das Ergebnis muss wieder demoduliert werden, wozu man den Schlüssel kennen muss, mit dem die Daten moduliert wurden. Der *IEEE-Standard* definiert ein Verfahren zur Bandbreitenspreizung, das im folgenden kurz erläutert wird.

- FHSS: *Frequency Hopping System* für Kommunikation mit *RFID*-Tags. Dafür wird der Übertragungsbereich in 79 Frequenzbänder mit jeweils 1MHz Bandbreite aufgeteilt [6]. Beim FHSS wird die Bandbreite der Daten nicht größer gewählt als die Breite des Kanals, also 1MHz. Das schränkt die Übertragungsgeschwindigkeit ein, erlaubt andererseits aber den Betrieb vieler parallel arbeitender Teilnehmer [23].

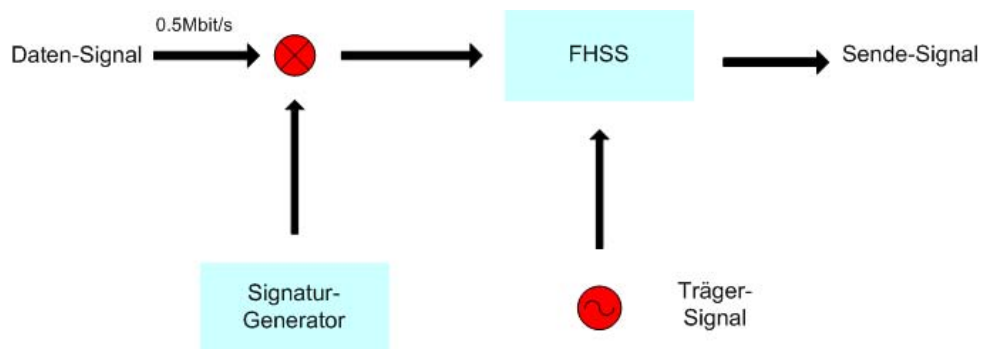


Abbildung 2.6: Erzeugung des Sendesignals

2.4 Anwendung RFID-System

Die ersten kommerziellen Vorläufer der *RFID*-Technologie kamen bereits in den 1960er-Jahren auf den Markt. Erst die Forschung und Weiterentwicklung in jüngerer Zeit haben ihr zum Durchbruch verholfen. Inzwischen hat *RFID* in zahlreiche Branchen Einzug gehalten (siehe Abbildung 2.7).

2 RFID-System



Abbildung 2.7: Anwendungsbereiche RFID-Technologie

2.4.1 Logistik

RFID-Systeme bieten in der gesamten Logistik und im Transportwesen vielfältige Anwendungsmöglichkeiten, beispielsweise in der Automobilindustrie oder in Transportprozessen beim Management von wieder verwendbaren Transportbehältern (siehe Abbildung 2.8). Auch an Flughäfen kommt *RFID* zum Einsatz. Alle Gepäckstücke erhalten einen *RFID*-Tag. Sie lassen sich so schneller und zuverlässiger verladen als mithilfe der herkömmlichen Barcode Technik. Sollte trotzdem einmal ein Gepäckstück verloren gehen, kann das Flughafenpersonal es leichter auffinden.

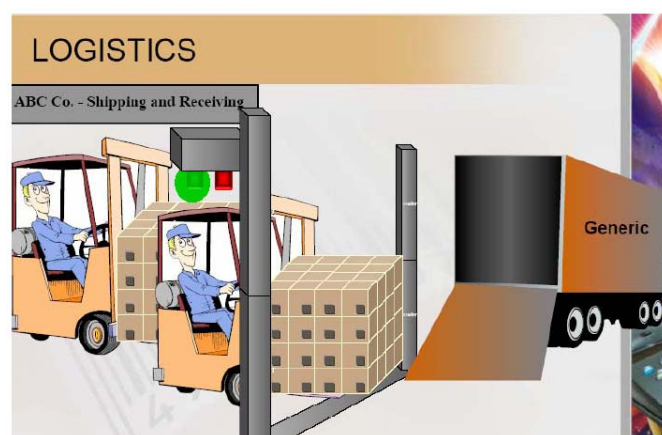


Abbildung 2.8: Anwendung RFID-System in Logistik

3 Kryptographische Maßnahmen für RFID-System

3.1 Einleitung

Mit der zunehmenden Verbreitung von *RFID*-Systemen, die eine kontaktlose automatische Erfassung von Daten erlauben, wächst die Bedeutung von Datenschutz und Datensicherheit. Für sicherheitsrelevante Anwendungen können kryptographische Verfahren den Schutz der übertragenen Daten gewährleisten. Bei *RFID*-Systemen ist die Authentisierung von Tags durch das Lesegerät und umgekehrt die Aufgabe diese Verfahren. *RFID*-Systeme müssen beim Erfassen eines Tags dessen Identität überprüfen, um festzustellen, ob dieses Tag zur Teilnahme berechtigt ist. Die folgenden Verfahren zur Authentisierung des Lesegeräts gegenüber dem Tag sind symmetrische Authentifikationsverfahren oder asymmetrische Authentifikationsverfahren.

3.2 Kryptographische Theorie

Seit 1967 erschienen viele kryptographische Arbeiten in öffentlich zugänglichen Publikationen. Den endgültigen Durchbruch zum Einsatz kryptographischer Verfahren auf breiter Front dürften die Entdeckung der sogenannten asymmetrischen Kryptographie durch Diffie und Hellman im Jahre 1976 und die Einführung des *Data Encryption Standard* (DES) im folgenden Jahr gebracht haben.

3.2.1 Ziele der Kryptographie

An moderne kryptographische Verfahren werden im wesentlichen vier Anforderungen gestellt. Sie müssen nicht bei jeder Anwendung gleichzeitig erfüllt werden.

- Vertraulichkeit: Der Inhalt eines Dokuments soll nur von dazu befugten Personen gelesen werden können.
- Integrität: Der Inhalt eines Dokuments soll nicht unbemerkt verändert werden können.

- Authentikation: Der Urheber eines Dokuments soll feststellbar sein; kein anderer soll sich als Urheber ausgeben können.
- Verbindlichkeit: Der Urheber eines Dokuments soll seine Urheberschaft nicht abstreiten können.

3.3 Das Konzept der digitalen Signaturen

Digitale Signaturen werden, die in der virtuellen Welt das Substitut für eine eigenhändige Unterschrift bilden sollen, mit digitalen Signaturen realisiert. Der Begriff *digital Signatur* wurde 1976 von Diffie und Hellman in einem bahnbrechenden Aufsatz eingeführt und beruht auf dem darin erstmals veröffentlichten Konzept der asymmetrischen Kryptografie, auch als *Public-Key-Kryptographie* bezeichnet. Im Gegensatz zu symmetrischen Verfahren, wo zwei Kommunikationspartner einen Schlüssel für einen manipulationssicheren Austausch von elektronischen Dokumenten benötigen, verfügen in einem *Public-Key-System* die einzelnen Teilnehmer über zwei unterschiedliche Schlüssel, mithilfe derer sie mit beliebig vielen verschiedenen Kommunikationspartnern kommunizieren können. Einer der beiden Schlüssel ist öffentlich bekannt und wird daher als öffentlicher Schlüssel bezeichnet *public key*, daher auch die Bezeichnung *Public-Key-Kryptographie*. Der andere Schlüssel sollte sich unter der alleinigen Kontrolle seines Inhabers befinden und wird daher privater Schlüssel genannt.

Zwischen den beiden Schlüsseln besteht ein komplexer mathematischer Zusammenhang, so dass aus einem hinreichend langen öffentlichen Schlüssel der zugehörige private Schlüssel praktisch d.h. mit angemessenem Ressourcenaufwand nicht ermittelt werden kann. Mit dem öffentlichen Schlüssel des Empfängers lassen sich Klartexte verschlüsseln, die dann nur mit dem dazugehörigen privaten Schlüssel zu entschlüsseln sind. Für die Gewährleistung von Integrität und Authentizität werden die Rollen der Schlüssel getauscht.

Der Absender chiffriert die Nachricht mit seinem privaten Schlüssel (Signierung), die erfolgreiche Dechiffrierung mit dem öffentlichen Schlüssel beweist die Unverfälschtheit des Dokumentes sowie die Tatsache, dass nur der Inhaber des Schlüsselpaares als Nachrichturheber. Der Nachteil von asymmetrischen Verfahren ist der hohe Aufwand an Rechenzeit. Da das zu signierende Dokument sehr lang sein kann und somit der Signaturvorgang sehr lange dauern würde, wird das sogenannte Hash and Sign Prinzip angewandt. Zunächst wird ein Fingerabdruck des Dokuments Hash-Wert erstellt und nur dieser signiert. Das ursprüngliche Dokument wird zusammen mit der Signatur dem signierten Hash-Wert an den Empfänger übermittelt.

Der Empfänger, der ein signiertes Dokument erhält, ermittelt zunächst den Hash-Wert des Dokuments. Dann entschlüsselt er die Signatur mit dem öffentlichen Schlüssel des Absenders. Nach der Entschlüsselung der Signatur bleibt der Hash-Wert des Dokuments übrig, so wie er beim Absender vor der Versendung gebildet wurde. Der Empfänger prüft die Übereinstimmung dieses Hash-Werts mit dem von ihm gebildeten Hash-Wert. Stimmen beide überein, wurden die

3 Kryptographische Maßnahmen für RFID-System

Daten nicht verändert und das Dokument ist authentisch. Diese prinzipielle Funktionsweise der digitalen Signatur ist in der (Abbildung 3.1) dargestellt.

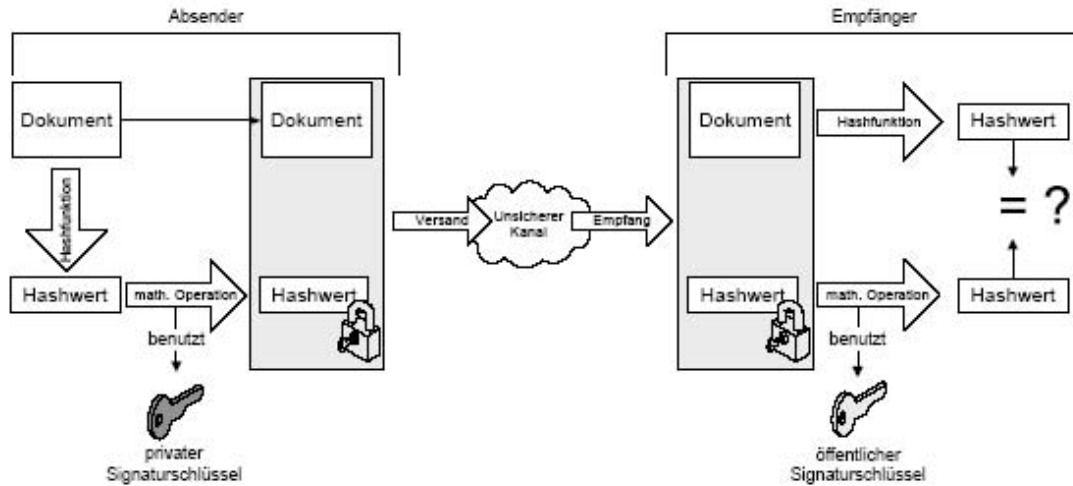


Abbildung 3.1: Signierungsvorgang

3.4 Authentifikationsverfahren

Kryptographische Verfahren können die Nachrichten vor unbefugtem Mitlesen absichern helfen. Solche Verfahren dienen dem Ziel der Vertraulichkeit. neue kryptographische Verfahren lassen sich aber auch einsetzen, um die anderen drei Ziele Integrität, Authentikation und Verbindlichkeit zu erreichen. Man spricht dabei von Authentifikationsverfahren.

3.4.1 Symmetrische Authentifikationsverfahren

Ein symmetrisches Authentifikationsverfahren funktioniert wie folgt: Der Sender berechnet aus der Nachricht mit Hilfe einer One-Way Funktion einen kleinen Datenblock, den man Hash-Wert nennt. Es ist nicht möglich, ohne Kenntnis der Nachricht eine zu einem solchen Hash-Wert passende Nachricht zu finden. Auch ist es praktisch nicht möglich, zwei Nachrichten zu finden, die denselben Hash-Wert haben. Nun signiert der Sender den Hash-Wert mit dem vereinbarten Schlüssel und sendet die Nachricht und signierten Hash-Wert an den Empfänger. Der Empfänger kann dann den Hash-Wert wieder herstellen. Durch erneute Anwendung der öffentlichen One-Way Funktion auf die Nachricht und Vergleich des Ergebnisses mit dem entschlüsselten Hash-Wert kann er feststellen, ob die Nachricht während der Übertragung verändert wurde oder die Nachricht von richtigem Absender gesendet wurde. Denn ein Angreifer könnte zwar die Nachricht verändern und aus der neuen Nachricht den zugehörigen Hash-Wert berechnen.

3 Kryptographische Maßnahmen für RFID-System

Er ist jedoch nicht dazu in der Lage, diesen neuen Hash-Wert korrekt zu verschlüsseln, da er den zwischen Sender und Empfänger vereinbarten Schlüssel nicht kennt. Jede Veränderung der Nachricht muss dem Empfänger also auffallen, weil die Nachricht dann nicht mehr zum Hash-Wert passt. Integrität und Authentizität einer Nachricht werden nur gegen Angriffe gesichert, Verbindlichkeit dagegen wird überhaupt nicht erreicht.

3.4.2 Asymmetrische Authentikationsverfahren

Bei einem asymmetrischen Authentikationsverfahren besitzt jeder Teilnehmer zwei Schlüssel: einen privaten Signaturschlüssel und einen öffentlichen Verifizierungsschlüssel. Will jemand eine Nachricht als von ihm erstellt ausweisen, so berechnet er den Hash-Wert einer Nachricht und verschlüsselt diesen mit seinem Signaturschlüssel. Jeder, der im Besitz des zugehörigen öffentlichen Verifizierungsschlüssels hat, kann nun die Echtheit der Nachricht überprüfen, sie verifizieren. Signieren kann die Nachricht allerdings nur der Besitzer des privaten Signaturschlüssels, so dass Integrität, Authentikation und Verbindlichkeit realisiert werden können (siehe Abbildung 3.2).

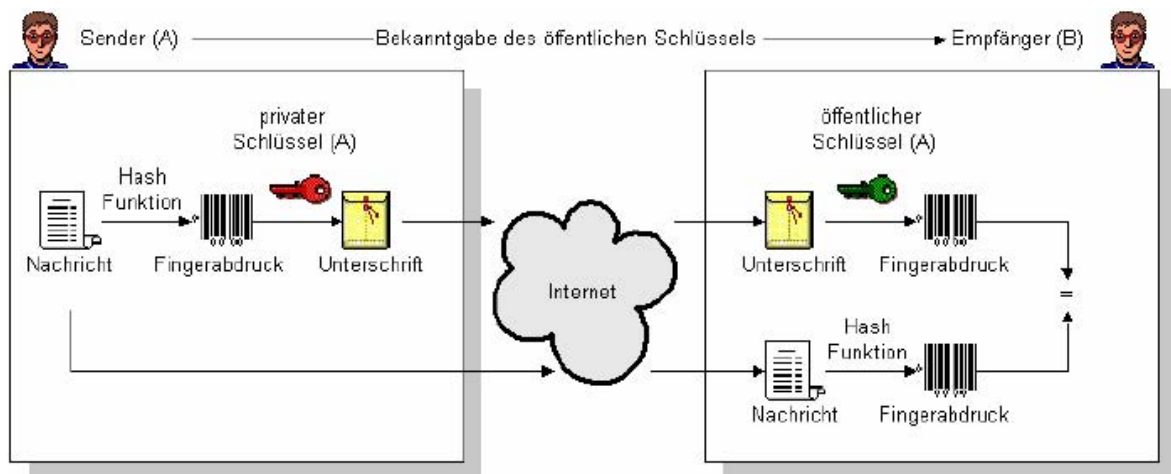


Abbildung 3.2: Asymmetrische-Authentikation

3.5 Sicherheit von RFID-Systemen

Wie jedes andere System der Nachrichten und Informationstechnik, so sind auch *RFID*-Systeme potentiell gefährdet, von einem Angreifer ausgespäht oder manipuliert zu werden. Um die möglichen Risiken des Einsatzes von *RFID*-Systemen etwas besser einschätzen zu können, werden wir daher einige der gängigen Angriffsarten auf *RFID*-Systeme etwas genauer betrachten. Im Anschluß daran werden kryptographische Verfahren zum Schutz gegen gängige Angriffe vorgestellt. Ein *RFID*-System ist darauf angewiesen, dass die vom Lesegerät erfassten Daten über

weitere Kommunikationskanäle mit anderen Datenbeständen verknüpft werden. Wir beschränken uns daher im Wesentlichen auf Angriffe auf die Luftschnittstelle zwischen dem Lesegerät und den *RFID*-Tags, sowie Angriffe auf das Tag selbst.

3.5.1 Angriffe auf RFID-Systeme

Ein Blick auf die Abbildung 3.3 zeigt uns verschiedene grundlegende Angriffsarten auf verschiedenen Komponenten eines *RFID*-Systems. Grundsätzlich kann ein Angriff dabei auf das Tag, das Lesegerät oder auch das Kommunikation Interface zwischen das Tag und das Lesegerät erfolgen.

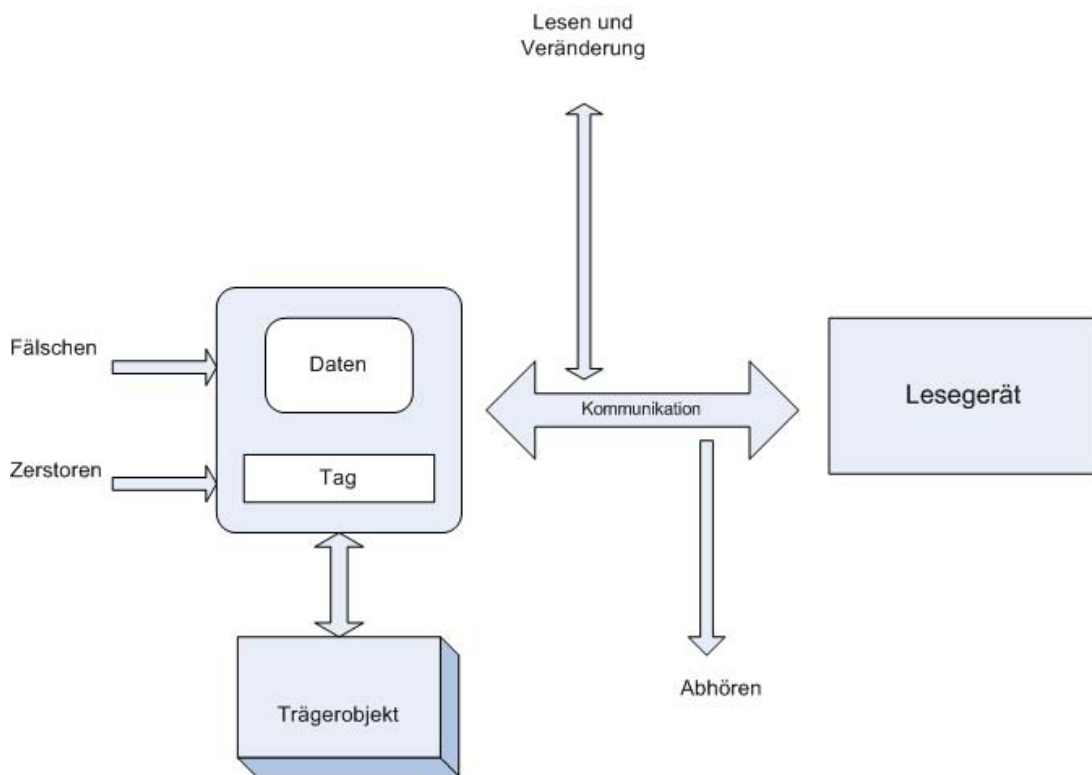


Abbildung 3.3: Angriffsarten eines RFID Systems bei der Kommunikation

Es lassen sich für die nachfolgend beschriebenen Angriffe Angriffsarten klassifizieren.

- **Ausspähen:** Hier versucht der Angreifer, sich unberechtigten Zugang zu Informationen und Daten zu verschaffen.
- **Täuschen:** Hierbei versucht der Angreifer, unzutreffende Informationen in das *RFID*-System einzuspeisen, um den Benutzer eines *RFID*-Systems zu täuschen.

3 Kryptographische Maßnahmen für RFID-System

- Denial of Service: Bei diesem Angriff wird die Verfügbarkeit von Funktionen eines *RFID*-Systems beeinträchtigt.

Angriffe auf den RFID-Tag

Am leichtesten zugänglich ist in der Regel das Tag, das auf Waren oder als Ticket für einen Angreifer jederzeit und in den meisten Fällen zeitlich unbegrenzt zur Verfügung steht. Gegenüber dem Tag existiert daher eine Vielfalt an unterschiedlich wirksamen Angriffen.

Zerstören des Tags Die einfachste Möglichkeit eines Angriffes auf ein *RFID*-System besteht in der mechanischen Zerstörung eines Tags. So kann die Antenne meist mit einfachen Hilfsmitteln durchtrennt oder abgeschnitten werden. Auch der Chip kann durch Knicken leicht zerstört werden.

Abschirmen des Tags Man könnte beispielsweise verhindern, dass der Tag überhaupt Energie erhält. Bei aktiven Tags könnte man die Batterie entfernen, so dass diese keine Energie zum Modulieren der Frequenz haben.

Angriffe über die Kommunikation zwischen RFID-Tag und Lesegerät

RFID-Systeme sind Funksysteme und kommunizieren über elektromagnetische Wellen im Nah und Fernfeld. Für einen Angreifer liegt es daher nahe zu versuchen, ein *RFID*-System über das *High Frequency Interface* anzugreifen. Der besondere Reiz liegt darin, dass bei einem Angriff über das HF-Interface kein physischer Zugriff auf ein Lesegerät oder ein Tag nötig ist, sondern aus der Entfernung agiert werden kann. Derzeit sind folgende Angriffe bekannt:

Abhören der Kommunikation Ein Abhören der Kommunikation zwischen das Lesegerät und das *RFID*-Tag ist, wie man es aus dem Bereich der Kommunikation kennt ist prinzipiell möglich. Und das Risiko wächst natürlich, je größer die Übertragungreichweite des betreffenden *RFID*-Systems ist. *RFID*-Systeme mit aktiven Tags sind mit hoher Reichweite einem höheren Risiko ausgesetzt als passive Tags mit nur wenigen Zentimetern Reichweite. Da viele *RFID*-Systeme im Klartext senden, ist es ein Leichtes, benötigte Informationen zu erhalten.

Unautorisiertes Auslesen der Daten Ein Auslesen durch eine dritte unberechtigte Partei ist auch nicht auszuschließen, wenn man bedenkt, dass es Lesegeräte und sogar spezielle Software zu kaufen gibt. So könnten sich Angreifer unerlaubt Informationen über Personen oder andere Dinge verschaffen.

Unautorisiertes Verändern der Daten Es ist natürlich auch möglich, dass man ein Lese-/Schreibgerät besitzt, Daten auf dem Tag zu verändern. Hierbei verändert der Angreifer einfach die Datenblöcke und ermittelt, wenn nötig, sogar neue Prüfsummen, so dass der Empfänger nichts von der Veränderung bemerkt.

3.6 Sicherheitsverfahren für die RFID-Systeme

Vertraulichkeit, Integrität, Authentikation sind Bedingungen, die bei einer Übertragung von Daten zwischen das *RFID*-Tag und den Lesegerät erfüllt werden müssen.

3.6.1 Vertraulichkeit der Informationen

Um die Vertraulichkeit von Informationen zu gewährleisten, gibt es verschiedene Verschlüsselungsverfahren, wie zum Beispiel symmetrische Verschlüsselungsverfahren, asymmetrische Verschlüsselungsverfahren oder Hash-Funktion. Die am häufigsten verwendete Verschlüsselung bei *RFID*-Systemen ist die symmetrische Verschlüsselung. Dabei werden Verschlüsselungsalgorithmen wie Block Ciphers, Stream Ciphers genutzt. Noch sicherer sind asymmetrische Verschlüsselungsalgorithmen wie RSA oder ECDSA elliptische Kurven.

3.6.2 Authentifizierung der Informationen des Tags

Zur Authentifizierung eines Tags stehen folgende Verfahren zur Verfügung:

Das digital Signatur Verfahren, One-Time Signatur Verfahren. Ein Signaturverfahren besteht aus zwei Bestandteilen:

- einem Algorithmus zur Signaturerzeugung, im weiteren als Signieralgorithmus bezeichnet.
- einem Algorithmus zur Signaturverifikation, im weiteren als Verifizieralgorithmus bezeichnet.

Jedes Signaturverfahren hängt von einem Schlüsselpaar ab, bestehend aus einem privaten (geheimen) Schlüssel X und einem öffentlichen Schlüssel Y . Das Schlüsselpaar (X, Y) wird mit einem Schlüsselerzeugungsalgorithmus generiert. Jedes Signieralgorithmus bildet eine kryptografische Funktion S , die im weiteren als Signierfunktion bezeichnet wird. Im Verifizieralgorithmus wird die sogenannte Verifikationsfunktion V verwendet. Die Signierfunktion S erhält als Eingabe die zu signierenden Daten M , den privaten Schlüssel X sowie weitere Parameter und berechnet die digitale Signatur

$$s = S(X, M)$$

3 Kryptographische Maßnahmen für RFID-System

Die Verifikationsfunktion V liefert als Verifikationsergebnis TRUE (d.h. Signatur gültig) oder FALSE (d.h. Signatur ungültig) mit Hilfe des öffentlichen Schlüssels Y , weiterer Parameter sowie digital signierter Daten M . Mathematisch gilt:

$$V(Y, s, M) \in \{true, false\},$$

mit

$$V(Y, s, M) = true \iff s = S(X, M)$$

Die Daten, die die Eingabe für die Signier- bzw. Verifikationsfunktion bilden, werden aus der zu signierenden Nachricht M normalerweise mit Hilfe einer Hash-Funktion erstellt. Die Hash-Funktion ist somit eine weitere Komponente eines Signaturverfahrens. Ein solches Verfahren beinhaltet meistens die Anwendung einer Hash-Funktion auf die Nachricht und die anschließende Formatierung des berechneten Hash-Wertes.

3.6.3 Integrität der Informationen

Um zu Prüfen, ob empfangene Daten integer sind, gibt es verschiedene Prüfverfahren, da die Daten während der Übertragung durch Störungen nicht vollständig sein könnten bzw. die Daten abgefangen und manipuliert sein könnten. Da Angreifer Übertragungsdaten abhören und auch diese Daten leicht verändern könnten, wäre es notwendig, dass jede Nachricht eine digital Signatur erhält (einen elektronischen Fingerabdruck). Ein solcher Fingerabdruck könnte beispielsweise ein errechneter Wert aus gesendeten Daten sein, welcher sich verändert, wenn die Daten geändert werden.

3.7 Kryptographie Implementation

In den letzten Jahre werden viele Verschlüsselungsimplementierungen vorgeschlagen. Sie werden in der Software oder in den Hardware eingeführt. Die Wahl der Implementierung hängt davon ab, welcher Algorithmus angewendet wird.

3.8 Symmetrische Authentifizierung

3.8.1 Get-Challenge-Response

Die ISO *International Organization for Standardization* [29], definiert verschiedene Challenge-Response-Verfahren für die starke Authentifizierung bei kontaktlosen Chipkarten wie *RFID*-Tag.

3 Kryptographische Maßnahmen für RFID-System

Idee: Ein Lesegerät möchte sich einem *RFID*-Tag gegenüber identifizieren.

- Lesegerät \rightarrow *RFID*-Tag : Lesegerät sendet an das *RFID*-Tag eine Aufgabe (*challenge*).
- *RFID*-Tag \rightarrow Lesegerät : *RFID*-Tag liefert eine Antwort (*response*), die nur ein *RFID*-Tag selbst geben kann.
- Lesegerät : Das Lesegerät akzeptiert die Antwort, wenn sie mit der erwarteten Lösung übereinstimmt.

Bei diesem Verfahren besitzen die beiden Lesegerät und *RFID*-Tag gleich geheim kryptographischen Schlüssel K symmetrisches Verfahren (siehe Abbildung 3.4). Das Lesegerät erzeugt eine Zufallszahl random number r_R und sendet sie an das Tag. Die Verwendung des gemeinsamen geheimen Schlüssels K und eines gemeinsamen Schlüsselalgorithmus E_k berechnet verschlüsselt das Tag die Zahl r_R ; ($E_K(r_R)$) und sendet diesen Datenblock an das Lesegerät. Weiterhin muss Das Lesegerät die empfangenen Daten entschlüsseln und vergleicht sie mit der gesendeten Zahl r_R . Wenn diese übereinstimmen, kann das Lesegerät die Echtheit des Tags versichern. Ein Nachteil des beschriebenen Authentifizierungsverfahren besteht darin, dass alle Tags mit einem identischen kryptographischen Schlüssel K gespeichert sind. Da solche Tags für jedermann in unkontrollierbarer Anzahl zugänglich sind, muss mit einer geringen Wahrscheinlichkeit damit gerechnet werden, dass der Schlüssel eines Tags aufgedeckt wird. Das oben beschriebene Verfahren wäre damit vor Manipulationen ungeschützt und ausgeliefert. Das andere Problem ist das Austauschen des Schlüssels zwischen dem Lesegerät und dem Tag.

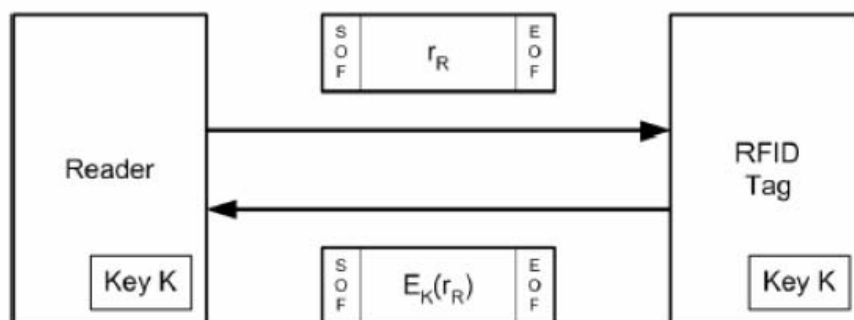


Abbildung 3.4: Ablauf einer Authentifizierung zwischen Tag und Lesegerät

3.9 Asymmetrische Authentifizierung

Die häufigste Kritik der Verwendung von *Public Key cryptographic* in den *RFID*-Systems ist die Berechnungskompliziertheit, die niedrige Speicherkapazität und die Größe des Overheads zwischen dem Lesegerät und dem *RFID*-Tag.

3.9.1 RSA Verfahren

1977 wurde der asymmetrische (RSA) Verfahren von Rivest, Shamir und Adleman vorgestellt, der auf Ideen von den Herren Diffe und Hellman basiert. Dieser wird bis heute in der Praxis eingesetzt. Das Schlüsselpaar besteht aus dem Produkt von zwei Primzahlen, wobei die Sicherheit darin besteht, dass es sehr einfach ist zwei Primzahlen miteinander zu multiplizieren, aber es bei einer gewissen Zahlengröße sehr zeitaufwendig ist, dieses Produkt zu faktorisieren [9].

RSA Schlüsselerzeugung

Um RSA-Verfahren auf das *RFID*-Tag anwenden zu können, muss das Tag einen geheimen Schlüssel und einen öffentlichen Schlüssel erzeugen. Die Sicherheit hängt davon ab, wie gross p und q sind.

1. Wähle zwei Primzahlen p und q .
2. Berechne $n = p \cdot q$ und $\phi = (p - 1) \cdot (q - 1)$.
3. Suche e mit $1 < e < \phi$, so dass e Teilerfremd zu ϕ ist.
4. Suche d mit $1 < d < \phi$, so dass gilt, dass $(e \cdot d) \bmod \phi = 1$.
5. Der geheime Schlüssel ist (n, d) und der öffentliche Schlüssel ist (n, e) .

3.9.2 RSA Signatur

Das Lesegerät sendet eine Nachricht m an das Tag, der die Nachricht m signieren muss. Die Hash-Funktion

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^k$$

wandelt m in ein geeignetes Format um.

1. Berechne $\tilde{m} = H(m)$, Nachricht digest mit Ausgabe k bits.
2. Berechne $s = \tilde{m}^d \bmod n$.
3. das Tag sendet die Signatur s an das Lesegerät.

3.9.3 RSA Verifikation

Das Lesegerät muss die Signatur s von den *RFID*-Tag überprüfen.

1. Berechne

$$\tilde{m} = s^e \text{ mod } n$$

2. Berechne

$$\hat{m} = H(m)$$

3. Vergleichen ob $\hat{m} = \tilde{m}$, wenn nicht verwerfe die Nachricht m

Bei der Implementierung des RSA-Verfahrens auf das *RFID*-Tag muss beachtet werden, dass zum Teil große Anforderungen an die Hardware gestellt werden, da das Finden von großen Primzahlen sehr komplex ist und der Euklidische Algorithmus zur Bestimmung des $ggT()$ und das Potenzieren von $\text{mod } n$ notwendig ist. Um diesen Algorithmus auf ein *RFID*-Tag Implementieren zu können, brauchen wir ein *RFID*-Tag mit Mikroprozessor, der alle komplexen Berechnungen ausführen könnte, die RSA-Algorithmus wie potenzieren und modulo n durchführen. Anwendung des RSA-Verfahrens mit private Schlüssel 512 bits auf ein *RFID*-Tag benötigt mindestens 32 kbyte Speicherkapazität . Alle mathematischen Operationen wie squar, Adder, Multiply und Shift, auf die der RSA-Algorithmus beruht, kann das *RFID*-Tag Schaltung nicht diese Operationen durchführen (siehe Abbildung 3.5).

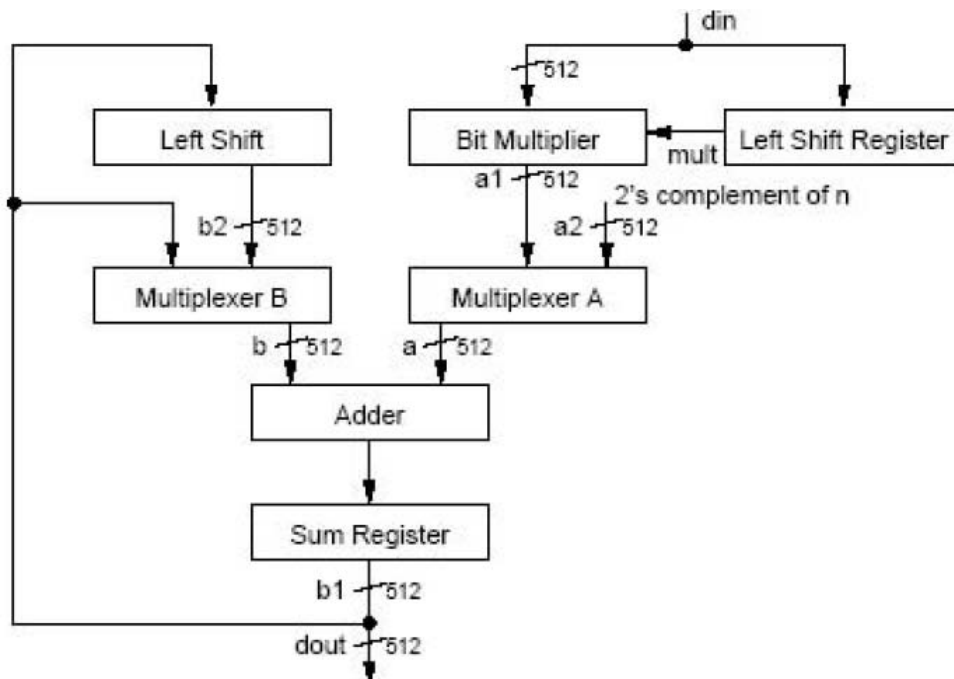


Abbildung 3.5: Block Diagram für RSA

3 Kryptographische Maßnahmen für RFID-System

Die Abbildung 3.5 zeigt die Architektur für one squaring Operation. Ein *Left Shift Register*, ein *Bit Multiplier*, ein *Left Shift unit*, und die haupt units *Adder*, *Sum Register*, und two *multiplexers*, um eine mod operation zu berechnen. Diese Architekur überragt der Leistung des Tag Mikroprozessors [8].

3.10 Hash-Funktion

Im Signaturverfahren werden nicht die Daten selbst, sondern der sogenannte Hash-Wert dieser Daten, als eine Folge fester kurzer Längen, signiert. Dadurch müssen die Signaturen nicht gleich lang wie die signierten Nachrichten sein und ihre Berechnung verläuft wesentlich schneller, als wenn das ganze Nachricht signiert werden müsste. Ein Verfahren zur Berechnung eines Hash-Wertes wird als Hash-Funktion bezeichnet.

Idee: Eine Hash-Funktion ist eine Abbildung H , die für einen Eingabestring m variabler Bitlänge eine Ausgabe $H(m)$ mit fester kürzerer Bitlänge erzeugt:

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^n$$

Die Ausgabe $H(m)$ wird als Hash-Wert, Nachricht digest bezeichnet. Die Berechnung von $H(M)$ muss sich leicht durchführen lassen. Für kryptografische Zwecke müssen die Hash-Funktionen zwei Anforderungen erfüllen:

- kollisionsfrei, kollisionsresistent sein.
- Unumkehrbar, One-Way Funktionen sein.

SHA-1 ist eine Hash-Funktion, der in dem Merkle Signatur Schema verwendet wird und der den Authentifizierungsbaum aufbauen könnte (siehe Abbildung 3.6).

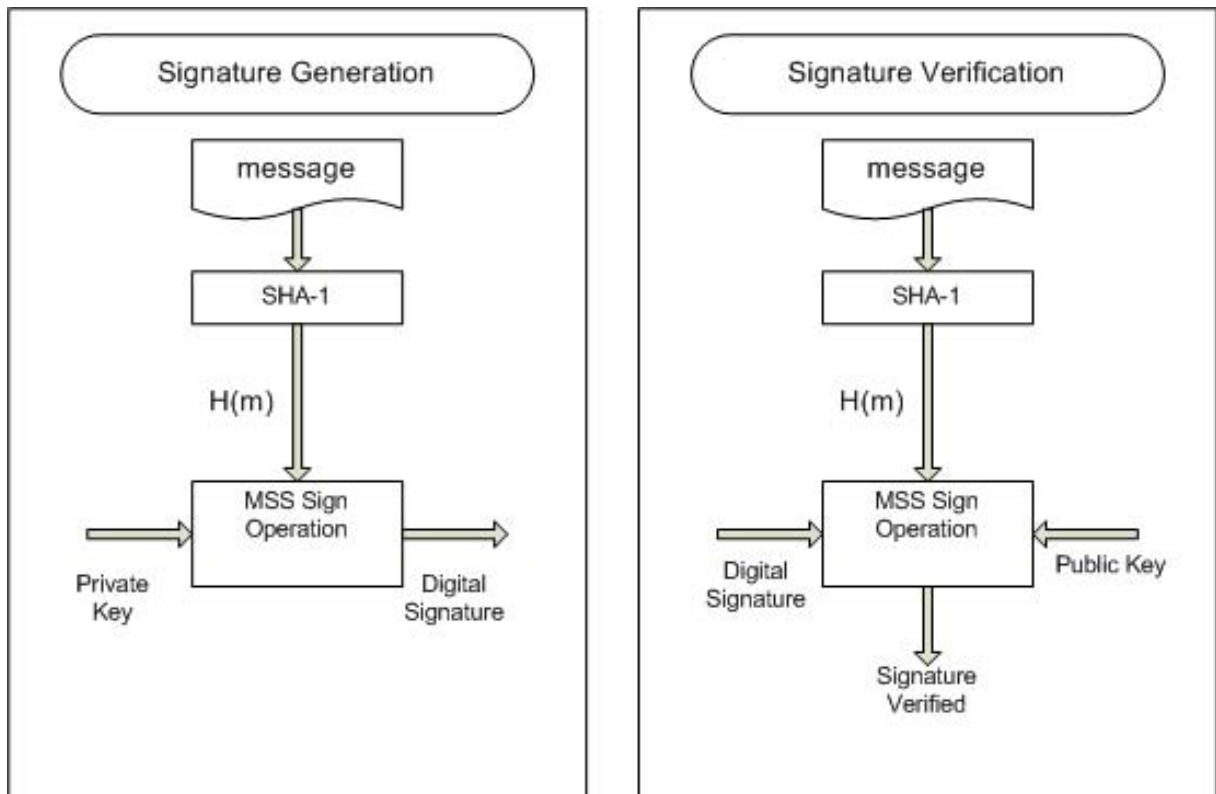


Abbildung 3.6: SHA-1 mit MSS

3.10.1 Secure Hash Algorithm SHA-1

SHA – 1 könnte Hash-Werte von Nachrichten der Länge bis $< 2^{64}$ berechnen. Die Blockgröße beträgt in SHA-1 512 bits. Das Paddingverfahren füllt die Eingabe auf ein Vielfaches von 512 auf. Zuerst wird an den Eingabestring eine Eins angehängt, dann so viele Nullen, dass die Länge einem Vielfachen von 512 bzw. minus 64 bit entspricht. Als letztes wird eine 64 Bit-Darstellung der Länge der Nachricht vor dem Auffüllen angehängt. Die Auffüllung wird auch dann vorgenommen, wenn die Länge der ursprünglichen Eingabe bereits ein Vielfaches der Blockgröße ist [17].

SHA-1 Vorbereitungsphase

- Eingabe: Eingabestring M der Länge $L_M < 2^{64}$.
- Ausgabe: Der formatierte Eingabestring M der Länge $L_M = t \cdot 512$.
- Verlauf:
 $d := (447 - L_M) \bmod 512$.

3 Kryptographische Maßnahmen für RFID-System

$l :=$ Binärdarstellung der Länge von M ; $M := M\|0^d\|l$.

SHA-1 Berechnungsphase SHA-1 benutzt für die Berechnung des Hash-Wertes drei Grund-Funktionen [28].

- $x \oplus y$: bitweises exklusives OR von x und y .
- $x + y$: Addition modulo 2^{32} .
- $ROTL^n(x)$: Zyklischer Linksshift von w -bit x um n Positionen.

Alle komplexen Berechnungsoperationen, die in *Public Key Cryptographie* wie RSA und ECD-SA vorhanden sind, können durch Anwendung SHA-1 vermieden werden können, um eine Nachricht auf das *RFID*-Tag zu signieren.

4 MSS Merkle Signatur Schema

Einleitung Die Sicherheit aller dargestellten *public Key cryptographien* wie RSA, ECDSA Signatureverfahren beruht entweder auf dem Faktorisierungsproblem oder auf dem Problem des diskreten Logarithmus. Die Fortschritte bei der Entwicklung von *Quantencomputern* [19]. würden allerdings dazu führen, dass diese Algorithmen nicht mehr sicher wären. Schon 1994 hat Peter Shor bewiesen, dass die *Quantencomputer* in polynomialer Zeit sowohl ganze Zahlen faktorisieren als auch diskrete Logarithmen in den relevanten Gruppen berechnen können [20]. Dass man an die Alternativen dringend denken muss, hat die Vorstellung des ersten von der Industrie gefertigten *Quantenprozessors* mit 16 Qubits (*Quanten-Bits*) durch die kanadische Firma D-Wave Systems in Februar 2007 bewiesen [31]. Zwar bedroht das noch nicht die Sicherheit von RSA, DSA und ECDSA Signaturen, aber es wird zur Zeit nach Quantencomputer-resistenten Lösungen gesucht [21].

4.1 One-Time Signatur Schema OTSS

One-Time Signatur Schema (OTSS) ist eine entwicklungsfähige Alternative zur *public key digital Signatur*. OTSS Sicherheit basiert nur auf der Stärke der *underlying One-Way Funktion* wie *DES* [11], *MD5* [12], *SHA* [10], und hängt nicht von der vermuteten Schwierigkeit eines mathematischen Problems ab.

One-Time Signatur Schema OTSS ist eine Art des digital Signatur Schemes und wird verwendet, um nur eine einzelne Nachricht mit einem gegebenen Stück privaten und öffentlichen Informationen zu signieren [25]. Durch konventionelle Signatur Schemata wie *RSA*, *ECDSA* kann das gleiche Schlüsselpaar verwendet werden, um mehrfache Nachrichten zu signieren und zu verifizieren. Einer der Vorteile ist, dass die One- Time Signatur Generation und Verifikation sehr leistungsfähig sind und es für Chipkarten oder *RFID*-Tag nützlich ist [26]. Niedrige Berechnungen, nicht komplexe Operationen, und niedriger Speicher werden angefordert, um OTSS auf das *RFID*-Tag anwenden zu können. Es gibt eine verhältnismäßig leistungsfähige Implementierung des One-Time Signatur Schema durch Merkle-Baum, das durch den Merkle-Baum angerufen wird, der neue Schlüsselpaare für jede Nachricht erfordert. Merkle One-Time Signatur Schema liefert eine Signaturschwindigkeit bis 35 mal schneller als ein 2048-Bits *RSA* Signatur Schema, für ungefähr eine Million Signaturen und eine Signaturgröße nur einiger Kilobytes.

Lamport erfand zuerst ein One-Time digital Signatur Schema, der auf One-Way Funktion basierte [27]. Jedoch erfordert der Lamport OTSS einen großen Platz für die Ablage der authentischen Informationen, wenn viele Nachrichten signiert werden. In diesem Kapitel stellen wir

4 MSS Merkle Signatur Schema

ein leistungsfähiges One-Time Signatur Schema Winternetz OTSS vor, um eine Nachricht zu signieren.

Anwendung der One-Time Signatur One-Time Signatur Schema, OTSS soll im Folgenden zunächst kurz erläutert werden.

Schlüssel Erzeugung: Für eine OTSS Signatur werden pro zu signierendem Bit der Nachricht zwei Zufallszahlen $x_1(0), x_2(1)$ generiert. Die Folge dieser Zahlen ist der geheime Signaturschlüssel X des OTSS. Daraus werden die Hash-Werte $y_1 = H(x_1), y_2 = H(x_2)$ berechnet. Die Folge dieser Werte ist der Verifikationsschlüssel Y .

Signatur Erzeugung: Für jedes Bit i der Nachricht m wird eine Signatur s_i wie folgt gebildet.

$$s_i = \begin{cases} x_{i,0}, & \text{falls } m_i = 0 \\ x_{i,1}, & \text{falls } m_i = 1 \end{cases}$$

Verifikation Erzeugung: Für jeden Wert i wird die Signatur s_i überprüft, ob

$$\begin{cases} H(s_i) = y_{i,0}, & \text{falls } m_i = 0 \\ H(s_i) = y_{i,1}, & \text{falls } m_i = 1 \end{cases}$$

4.1.1 Lamport One-Time Signatur Schema

Lamport erfand zuerst ein One-Time digital Signatur Schema, das auf One-Way Funktion $Y = H(X)$ beruht [13].

Definition: Sei

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^n$$

eine Hash-Funktion, die n Bit Hash-Werte erzeugt. Für Lamport OTSS werden $2 \cdot n$ Zufallszahlen

$$x_1(0), x_1(1), x_2(0), x_2(1), \dots, x_n(0), x_n(1) \in \{0, 1\}^n$$

generiert. Die Folge dieser Zahlen bildet den geheimen Signaturschlüssel X des Lamport OTSS. Daraus werden die Hash-Werte

$$y_{i,j} = H(x_{i,j}), \text{ mit } 1 \leq i \leq n \text{ und } j \in \{0, 1\}$$

4 MSS Merkle Signatur Schema

berechnet. Die Folge dieser Werte bildet den Verifikationsschlüssel Y . Für die zu signierende Nachricht d wird der Hash-Wert

$$H(d) = (h_1, \dots, h_n) \in \{0, 1\}^n$$

berechnen. Lamport OTSS der Nachricht d ist die Folge

$$s = (b_1, \dots, b_n) = (x_1(h_1), x_2(h_2), \dots, x_n(h_n))$$

. Jedoch erfordert das Lamport One-Time Schema einen großen Platz für die Ablage der authentischen Informationen, wenn viele Nachricht signiert werden. Das Lamport One-Time Signatur Schema ist zwischen einem einzelnen Paar Benutzern praktisch, die bereit sind, um die großen Mengen von Daten austauschen zu können. Lamport OTSS ist nicht praktisch für die meisten Anwendungen wie Wireless Sensor Networks WSN und $RFID$ -Tag. Im $RFID$ -System könnte ein Signatur System für n mögliche Nachrichten mit der Länge s bits wie folgt entworfen werden.

$$\begin{pmatrix} Y_1 = y_{1,1}, y_{1,2}, \dots, y_{1,2 \cdot s} \\ Y_2 = y_{2,1}, y_{2,2}, \dots, y_{2,2 \cdot s} \\ \vdots \\ Y_n = y_{n,1}, y_{n,2}, \dots, y_{n,2 \cdot s} \end{pmatrix}$$

Das $RFID$ -Tag muss

$$y_{i,j} = H(x_{i,j})$$

berechnet und das $x_{i,j}$ als Zufall gewählt werden. Jedoch vor dem Verwenden dieser Methode, müssen das Lesegerät und der $RFID$ -Tag darin übereinstimmen, dass

$$Y = Y_1, Y_2, \dots, Y_n$$

für eine Signatur verwendet werden sollte, und das Lesegerät wird Y kopieren müssen. Wenn jede $y_{i,j}$ nur (bei Anwendung SHA-1) 160 bits lang ist, wenn $s = 160$, $n = 1000$ dann beträgt

$$Y = n \cdot 2 \cdot s \cdot 160 = 1000 \cdot 2 \cdot 160 \cdot 160 = 512,00,000$$

bits oder 6,4 Megabytes. Pro zu signierendem Bit der Nachricht macht die Anwendung der Lamport One-Time Signatur auf den $RFID$ -Tag unmöglich. Die Speicherkapazität der Lamport Signatur Generation ist zu hoch. Das Tag muss für eine Nachricht digest $H(m) = (h_1, \dots, h_n)$, n Signature erzeugen, so dass das Tag die Signatur

$$\sigma = (\sigma_1, \dots, \sigma_n) \text{ wobei } \sigma_i = x_i(h_i)$$

berechnen muss.

Bei der Anwendung des Algorithmus SHA-1; $n = 160$ ist die Signatur

$$\sigma = (\sigma_1, \dots, \sigma_{160}) \text{ wobei } \sigma_i \in \{0, 1\}^n$$

Die Speicherkapazität, die der Tag im EEPROM benötigt, um eine Signatur zu sichern, beträgt $160 \cdot 160 = 25600$ bits. Das heißt, dass Lamport OTSS unflexibel für $RFID$ -Tag ist.

4.1.2 Winternitz One-Time Signatur Schema

Wir stellen das Winternitz One-Time Signatur Schema, eine neue One-Time Signatur vor, das One-Way Funktion ohne trapdoors wie SHA-1 verwendet [15]. Winternitz OTSS kennzeichnet niedrige authentische Informationen mit verhältnismäßig kleiner Signaturgröße. Im Vergleich zu anderen One-Time Signatur Schemata, erzeugt das Winternitz OTSS eine kleine Signatur und ist so schnell zu verifizieren. Auf der Unterseite ist der öffentliche Schlüssel von Winternitz OTSS nicht so groß; 160 bits beträgt es durch die Verwendung einer Hash-Funktion wie SHA-1. Die Signaturgröße ist kleiner als das vorhergehende Schema, das auf One-Way Funktion ohne trapdoors basierte. Ein ideales Signierungs-Authentifizierungs-Protokoll sollte für den Absender und den Empfänger effizient sein. Die Größe des Overheads, das die Verifizierungsinformationen hat, muss klein sein.

Die Sicherheit des Winternitz OTSS kommt von der Schwierigkeit des Findens von k Kollision für eine One-Way Funktion. Winternitz OTSS liefert eine Signatur und ist schneller als Lamport OTSS, in der Signierung und Authentifizierung einer Nachricht. Winternitz OTSS bietet superschnelle Authentifizierung an. Es wird geglaubt, dass er heute die schnellste Signatur liefert. Bevor das Merkle Signatur Schema MSS beschrieben wird, wird das Winternitz OTSS erklärt, das bei der Anwendung des Merkle Signatur Schema MSS benutzt werden könnte. Winternitz OTSS verwendet einen Parameter w , welches die Zahl der bits, die gleichzeitig verarbeitet werden müssen bezeichnet.

Wir beschreiben das Winternitz OTSS.

- Schlüsselpaar Generation.
- Signatur Generation.
- Signatur Verifikation.

Winternitz OTSS Schlüsselpaar Generation: Die Erzeugung des Winternitz OTSS Schlüsselpaars (X, Y) , und der X Signaturschlüssel (private Schlüssel) ist Reihenfolge des OTSS Signaturschlüssels, Y ist der Verifikationsschlüssel.

Algorithmus 1 beschreibt die Generation des Winternitz OTSS Schlüsselpaars.

Systemparameter

1. Hash-Funktion $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$
2. $t_w = \lceil n/w \rceil + \lceil (\lfloor \log_2 \lceil n/w \rceil \rceil + 1 + w)/w \rceil$ wobei $w \in \mathbb{N}$

4 MSS Merkle Signatur Schema

Ausgabe Signaturschlüssel X , Verifikationsschlüssel Y

1. wähle zufällig

$$x_1, \dots, x_{t_w} \in_R \{0, 1\}^n$$

2. setze

$$X = (x_1, \dots, x_{t_w})$$

3. berechne

$$y_i = H^{2^w - 1}(x_i) \forall i \in \{1, \dots, t_w\}$$

4. berechne

$$Y = H(y_1 \parallel \dots \parallel y_{t_w})$$

wobei \parallel die Konkatenation ist

5. return (X, Y)

Winternitz OTSS Signatur Generation: Es wird die Berechnung des Winternitz OTSS für eine Nachricht d mit dem Schlüsselpaar (X, Y) . Das Winternitz OTSS $sign(d)$ wird durch den Signaturschlüssel X berechnet.

Algorithmus 2 beschreibt die Generation des Winternitz OTSS.

Eingabe: Nachricht d , Signaturschlüssel X

Ausgabe: One-Time Signatur σ von d

1. Berechne den Hash-Wert $H(d)$ von der Nachricht d
2. splite die Binärdarstellung von $H(d)$ in $\lceil n/w \rceil$ Blöcke $b_1, \dots, b_{\lceil n/w \rceil}$ der Länge w , und padde $H(d)$ mit Nullen von Links wenn nötig.
3. Wandele die b_i -s Blöcke in Dezimalzahlen und berechne die Summe

$$C = \sum_{i=1}^{\lceil n/w \rceil} 2^{w \cdot i} - b_i$$

4. splite die Binärdarstellung von C in

$$\lceil (\lceil \log_2 \lceil n/w \rceil \rceil + 1 + w) / w \rceil$$

Blöcke

$$b_{\lceil n/w \rceil}, \dots, b_{t_w}$$

der Länge w und padde C mit Nullen von links wenn nötig

4 MSS Merkle Signatur Schema

5. Wandle die b_i -s Blöcke in Dezimalzahlen und berechne

$$\sigma_i = H^{b_i}(x_i) \forall i \in \{1, \dots, t_w\}$$

6. return $\sigma = (\sigma_1, \dots, \sigma_{t_w})$

Winternitz OTSS Verifikation Generation Um ein Winternitz One-Time Signatur verifizieren zu können, verifiziert der Verifier mit dem Verifikationsschlüssel Y die Signatur σ . Wenn diese Verifikation ausfällt, weist der Verifizierer die Signatur σ als Invalid zurück.

Algorithmus 3 beschreibt das Winternitz OTSS Verifikation Generation.

Eingabe: Nachricht d , Signatur $\sigma = (\sigma_1, \dots, \sigma_{t_w})$, Verifikationsschlüssel Y

Ausgabe: TRUE wenn die Signatur gültig ist, sonst FALSE

1. berechne die b_1, \dots, b_{t_w} wie in Algorithmus 2
2. berechne $\phi_i = H^{2^w - 1 - b_i}(\sigma_i) \forall i \in \{1, \dots, t_w\}$
3. berechne $\phi = H(\phi_1 \parallel \dots \parallel \phi_{t_w})$
4. wenn $\phi = Y$ ist die Ausgabe **TRUE** sonst **FALSE**

4.2 Merkle Signatur Schema

Die Notwendigkeit für schnelle und niedrige Berechnungen sowie die Authentifizierungs Lösungen für schwächere Applikation wie das *RFID*-Tag sind das Ziel. Perrig stellt ein neues One-Time Signatur Schema dar [7], das auf eine sehr schnelle Verifikationszeit, Signaturzeit und auf kleine Schlüsselgrößen abzielt. Reyzin stellt ein anderes Schema vor, das über schnellere Signatur und Verifikationszeiten verfügt [14]. Das neue Schema schlägt die Wiederverwendung mit einem gleichen Schlüssel mehr als einmal vor. *Several-Times Signatures* signieren einige Nachrichten mit den verschiedenen Schlüsseln, um eine angemessenere Sicherheit zu schaffen. Folglich bleibt die Effizienz und die Verwaltung von Schlüsselerzeugung im One-Time Signatur weiterhin ein ungelöstes Problem.

Merkle Signatur Schema wurden von Ralph Merkle 1979 beschrieben. Im Gegensatz zu allen anderen Signaturverfahren wie RSA, ECDSA beruht ihre Sicherheit nicht auf ein zahlen-theoretisches, algebraisches oder geometrisches Problem, das schwer lösbar ist. Benötigt wird ausschließlich, was andere Signaturverfahren ebenfalls voraussetzen:

- eine sichere kryptographische Hash-Funktion.

4 MSS Merkle Signatur Schema

- ein sicherer Zufallszahlengenerator.

Jede neue Hash-Funktion führt zu einem neuem Signaturalgorithmus, das das Problem der langfristigen Verfügbarkeit digitaler Signaturverfahren zu lösen versucht. Merkle verwendet in seiner Konstruktion so genannte One-Time Signatur Schema OTSS.

4.2.1 MSS Schlüsselpaare Generation

Die Idee des Merkle Signatur Schemas ist die Gültigkeit vieler Verifikationsschlüssel mittels eines Hash-Baums auf die Gültigkeit eines einzelnen öffentlichen Schlüssels zurückzuführen. Bei der Schlüsselgenerierung muss als Erstes die Maximalzahl möglicher Signaturen 2^h , wobei h die Höhe des Hash-Baums ist, festgelegt werden, die mit dem erzeugten öffentlichen Schlüssel verifizierbar sein sollen. Dann werden 2^h OTSS Schlüsselpaare

$$(X_i, Y_i) \forall i \in \{1, \dots, 2^h\}$$

generiert. Der private MSS Schlüssel ist die Folge aller X_i (Signaturschlüssel), und Y_i sind die (Verifikationsschlüssel).

Um den öffentlichen MSS Schlüssel zu bestimmen, wird ein binärer Authentifizierungsbaum konstruiert (siehe Abbildung 4.1). Von jedem Verifikationsschlüssel Y_i wird der Hash-Wert $H(Y_i)$ berechnet, der als Blatt des Authentifizierungsbaums dient. Jeder innere Knoten des Baums (einschließlich Wurzel) ist der Hash-Wert der Konkatenation seiner beiden Kinder

$$parent = H(left || right)$$

Der öffentliche MSS Schlüssel ist die Wurzel des Authentifizierungsbaums [22].

4 MSS Merkle Signatur Schema

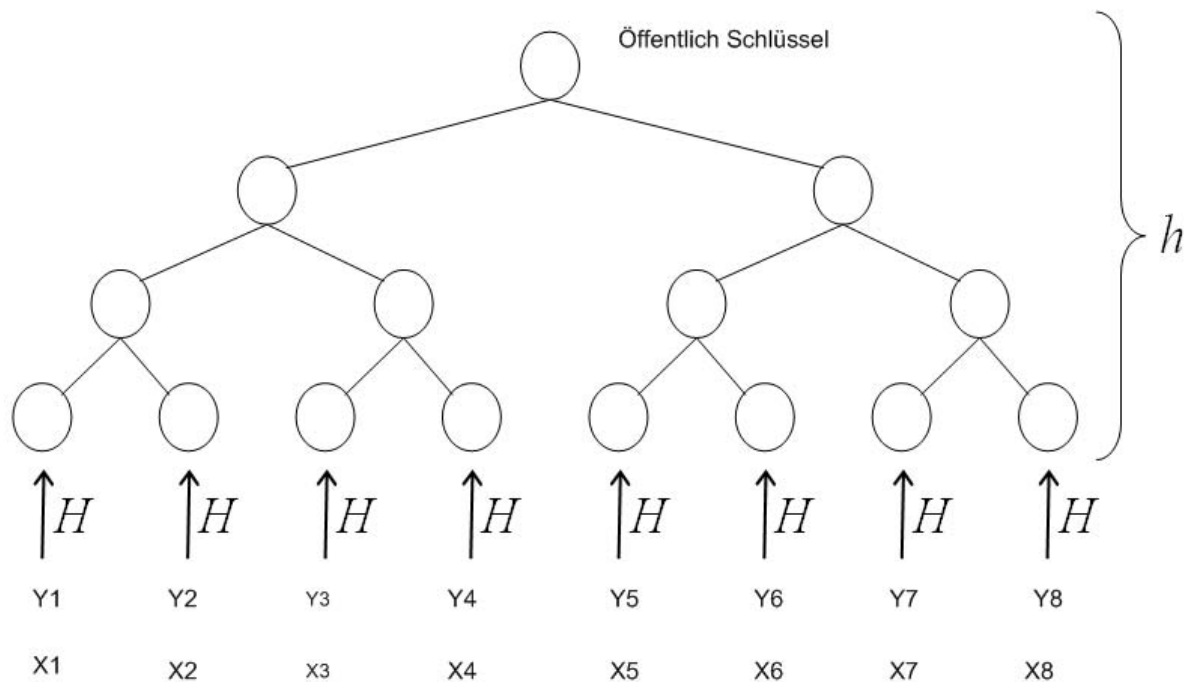


Abbildung 4.1: Merkle-Baum

4.2.2 MSS Signatur Generation

Beim Signieren werden die OTSS Schlüssel nacheinander benutzt. Die Signatur einer Nachricht d_i , welche unter Verwendung des i -ten Schlüsselpaars (X_i, Y_i) erzeugt wurde, enthält ein 4-Tupel

$$(i, Y_i, \sigma_i, Auth_i)$$

also den Index i , den i -ten Verifikationsschlüssel Y_i und die OTSS Signatur σ_i von d_i , die mit dem i -ten Signaturschlüssel X_i erstellt worden ist. Das vierte Element ist der sogenannte Authentifizierungspfad $Auth_i$. Für den Verifikationsschlüssel Y_i ist die Folge hier a_h, \dots, a_1 der Geschwister aller Knoten auf dem Pfad vom i -ten Blatt des Authentifizierungsbaums zur Wurzel. Der erste Knoten in dieser Folge ist das Blatt, das den gleichen Vorgänger hat wie i und sich vom i -ten Blatt unterscheidet (siehe Abbildung 4.2). Ein Beispiel ist einen Authentifizierungspfad bei $h = 2$. Hier ist der Authentifizierungspfad für Y_2 die Folge $Auth_2 = (a_2, a_1)$.

4 MSS Merkle Signatur Schema

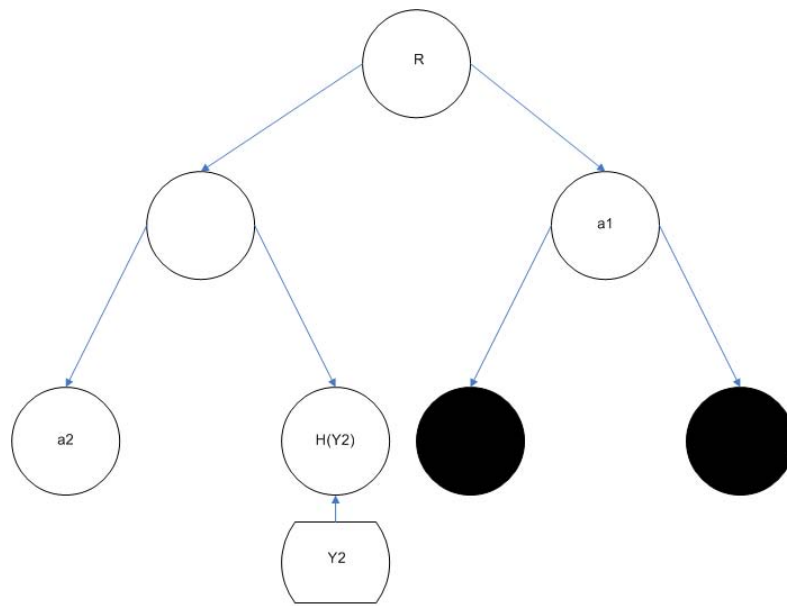


Abbildung 4.2: Merkle-Authentifizierungspfad

4.2.3 MSS Signature Verifikation

Um das MSS Signatur $(i, Y_i, \sigma_i, Auth_i)$ zu verifizieren, überprüft der Verifier zuerst das OTSS Signatur σ_i mit dem Verifikationsschlüssel Y_i . Danach wird dieser Verifikationsschlüssel selbst mit Hilfe des Authentifizierungspfads überprüft, indem der Pfad vom i -ten Blatt $H(Y_i)$ zur Wurzel konstruiert wird. Ist der letzte Knoten in diesem Pfad gleich dem öffentlichen MSS Schlüssel, so ist die Signatur gültig. In dem Beispiel aus der Abbildung 4.2. wäre die Signatur anerkannt, falls öffentlich Schlüssel = $H(H(H(Y2)||a_2)||a_1)$ gelten würde.

4.3 GMSS Merkle Signatur Schema

Einleitung: Es wurde bereits herausgefunden, dass für ein $h \in \mathbb{N}$ könnte MSS $N = 2^h$ Nachrichten durch Anwendung 2^h Schlüsselpaare von One-Time Signatur Schema signiert werden könnte. Für $N > 2^{25}$ ist das Merkle Signatur Verfahren ineffizient, weil der geheime Schlüssel zu groß ist und die Schlüsselerzeugung zu lang dauert. In den spezifischen Anwendungen wie *RFID*-Tag wird das MSS Verfahren ineffizient, weil der Platzbedarf, um alle Signaturschlüssel X_i speichern zu können, viel größer ist als die *RFID*-Tag Speicherkapazität. Das MSS Verfahren wurde entwickelt, um sehr effizient für alle Applikation zu sein. Das GMSS One-Time Signatur Schema ist eine Verbesserung des MSS One-Time Signatur Schemas, so dass die Zeit

4 MSS Merkle Signatur Schema

für die Schlüsselerzeugung und die Größe des geheimen MSS Schlüssels drastisch reduziert wird.

Idee: GMSS-Authentifizierungsbaum besteht aus einem Baum mit T Schichten. Die Knoten von diesem Baum sind der Reihenfolge von Merkle Bäume. Die Höhe aller Merkle Bäume in einer bestimmten Schicht i ist bezeichnet durch h_i (siehe Abbildung 4.3).

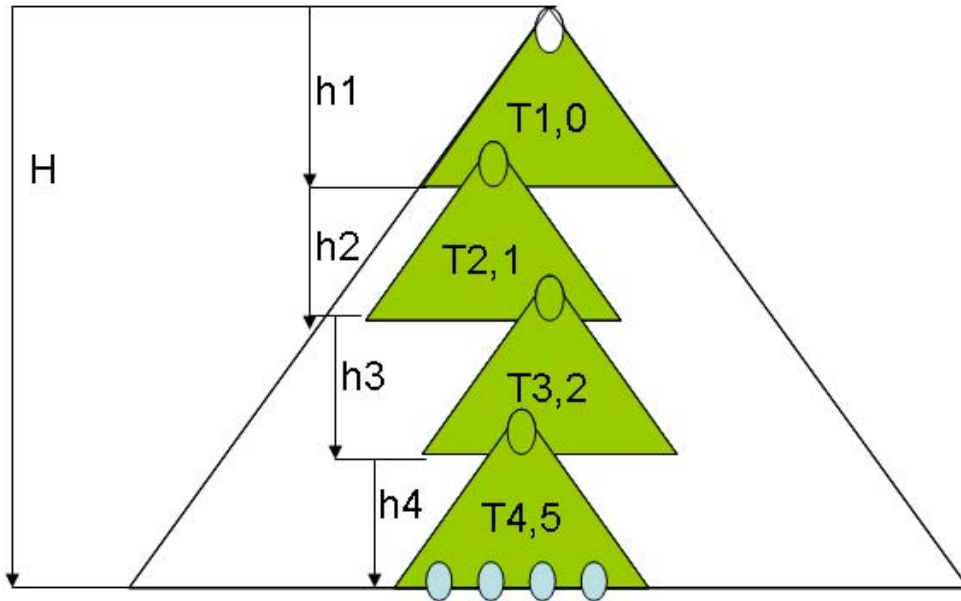


Abbildung 4.3: GMSS-Baum

Die Abbildung zeigt uns eine GMSS-Baum mit $T = 4$ Schichten. Jede Schicht hat einen MSS-Baum mit eine höhe h_i . Bäume in den unterschiedlichen Schichten können unterschiedliche Höhen haben. Jeder MSS-Baum in Schicht $i = 1, \dots, T-1$ ist Vater zu 2^{h_i} Merkle Bäumen. $\tau_{1,0}$ bezeichnet den einzelnen Merkle-Baum in Schicht $T = 1$. Die Zahl der Merkle-Bäumen in den Schichten

$$i = 2, \dots, T \text{ ist } 2^{h_1 + \dots + h_{i-1}}$$

und werden von

$$\tau_{i,j}, j = 0, \dots, 2^{h_1 + \dots + h_{i-1}} - 1$$

entsprechend ihrer Position von links nach rechts bezeichnet.

GMSS-Baum kann $N = 2^{h_1 + \dots + h_T}$ Nachrichten signieren. Für diese Ziel, die Wurzel eines Merkle-Baums in der Schicht i ist $ROOT_{\tau_{i,j}}$. $\text{Sign}(\tau_{i,j})$ ist ein One-Time Signatur von $ROOT_{\tau_{i,j}}$. $\text{Sign}(d_i)$ ist ein One-Time Signatur einer Nachricht d_i , die bei der Verwendung Blätter der Merkle-Bäumen auf der tiefsten Schicht T signiert wird (siehe Abbildung 4.4).

4 MSS Merkle Signatur Schema

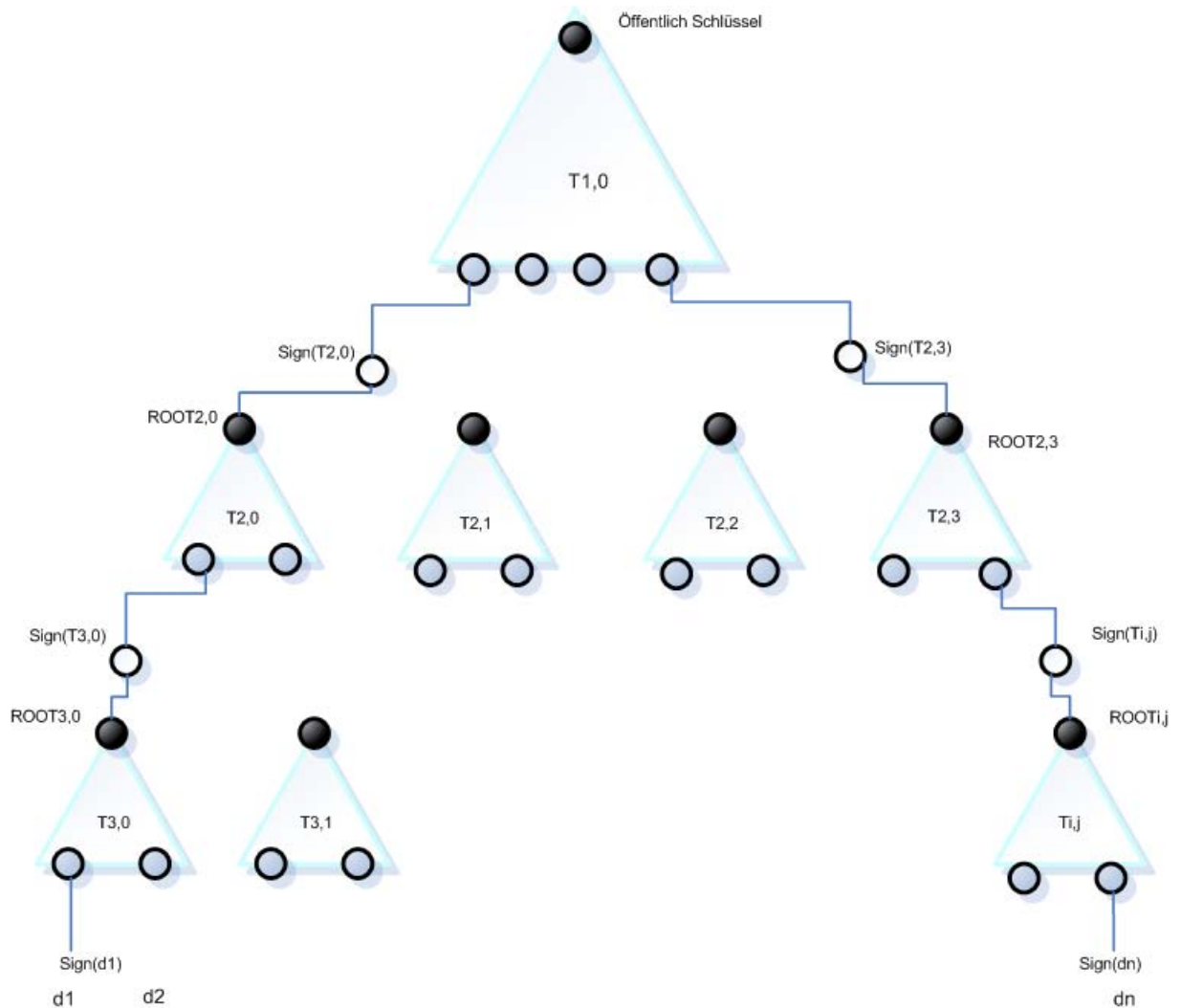


Abbildung 4.4: Konstruktion-GMSS

Der öffentliche Schlüssel ist eine Wurzel des Merkle-Baums in der Schicht $T = 1$. Die Nachrichten werden durch Blätter des MSS-Baums signiert. Die Wurzel der Bäume sind nicht der öffentliche Schlüssel für diese Signatur (siehe Abbildung 4.5).

4 MSS Merkle Signatur Schema

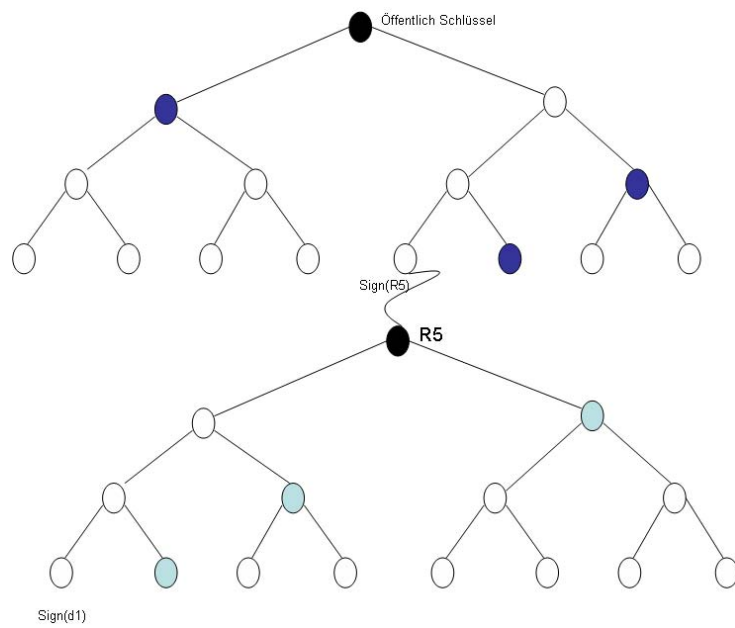


Abbildung 4.5: GMSS Signatur

Die Abbildung 4.5 zeigt uns die GMSS Signatur für eine Nachricht $Sign(d_1)$ und seinen Authentifizierungspfad. Nachdem erstes $2^{h_2} = 2^3 = 8$ Signaturen in der Schicht $T = 2$ erzeugt wurde, wird ein neuer Merkle-Baum konstruiert, der auch zur Erzeugung des nächsten neuen 8 Signaturen benutzt wird. In der Schicht $T = 1$ wird dem nächsten Blatt $i = 6$ verwendet (siehe Abbildung 4.6).

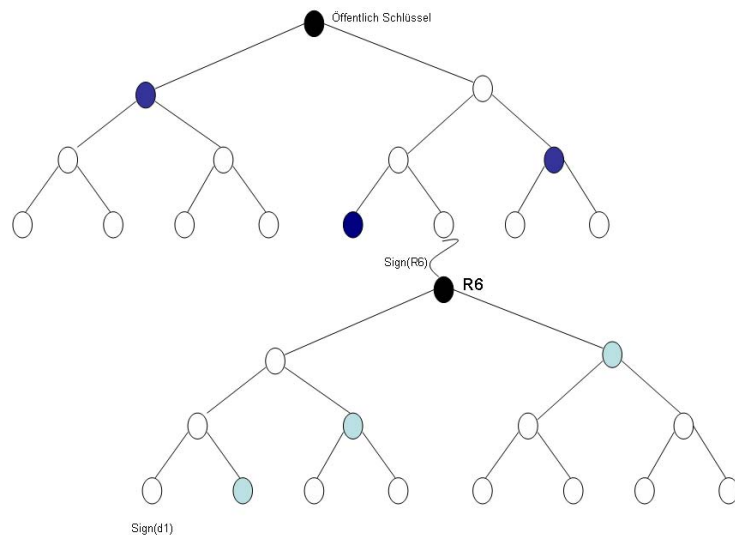


Abbildung 4.6: GMSS-Baum nach 2^3 Signaturen

4 MSS Merkle Signatur Schema

Für gegebene Signatur $s \in 1, \dots, 2^{h_1+\dots+h_T}$, gibt es einen einzigartigen Pfad P vom Merkle-Baum $\tau_{i,j}$ von der Schicht T zum einzelnen Merkle-Baum $\tau_{1,0}$ in der oberen Schicht $T = 1$. Dieser Pfad bezieht einen Merkle Authentifizierungsbaum an jeder Schicht $i = 1, \dots, T$.

Eine GMSS Signatur enthält die einmalige Signatur $\text{Sign}(d_i)$ einer Nachricht digest d_i und die einmalige Signatur $\text{Sign}(\tau_{i,j})$ der Wurzeln aller Merkle-Bäume auf dem Pfad P , außer $\tau_{1,0}$ Merkle-Baum in der Schicht $T = 1$. Für alle Bäume $\tau_{i,j}$ auf Pfad P , enthält eine GMSS Signatur auch den Authentifizierungspfad $\text{Auth}_{\tau_{i,j}, l}$ des Blattes l , der benutzt werden könnte, um die Nachricht d_l signieren zu können. Ein Authentifizierungspfad $\text{Auth}_{\tau_{i,j}, l}$ wird definiert, während die Reihenfolge der Geschwister aller Knoten auf dem Pfad von Blatt l zur Wurzel des Merkle-Baums $\tau_{1,0}$ aufsteigt (siehe Abbildung 4.7).

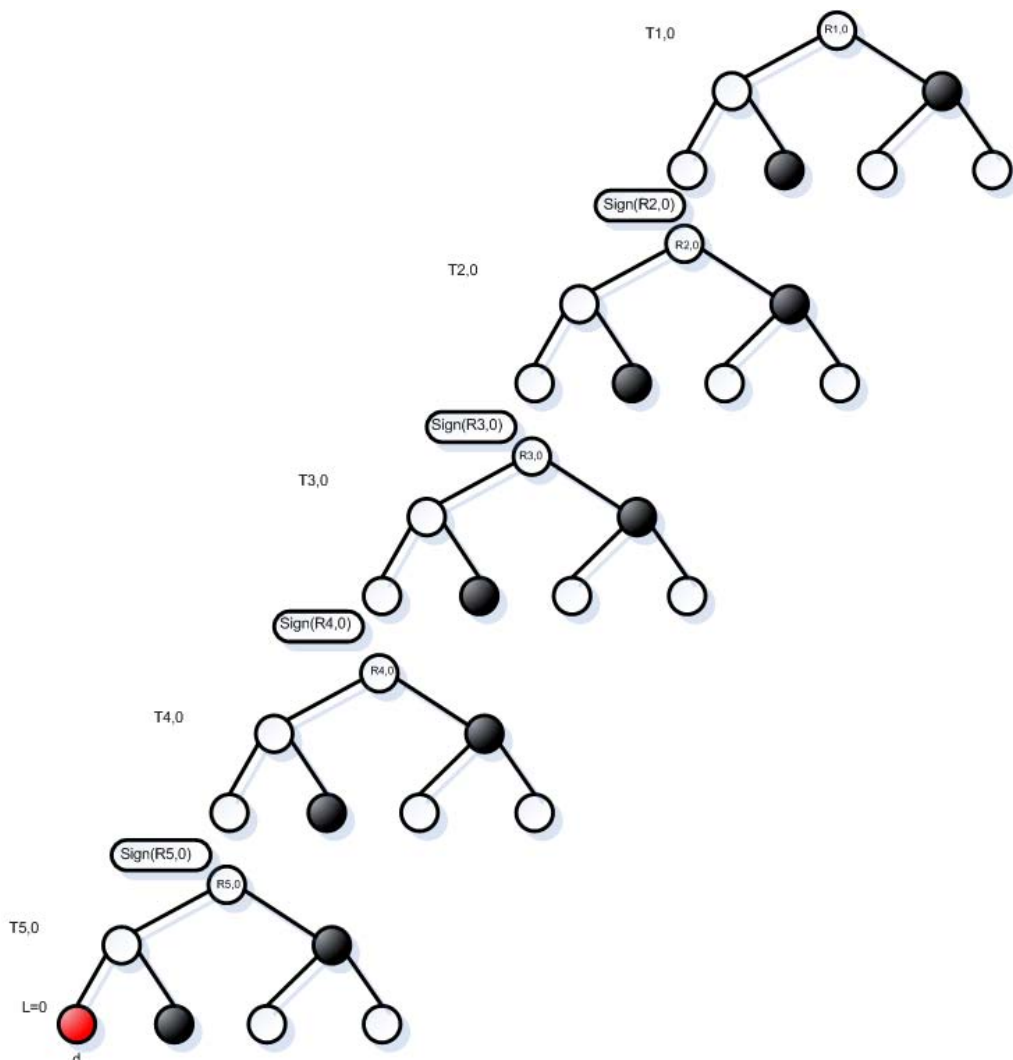


Abbildung 4.7: GMSS-Authentifizierungspfad

4 MSS Merkle Signatur Schema

Die Abbildung 4.7 erklärt uns ein Beispiel für einen GMSS-Authentifizierungsbaum und seinen Authentifizierungspfad, so dass die Zahl der Schichten $T = 5$ und jeder MSS-Baum in jede Schicht die Höhe $h = 2$ hat. Die Anzahl der Signatur ist gleich 2^{10} . Der rote Knoten bezeichnet die aktuelle Nachricht d , die wir zu signieren versuchen. Die Reihenfolgen der schwarzen Knoten sind Authentifizierungspfade für die Nachricht d .

4.3.1 GMSS Schlüsselpaare Generation

Es ist nicht praktisch, wenn wir die ganze GMSS privaten Schlüssel, Signaturschlüssel im *RFID*-Tag speichern. Um die GMSS privaten Schlüssel X_i kleiner machen zu können, um viel Speicher zu sparen, werden die OTSS Signaturschlüssel x_i mit *Pseudo Random Number Generation PRNG* erzeugt. *PRNG* ist eine Funktion, die auf eine Hash-Funktion beruht. Die Eingabe des *PRNG* ist eine Blockgröße n bits und die Ausgabe sind zwei Blöcke jeweils mit n bits.

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^n \times \{0, 1\}^n$$

. Das Seed der *PRNG*-Funktion wird nur im EEPROM für GMSS One-Time Signatur Schema gespeichert, um den GMSS Signaturschlüssel erzeugen zu können. In dem nächsten Kapitel werden wir die Größe des Seeds 160 bits sehen, welches man im *RFID*-Tag speichern kann. Algorithmus 6 beschreibt die Bildung der GMSS Schlüsselpaare [16]. Der Algorithmus verwendet zwei Subroutinen, die in Algorithmen 4 und 5 beschrieben werden [16].

4.3.2 GMSS Signatur Generation

GMSS Signatur Generation wird in verschiedene Teile durchgeführt.

- Zuerst wird die MSS Signatur der Nachricht d mit dem MSS-Baum $\tau_{i,j}$ in der letzter Schicht i erzeugt.
- Dann wird die MSS Signatur von der Wurzel des MSS-Baumes $\tau_{i,j}$, bei Anwendung eines Blatters des nächstem MSS-Baums $\tau_{i-1,j}$ in der Schicht $i - 1$ berechnet, und so weiter und so fort bis den MSS-Baum $\tau_{1,0}$ in der Schicht $T = 1$ ist.

Algorithmus 8 beschreibt die Bildung der GMSS Signatur Generation [16]. Dieser Algorithmus verwendet alle vorherige Subroutinen, die in Algorithmen 2, 4, 5 beschrieben werden [18].

4.3.3 GMSS Signatur Verifikation

GMSS signatur Verifikation fährt in zwei Schritten fort.

4 MSS Merkle Signatur Schema

- Zuerst werden alle Authentifizierungspfade für jeden MSS-Baum validiert.
- Dann wird die Gültigkeit aller One-Time Signaturen der $ROOT_{\tau_i,j}$ verifiziert.

Die Details der GMSS Signatur Verifikation wird in Algorithmus 9 beschrieben [16].

5 GMSS Signatur Generation für RFID-Tags

5.1 Einleitung

Der beste Weg eine Authentifizierung auf ein *RFID*-Tag einführen zu können ist, ein kryptographier Algorithmus hinzuzufügen. Momentan gibt es keine absicherte Verfahren, die Kryptographische Authentikation für *RFID*-Tag anwenden können. Dies ist wegen der niedrigen Speicherkapazität und der Anforderung des Niedrigleistungsverbrauchs ausgeschlossen. Asymmetrische Authentifizierungsverfahren wie *RSA* oder *ECDSA* sind bis heute sichere Verfahren. Aber es ist zu schwer, sie auf *RFID*-Tag implementieren zu können, da das Finden einer großen Primzahl sehr komplex ist. Außerdem ist der Euklidische Algorithmus zur Bestimmung des $\text{ggT}()$ und die Berechnung des Potenzierens von $\text{mod } n$ mehr als die Fähigkeit des Mikroprozessors eines *RFID*-Tags. One-Time Signatur wird in *High-Performance-Application* verwendet. One-Time Signatur Schema OTSS ist eine digitale Signatur, welche auf der One-Way Funktion beruht. One-Way Funktionen sind einfache Verfahren zur Implementierung und effizienten Berechnen der *RSA* digital Signatur. Das GMSS Merkle Signatur Schema ist eine gute alternative für das Authentifizierungsverfahren, das auf dem *RFID*-Tag implementiert werden kann. Dabei besorgt dann das GMSS Signatur Schema eine schnellere Signatur als das *RSA* Signatur Schema und die Signaturgröße nur einige kilobytes. Die Implementierung des GMSS Verfahrens auf dem *RFID*-Tag fordert niedrigen Speicher und einen Mikroprozessor mit kleiner Fähigkeit, so dass er die Grund-Operationen, die die GMSS Signatur braucht, durchführen kann.

5.2 Eigenschaften des RFID-Tags für GMSS

In unserer Umgebung gibt es verschiedene Arten von *RFID*-Tags, die jeweils einer Speicherkapazität klassifiziert werden. Einige Tags haben EEPROM Speicher mit 128 bit und andere mit 2 kbyte. In dieser Zeit haben wir einige Arten von *RFID*-Tags, die über eine große Speicherkapazität EEPROM von bis zu 4 kbyte verfügen. Es wird *mifare MF1 ICS70* Tag 4 kbyte als Beispiel angeboten [30]. Das GMSS Merkle Signatur Schema braucht einen Speicher wie alle vorherige Algorithmen, um seine Signatur zu berechnen und zu speichern.

5.3 Optimal Parameter group

Um das GMSS Merkle Signatur Schema auf das *RFID*-Tag anwenden zu können, werden die zu den *RFID*-Tagsfähigkeiten passenden Parameter der GMSS benötigt. Das GMSS One-Time Signatur Schema beruht auf verschiedene Parameter. Es benutzt sie zur Erzeugung von Signaturen. Dabei werden Analysen von praktischen Parametern für das GMSS Signatur Schema vorgesehen und dabei aufgezeigt welche, Parameterwerte, die schnelle Signaturzeit und passende Signaturgröße produzieren. Dieser Abschnitt beschreibt alle Parameter der GMSS-Implementation, die mit dem *RFID*-Tag zusammenpasst, so dass das Tag eine Nachricht signieren kann.

5.4 Auswahl der Hash-Funktion

Für die Implementierung des GMSS Signatur Schemas, müssen einige One-Time-Signaturen ausgewählt werden, sowie eine passende Hash-Funktion. Die Hash-Funktionen, die im GMSS signatur Schema verwendet werden, stellen die Leistung die Geschwindigkeit und die Signaturgröße, sowie seine Sicherheit fest. Es gibt verschiedene Hash-Funktionen mit verschiedenen Größen der Ausgabe. Es werden SHA-1 als Hash-Funktion ausgewählt. SHA-1 wechselt die Nachricht zu $n = 160$ bits.

$$H : \{0, 1\}^* \longrightarrow \{0, 1\}^n$$

Das *RFID*-Tag hat EEPROM mit nur 4 kByte, dabei wird eine Aufteilung der 4 kByte unternommen, die der SHA-1 zusagt. Das bedeutet, dass die 4 kByte in viele Blöcke aufgeteilt werden und jede Blockgröße gleich $n = 160$ bits ist. Somit sind die 4 kByte = $(32000/160) = 200$ Blöcke. Jeder Wert der im GMSS berechnet wird, braucht einen Block, welcher 160 bits entspricht, zum speichern.

5.5 Auswahl der One-Time Signatur

One-Time Signatures sind digitale Signatur Schemata, die das *RFID*-Tag ermöglichen, um eine Nachricht mit einem gegebenen Set Schlüsseln, den sogenannte Geheimschlüssels zu signieren. Um eine Nachricht signieren zu können, werden unterschiedliche Schlüssel erzeugt. Die entsprechende One-Time-Signatur in dieser Arbeit ist das Winternitz One-Time Signatur Schema. Dieses Schema zielt darauf hin, die Bandbreite der Signatur und die Kompliziertheit der Verifikation zu senken, aber erfordert dabei eine hohe Zahl von Hash-Funktionen, die zu berechnen sind. Das Winternitz OTSS beruht auf die Hash-Funktion SHA-1. Der Parameter w macht das Winternitz OTSS sehr flexibel. Die Aufgaben dieses Parameters, sind die Bestimmung der Signaturgröße und der Signatur Schlüsselpaar Generationszeit. Die folgende Gleichung beschreibt die Beziehung zwischen dem Winternitz Parameter w und der Signaturgröße.

5 GMSS Signatur Generation für RFID-Tags

$$t_w = \lceil n/w \rceil + \lceil (\lceil \log_2 \lceil n/w \rceil \rceil + 1 + w)/w \rceil$$

t_w stellt die Zahl der Geheimschlüsseln (x_1, \dots, x_{t_w}) , welche einen One-Time Signaturschlüssel

$$X = (x_1, \dots, x_{t_w})$$

erzeugen. Für eine 160 bits Nachricht digest d ist die Signatur

$$\sigma = (\sigma_1, \dots, \sigma_{t_w}) = (H^{k_1}(x_1), \dots, H^{k_{t_w}}(x_{t_w}))$$

Die Signaturgröße gleicht $t_w \cdot n$ bits, so dass die Größe des Geheimschlüssels x_i 160 bits beträgt. Um den One-Time Verifikationsschlüssel Y berechnen zu können, spielt w eine große Rolle, um die Zahl der Hash-Operationen, die zur Erzeugung von Y durchzuführen. Die nächste Gleichung zeigt uns die Zahl der Hash-Operationen an.

$$Y = H(H^{2^w-1}(x_1) \parallel \dots \parallel H^{2^w-1}(x_{t_w}))$$

Es kann gesagt werden, dass wenn der Winternitz-Parameter w groß ist, mehr bits of $H(d)$ behandelt werden. Wenn die Variable t_w klein ist dann ist die Signaturgröße zu klein und die Größe des Frams, der die Signatur Information beinhaltet zu klein und wird schnell übertragen wird. Die größten Nachteile von der Erhöhung des w Wertes sind, dass die Signatur Generation viel zu lange dauert, weil mehr Hash-Werte berechnet werden, um den Verifikationsschlüssel Y und um eine Signatur zu erzeugen durchgeführt werden müssen. (siehe Abbildung 5.1).

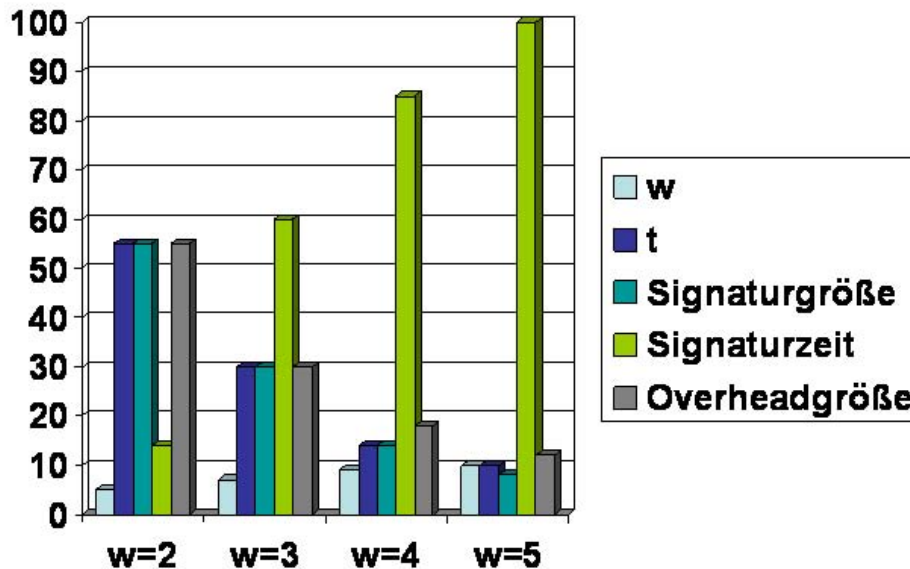


Abbildung 5.1: Beziehung zwischen w , t_w , Signaturgröße, Signaturzeit

5 GMSS Signatur Generation für RFID-Tags

Wenn das Winternitz-Parameter w klein ist, ist die Variable t_w groß, und das bedeutet auch, dass die Signaturgröße zu groß ist. Deshalb ist die Signaturübertragung zu langsam und die Overheadgröße zu groß. Der Vorteil ist, dass die Hash-Funktion nur $2^w - 1$ mal verwendet werden kann und die Signaturzeit zu kurz ist. In unserem Schema definieren wir den Wert des Winternitz One-Time Signatur als $w = 4$, so dass die Signaturgröße zu den EEPROM des *RFID*-Tags passt. Der Wert der Variable t_w ist gleich 43, das heißt die Anzahl der Geheimschlüssel $43 (x_1, \dots, x_{43})$ betragen, jeder (x_i) benötigt einen Block zu Speicherung im EEPROM, die Blockgröße beträgt 160 bits.

5.6 GMSS-Baum Parameter

Nachdem die Analyse der GMSS Merkle Signatur und der ihr zugehörigen Parameter erläutert wurde, werden zwei verschiedene Verfahren aufgezeigt, die das *RFID*-Tag zur Erzeugung einer Signatur benutzt. Das erste Verfahren, ist das GMSS Merkle Schema mit zwei Schichten $T = 2$. Die erste Schicht $T = 1$ hat einen Merkle-Baum $\tau_{1,0}$ mit der Höhe von $h_1 = 5$. Die zweite Schicht $T = 2$ hat die $2^{h_1} = 2^5 = 32$ Merkle-Bäume, die mit $\tau_{2,j} ; j = 0, \dots, 31$ gekennzeichnet werden. Jeder Merkle-Baum hat die Höhe $h_2 = 4$ (siehe Abbildung 5.2).

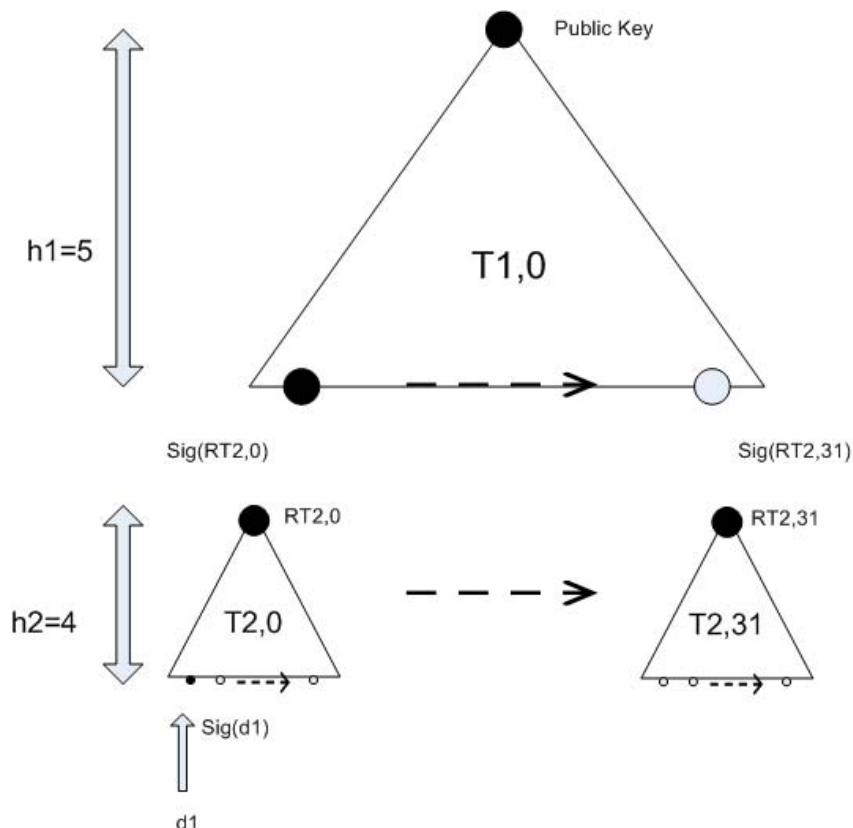


Abbildung 5.2: GMSS-Baum

5 GMSS Signatur Generation für RFID-Tags

Das *RFID*-Tag benutzt dieses Verfahren, um den GMSS-Baum aufzubauen und um $2^{h_1+h_2} = 2^9 = 512$ Nachrichten signieren zu können (siehe Abbildung 5.3). Alle Hash-Werte müssen in dem EEPROM des *RFID*-Tags gespeichert werden. Das heisst, alle Knoten und alle Blätter des GMSS-Baums müssen im EEPROM gespeichert werden.

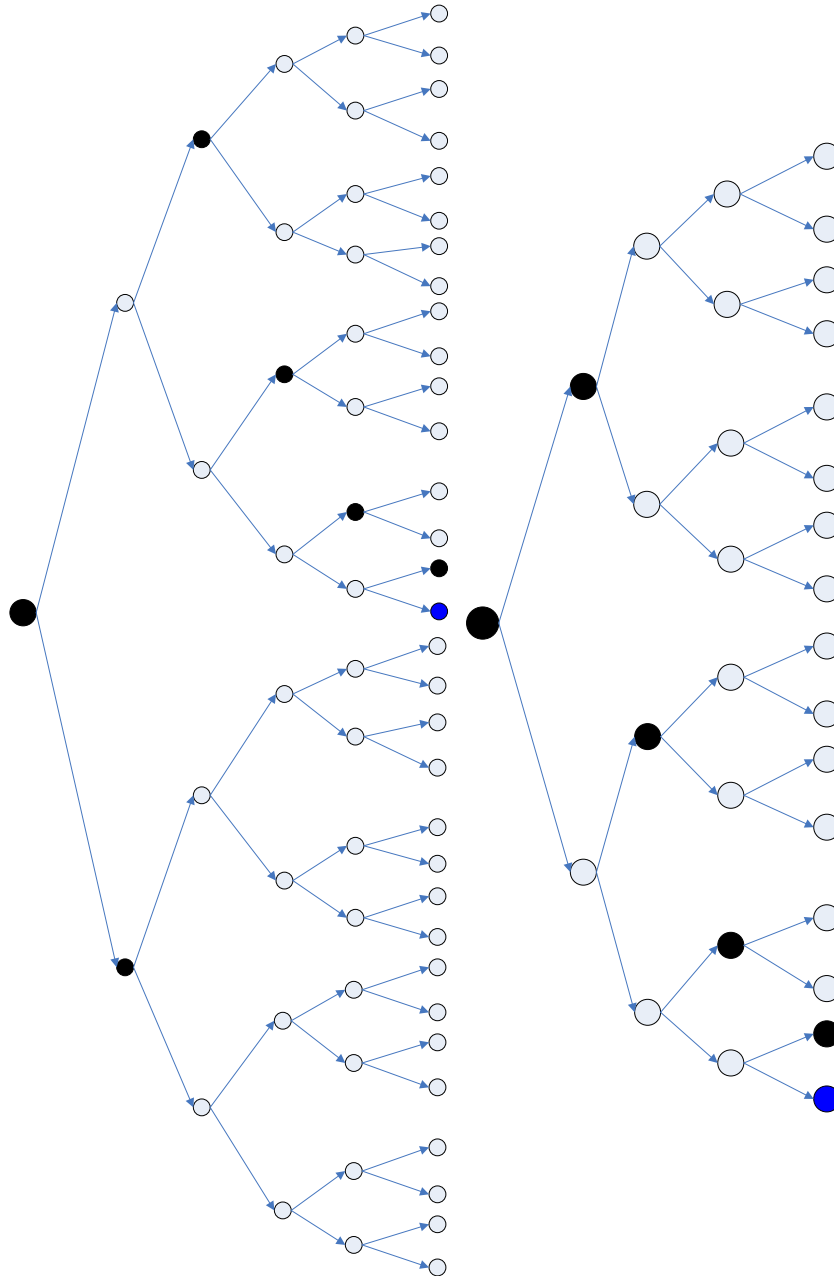


Abbildung 5.3: GMSS-Baum für RFID-Tag

5.7 Implementation GMSS Signatur auf RFID-Tag

Um das erste Verfahren auf das *RFID*-Tag anwenden zu können, müssen alle Knoten und alle Blätter des GMSS-Baums in dem EEPROM gesichert werden.

5.7.1 Anforderungsspeicher EEPROM für GMSS-Baum

Jetzt kann die Zahl der Knoten und Blätter des GMSS-Baums berechnet und ihre Größe in dem Tag bestimmt werden. Es wird $2^{h_1} = 2^5 = 32$ Blöcke für die Blätter und $2^5 - 1 = 31$ Blöcke für die Knoten in der Schicht $T = 1$ benötigt (siehe Abbildung 5.4).

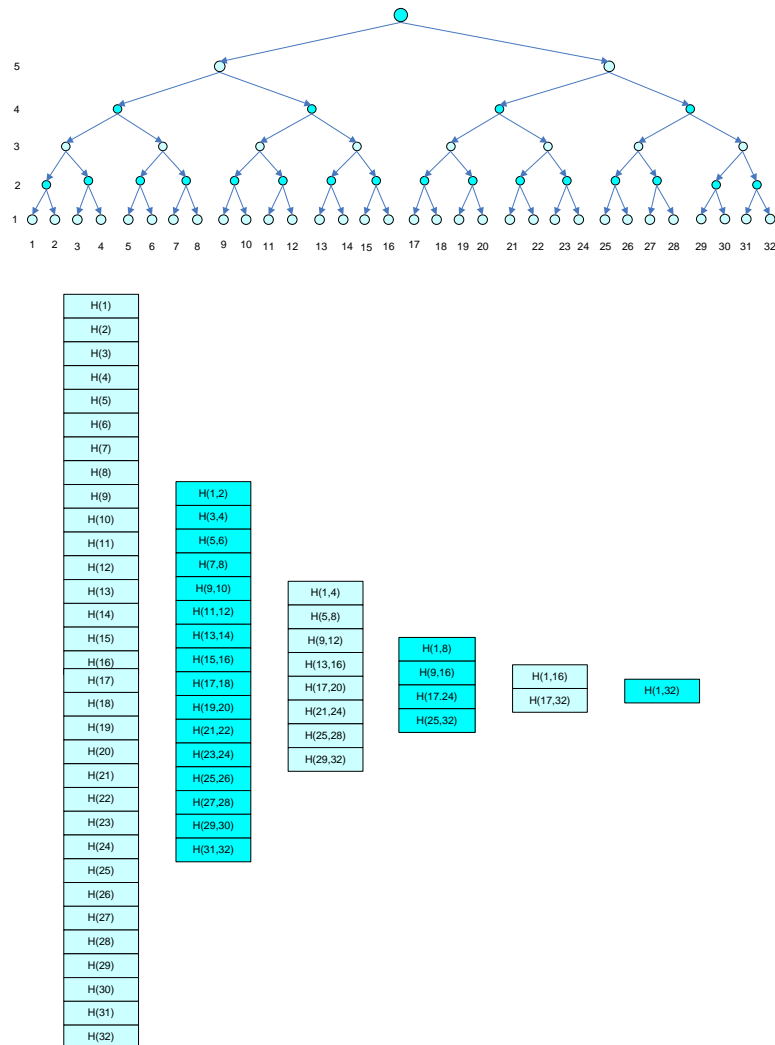


Abbildung 5.4: Merkle-Baum $\tau_{1,0}$ mit allen Blöcken

5 GMSS Signatur Generation für RFID-Tags

Um alle Knoten und Blättern Merkle-Baum speichern zu können, braucht das Tag $31 + 32 = 63$ Blöcke. Das heißt, die Speicherkapazität, die zur Speicherung des Merkle-Baums $\tau_{1,0}$ gebraucht wird, ist gleich: Die Zahl der Blöcke $\cdot 160 = 63 \cdot 160$ bits.

Die Abbildung 5.4 zeigt alle Blöcke im EEPROM, die über Hash-Werte der Blätter $H(Y_i); i = 1, \dots, 32$ und über die Hash-Werte der Knoten $H(H(Y_i) \| H(Y_i + 1))$ verfügt. Im nächsten Schritt, es wird $2^{h_2} = 2^4 = 16$ Blöcke für die Blätter und $2^{h_2} - 1 = 2^4 - 1 = 15$ Blöcke für die Knoten in der zweite Schicht $T = 2$ benötigt. Um alle Knoten und Blätter des Merkle-Baums $\tau_{2,0}$ speichern zu können, benötigt *RFID*-Tag 31 Blöcke (siehe Abbildung 5.5). Das heißt, die Speicherkapazität, die für den Merkle-Baum $\tau_{2,0}$ zur Speicherung gebraucht wird, ist gleich: Die Zahl der Blöcke $\cdot 160 = 31 \cdot 160$ bits.

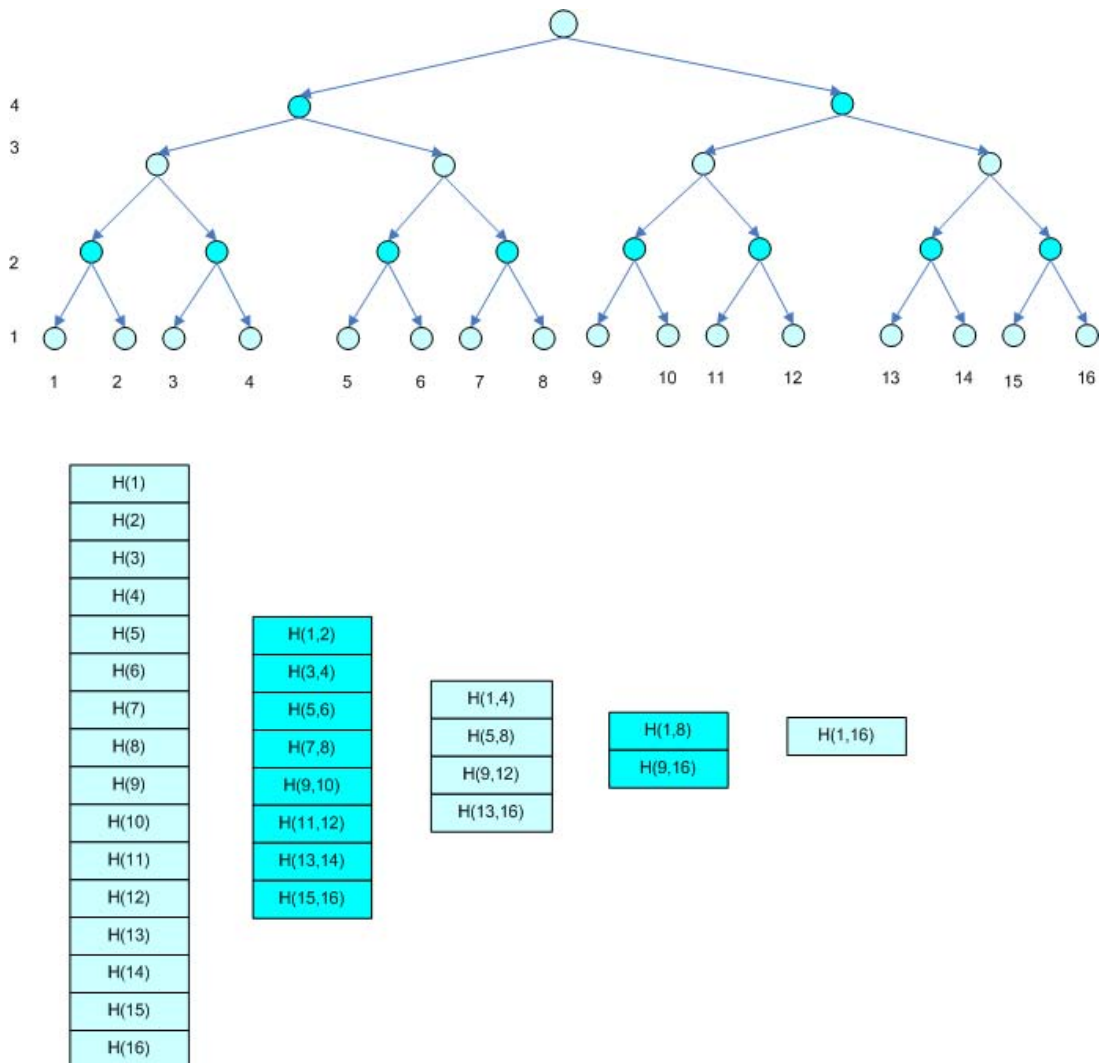


Abbildung 5.5: Merkle-Baum $\tau_{2,0}$ mit allen Blöcken

In diesem Verfahren beträgt die Zahl der Signature $d \in \{1, \dots, 512\}$. Die Zahl der Blöcke, die zur Speicherung aller Knoten und Blätter des GMSS-Baums gebraucht werden, ist gleich $63 + 31 = 94$ Blöcke. Die Speicherkapazität von EEPROM für GMSS-Baum ist gleich $94 \cdot 160$ bits.

5.7.2 Anforderungsspeicher EEPROM um einen GMSS-Baum zu bilden

Es wird eine Methode für Winternitz OTSS Signaturschlüsseln zu Erzeugung angeboten, um den GMSS-Baum erreichen zu können. Das *RFID*-Tag:

- erzeugt ein fest von $t_w = 43$ random Geheimschlüsseln $\{x_i\}_{i=1}^{t_w}$
Um diese Aufgabe erledigen zu können, wird eine *Pseudo Random Number Generator* PRNG angewandt

$$f : \{0, 1\}^n \longrightarrow \{0, 1\}^n \times \{0, 1\}^n$$

- berechnet $y_i = H^{2^w-1}(x_i)$
- berechnet den Wert $Y = H(y_1 \parallel \dots \parallel y_{t_w})$
- berechnet den Wert des Blatters $H(Y)$

Es wird ein Block für den Merkle-Baum $\tau_{1,0}$ in der erste Schicht $T = 1$ benötigt, um

$$Seed_{\tau_{1,0},l}; l = 1, \dots, 32$$

speichern zu können. In der zweiten Schicht $T = 2$ werden zwei Blöcke benötigt. Ein Block, um

$$Seed_{\tau_{2,0},l}; l = 1, \dots, 16$$

zu speichern, den zum aktuellen Merkle-Baum $\tau_{2,0}$ gehört und als Merkle-Online-Baum bezeichnet wird, sowie ein Block, der zum zweiten

$$Seed_{\tau_{2,1},l}; l = 1, \dots, 16$$

gehört und als zweiten Merkle-Baum $\tau_{2,1}$ in der zweiten Schicht, welcher als Merkle-Offline-Baum bezeichnet wird (siehe Abbildung 5.6).

5 GMSS Signatur Generation für RFID-Tags

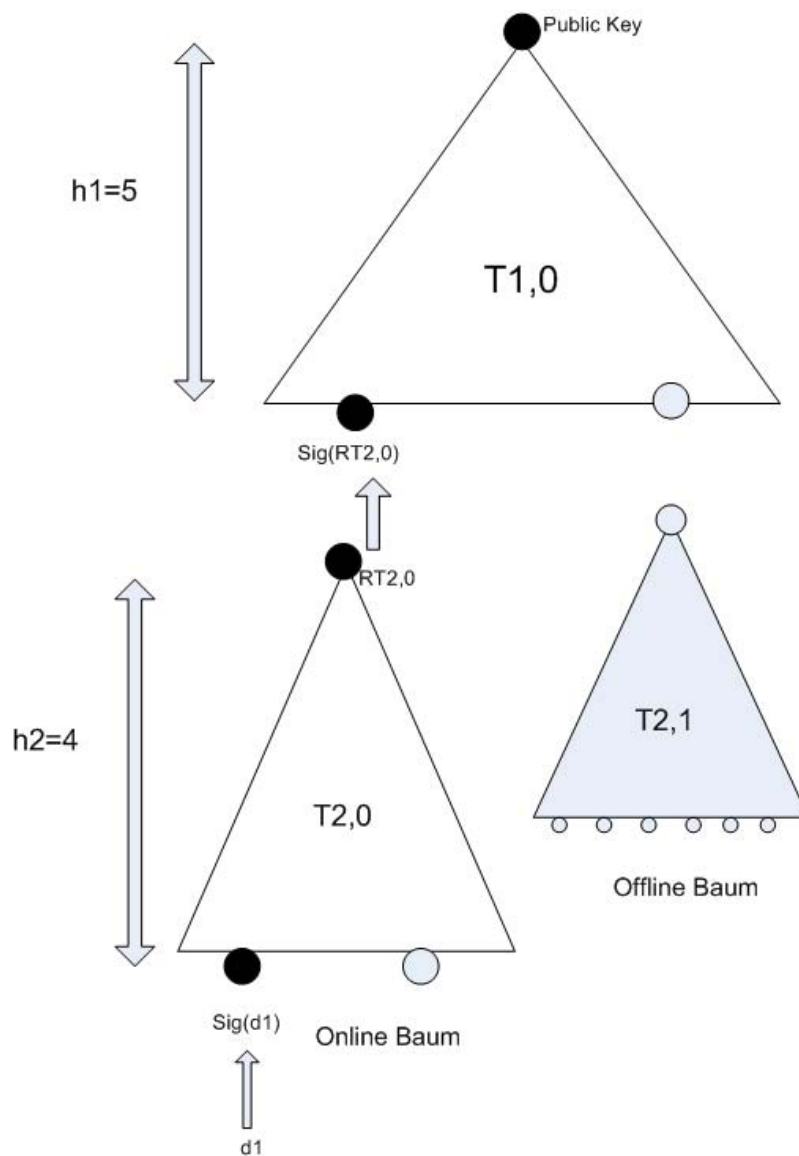


Abbildung 5.6: Merkle-Online-Baum $\tau_{2,0}$, Merkle-Offline-Baum $\tau_{2,1}$

Ebenso werden in der zweiten Schicht $T = 2$ vier vorläufige Blöcke benötigt, um einen Winternitz One-Time Signaturschlüssel zu erzeugen. Das heißt, in der zweiten Schicht braucht der Merkle-Online-Baum zwei vorläufig Blöcke, um seinen Signaturschlüssel erzeugen zu können. Ein Block für das $Seed_{OTS}$. Die aktualisierten $Seed_{OTS}$ werden in diesem Block gespeichert und der Geheimschlüssel x_i wird erzeugt. Der zweite Block für

$$Seed_{i,j}, l + 1 = Seed_{Next}$$

Das aktualisierte $Seed_{Next}$ wird in diesen Block gesichert, um den nächsten OTSS Signaturschlüssel für die nächste Nachricht generieren zu können. Dasselbe gilt auch für den Merkle-Offline-Baum (siehe Abbildung 5.7). Das *RFID*-Tag reserviert einen allgemeinen Block, der

5 GMSS Signatur Generation für RFID-Tags

über den temporären Wert des Geheimschlüssels $\{x_i\}_{i=1}^{43}$ verfügt. Das Tag nimmt den Wert des x_i und berechnet

$$y_i = H^{2^w-1}(x_i), \sigma_i = H^k(x_i)$$

Diese Werte werden in einigen Blöcken gespeichert, von denen ein neuer Wert x_{i+1} erzeugt wird, der diesen Block besitzt.

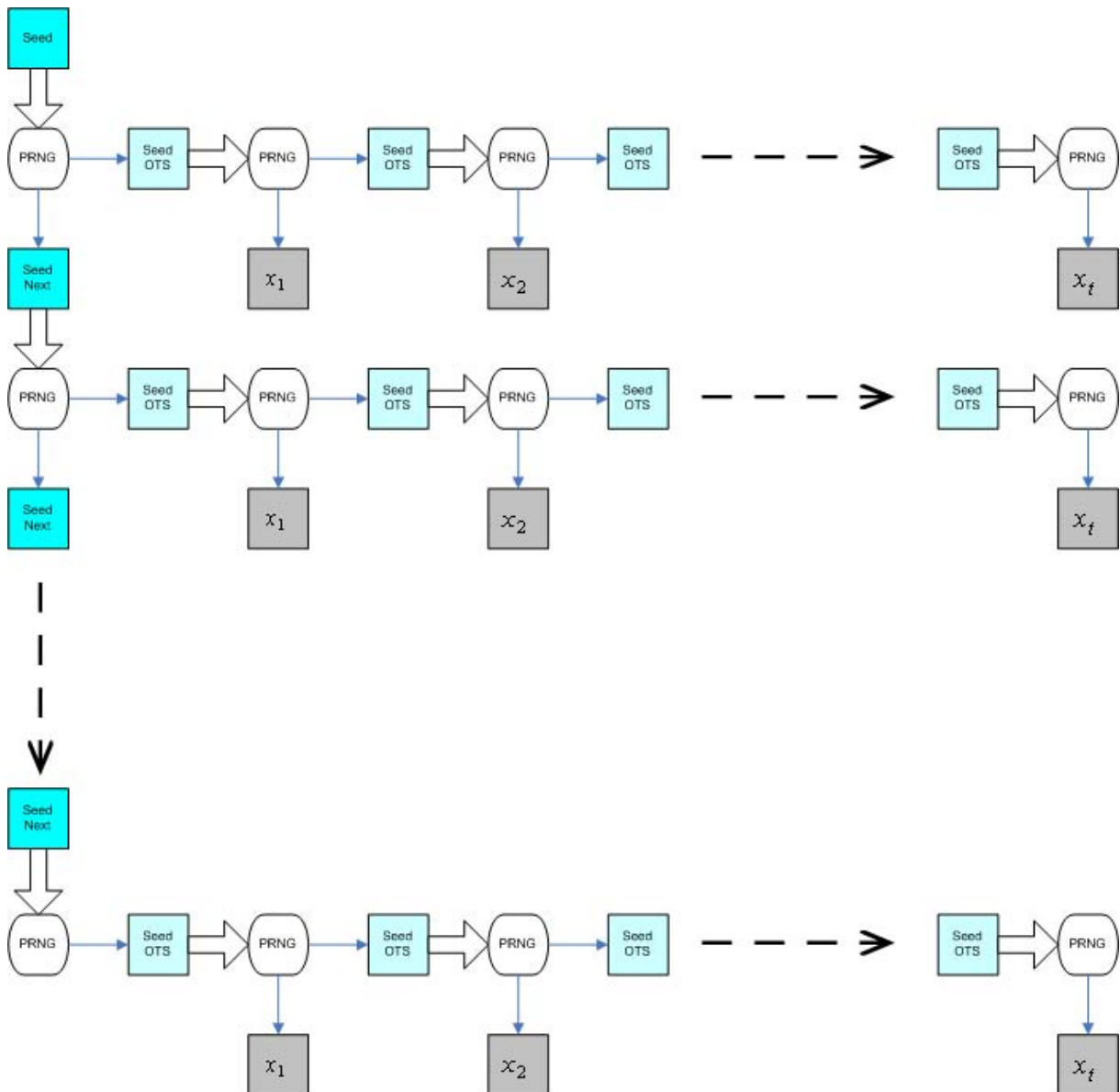


Abbildung 5.7: Anwendung PRNG

Die Abbildung 5.7 zeigt auf, wie die PRNG- Funktion benutzt werden kann, um den Signaturschlüssel für das aktuelle Blatt und für das nächste Blatt zu erzeugen. In diesem Schema

5 GMSS Signatur Generation für RFID-Tags

braucht jeder Merkle-Baum nur drei Blöcke, um die PRNG-Funktion anwenden zu können.

$$PRNG(Seed) \rightarrow (Seed_{OTS}, Seed_{Next})$$

$$PRNG(Seed_{OTS}) \rightarrow (Seed_{OTS}, x_i); i = 1, \dots, 43$$

Das heisst, der Merkle-Baum $\tau_{1,0}$ in der erste Schicht $T = 1$, im EEPROM $3 \cdot 160$ bits braucht, $T = 2$ wird $6 \cdot 160$ bits im EEPROM benötigt. Es werden nur zwei vorläufige Blöcke erfordert, um jedes Blatt zu erzeugen (siehe Abbildung 5.8).

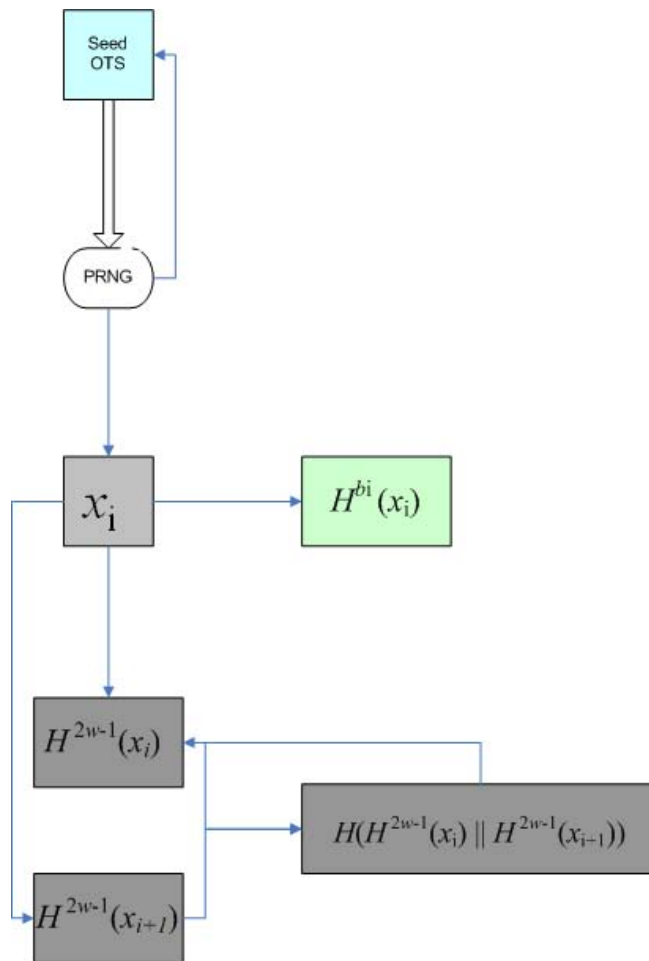


Abbildung 5.8: Berechnung eines Merkle Knotens

Um im selben Zeitpunkt die Schlüssel Generation und den Aufbau des GMSS-Baumes zu ermöglichen, wird das Seed in der zweiten Schicht erfasst. Die Eingabe der PRNG-Funktion ist das $Seed_{1,0,l}$ und die Ausgabe, ist das $Seed_{OTS}$ und $Seed_{1,0,l+1}$. Der zweite Schritt ist die Eingabe der PRNG-Funktion; das $Seed_{OTS}$ und die Ausgabe ist das neue $Seed_{OTS}$, welches im

5 GMSS Signatur Generation für RFID-Tags

selben Block des alten $Seed_{OTS}$ gespeichert wird. Die zweite Ausgabe ist der Geheimschlüssel x_i , welcher in einem allg. Block gesichert wird. Der dritte Schritt wird x_i genommen und durch

$$H^{2^w-1}(x_i); i = 1, \dots, 43$$

berechnet. Das Ergebnis y_i wird in einem Block gespeichert. Danach erfolgt die Eingabe der PRNG-Funktion, die das neue $Seed_{OTS}$ ist. Die Ausgabe des neuen $Seed_{OTS}$ wird im gleichen Block gespeichert und erzeugt einen neuen Geheimschlüssel x_{i+1} welcher im Block des alten Geheimschlüssel x_i gespeichert wird. Dieser Wert wird entnommen und anhand der Formel

$$y_{i+1} = H^{2^w-1}(x_{i+1})$$

berechnet. Das Ergebnis wird in einen neuen Block gespeichert. Als nächstes wird

$$z_i = H(y_i || y_{i+1})$$

berechnet und das Ergebnis in dem Block y_i gespeichert. Diese Schrittfolge wird bis $t_w = 43$ wiederholt. Bei jeder Wiederholung wird der Wert

$$z_{i+1} = H(z_i || y_{i+2})$$

berechnet. Das Endergebnis des Werts

$$H(z_{t_w-1})$$

ist der Wert des ersten Blattes.

5.7.3 Anforderungsspeicher EEPROM für Signatur Generation

Die Signaturgröße beruht auf das Winternitz OTSS Parameter w und der Benutzung der Hash-Funktion. Um eine Signatur erzeugen zu können, sind zwei Phasen vornöten. In der ersten Phase wird die Nachricht d_i signiert

$$Sig(d_i); i = 1, \dots, 16$$

In der zweiten Phase wird die Wurzel des Merkle-Baums

$$Sig(\tau_{2,j}); j = 0, \dots, 31$$

signiert. Für die Erzeugung einer Signatur werden zwei Merkle-Bäume erfordert. Der erste Baum in der zweiten Schicht, der Merkle-Online-Baum, hat 2^4 Blätter. Der zweite Baum in der erste Schicht hat 2^5 Blätter. In der zweite Schicht wird jeden Geheimschlüssel $x_i; i = 1, \dots, 43$ signiert. Jede OTSS Signatur braucht einen Block. Das heißt, es wird

$$\sigma_i = (H^{b_1}(x_1), \dots, H^{b_{43}}(x_{43}))$$

berechnet. Das Tag braucht $43 \cdot 160$ bits im EEPROM, um eine Nachricht der Signatur $Sig(d_i)$ zu speichern. Das Tag braucht $43 \cdot 160$ bits im EEPROM, um die Signatur der Merkle Wurzel

$\text{Sig}(\tau_{2,j})$ zu sichern. Der Große Vorteil des Verfahrens ist der Authentifizierungspfad $\text{Auth}_{\tau_{1,0}}$, welcher während dem Aufbau des GMSS-Baums berechnet und im EEPROM gespeichert wird. Die Größe des $\text{Auth}_{\tau_{1,0}}$ ist gleich $h_1 = 5 \cdot 160$ bits. Dasselbe gilt für den Authentifizierungspfad $\text{Auth}_{\tau_{2,j}}$, aber ihre Größe ist $h_2 = 4 \cdot 160$ bits. Die beiden Authentifizierungspfade erfordern nicht neue Blöcke, um sie im EEPROM zu speichern. Sie werden während des Aufbaus des GMSS-Baums berechnet und im EEPROM gespeichert.

In diesem Verfahren beträgt die Speicherkapazität, die das *RFID*-Tag braucht, um eine Signatur herstellen zu können $(43 + 43 = 86) \cdot 160$ bits.

5.8 Merkle-Offline-Baum Generation

In diesem Schritt wird versucht eine Signatur der Nachricht zu erstellen, welche im selben Moment den Authentifizierungspfad für diese Signatur berechnet. Dabei wird das neue Algorithmus (Offline-Baum) für die Erzeugung des Merkle-Offline-Baums in der zweiten Schicht genutzt. Der Vorteil des Algorithmus ist die zeitliche Verkürzung, das heißt, es wird nicht auf die Signierung $2^{h_2} = 2^4 = 16$ gewartet, um einen neuen Merkle-Online-Baum erzeugen zu können, sondern es wird signiert und gleichzeitig auch erbaut. Dies entsteht an Hand einer Methode, die mit der Speicherverwaltung im Zusammenhang steht. Dabei bedeutet die Speicherverwaltung in diesem Falle die Entfernung aller Blätter und Knoten, die in der Signatur bereits genutzt wurden, da sie nun nicht mehr brauchbar sind. Die leeren Blöcke werden für den Aufbau des neuen Merkle-Baums genutzt. Nach der $2^4 = 16$ Signatur wird in der zweiten Schicht der Merkle-Offline-Baum vollständig. Um den Offline-Baum benutzen zu können, definieren wir die folgende Variable.

- $H_{i,j}$: ist die Position der Knoten und Blätter in dem Merkle-Baum. $i = 1, \dots, h + 1$ ist der Index, welcher die höheren Knoten in dem Merkle-Baum bezeichnet.
 $j = 1, \dots, 2^{h-i+1}$ ist der Index, welcher den Platz der Knoten bezeichnet.
- $d = 1, \dots, 2^h$ ist die Nummer der Signatur.
- $s \in \{1, \dots, 2^h\}$ ist die Anzahl der Signatur.
- $t \in \mathbb{N}; t \leftarrow \max\{t : 2^t | s - 1\}$.

Algorithmus Offline-Tree.

Eingabe: Die Nummer des Blatts d_i und Blöcke des Merkle-Online-Baums.

Ausgabe: Ein Merkle-Offline-Baum.

1. Berechnung $t \in \mathbb{N}; t \leftarrow \max\{t : 2^t | d_i - 1\}$.

5 GMSS Signatur Generation für RFID-Tags

2. $j \leftarrow d_i - 1$.
3. For $i = 1, \dots, t$.
 - a) löschen die Knoten $H(i, j)$ und $H(i, j - 1)$ von ihren Blöcken.
 - b) $j \leftarrow \lfloor j \rfloor / 2$.
4. wendet den Algorithmus Merkle-Baum Generation an.

In den folgenden Abbildungen wird dies verdeutlicht. Nach der 16. Signatur wird in der zweiten Schicht der Merkle-Offline-Baum aufgebaut. Es wird ein kleines Beispiel gegeben, wie der Offline-Baum Algorithmus funktionieren kann. Die Höhe des Merkle-Online Baums ist gleich $h = 3$, die Zahl der möglichen Signatur $2^3 = 8$. Im ersten Schnitt signiert das Tag die erste Nachricht d_1 und bestimmt ihren passenden Authentifizierungspfad. Der Wert der Variable $t = 0$, der Algorithmus (Offline-Baum) ist noch nicht aktiviert. (siehe Abbildung 5.9).

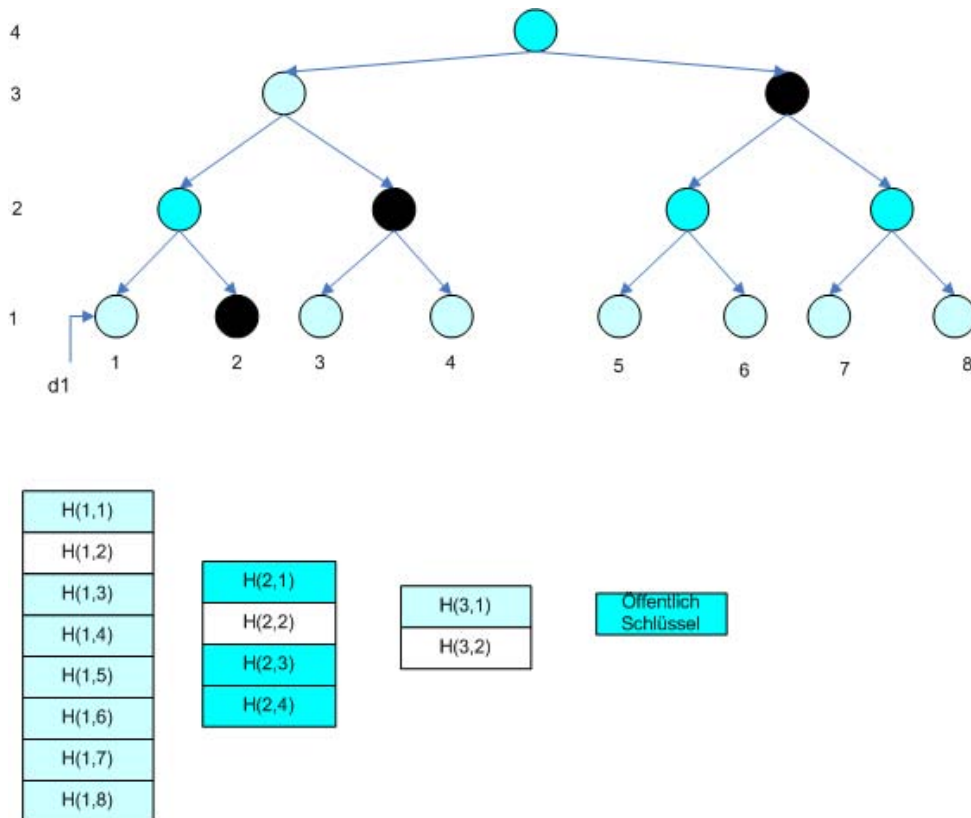


Abbildung 5.9: Authentifizierungsbaum für d_1

Die Abbildung 5.9 zeigt den Merkle-Online-Baum, der in dem EEPROM schon gespeichert wird. Das Tag signiert die Nachricht d_1 und nimmt die Hash-Werte, die in dem EEPROM warten, als Authentifizierungspfad ($H(1, 2)$, $H(2, 2)$, $H(3, 2)$) an.

5 GMSS Signatur Generation für RFID-Tags

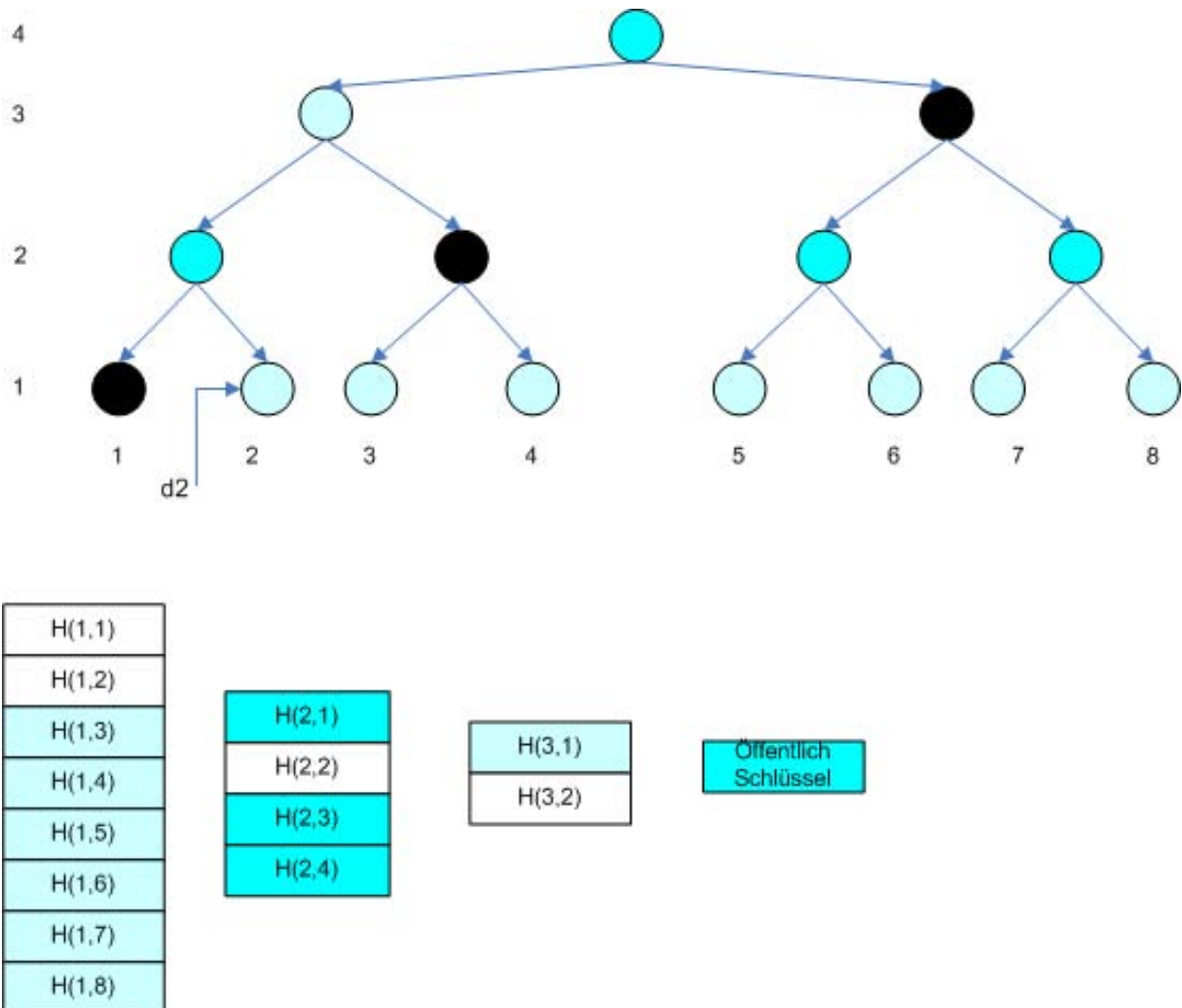


Abbildung 5.10: Authentifizierungsbaum für d_2

Die Abbildung 5.10 verdeutlicht, dass das Tag die Nachricht d_2 signiert und die Hash-Werte im EEPROM als Authentifizierungspfad ($H(1,1), H(2,2), H(3,2)$) benutzt. Der Wert der Variable $t = \max\{2^t | 2 - 1\} = 0$.

Vor der Signierung der Nachricht d_3 beträgt der Wert der Variable $t = 1$. Das Algorithmus Offline-Baum löscht die beiden Blätter $H(1,1), H(1,2)$ und gibt zwei neue Blöcke frei. Das Tag braucht diese Hash-Werte nicht mehr im EEPROM (siehe Abbildung 5.11).

5 GMSS Signatur Generation für RFID-Tags

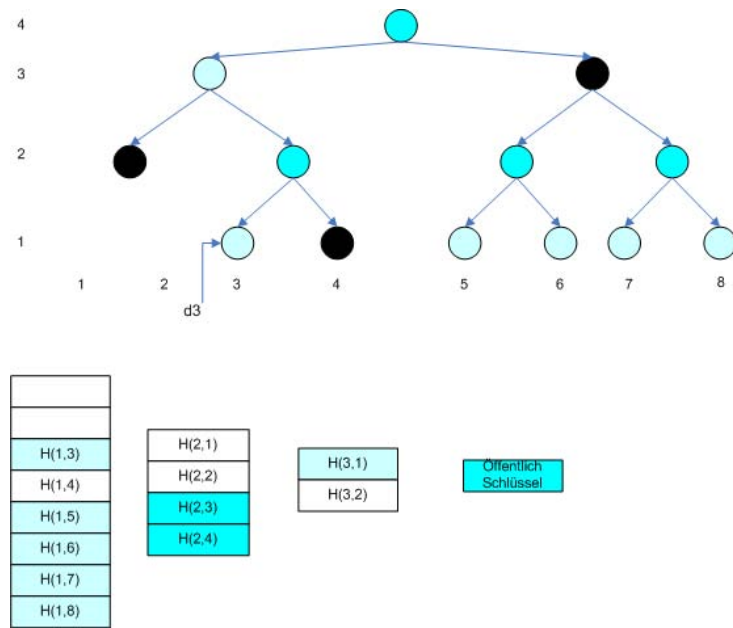


Abbildung 5.11: Authentifizierungsbaum für d_3

Das Tag signiert der Nachricht d_3 und bestimmt ihren Authentifizierungspfad $H(1, 4), H(2, 1), H(3, 2)$. Für die Nachricht d_4 , ist der Authentifizierungspfad $H(1, 3), H(2, 1), H(3, 2)$ (siehe Abbildung 5.12).

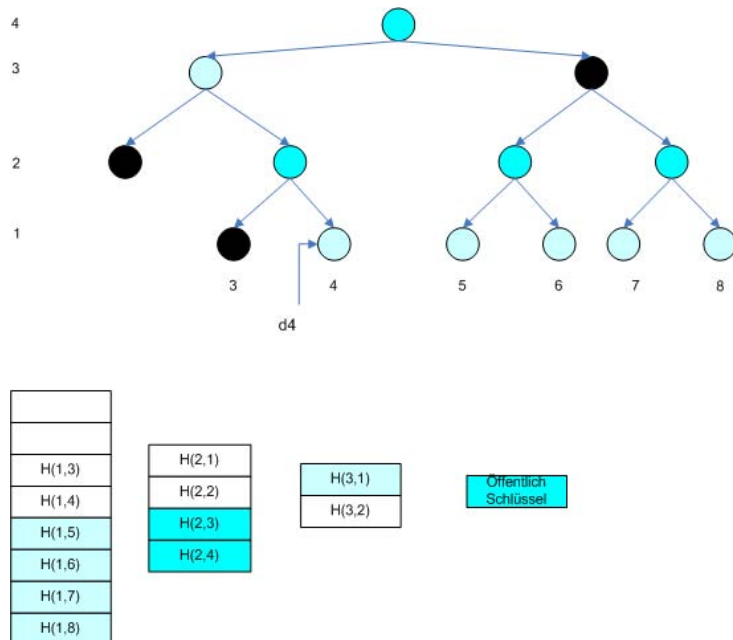


Abbildung 5.12: Authentifizierungsbaum für d_4

5 GMSS Signatur Generation für RFID-Tags

Vor der Signierung der Nachricht d_5 , beträgt der Wert der Variable $t = 2$. Der Algorithmus Offline-Baum löscht die Blätter $H(1, 3)$, $H(1, 4)$ und die Knoten $H(2, 1)$, $H(2, 2)$ und gibt vier Blöcke frei. An dieser Stelle fängt das Tag an, einen neuen Merkle-Baum aufzubauen und alle freien Blöcke zu benutzen, um in den neuen Blättern und neuen Knoten speichern zu können. (siehe Abbildung 5.13).

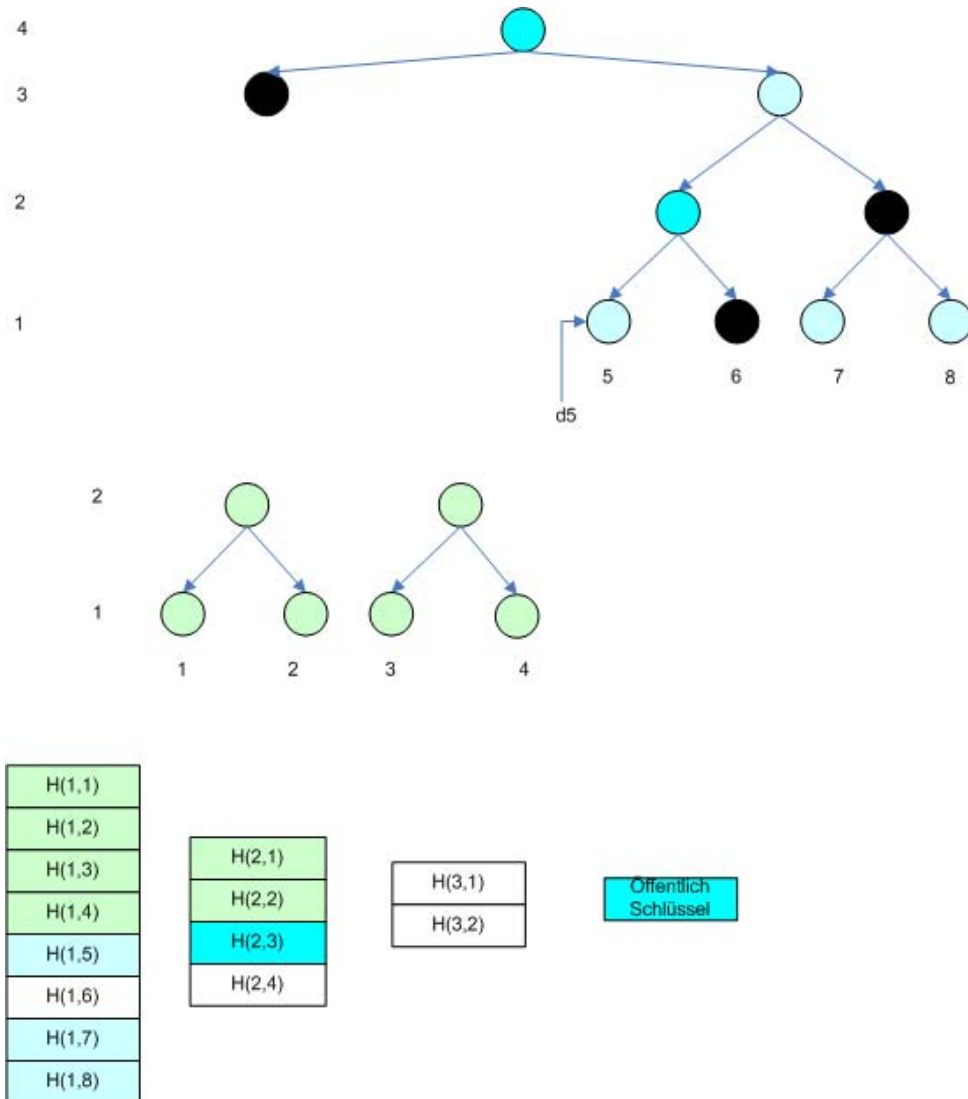


Abbildung 5.13: Authentifizierungsbaum für d_5

Das Tag benutzt die 6 freien Blöcke, um die neuen Hash-Werte des neuem Merkle-Baums

$$H(1, 1), H(1, 2), H(1, 3), H(1, 4), H(2, 1), H(2, 2)$$

speichern zu können. Die Abbildung 5.13 zeigt einen Teil des Aufbaus eines Merkle-Offline-Baums während der Anwendung des Merkle-Online-Baums auf. Im nächsten Schnitt signiert

5 GMSS Signatur Generation für RFID-Tags

das Tag die Nachricht d_5 und bringt ihren Authentifizierungspfad

$$H(1, 6), H(2, 4), H(3, 1)$$

Dasselbe auch für d_6 (siehe Abbildung 5.14). Der Authentifizierungspfad

$$H(1, 5), H(2, 4), H(3, 1)$$

wurde schon im EEPROM vorbereitet.

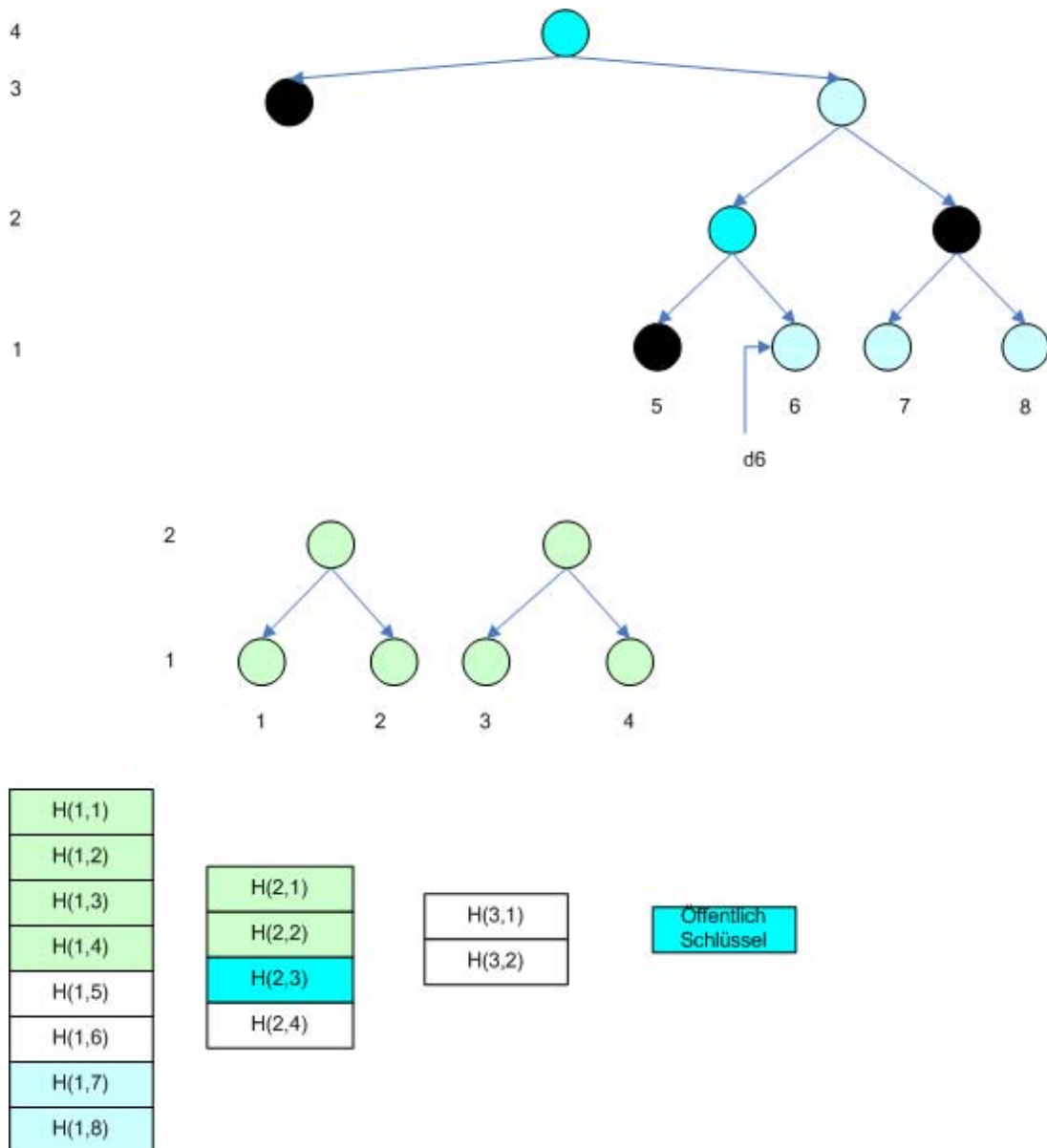


Abbildung 5.14: Authentifizierungsbaum für d_6

Vor der Signierung d_7 nimmt die Variable den Wert $t = 1$. Die Blätter $H(1, 5)$, $H(1, 6)$ werden gelöscht und das Tag frei für diesen zweiten Stock (siehe die Abbildung 5.15).

5 GMSS Signatur Generation für RFID-Tags

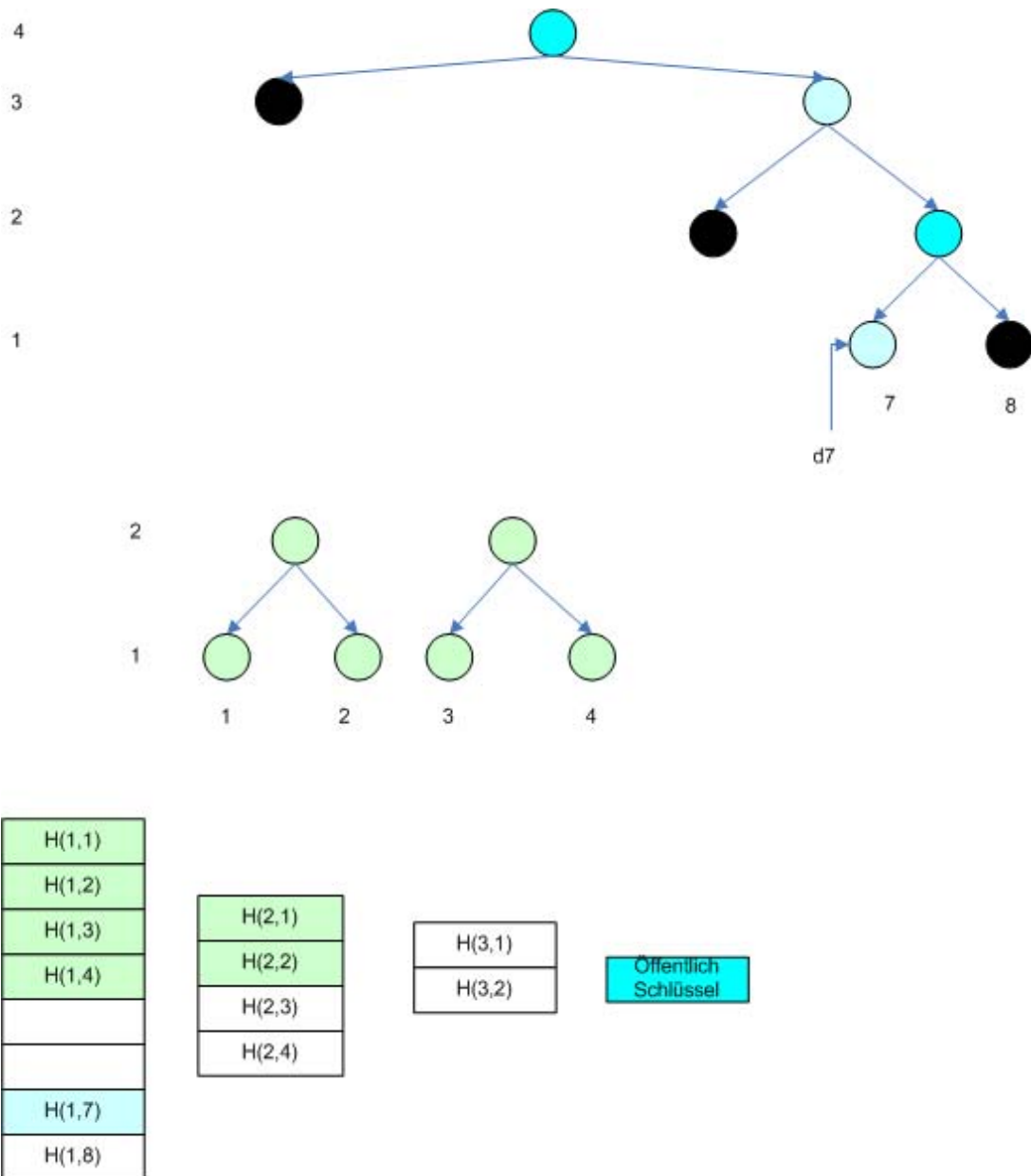


Abbildung 5.15: Authentifizierungsbaum für d_7

Der Authentifizierungspfad der Signatur d_7 ist

$$H(1, 8), H(2, 3), H(3, 1)$$

wird schon während des Aufbaus eines Merkle-Online-Baums berechnet. Wenn die Nachricht d_8 signiert wird, sendet das Tag diese Signatur mit ihrem Authentifizierungspfad

$$H(1, 7), H(2, 3), H(3, 1)$$

5 GMSS Signatur Generation für RFID-Tags

an das Lesegerät (siehe Abbildungen 5.16) zu.

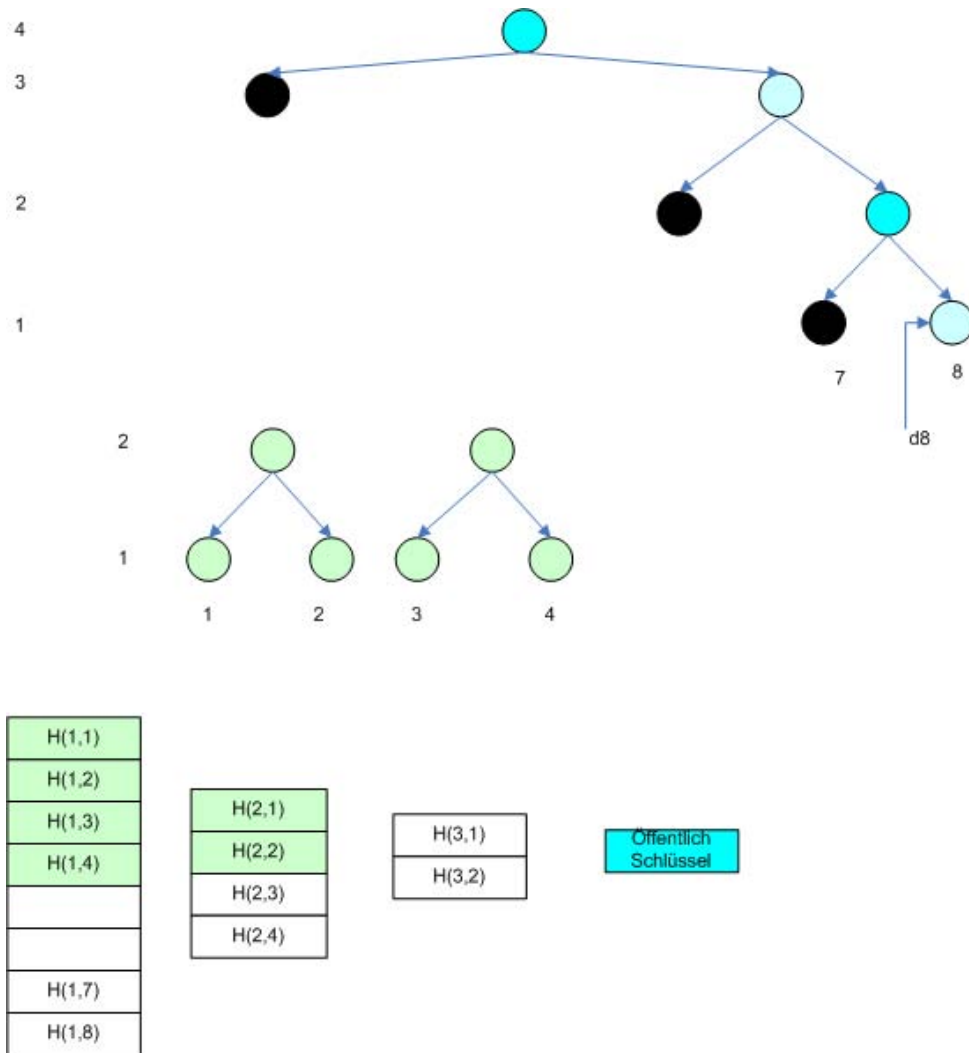


Abbildung 5.16: Authentifizierungsbaum für d_8

Der Wert der Variablen $t = 3$. Die Blätter

$$H(1, 7), H(1, 8)$$

und die Knoten

$$H(2, 3), H(2, 4), H(3, 1), H(3, 2)$$

werden vom EEPROM gelöscht. Danach werden die leeren Blöcke zur Erzeugung des kompletten Merkle-Offline-Baums angewandt (siehe Abbildung 5.17).

5 GMSS Signatur Generation für RFID-Tags

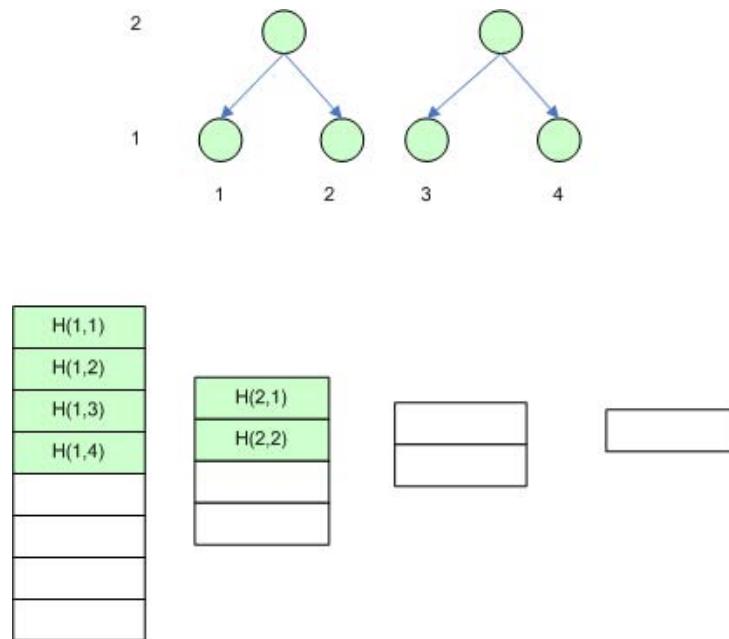


Abbildung 5.17: Berechnung des Merkle-Offline-Baums

Das Tag ruft weiter die vorhandenen Algorithmen wie PRNG, Hash-Funktion und MSS-Baum Generation, die bereits erläutert wurden, auf [16].

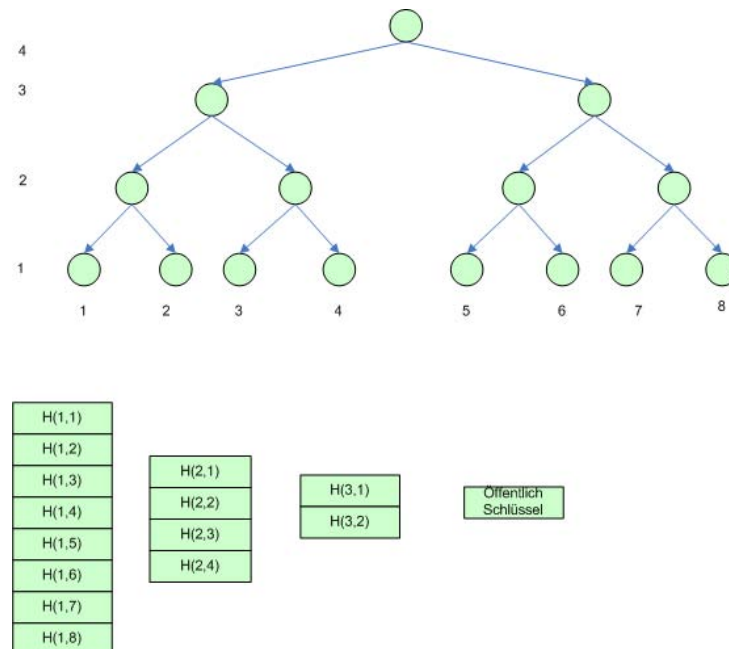


Abbildung 5.18: Merkle-Offline-Baum

5 GMSS Signatur Generation für RFID-Tags

Für das erste Verfahren (Ganz-Baum im EEPROM) werden die totalen Kosten berechnet, die das *RFID*-Tag benötigt, um ein GMSS-Baum aufbauen zu können und 512 Signaturen zu erzeugen.

GMSS-Baum Generation Der Aktuelle GMSS-Baum besteht aus drei Bäumen. Der Merkle-Baum in der Schicht $T = 1$, welcher über $2^5 = 32$ Blätter verfügt. Jedes Blätter erfordert

$$(2^w - 1) \cdot t_w + (t_w - 1) = (2^4 - 1) \cdot 43 + 42 = 687$$

Hash und

$$(t_w + 1) = (43 + 1) = 44$$

PRNG, für alle Blätter des Merkle-Baums ($687 \cdot 32 = 21984$) Hash und ($44 \cdot 32 = 1408$) PRNG. Merkle-Baum mit 2^5 erfordert ($2^5 - 1 = 31$) Hash, um alle Knoten zu berechnen. Der Merkle-Baum in der ersten Schicht braucht $21984 + 31 = 22015$ Hash und 1408 PRNG für die Berechnung. Der Merkle-Online-Baum verfügt in der Schicht $T = 2$, der über $2^4 = 16$ Blätter und $2^4 - 1 = 15$ Knoten. Die Nummer von Hash, die der Merkle-Online-Baum braucht, ist $16 \cdot [(2^4 - 1) \cdot 43 + 42] + 15 = 11007$ Hash und $44 \cdot 16 = 704$ PRNG. Der Merkle-Offline-Baum in der zweiten Schicht wird während der Ersten $2^4 = 16$ Signatur aufgebaut. Ein Merkle-Offline-Baum kostet 11007 Hash und 704 PRNG. Die totalen Kosten der GMSS-Baum Generation in diesem Verfahren beträgt $22015 + 11007 + 11007 = 44029$ Hash und $1408 + 704 + 704 = 2816$ PRNG.

Größe des GMSS-Baums Um den GMSS-Baum speichern zu können, werden im EEPROM $2^{5+1} - 1 = 63$ Blöcke für Merkle-Baum in der ersten Schicht und 3 Blöcke für PRNG-Funktion reserviert. Jede Blockgröße beträgt 160 bits. Für den Merkle-Online und Offline-Baum in der zweite Schicht werden im EEPROM $2^{4+1} - 1 = 31$ Blöcke und 6 Blöcke für PRNG-Funktion, 2 allgemeine Blöcke für die Berechnung jedes Knotens und einen allg. Block für die Geheimschlüssel behalten. Zusammen betragen diese die Größe des GMSS-Baums $(63 + 31 + 3 + 3 + 3 + 2 + 1) = 106$ Blöcke, $(106 \cdot 160)$ bits.

Größe der Signatur und des Overheads Die Anzahl der Signatur für eine Nachricht beträgt in diesem Verfahren 2. Die erste Signatur ist die sogenannte Online-Signatur, welche die Nachricht digest $d_i; i = 1, \dots, 16$ bei der Anwendung der Blätter des Merkle-Online-Baums in der zweiten Schicht signiert. Die zweite Signatur heißt, Offline-Signatur, welche die Wurzel des Merkle-Online-Baums in der zweite Schicht signiert. Um eine Signatur zu erzeugen, sind zwei Phasen vorhanden. In der ersten Phase wird die Nachricht digest signiert $Sig(d_i)$, die

$$(2^w - 1) \cdot t_w / 2 = (2^4 - 1) \cdot 43 / 2 = 323$$

Hash Kostet. In der zweiten Phase wird die Wurzel des Merkle-Online-Baums signiert

$$Sig(\tau_{2,j}); j = 0, \dots, 31$$

5 GMSS Signatur Generation für RFID-Tags

die 323 Hash Kostet. Um $\text{Sig}(d_i)$ erzeugen zu können, braucht *RFID*-Tag einen Platz im EEPROM $t_w = 43$ Blöcke, um diesen zu speichern. Dasselbe gilt auch für $\text{Sig}(\tau_{2,j})$. Die Größe der Signatur einer Nachricht beträgt $(43 + 43 = 86) \cdot 160$ bits. Nach der vorherigen Analyse des GMSS One-Time Signatur Schema ist die Speicherkapazität des EEPROM, die das *RFID*-Tag benötigt, um GMSS One-Time Signature Schema anwenden zu können bei $(106 + 86 = 192) \cdot 160$ bits.

Zur Authentifizierung des Lesegeräts gegenüber dem Tag muss dieses Tag Verifikationsdaten zum Lesegerät übertragen. Die Verifikationsdaten bestehen aus:

- One-Time Signatur der Nachricht digest $\text{Sig}(d_i)$ für das Blatt d_i und dem Authentifizierungspfad für dieses Blatt $\text{Auth}_{\tau_{2,j},l}$. Die Größe des $\text{Auth}_{\tau_{2,j},l}$ Pfads ist gleich der Höhe des Merkle-Online-Baums $h_2 = 4$ Blöcke.
- One-Time Signatur der Wurzel des Merkle-Online-Baums $\text{Sig}(\tau_{2,j})$ und den Authentifizierungspfad für diese Wurzel $\text{Auth}_{\tau_{1,0},l}$. Die Größe des $\text{Auth}_{\tau_{1,0},l}$ Pfads ist gleich der Höhe des Merkle-Baums in der ersten Schicht $h_1 = 5$ Blöcke.

Das *RFID*-Tag stellt ein Frame mit Overhead her, um die Verifikationsdaten in dem Overhead hochzuladen und an das Lesegerät zu senden.

Die Größe des Overheads beträgt

$$\begin{aligned} & (\text{Sig}(d_i) + \text{Sig}(\tau_{2,j}) + \text{Auth}_{\tau_{2,j},l} + \text{Auth}_{\tau_{1,0},l}) \\ & = (43 + 43 + 4 + 5) = 95 \cdot 160 \end{aligned}$$

bits. Um das Tag des Frames an das Lesegerät senden zu können, definiert *IEEE 802.11* ein neues Protokoll, um die Verifikationsdaten zwischen Lesegerät und *RFID*-Tag zu übertragen [5]. (siehe Tabelle 2.5).

Die Größe dieses Frames könnte bis zu 0.5 Mbit/s erreichen. Es wird eine Tabelle 6.1 angeboten, die die Anwendung des ersten Verfahrens (Ganz-Baum im EEPROM) GMSS One-Time Signatur auf *RFID*-Tag zusammenfasst.

5 GMSS Signatur Generation für RFID-Tags

T	2
h_1	5
h_2	4
w	4
$SHA - 1, n$	160
t_w	43
Anzahl der Signatur	512
Größe des GMSS-Baums	$106 \cdot 160 \text{ bits} = 2 \text{ Kbyte}$
Größe der GMSS-Signatur	$86 \cdot 160 \text{ bits} = 1,7 \text{ Kbyte}$
GMSS Generation	44029 Hash
GMSS Generation	2816 PRNG
One Signatur Generation	646 Hash
Größe des Authentifizierungspfads	$9 \cdot 160 \text{ bits} = 1,44 \text{ Kbit}$
Komplette Speicherkapazität	$192 \cdot 160 \text{ bits} = 3,8 \text{ Kbyte}$
Größe des Overheads	$95 \cdot 160 \text{ bits} = 2 \text{ Kbyte}$

Tabelle 5.1: GMSS One-Time Signatur Parameter

Anmerkung Der Vorteil dieses GMSS One-Time Signatur Schemas ist, dass dieser für die Signatur einer Nachricht sehr schnell ist. Das Tag hat alle Authentifizierungspfade für alle Blätter in den beiden Schichten berechnet und sie im EEPROM gespeichert. Für eine neue Nachricht braucht das Tag keine Zeit, um den neuen Authentifizierungspfad zu erzeugen. Denn es wählt einfach, aus dem EEPROM die passenden Blöcke, die über die passenden Authentifizierungspfade verfügen.

Der Nachteil dieses GMSS One-Time signatur Schemas ist, die Anzahl der Signaturen die eigentlich zu klein sind, so dass das Tag nur 512 Signaturen erzeugen kann.

Um im Tag mehrere Signatur erzeugen zu können, wird nun ein neues Verfahren angeboten. Wir definieren neue Parameter für den GMSS-Baum. Dieses Verfahren beruht auf einige Prinzipien. Das Tag speichert nicht mehr alle Knoten des GMSS-Baums, sondern es speichert nur noch die Knoten und die Blätter, die einen geraden Index in jeder Stufe haben. Mit diesem Verfahren könnte das Tag mehr Platz im EEPROM sparen und sich die Zahl von Signaturen erhöhen. Der Nachteil dieses Verfahrens ist, dass die Signaturzeit einer Nachricht länger dauert als im erstes Verfahren, weil das Tag für jede Nachricht den Authentifizierungspfad berechnen und jedes Mal einen Update der Blöcke im EEPROM vorbereiten muss.

5.9 GMSS-Baum Parameter

Das zweite Verfahren konstruiert einen GMSS-Baum mit drei Schichten $T = 3$. Die erste Schicht $T = 1$ hat einen Merkel-Baum $\tau_{1,0}$ mit der Höhe $h_1 = 5$. Die zweite Schicht $T = 2$

5 GMSS Signatur Generation für RFID-Tags

hat $2^{h_1} = 2^5 = 32$ Merkle-Bäume, die mit $\tau_{2,j}$; $j = 0, \dots, 31$ gekennzeichnet werden. Jeder Merkle-Baum hat die Höhe $h_2 = 4$. Die dritte Schicht $T = 3$ hat $2^{h_1+h_2} = 2^9 = 512$ Merkle-Bäume, die mit $\tau_{3,j}$; $j = 0, \dots, 511$ gekennzeichnet werden. Jeder Merkle-Baum hat die Höhe $h_3 = 3$ (siehe Abbildung 5.19).

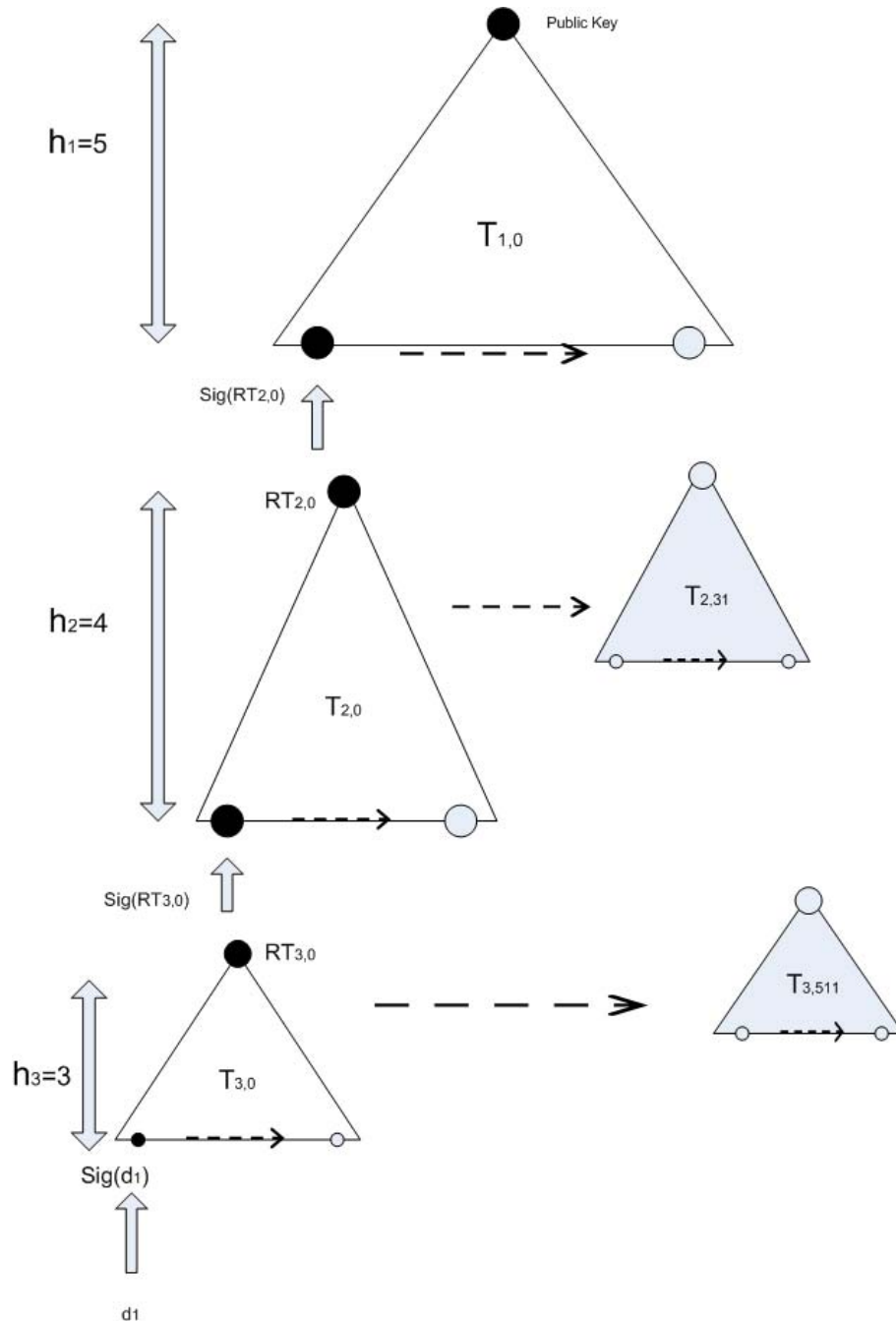


Abbildung 5.19: GMSS-Baum

5 GMSS Signatur Generation für RFID-Tags

Der *RFID*-Tag benutzt dieses Verfahren, um den GMSS-Baum aufzubauen und

$$2^{h_1+h_2+h_3} = 2^{12} = 4096$$

Nachrichten signieren zu können (siehe Abbildung 5.20).

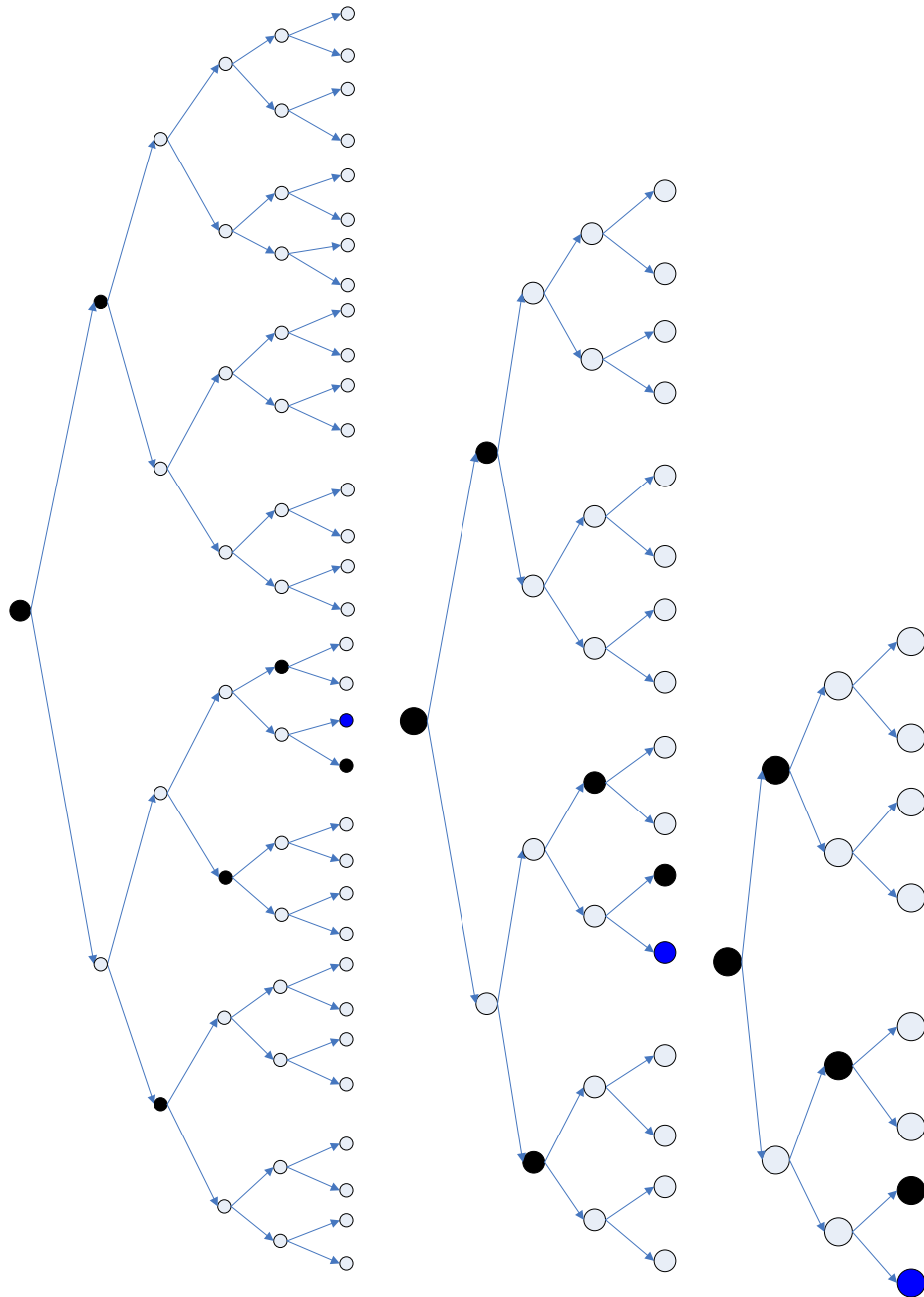


Abbildung 5.20: GMSS-Baum für RFID-Tag

5.10 Implementation GMSS Signatur auf RFID-Tag

Um das zweite Verfahren auf *RFID*-Tag anwenden zu können, müssen wir alle Knoten und Blätter des GMSS- Baums, die einen geraden Index haben, in dem EEPROM sichern.

5.10.1 Anforderungsspeicher EEPROM für GMSS-Baum

Jetzt können wir die Zahl der Knoten und Blätter des GMSS-Baums berechnen und ihre Größe in dem Tag bestimmen. Es wird $2^{h_1} - 1 = 2^5 - 1 = 31$ Blöcke für die erste Schicht $T = 1$ benötigt, um alle Knoten und Blättern des Merkle-Baums speichern zu können (siehe Abbildung 5.21). Das heisst, die Speicherkapazität, die der Merkle-Baum $\tau_{1,0}$ zum speichern braucht, ist gleich: Die Zahl der Blöcke * 160 = $31 \cdot 160$ bits.

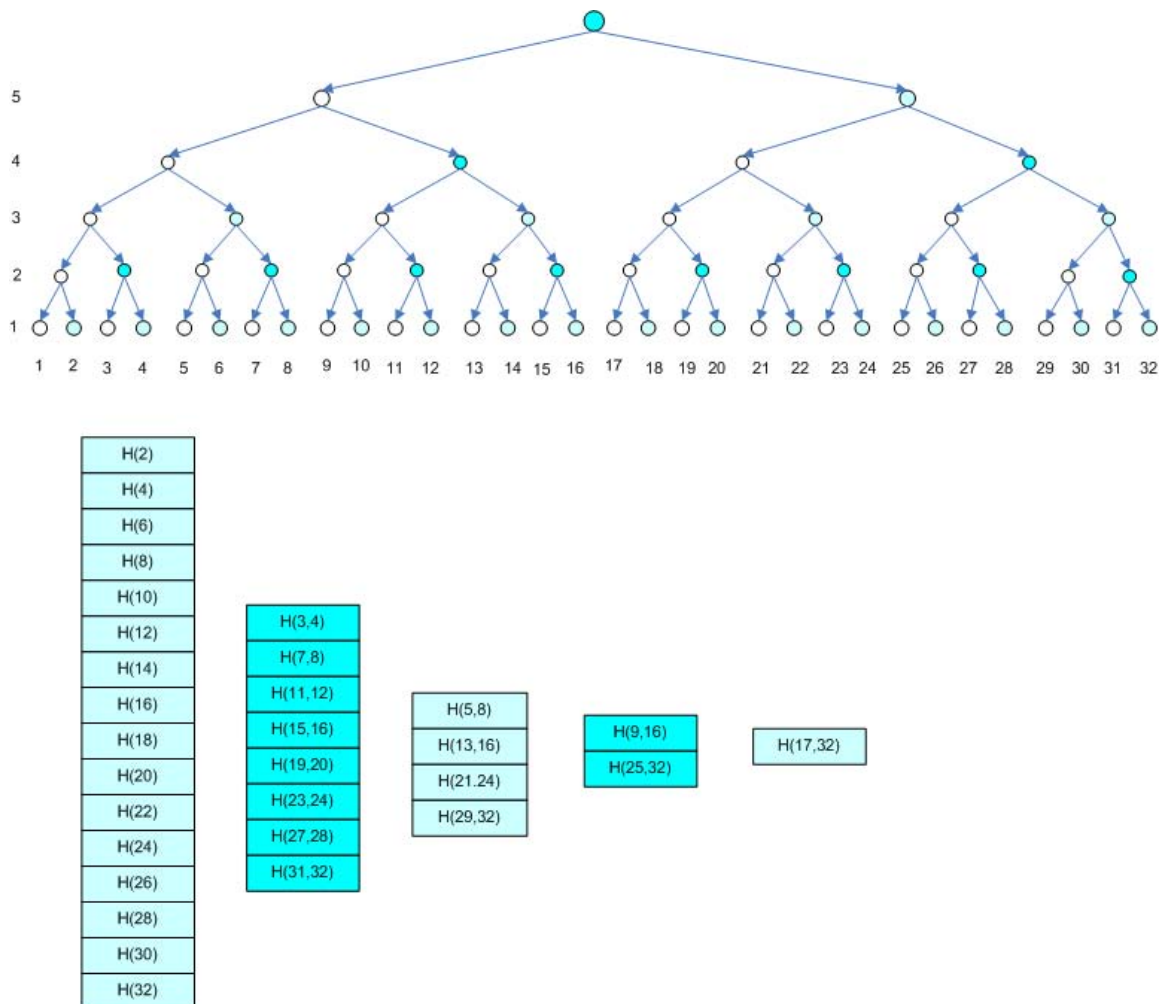


Abbildung 5.21: Merkle-Baum $\tau_{1,0}$ mit allen Blöcken

5 GMSS Signatur Generation für RFID-Tags

Im nächsten Schritt wird $2^{h_2} - 1 = 2^4 - 1 = 15$ Blöcke für die zweite Schicht $T = 2$ benötigt, um die Knoten und Blätter des Merkle-Online-Baums $\tau_{2,0}$ speichern zu können (siehe Abbildung 5.22). Das heisst, die Speicherkapazität, die für den Merkle-Online-Baum $\tau_{2,0}$ zum speichern benötigt wird, ist gleich: die Zahl der Blöcke * 160 = 15 · 160 bits.

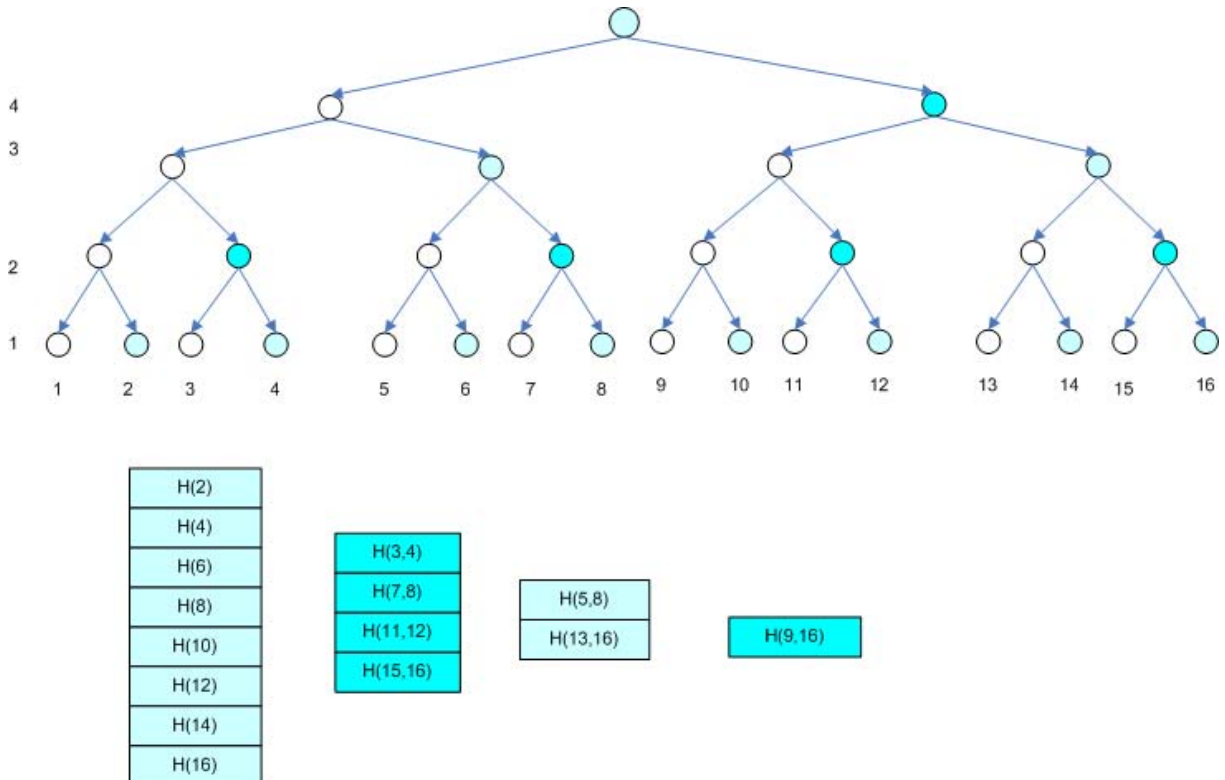


Abbildung 5.22: Merkle-Baum $\tau_{2,0}$ mit allen Blöcken

Es wird $2^{h_3} - 1 = 2^3 - 1 = 7$ Blöcke für die dritte Schicht $T = 3$ benötigt, um die Knoten und Blätter des Merkle-Online-Baums $\tau_{3,0}$ speichern zu können (siehe Abbildung 5.23). Das heisst, die Speicherkapazität, die für den Merkle-Online-Baum $\tau_{3,0}$ zum speichern erfordert, ist gleich: Zahl der Blöcke · 160 = 7 · 160 bits.

5 GMSS Signatur Generation für RFID-Tags

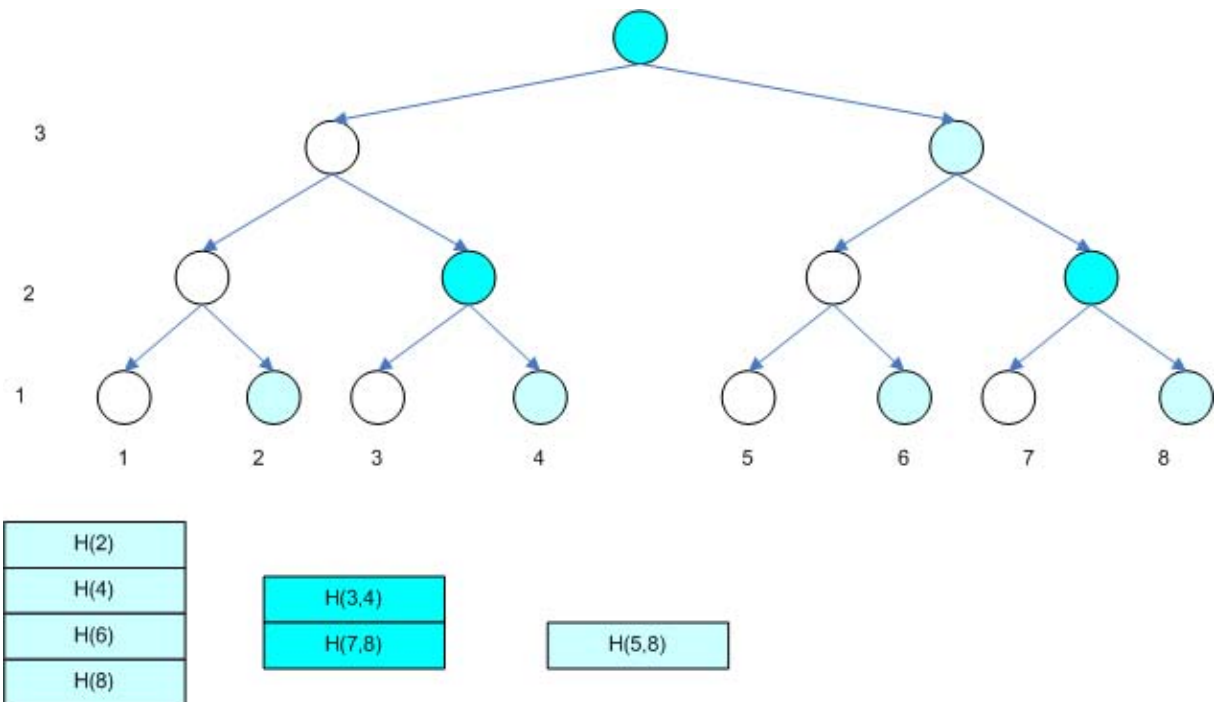


Abbildung 5.23: Merkle-Baum $\tau_{3,0}$ mit allen Blöcken

In diesem Verfahren ist die Zahl der Signatur ist $d \in \{1, \dots, 4096\}$. Die Zahl der Blöcke, die zur Speicherung des GMSS-Baums gebraucht werden, ist gleich $(31 + 15 + 7 = 53)$ Blöcke. Die Speicherkapazität des EEPROM ist gleich: $(53 \cdot 160)$ bits.

5.10.2 Anforderungsspeicher EEPROM um einen GMSS-Baum zu bilden

Es wird ein Block für den Merkle-Baum $\tau_{1,0}$ in der erste Schicht $T = 1$ benötigt, um

$$Seed_{\tau_{1,0},l}; l = 1, \dots, 32$$

speichern zu können. In der zweiten Schicht $T = 2$ werden zwei Blöcke benötigt, um

$$Seed_{\tau_{2,0},l}; l = 1, \dots, 16$$

zu speichern, der zum aktuellen Merkle-Baum $\tau_{2,0}$ gehört, welcher als Merkle-Online-Baum bezeichnet wird. Zum zweiten

$$Seed_{\tau_{2,1},l}; l = 1, \dots, 16$$

gehört, der zweite Merkle-Baum $\tau_{2,1}$ in der zweiten Schicht, welcher als Merkle-Offline-Baum bezeichnet wird. In der dritte Schicht $T = 3$ werden auch zwei Blöcke benötigt, um

$$Seed_{\tau_{3,0},l}; l = 1, \dots, 8$$

5 GMSS Signatur Generation für RFID-Tags

zu speichern. der zum aktuellen Merkle-Baum $\tau_{3,0}$ gehört, welcher auch Merkle-Online-Baum bezeichnet wird. Zum zweiten

$$Seed_{\tau_{3,1},l}; l = 1, \dots, 8$$

gehört, der zweite Merkle-Baum $\tau_{3,1}$ in der dritten Schicht, welcher als Merkle-Offline-Baum bezeichnet wird (siehe Abbildung 5.24).

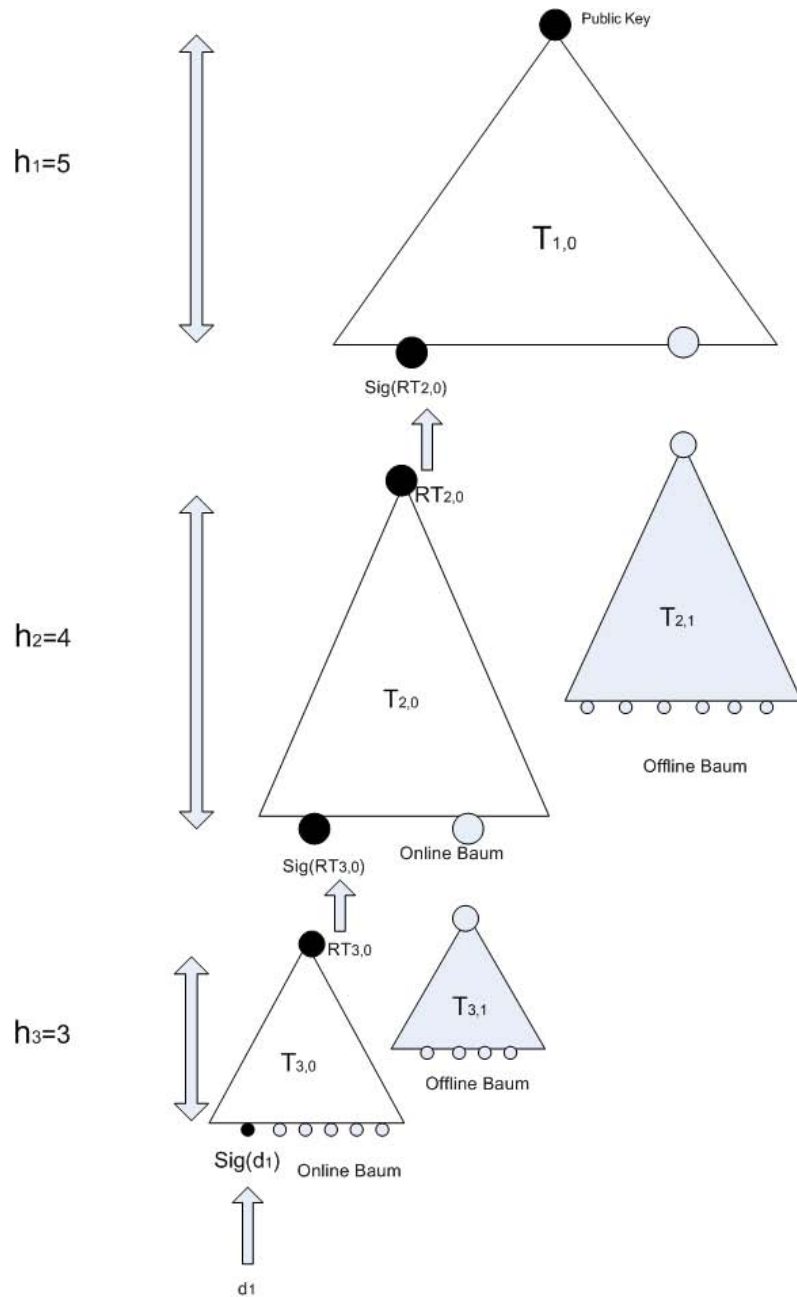


Abbildung 5.24: Merkle-Online-Baum $\tau_{2,0}$, Merkle-Offline-Baum $\tau_{2,1}$, Merkle-Online-Baum $\tau_{3,0}$, Merkle-Offline-Baum $\tau_{3,1}$

5 GMSS Signatur Generation für RFID-Tags

Ebenso werden in der zweiten Schicht $T = 2$ zwei vorläufige Blöcke benötigt, um einen Winternitz One-Time Signaturschlüssel zu erzeugen. Das heißt, in der zweiten Schicht braucht der Merkle-Online-Baum einen vorläufigen Block für $Seed_{OTS}$, um seinen Geheimschlüssel generieren zu können. Einen Block $Seed_{Next}$ für die nächste GMSS Signatur vorzubereiten und einen allgemeinen Block für Geheimschlüssel $\{x_i\}_{i=1}^{43}$. Dasselbe gilt auch für den Merkle-Offline-Baum. In der dritte Schicht $T = 3$ werden auch zwei vorläufigen Blöcke benötigt, um einen Winternitz One-Time Signaturschlüssel zu erzeugen. Das heißt, in der dritten Schicht braucht der Merkle-Online-Baum zwei vorläufig Block für $Seed_{OTS}$ und $Seed_{Next}$. Dasselbe gilt auch für den Merkle-Offline-Baum.

In unserem Verfahren braucht jeder GMSS-Baum nur zwei Blöcke und einen allg. Block, um die PRNG-Funktion anzuwenden und eine Signatur zu erzeugen. Der Merkle-Baum $\tau_{1,0}$ in der ersten Schicht, $3 \cdot 160$ bits in dem EEPROM wird besetzt und in $T = 2$ brauchen wir $6 \cdot 160$ bits in dem EEPROM, sowie in $T = 3$ brauchen wir $6 \cdot 160$ bits und $1 \cdot 160$ für x_i . zwei vorläufige Blöcke wie in ersten Verfahren (siehe Abbildungen 5.7, 5.8) erfordert, um jeden Knoten und Blätter zu erzeugen. Um im selben Zeitpunkt die Schlüssel Generation und den Aufbau des GMSS-Baumes zu ermöglichen, lesen sie das erste Verfahren.

5.10.3 Anforderungsspeicher EEPROM für Signatur Generation

Die Größe der Signatur beruht auf den Winternitz OTSS Parameter w welche auch als Hash-Funktion benutzt wird.

Um eine Signatur erzeugen zu können, gibt es drei Phasen. In der ersten Phase wird die Nachricht d_i signiert

$$Sig(d_i); i = 1, \dots, 8$$

In der zweiten Phase wird die Wurzel des Merkle-Baums

$$Sig(\tau_{3,j}); j = 0, \dots, 511$$

signiert. In der dritter Phase wird die Wurzel des Merkle-Baum

$$Sig(R_{\tau_{2,j}}); j = 0, \dots, 31$$

signiert. Für die Erzeugung einer Signatur werden drei Merkle-Bäume erfordert. Der erste Baum in der dritten Schicht Merkle-Online-Baum, hat $2^3 = 8$ Blätter. Der zweite Merkle-Online-Baum in der zweite Schicht hat $2^4 = 16$ Blätter. Der dritte Merkle-Baum in der erste Schicht hat $2^5 = 32$ Blätter. Das Tag braucht $43 \cdot 160$ bits in dem EEPROM, um eine Signatur der Nachricht $Sig(d_i)$ zu speichern und braucht $43 \cdot 160$ bits in dem EEPROM, um die Signatur einer Wurzel des Merkle-Online-Baums $Sig(\tau_{3,j})$ zu sichern, sowie $43 \cdot 160$ bits in dem EEPROM, um die Signatur der Wurzel des Merkle-Online-Baums $Sig(\tau_{2,j})$ zu speichern. Der Nachteil dieses Verfahren ist, dass die Authentifizierungspfade

$$Auth_{\tau_{1,0}}, Auth_{\tau_{2,j}}, Auth_{\tau_{3,j}}$$

5 GMSS Signatur Generation für RFID-Tags

während des Aufbaus des GMSS-Baums nicht berechnet werden. Sondern während der Signatur einer Nachricht berechnet das Tag einen Authentifizierungspfad und bereitet einen Authentifizierungspfad für die nächste Signatur vor. Mit diese Methode kann das Tag Speicher sparen, um mehr Signaturen zu erzeugen; aber das kostet mehr Berechnungen. In diesem Verfahren beträgt, die Speicherkapazität, die das *RFID*-Tag braucht, um ein Signatur herzustellen, $(43 + 43 + 43 = 129) \cdot 160$ bits.

GMSS-Baum Generation Der aktuelle GMSS-Baum besteht aus fünf Bäume. Ein Merkle-Baum in der Schicht $T = 1$, der $2^5 = 32$ Blätter hat: Jedes Blatt fordert

$$(2^w - 1) \cdot t_w + (t_w - 1) = (2^4 - 1) \cdot 43 + 42 = 687$$

Hash und

$$(t_w + 1) = (43 + 1) = 44$$

PRNG. Für alle Blätter Merkle-Baum ($687 \cdot 32 = 21984$) Hash und ($44 \cdot 32 = 1408$) PRNG. Ein Merkle-Baum mit 2^5 erfordert ($2^5 - 1 = 31$) Hash, um alle Knoten zu berechnen. Ein Merkle-Baum in der erster Schicht braucht ($21984 + 31 = 22015$) Hash, um 1408 PRNG zu berechnen. Der Online-Merkle-Baum in der Schicht $T = 2$, der $2^4 = 16$ Blättern und $2^4 - 1 = 15$ Knoten hat. Die Zahl von Hash, die ein Online-Merkle-Baum braucht, ist $16 \cdot [(2^4 - 1) \cdot 43 + 42] + 15 = 11007$ Hash und $44 \cdot 16 = 704$ PRNG. Der Merkle-Offline-Baum in der zweite Schicht wird während der ersten 2^4 Signatur aufgebaut. Der Merkle-Offline-Baum kostet 11007 Hash und 704 PRNG. Der Merkle-Online-Baum hat in der Schicht $T = 3$, der $2^3 = 8$ Blätter und $2^3 - 1 = 7$ Knoten. Die Zahl der Hash, die einen Merkle-Online-Baum braucht, ist $8 \cdot [(2^4 - 1) \cdot 43 + 42] + 7 = 5503$ Hash und $44 \cdot 8 = 352$ PRNG. Der Merkle-Offline-Baum in der dritte Schicht wird während des ersten 2^3 Signatur aufgebaut. Der Merkle-Offline-Baum kostet 5503 Hash und 352 PRNG. Die totalen Kosten der GMSS-Baum Generation in diesem Verfahren beträgt $22015 + 11007 + 11007 + 5503 + 5503 = 55035$ Hash und $1408 + 704 + 704 + 352 + 352 = 3520$ PRNG.

Um Speicherkapazität zu gewinnen und mehr Signaturen erzeugen zu können, muss das Tag während einer Signatur noch mal die PRNG-Funktion aufrufen und den Hash-Wert wieder berechnen. Für den Merkle-Online-Baum in der Schicht $T = 3$ berechnet das Tag die Werte der Blätter und Knoten, die nicht in dem EEPROM gesichert werden. In der erste Stufe haben wir 4 Blätter und in der zweite Stufe haben 2 Knoten, sowie dritte Stufe 1 knote. Die Zahl der berechnungen $4 \cdot [(2^4 - 1) \cdot 43 + 42] + 3 = 2751$ Hash und $44 \cdot 8 = 352$ PRNG. Für den Merkle-Online-Baum in der Schicht $T = 2$ berechnet das Tag die Werte der Blätter und Knoten, die nicht in dem EEPROM gespeichert werden. In der erste Stufe haben wir 8 Blätter, in der zweite Stufe haben 4 Knoten und in der dritten Stufe 2 Knoten, sowie in der vierten Schicht 1 Knoten. Die Zahl der Berechnungen $8 \cdot [(2^4 - 1) \cdot 43 + 42] + 7 = 5503$ Hash und $44 \cdot 16 = 704$ PRNG, dasselbe gilt für Merkle-Baum in der Schicht $T = 1$, Die Zahl der Berechnungen $16 \cdot [(2^4 - 1) \cdot 43 + 42] + 15 = 11007$ Hash und $44 \cdot 32 = 1408$ PRNG.

Größe des GMSS-Baums Um den GMSS-Baum speichern zu können, reservieren wir in EEPROM $2^5 - 1 = 31$ Blöcke für den Merkle-Baum in der erste Schicht und 3 Blöcke, um

die PRNG-Funktion zu berechnen. Die Blockgröße ist 160 bits. Für den Merkle-Online und Offline-Baum in der zweiten Schicht behalten wir in dem EEPROM $2^4 - 1 = 15$ Blöcke und 6 Blöcke für PRNG. Für den Merkle-Online und Offline-Baum in der dritten Schicht behalten wir in EEPROM $2^3 - 1 = 7$ Blöcke und 6 Blöcke für PRNG. 2 allg. Blöcke für Berechnung jedes Knotens und einen allg. Block für Geheimschlüssel x_i . Insgesamt ist die Größe GMSS-Baum $(31 + 15 + 7 + 3 + 6 + 6 + 2 + 1) = 71$ Blöcke also $(71 \cdot 160)$ bits.

Größe der Signatur und des Overheads Die Nummer einer Signatur für eine Nachricht in diesen Verfahren beträgt 3. Die erste Signatur ist die sog. Online-Signatur, welche die Nachricht digest d_i ; $i = 1, \dots, 8$ bei Anwendung der Blätter des Merkle-Online-Baums in der dritten Schicht signiert hat. Die zweite Signatur heißt, Offline-Signatur, welche die Wurzel des Merkle-Online-Baums in der zweiten Schicht signiert hat. Die dritte Signatur, Offline-Signatur, welche die Wurzel des Merkle-Online-Baums in der zweiten Schicht signiert hat. Um eine Signatur erzeugen zu können, sind drei Phasen vorhanden. In der ersten Phase wird die Nachricht digest signiert $Sig(d_i)$ $i = 1, \dots, 8$, die 323 Hash kostet. In der zweite Phase wird die Wurzel des Merkle-Online-Baums signiert $Sig(\tau_{3,j})$; $j = 0, \dots, 511$, die 323 Hash kostet. In der dritte Phase wird die Wurzel des Merkle-Online-Baums signiert $Sig(\tau_{2,j})$; $j = 0, \dots, 31$, die 323 Hash kostet.

Um $Sig(d_i)$ erzeugen zu können, braucht das *RFID*-Tag einen Platz im EEPROM t_w mit 43 Blöcken. Dasselbe gilt auch für $Sig(\tau_{3,j})$ und $Sig(\tau_{2,j})$. Die Größe der Signatur einer Nachricht ist $(43 + 43 + 43 = 129) \cdot 160$ bits. Nachder vorherigen Analyse der GMSS Signatur ist die Speicherkapazität des EEPROM, die das *RFID*-Tag braucht, um das GMSS One-Time signatur Schema anzuwenden $(71 + 129 = 200) \cdot 160$ bits.

Zur Authentifizierung des Lesegeräts gegenüber dem Tag muss dieses Tag Verifikationsaten zum Lesegerät übertragen. Die Verifikationsdaten bestehen aus:

- der One-Time Signatur der Nachricht digest $Sig(d_i)$ für das Blatt d_i und für den Authentifizierungspfad für dieses Blatt $Auth_{\tau_{3,j},l}$. Die Größe des $Auth_{\tau_{3,j},l}$ Pfads ist gleich der Höhe des Merkle-Online-Baums $h_3 = 3$ Blöcke.
- der One-Time Signatur der Wurzel des Merkle-Online-Baum $Sig(\tau_{3,j})$ und des Authentifizierungspfads für diese Wurzel $Auth_{\tau_{2,j},l}$. Die Größe des $Auth_{\tau_{2,j},l}$ Pfads ist gleich der Höhe des Merkle-Online-Baums in der zweiten Schicht $h_2 = 4$ Blöcke.
- der One-Time Signatur der Wurzel des Merkle-Online-Baums $Sig(\tau_{2,j})$ und des Authentifizierungspfads für diese Wurzel $Auth_{\tau_{1,0},l}$. Die Größe des $Auth_{\tau_{1,0},l}$ Pfads ist gleich der Höhe des Merkle-Baums in der ersten Schicht $h_1 = 5$ Blöcke.

Das *RFID*-Tag stellt ein Frame mit overhead her, um die Verifikationsdaten in dem overhead hochzuladen und an das Lesegerät zu senden. Die Größe des overheads beträgt:

$$(Sig(d_i) + Sig(\tau_{3,j} + Sig(\tau_{2,j}) + Auth_{\tau_{3,j},l} + Auth_{\tau_{2,0},l} + Auth_{\tau_{1,0},l})$$

5 GMSS Signatur Generation für RFID-Tags

$$= (43 + 43 + 43 + 3 + 4 + 5) = 141 \cdot 160$$

bits.

IEEE 802.11 definiert ein neues Frame, um die Verifikationsdaten zwischen Lesegerät und *RFID*-Tag zu übertragen (siehe Tabelle 2.5). Die Größe dieses Frames könnte bis 0.5 Mbit/s erreichen.

Es wird eine Tabelle 6.2 angeboten, die die Anwendung des zweiten Verfahrens GMSS One-Time Signatur auf *RFID*-Tag zusammenfasst.

T	3
h_1	5
h_2	4
h_2	3
w	4
$SHA - 1, n$	160
t_w	43
Anzahl der Signatur	4096
Größe des GMSS-Baums	$71 \cdot 160 \text{ bits} = 1,42 \text{ kbyte}$
Größe der GMSS-Signatur	$129 \cdot 160 \text{ bits} = 2,85 \text{ Kbyte}$
GMSS Generation	55035 Hash
GMSS Generation	3520 PRNG
One Signatur Generation	$323 \cdot 3 = 969 \text{ Hash}$
Update des Merkle-Baums $\tau_{3,j}$	2751 Hash
Update des Merkle-Baums $\tau_{3,j}$	352 PRNG
Update des Merkle-Baums $\tau_{2,j}$	5503 Hash
Update des Merkle-Baums $\tau_{2,j}$	704 PRNG
Update des Merkle-Baums $\tau_{1,0}$	11007 Hash
Update des Merkle-Baums $\tau_{1,0}$	1408 PRNG
Komplette Speicherkapazität	$200 \cdot 160 \text{ bits} = 4 \text{ kbyte}$
Größe des Overheads	$141 \cdot 160 \text{ bits} = 2,82 \text{ Kbyte}$

Tabelle 5.2: GMSS One-Time Signatur Parameter

5.11 Berechnung der Authentifizierungspfade

Wie bereits erklärt berechnet dieses Verfahren nicht alle Authentifizierungspfade während der Generation des GMSS-Baums, sondern es benutzt eigene Algorithmen, um den Authentifizierungspfad für eine Signatur zu berechnen. Das Tag hat nur die geraden Knoten und Blätter

5 GMSS Signatur Generation für RFID-Tags

während der GMSS-Baum Generation berechnet und in seinem EEPROM gespeichert. Das Algorithmus liest den Index einer Signatur

$$d_i ; i = 1, 3, 5, \dots, 2^h - 1$$

Wenn dieser Index ungerade ist, ruft der Algorithmus die PRNG-Funktion auf, um den Winternitz One-Time Signaturschlüssel für dieses Blatt berechnen zu können. Das Ergebnis ist, der Wert des Blatts wurde berechnet. Danach signiert das Tag diese Nachricht und updatet seinen EEPROM, um einen neuen Authentifizierungspfad für eine nächste Signatur vorzubereiten. (siehe Abbildung 5.25) zeigt uns, dass bevor das Tag den $Auth_{\tau_{3,0,1}}$ Pfad bestimmt, es den Wert des Blatts d_1 bei Anwendung der PRNG-Funktion $H(1)$ [687 Hash und 44 PRNG] berechnet. Dieser Wert bleibt gespeichert in dem selben allg. Block (siehe Abbildung 5.8), weil das Tag weiss, dass es $H(1)$ für den nächste Authentifizierungspfad braucht.

Der $Auth_{\tau_{3,0,1}}$ Pfad besteht aus den Werten ($H(2), H(3, 4), H(5, 8)$) für d_1 in der Schicht $T = 3$ und dann sendet das Tag die $Sig(d_1)$ und die $Auth_{\tau_{3,0,1}}$ an das Lesegerät.

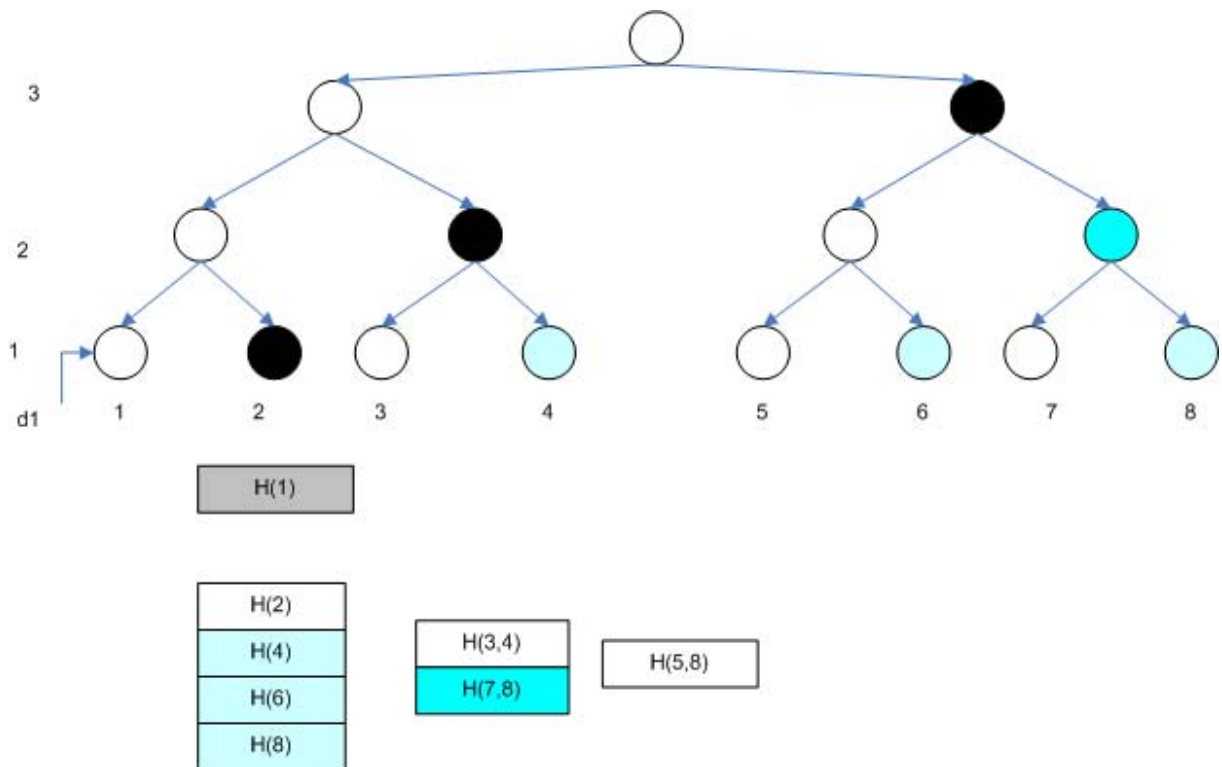


Abbildung 5.25: Authentifizierungsbaum für d_1 und die Blöcke

Bevor das Tag die Signatur d_2 erzeugen kann, updatet es seinen EEPROM. Das Tag speichert die neuen Hash-Werte in den selben Blöcken, die die geraden Index Knoten oder Blätter haben. Es ersetzt die Blöcke, die die alten Hash-Werte haben, mit den neuen Hash-Werten. Die (Abbildung 5.26) erklärt, wie das Tag den

$$H(1, 2) = H(H(1) || H(2))$$

5 GMSS Signatur Generation für RFID-Tags

berechnet und speichert ihn in den selben Block, der den Wert $H(2)$ hat, weil das Tag nicht mehr dieses Blatt braucht. Der $Auth_{\tau_{3,0,2}}$ für d_2 besteht aus den folgenden Werten $H(1)$, $H(3, 4)$, $H(5, 8)$

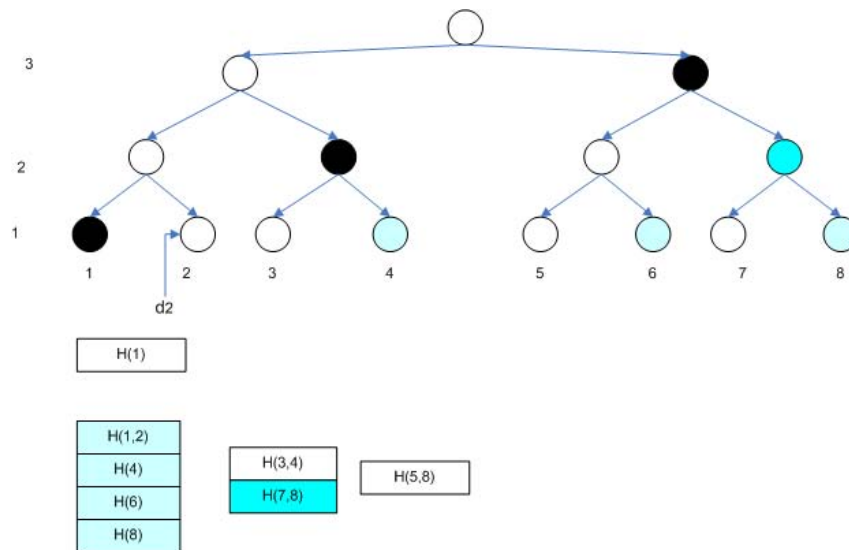


Abbildung 5.26: Authentifizierungsbaum für d_2 und die Blöcke

Vor der Erzeugung der Signatur d_3 berechnet das Tag den Wert

$$H(1, 4) = H(H(1, 2) || H(3, 4))$$

und löscht den Wert $H(3, 4)$ und wird ersetzt durch diesen.

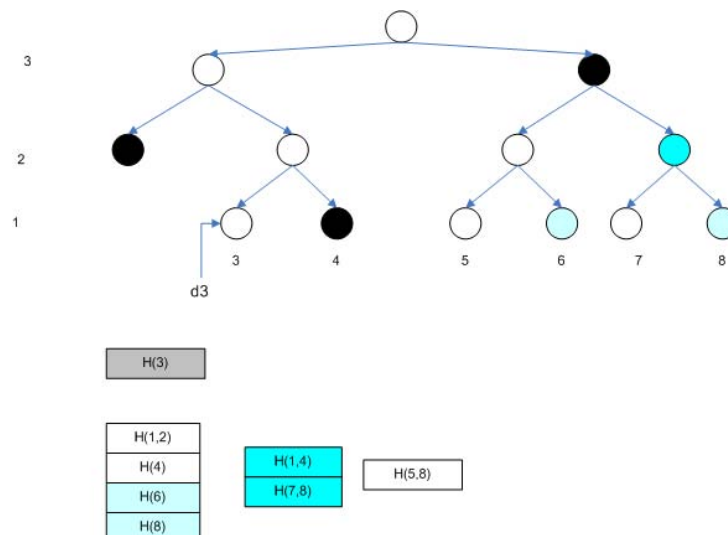


Abbildung 5.27: Authentifizierungsbaum für d_3 und die Blöcke

5 GMSS Signatur Generation für RFID-Tags

Das Tag bestimmt den $Auth_{\tau_3,0,3}$ und berechnet den Wert des Blatts d_3 bei Anwendung der PRNG-Funktion $H(3)$ [687 Hash und 44 PRNG]. Dieser Wert bleibt gespeichert im allg. Block. Der $Auth_{\tau_3,0,3}$ Pfad besteht aus den Werten $(H(4), H(1, 2), H(5, 8))$ für d_3 dann sendet das Tag die $Sig(d_3)$ und $Auth_{\tau_3,0,3}$ an das Lesegerät.

Um die nächste Nachricht d_4 signieren zu können, hat das Tag seinen $Auth_{\tau_3,0,4}$ Pfad $(H(3), H(1, 2), H(5, 8))$ schon im EEPROM vorbereitet (siehe Abbildung 5.28).

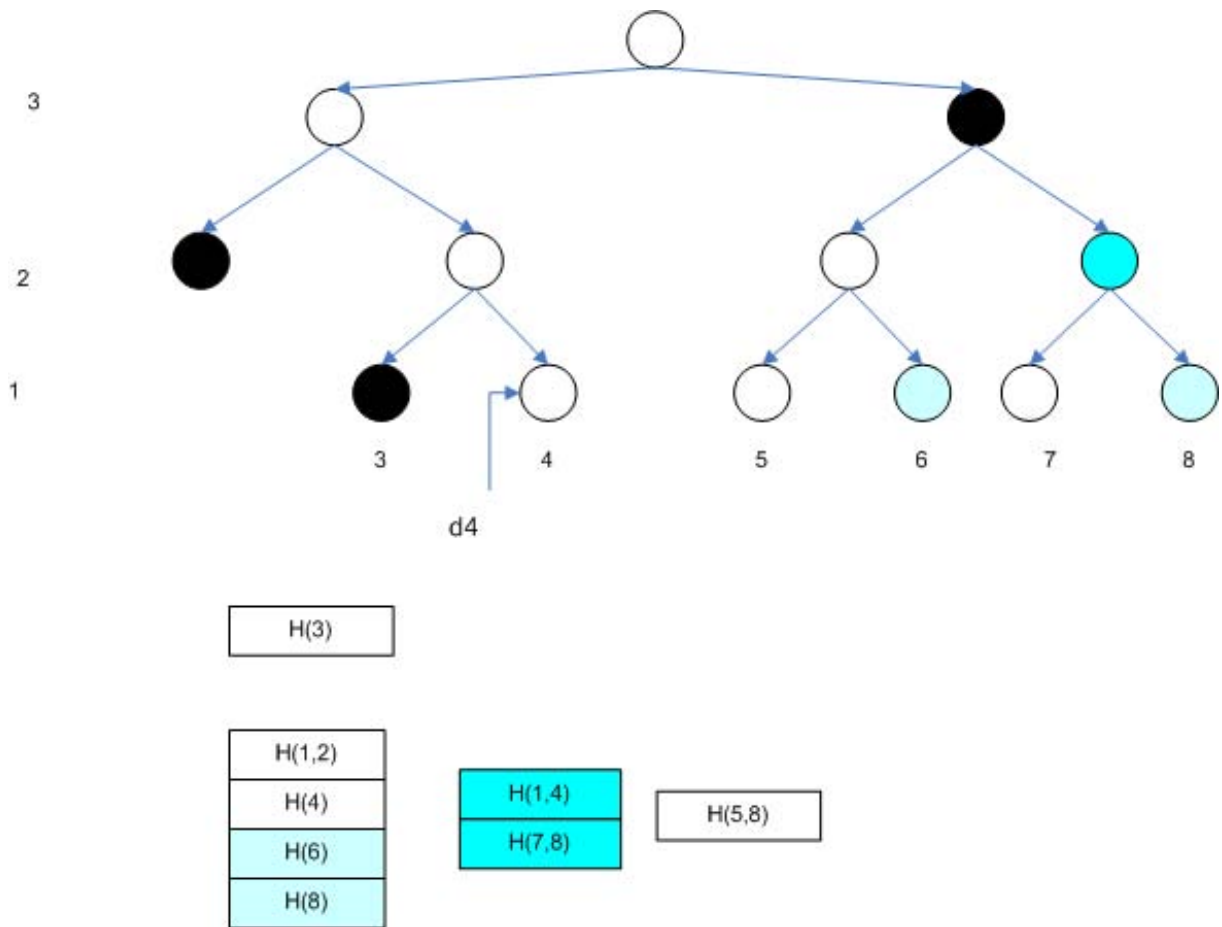


Abbildung 5.28: Authentifizierungsbaum für d_4 und die Blöcke

Es ist klar, dass diese GMSS One-Time Signatur auch auf den Algorithmus Offline-Baum beruht, um die alle Knoten und Blätter zu löschen und seine Speicher zu leeren, die das Tag nicht mehr braucht (siehe Abbildung 5.29)

5 GMSS Signatur Generation für RFID-Tags

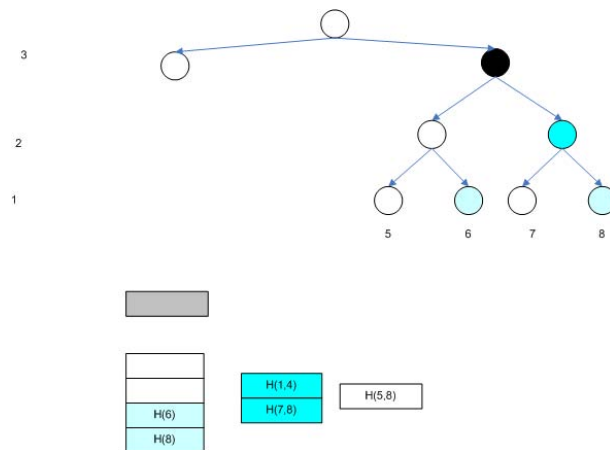


Abbildung 5.29: Anwendung der Offline-Baum

An dieser Stelle werden die Werte $(H(1,2), H(4))$ von dem EEPROM gelöscht. Das Tag findet eine Gelegenheit, um einen Merkle-Offline-Baum in der Schicht $T = 3$ vorzubereiten (siehe Abbildung 5.30)

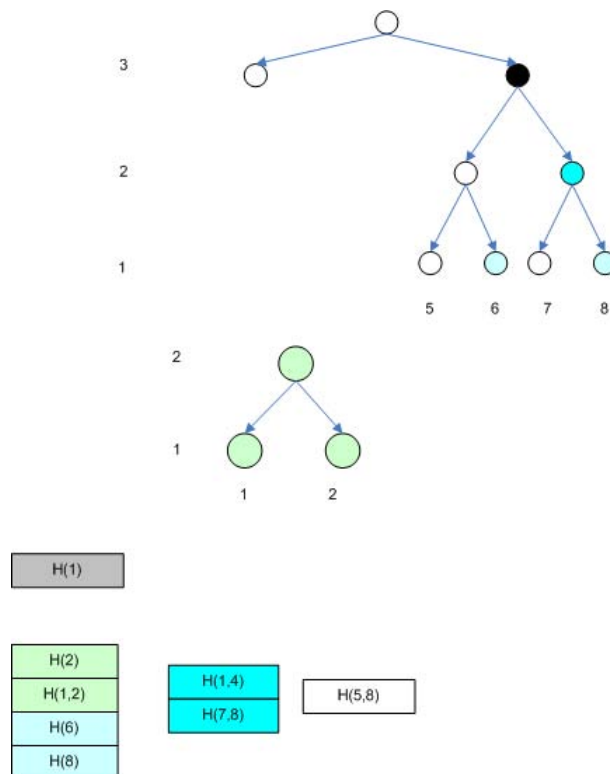


Abbildung 5.30: Erzeugung eines Offline-Baums

5 GMSS Signatur Generation für RFID-Tags

Das Tag berechnet die Werte $(H(1), H(2), H(1, 2))$ für den Merkle-Offline-Baum und fängt mit den geraden Knoten und Blättern an, um sie im EEPROM zu speichern an. Danach bestimmt das Tag den $Auth_{\tau_3,0,5}$ und berechnet den Wert des Blatts d_5 . Dieser Wert bleibt gespeichert im allg. Block. Der $Auth_{\tau_3,0,5}$ Pfad besteht aus den Werten $(H(6), H(7, 8), H(1, 4))$ für d_5 dann sendet das Tag die $Sig(d_5)$ und $Auth_{\tau_3,0,5}$ an das Lesegerät (siehe Abbildung 5.31).

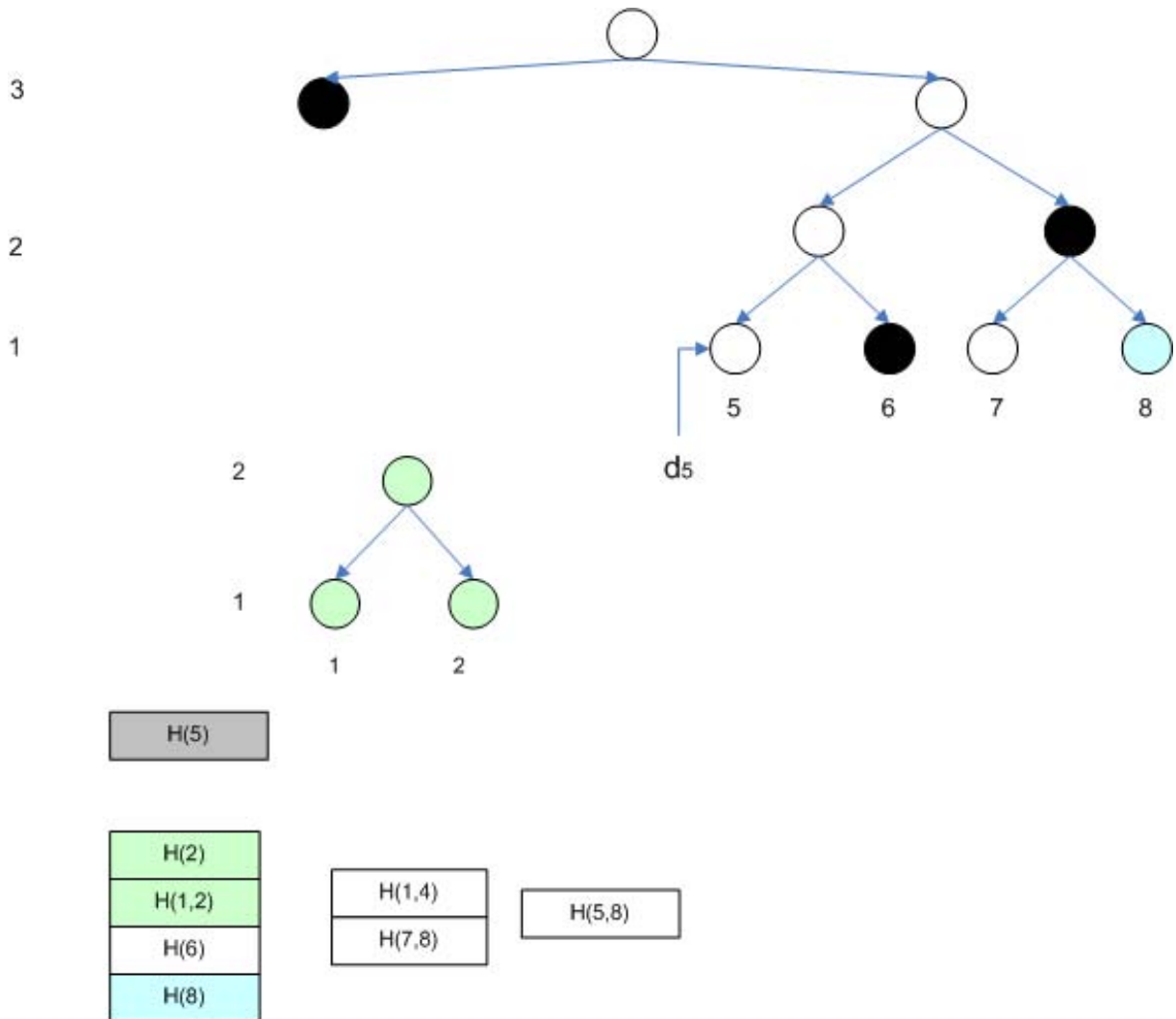


Abbildung 5.31: Authentifizierungsbaum für d_5 und die Blöcke

Um die nächste Nachricht d_6 signieren zu können, hat das Tag seinen $Auth_{\tau_3,0,6}$ Pfad $(H(5), H(7, 8), H(1, 4))$ schon im EEPROM vorbereitet (siehe Abbildung 5.32).

5 GMSS Signatur Generation für RFID-Tags

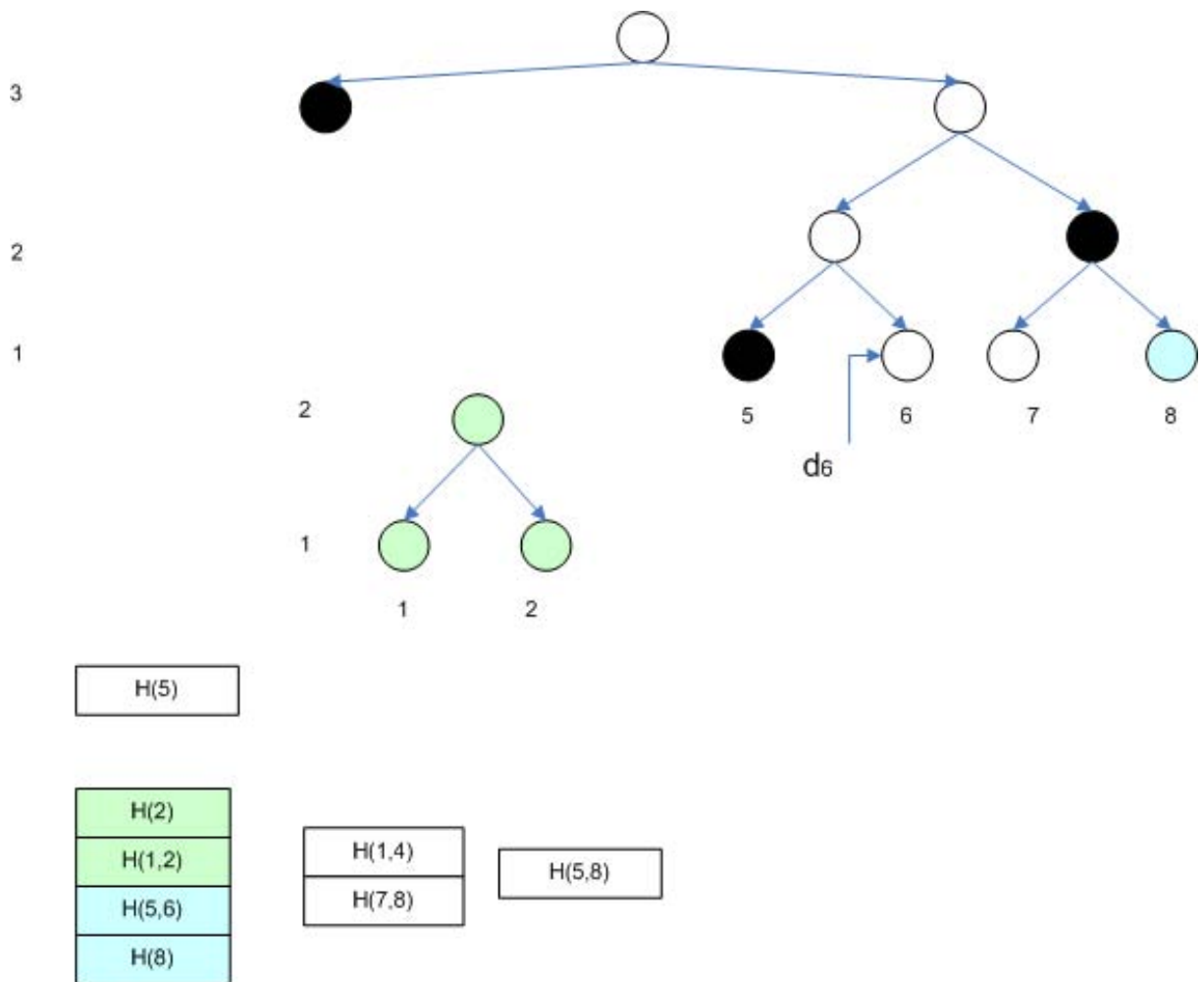


Abbildung 5.32: Authentifizierungsbaum für d_6 und die Blöcke

Die (Abbildung 5.32) erklärt, wie das Tag den

$$H(5, 6) = H(H(5) || H(6))$$

berechnet und ihn im selbem Block speichert, der den Wert $H(6)$ hat, weil das Tag nicht dieses Blatt braucht. Der $Auth_{\tau_{3,0,6}}$ für d_6 besteht aus die folgenden Werten $H(5)$, $H(7, 8)$, $H(1, 4)$ Vor der Erzeugung der Signatur d_7 berechnet das Tag den Wert $H(7)$. Danach signiert es d_7 und sendet $Sig(d_7)$ und $Auth_{\tau_{3,0,7}}$ an das Lesegerät (siehe Abbildung 5.33).

5 GMSS Signatur Generation für RFID-Tags

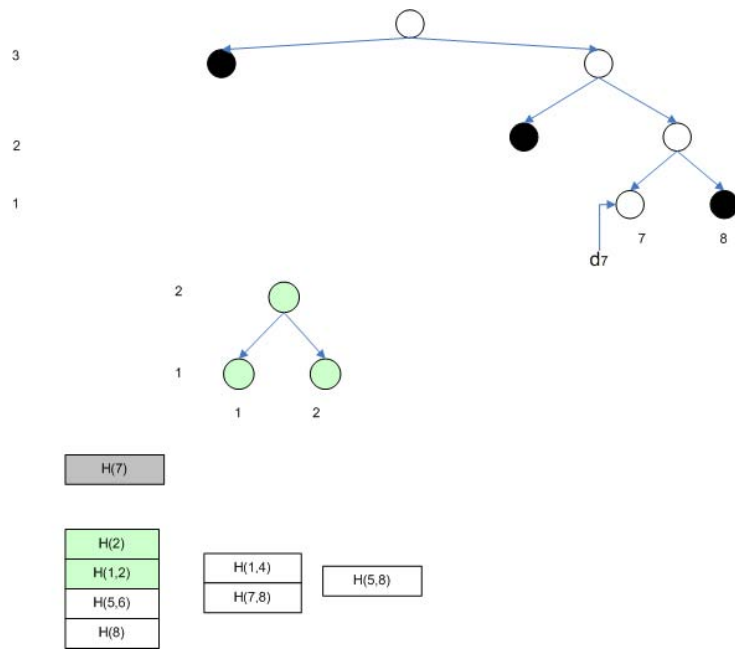


Abbildung 5.33: Authentifizierungsbaum für d_7 und die Blöcke

Der $Auth_{\tau_{3,0,8}}$ für d_8 besteht aus den folgenden Werten $H(7)$, $H(5,6)$, $H(1,4)$ (siehe Abbildung 5.34)

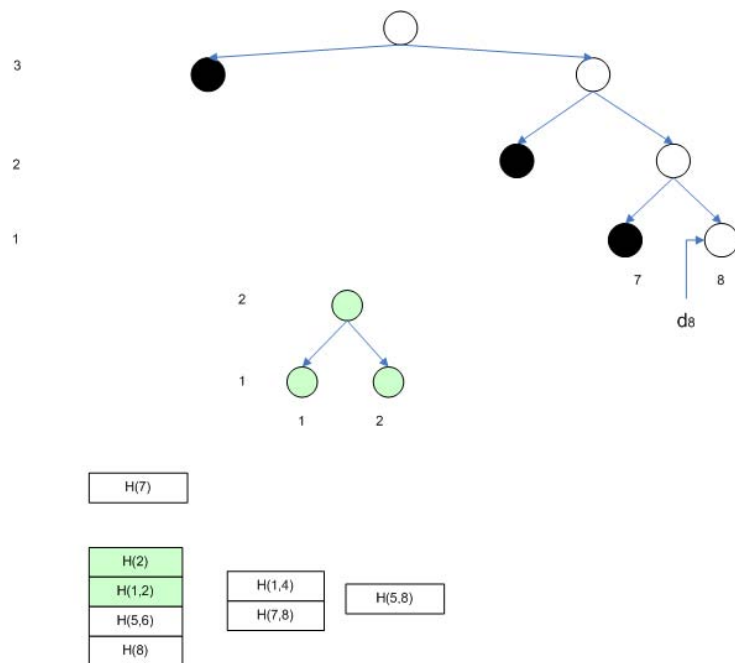


Abbildung 5.34: Authentifizierungsbaum für d_8 und die Blöcke

5 GMSS Signatur Generation für RFID-Tags

In dieser Stelle werden die Werte

$$(H(8), H(1, 4), H(5, 6), H(7, 8), H(5, 8))$$

von dem EEPROM gelöscht, und es werden neue Blätter und Knoten erzeugt, die in dem EEPROM gesichert werden (siehe Abbildung 5.35)

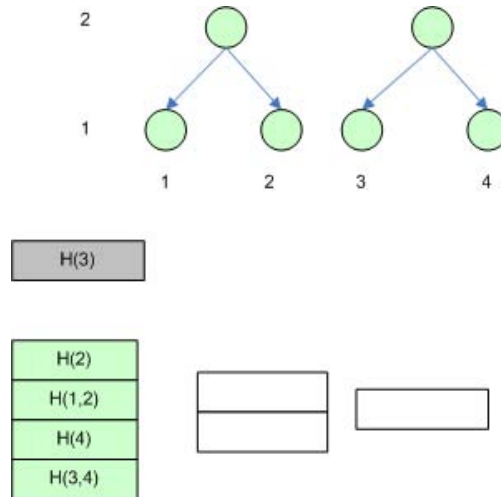


Abbildung 5.35: Erzeugung Merkle-Offline-Baum

Das Tag baut einen kompletten Merkle-Offline-Baum in der Schicht $T = 3$ (siehe die Abbildungen 5.36, 5.37, 5.38)

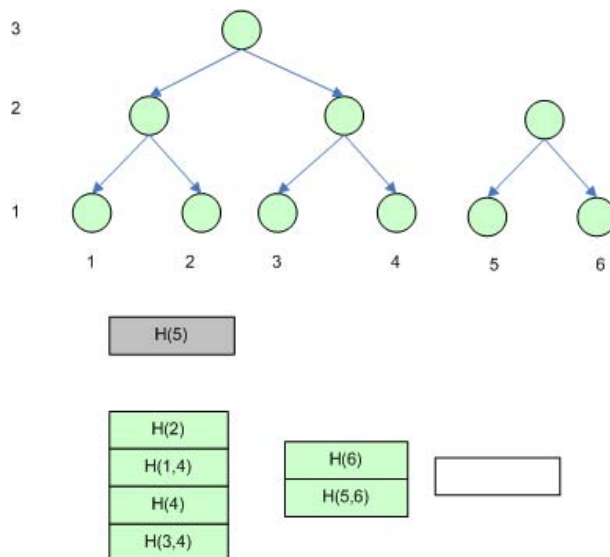


Abbildung 5.36: Erzeugung Merkle-Offline-Baum

5 GMSS Signatur Generation für RFID-Tags

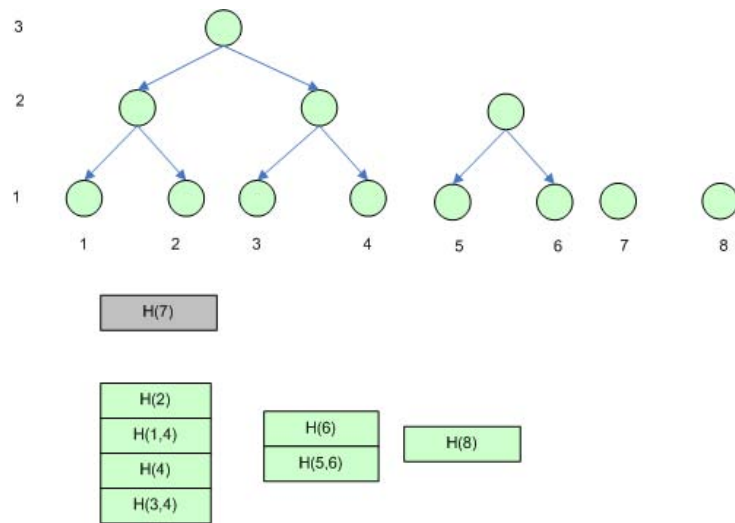


Abbildung 5.37: Erzeugung Merkle-Offline-Baum

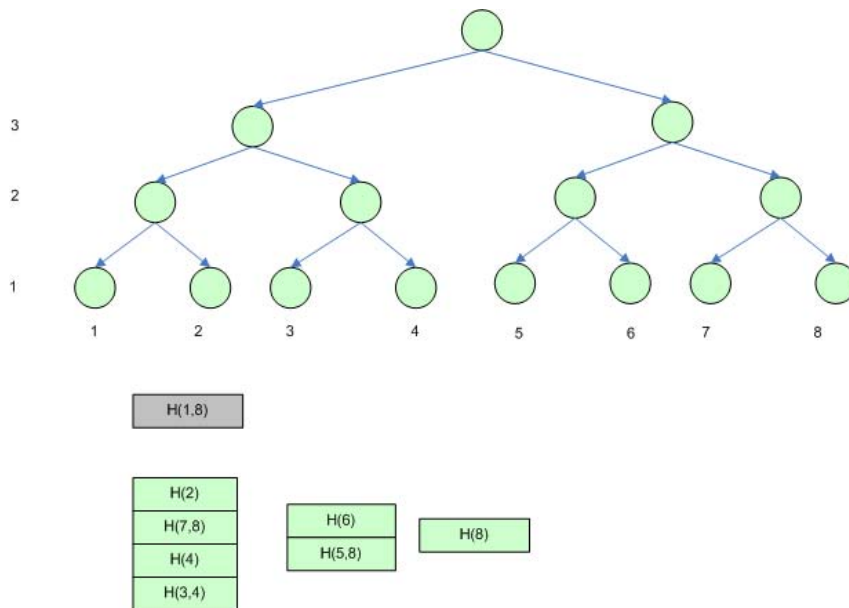


Abbildung 5.38: Merkle-Offline-Baum

In der Abbildung 5.38 sehen wir einen vollständigen Merkle-Offline-Baum $\tau_{3,1}$, der nach der achten Signatur aufgebaut wird. Der Merkle-Offline-Baum zum Online-Baum, damit das Tag in der nächsten Phase neue acht Nachrichten signieren kann. In der zweite Schicht $T = 2$ updatet das Tag den Merkle-Online-Baums $\tau_{2,0}$. Es nimmt das nächste Blatt d_2 und berechnet ihren Authentifizierungspfad $Auth_{\tau_{2,0},2}$. In der gleichen Methode nach der $2^4 = 16$. Signature wird den Merkle-Offline-Baum $\tau_{2,1}$ in der

5 GMSS Signatur Generation für RFID-Tags

Schicht $T = 2$ aufgebaut und das Tag updatet den Merkle-Baum $\tau_{1,0}$. Es nimmt das Blatt d_2 und berechnet ihren Authentifizierungspfad $Auth_{\tau_{1,0},2}$. Am Ende können wir sagen, es gibt zwei Phasen für das Update:

- In der Schicht $T = 2$ nach der Signierung der $2^3 = 8$ Nachrichten in der Schicht $T = 3$.
- In der Schicht $T = 1$ nach der Signierung der $2^4 = 16$ Nachrichten in der Schicht $T = 2$

6 Zusammenfassung

Zusammenfassend kann festgestellt werden. Bei der Anwendung des *RFID*-Tags möchte ein Kunde sicher sein, dass seine Ware von einer bestimmten Firma produziert wird. Die beste Methode ist, die kryptographische Authentikation auf *RFID*-Tag anzuwenden. Qualifizierte Signaturen gewährleisten Integrität und Authentizität der Nachrichten. Das *RFID*-Tag hat einen Mikroprozessor, der nur kleine Grund-Funktionen wie (AND, OR, XOR) durchführen kann, mit einem kleinen EEPROM von 4 Kbyte Speicherkapazität. Vorhandene digital Signatur Verfahren wie (*RSA*), (*DSA*) und (*ECDSA*) sind zu schwer auf dem *RFID*-Tag anzuwenden, weil die Sicherheit dieser Verfahren auf komplexen Berechnungen beruht. Die Einführung dieser Berechnungen überragt die Leistungsfähigkeit des Mikroprozessors des *RFID*-Tags und seiner Speicherkapazität. Die vorliegende Arbeit gibt einen Überblick über einen neuen Algorithmus, der zur Erzeugung von qualifizierten Signaturen auf *RFID*-Tag geeignet ist. Dieser Algorithmus passt die Leistungsfähigkeiten des *RFID*-Tag an.

Um das GMSS Merkle Signatur Schema auf das *RFID*-Tag anwenden zu können, werden die zu den *RFID*-Tagsfähigkeiten passenden Parameter der GMSS benötigt. Das GMSS One-Time Signatur Schema beruht auf verschiedene Parameter. Es benutzt sie zur Erzeugung von Signaturen. Dabei werden Analysen von praktischen Parametern für das GMSS Signatur Schema vorgesehen und dabei aufgezeigt welche, Parameterwerte, die schnelle Signaturzeit und passende Signaturgröße produzieren. Es werden alle Parameter der GMSS-Implementation beschrieben, die mit dem *RFID*-Tag zusammenpasst, so dass das Tag eine Nachricht signieren kann.

Das GMSS One-Time Signatur Schema ist ein gutes digital Signatur Schema für kleine Applikationen wie *RFID*-Tag. Es zielt auf eine schnelle Signatur und eine schnelle Verifizierung bei niedrigem Zeitaufwand und geringer Speicherkapazität ab. GMSS OTSS beruht bisher auf Hash-Funktionen und One-Time Signatur Schemata. In der vorliegenden Masterarbeit wendet GMSS OTSS die Hash-Funktion SHA-1 mit Ausgabe $n = 160$ bits und Winternitz One-Time Signatur Schema mit ausgewältem Parameter $w = 4$ an.

Die Variable t_w , welche die Signaturgröße feststellt, beträgt $t_w = 43$. Um GMSS OTSS auf *RFID*-Tag anwenden zu können, werden zwei Verfahren vorgeschlagen. Das erste Verfahren beruht auf dem Prinzip, dass der ganze GMSS-Baum im EEPROM des Tags gespeichert werden muss. Der GMSS-Baum hat zwei Schichten. Die erste Schicht hat einen Merkle-Baum mit der Höhe $h_1 = 5$, die zweite Schicht hat einen Merkle-Online-Baum mit Höhe $h_2 = 4$ und einen Merkle-Offline-Baum. Die Anzahl der Signaturen, die mit diesen Verfahren erzeugt werden können, sind klein und betragen 512 Signaturen. Aber die Zeit zur Erzeugung einer neuer Signatur ist zu schnell, weil alle Hash-Werte für die Knoten und die Blätter im EEPROM schon gesichert werden. Das Tag erfordert nicht, extra Berechnungen durchzuführen, um die Authentifizierungspfade bestimmen zu können. Für jede neue Signatur nimmt das Tag vom EEPROM

6 Zusammenfassung

die passenden Hash-Werte, die dieser Signatur gehören. Es wird eine Tabelle 6.1 angeboten, die die Anwendung der GMSS One-Time Signatur auf *RFID*-Tag zusammenfasst.

T	2
h_1	5
h_2	4
w	4
$SHA - 1, n$	160
t_w	43
Anzahl der Signatur	512
Größe des GMSS-Baums	$106 \cdot 160 \text{ bits} = 2 \text{ Kbyte}$
Größe der GMSS-Signatur	$86 \cdot 160 \text{ bits} = 1,7 \text{ Kbyte}$
GMSS Generation	44029 Hash
GMSS Generation	2816 PRNG
One Signatur Generation	646 Hash
Größe des Authentifizierungspfads	$9 \cdot 160 \text{ bits} = 1,44 \text{ Kbit}$
Komplette Speicherkapazität	$192 \cdot 160 \text{ bits} = 3,8 \text{ Kbyte}$
Größe des Overheads	$95 \cdot 160 \text{ bits} = 2 \text{ Kbyte}$

Tabelle 6.1: GMSS One-Time Signatur Parameter

Um die Anzahl der Signaturen erhöhen zu können, wird ein neues Verfahren angeboten. Das zweite Verfahren beruht auf dem Prinzip, dass der halbe GMSS-Baum im EEPROM des Tags gespeichert werden muss, so dass die Knoten und die Blätter, welche einen geraden Index haben, im EEPROM gesichert werden. Der GMSS-Baum hat drei Schichten. Die erste Schicht hat einen Merkle-Baum mit der Höhe $h_1 = 5$, die zweite Schicht hat einen Merkle-Online-Baum mit Höhe $h_2 = 4$ und einen Merkle-Offline-Baum. Die dritte Schicht hat einen Merkle-online-Baum mit Höhe $h_3 = 3$ und einen Merkle-Offline-Baum. Die Anzahl der Signaturen, die mit diesem Verfahren erzeugt werden können, sind größer als das erste Verfahren und betragen 4096 Signaturen. Aber die Zeit zur Erzeugung einer neuen Signatur dauert länger, weil die Hash-Werte für die Knoten und die Blätter, über einen die ungeraden Index verfügen, nicht im EEPROM vorher gespeichert werden. Das Tag erfordert zusätzliche Berechnungen von Hash-Funktion und PRNG-Funktion, um die Authentifizierungspfade bestimmen zu können. Für jede neue Signatur berechnet das Tag einige Hash-Werte, die dem Authentifizierungspfad dieser Signatur gehört. Es wird eine Tabelle 6.2 angeboten, die die Anwendung des zweiten Verfahrens GMSS One-Time Signatur auf *RFID*-Tag zusammenfasst.

6 Zusammenfassung

T	3
h_1	5
h_2	4
h_2	3
w	4
$SHA - 1, n$	160
t_w	43
Anzahl der Signatur	4096
Größe des GMSS-Baums	$71 \cdot 160 \text{ bits} = 1,42 \text{ kbyte}$
Größe der GMSS-Signatur	$129 \cdot 160 \text{ bits} = 2,85 \text{ Kbyte}$
GMSS Generation	55035 Hash
GMSS Generation	3520 PRNG
One Signatur Generation	$323 \cdot 3 = 969 \text{ Hash}$
Update des Merkle-Baums $\tau_{3,j}$	2751 Hash
Update des Merkle-Baums $\tau_{3,j}$	352 PRNG
Update des Merkle-Baums $\tau_{2,j}$	5503 Hash
Update des Merkle-Baums $\tau_{2,j}$	704 PRNG
Update des Merkle-Baums $\tau_{1,0}$	11007 Hash
Update des Merkle-Baums $\tau_{1,0}$	1408 PRNG
Komplette Speicherkapazität	$200 \cdot 160 \text{ bits} = 4 \text{ kbyte}$
Größe des Overheads	$141 \cdot 160 \text{ bits} = 2,82 \text{ Kbyte}$

Tabelle 6.2: GMSS One-Time Signatur Parameter

Literaturverzeichnis

- [1] Bundesamt für Sicherheit in der Informationstechnik *Risiken und Chancen des Einsatzes von RFID-Systemen Trends und Entwicklungen in Technologien, Anwendungen und Sicherheit*.
- [2] International Organization for Standardization. ISO/IEC 18000-3. Information Technology AIDC Techniques - RFID for Item Management, March 2003.
- [3] Technical report Micronetwork Interfaces for RFID Tags: A New Paradigm for Reader-Tag Communication Daniel W. Engels.
- [4] Raj Bridgelall Symbol Technologies Research and Development Holtsville, NY 11742-1300.
- [5] Bluetooth/802.11 Protocol Adaptation for RFID Tags Symbol Plaza, Holtsville, NY 11742, USA.
- [6] The Japan radio regulations regarding FHSS-type RFID were revised in 2002 and 2003.
- [7] Adrian Perrig. The BiBa one-time signature and broadcast authentication protocol. In ACM Conference on Computer and Communications Security, pages 2837, 2001.
- [8] Cryptography for Ultra-Low Power Devices by Jens-Peter Kaps A Dissertation Submitted to the Faculty of the *WORCESTER POLYTECHNIC INSTITUTE* In partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Electrical Engineering.
- [9] Einführung in die Kryptographie Reihe: Springer-Lehrbuch Buchmann, Johannes 3, erw. Aufl., 2004, XX, 266 S., Softcover ISBN: 978-3-540-40508-5
- [10] National Institute of Standards and Technology (NIST). FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.
- [11] National Institute of Standards and Technology (NIST) FIPS Publication 46-2: Data Encryption Standard, December 30, 1993.
- [12] Ronald L. Rivest. The MD5 message-digest algorithm. April 1992. RFC 1321.
- [13] Ralph C. Merkle Xerox PARC 3333 Coyote Hill Road, Palo Alto, Ca. 94304
- [14] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast

Literaturverzeichnis

- [15] Buchmann, Garica, Dahmen, Döring, Klintsevich, CMSS - An Improved Merkle Signature Scheme. In Cryptology ePrint Archive, Report 2006 / 320 (2006.) verifying. In Information Security and Privacy, 7th Australian Conference, ACISP 2002, pages 144-153. Springer, 2002.
- [16] CMSS - An Improved Merkle Signature Scheme. Johannes Buchmann, Luis Carlos Coronado, Erik Dahmen, Martin Döring, and Elena Klintsevich. Technische Universität Darmstadt
- [17] Federal Information Processing Standards Publication 180-1. 1995 April 17 Announcing the Standard for SECURE HASH STANDARD.
- [18] M. Szydło. Merkle tree traversal in log space and time. In C. Cachin and J. Camenisch, eds, Advances in Cryptology-Proc. EUROCRYPT 04, Volume 3027 of LNCS, PP.541 - 554. Springer - Verlage 2004. <http://szydlo.com/>.
- [19] <http://plato.stanford.edu/entries/qt-quantcomp/>.
- [20] SHOR, P. W: Algorithms for Quantum Computation: Discrete Logarithms and Factoring In: Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science.
- [21] Buchmann, Dahmen, May, Vollmer, Krypto 2020. Aussichten für langfristige kryptographische Sicherheit.
- [22] Buchmann, Garica, Dahmen, Döring, Klintsevich, CMSS - An Improved Merkle Signature Scheme. <http://eprint.iacr.org/>.
- [23] <http://www.embeddedinnovation.de/index.php?cID=ProdukteRFID>. United States Patent 20070085664. <http://www.freepatentsonline.com/20070085664.html>.
- [24] www.future-store.org.
- [25] <http://www.rsa.com/rsalabs/node.asp?id=2343>.
- [26] <http://cat.inist.fr/?aModele=afficheNcpsidt=17395980>.
- [27] <http://www.cs.uni-potsdam.de/ti/lehre/04-Kryptographie/inhalt.html>.
- [28] <http://www.itl.nist.gov/fipspubs/fip180-1.htm> 1995 April 17.
- [29] <http://engineers.ihs.com/document/abstract/SQQMCAAAAAAAAAAAAAA>.
- [30] <http://www.ieprox.com/files/Datasheets/060116-Mifare-DESfire-cards.pdf>.
- [31] <http://www.heise.de/newsticker/meldung/85299>.