ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Semesterarbeit

Broadcast-Authentifizierung für Pulsar

Fabian Hugelshofer
fabianhu@ee.ethz.ch

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Betreuer: Remo Meier, Thomas Locher, Stefan Schmid

Departement Informationstechnologie und Elektrotechnik
Eidgenössische Technische Hochschule (ETH) Zürich
Sommer 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die mittlerweile grosse Verbreitung von schnellen Internetzugängen ermöglicht auch Dienste mit höheren Bitraten. Video-Portale wie *YouTube*[1] oder *Google Video*[2] erfreuen sich grosser Beliebtheit. Auch herkömmliche TV-Angebote werden von den neuen Möglichkeiten beeinflusst. Diverse Sender stellen ihre Sendungen online zur Verfügung und digitale Videotheken wie *Maxdome*[3] oder *T-Online Video on Demand*[4] bieten Spielfilme nach Belieben. Neben den Angeboten auf Abruf (*on-demand*) existieren auch Angebote für die Liveübertragung von TV-Sendern. Beispiele hierfür sind *Cablecom Live-TV*[5] oder *ETH IPTV*[6]. Es ist damit zu rechnen, dass dieser Bereich auch in Zukunft starken Zulauf erhalten wird.

Die Übertragung der Videodaten kann in einem Peer-to-Peer (P2P) Netzwerk erfolgen. P2P hat den Vorteil einer sehr hohen Skalierbarkeit und äusserst geringen Infrastrukturkosten. Ein Beispiel für P2P-Livestreaming ist *Zattoo*[7], welches diverse TV-Sender an tausende von Zuschauern überträgt. Ein ebenfalls speziell für Livestreaming entwickeltes P2P-Protokoll ist das *Pulsar Protokoll* [LMSW07], welches im gleichnamigen Client[8] implementiert wird. Pulsar ermöglicht jedem Benutzer eigene Live-Videostreams einer grossen Anzahl von Teilnehmern zugänglich zu machen. Das Protokoll ist zuverlässig und effizient.

Da die Daten mit P2P von mehreren Peers weitergeleitet werden, besteht die Gefahr, dass ein bösartiger Benutzer Datenblöcke verändert oder selbst erzeugt. Er könnte damit versuchen seinen eigenen Stream einzuspielen oder einfach nur den Dienst zu stören. Um solche Angriffe auf die Verfügbarkeit zu verhindern, wird in dieser Arbeit ein Integritätsprotokoll für Pulsar entworfen, welches jedem Client ermöglicht, die Authentizität der Videodaten zu verifizieren. So kann verhindert werden, dass gefälschte Datenblöcke die Wiedergabe beeinträchtigen und möglicherweise im ganzen Netzwerk verbreitet werden.

Die Anforderungen an ein solches Integritätsprotokoll unterscheiden sich von herkömmlichen Signaturen. Um zusätzlichen Datenoverhead und Delay zu vermeiden, müssen die Signaturen möglichst klein sein und sofort verifiziert werden können. Es ist jedoch bereits ausreichend, wenn die Signaturen sicher sind, bis die Datenblöcke bei allen Empfängern eingetroffen sind. Signaturschemata wie FLASH [PCG01a], Quartz [PCG01b] und McEliece [JM96]

[1] `http://www.youtube.com`
[2] `http://video.google.com`
[3] `http://www.maxdome.de`
[4] `http://vod.t-online.de`
[5] `http://www.hispeed.ch/Movies___TV/Live_TV/`
[6] `http://www.iptv.ethz.ch`
[7] `http://www.zattoo.com`
[8] `http://www.getpulsar.com`

haben zwar sehr kleine Signaturen, sind jedoch nicht geeignet, da die Signierung und Verifikation zu lange dauert oder die Public-Keys in der Grössenordnung von einem MB sind. In dieser Arbeit werden Signaturschemata basierend auf den von Perrig [Per01] vorgestellten BiBa-Signaturen untersucht. Mit diesen auf Einwegfunktionen basierenden Signaturschemata lassen sich im Vergleich zu RSA oder DSA [MVO97] kleinere Signaturgrössen und deutlich geringerer Berechnungsaufwand erreichen.

2. Theorie

2.1. Pulsar

In diesem Abschnitt wird das Pulsar Protokoll vorgestellt, wie es von Locher et al. [LMSW07] beschrieben wird.

Das Pulsar Protokoll verteilt Livestreams auf eine effiziente Art an eine grosse Anzahl Clients. Dazu verwendet es einen Peer-to-Peer Overlay mit einem strukturierten Ansatz, um Datenblöcke möglichst schnell an möglichst viele Peers zu verteilen (*Push-Operation*), sowie einem unstrukturierten, der es Peers ermöglicht fehlende Blöcke gezielt anzufordern (*Pull-Operation*). Um an einem Stream teilzunehmen, nimmt ein Peer Kontakt mit einem Entry-Point auf und erhält von diesem eine Gruppe von zufälligen ersten Nachbarn zugewiesen. Da alle Peers den Stream gleichzeitig wiedergeben sollen, dient der Entry-Point auch der zeitlichen Synchronisation.

Datenblöcke von der Streamquelle werden in einer ersten Phase mittels Pushing entlang des strukturierten Overlays verteilt. Durch Präfixrouting wird für jeden Block ein aufspannender Baum induziert. Dieser bestimmt die Kanten, über welche das Paket im Netzerk verteilt wird. Um grosse Verzögerungen zu vermeiden, ist dabei wichtig, dass empfangene Blöcke möglichst schnell an die nötigen Nachbarn weitergeleitet werden. Grundsätzlich lassen sich auf diese Weise sämtliche Peers in kurzer Zeit und mit geringstem Overhead erreichen. Der Erfolg dieser Push-Operationen hängt von der Dynamik der Peers sowie der Zuverlässigkeit des Netzwerkes ab. Den Overlay verlassende Peers oder überlastete Leitungen führen zu Paketverlusten, wodurch einige Blöcke nicht alle Peers erreichen. In der Nähe der Quelle werden deshalb Blöcke von den Empfängern bestätigt, um sicherzustellen, dass sie weit genug gesendet wurden und somit eine grosse Anzahl Peers erreichen.

Fehlende Blöcke werden dann in einer zweiten Phase im unstrukturierten Overlay unter den Peers ausgetauscht. Dazu benachrichtigt jeder Peer seine Nachbarn über die Sequenznummern von neu erhaltenen Datenblöcken. Fehlende Blöcke können dann mit Pull-Operationen gezielt angefordert werden. Um den Overhead etwas zu verringern werden Benachrichtigungen und Anforderungen mit tatsächlichen Datenblöcken kombiniert, was zu einer begrenzbaren Verzögerung führt. Die Pull-Operationen machen das Protokoll sehr flexibel und verleihen ihm eine ausgesprochene Stabilität. Durch die nötigen Benachrichtigungen und Anforderungen entsteht aber auch eine zusätzliche Verzögerung sowie ein zusätzlicher Datenoverhead.

Ein typisches Videoframe hat eine Grösse von 1328 Bytes und wird mit einer Framerate von 36 Frames/s erzeugt. Die Framegrösse von 1328 Bytes ist insofern praktisch, da neben dem

Frame noch Benachrichtigungen über verfügbare Frames und Anforderungen von fehlenden Frames mitübertragen werden können, ohne dass ein zusätzliches IP Paket verschickt werden muss. Durch das Ausnutzen der gesamten MTU von 1500 B kann der Overhead für die Paketheader von IP und UDP reduziert werden. Da die Frames unterschiedliche Wege nehmen können und für die Pull-Operationen zusätzliche Zeit benötigt wird, ist ein Empfangspuffer nötig, um das Video flüssig abspielen zu können. Der Puffer im Pulsar Protokoll kann ca. 6 Sekunden zwischenspeichern.

Durch die Kombination von Push- und Pull-Operationen wird das Pulsar Protokoll zu einem schnellen, effizienten und robusten Livestreaming-Protokoll. Laut Locher et al. [LMSW07] dauert es in einem Netzwerk mit 100'000 Peers vom Erzeugen eines Datenblocks an der Quelle bis dieser alle Peers erreicht hat nicht länger als 1.5 Sekunden. Der Datenoverhead beträgt lediglich zwischen 1% und 3% und es ist möglich, dass 75% der Peers gleichzeitig den Overlay verlassen oder dass 5% der Pakete verloren gehen, ohne dass Frames zum Wiedergabezeitpunkt fehlen.

2.2. BiBa Signaturschema

Die von Perrig [Per01] vorgestellte BiBa Signatur basiert auf Einwegfunktionen ohne Trapdoor[1]. Sie ist einfach zu verifizieren und ist etwas kleiner als eine RSA Signatur. Auf der anderen Seite ist der Public-Key gross und das Erzeugen einer Signatur aufwändiger, als bei anderen auf Einwegfunktionen basierenden Signaturschemata.

Das Prinzip der BiBa Signatur basiert auf dem Geburtstagsparadoxon. Dieses Beispiel aus der Wahrscheinlichkeitstheorie zeigt, dass in einer Gruppe von 23 Personen zwei Personen mit mehr als 50% Wahrscheinlichkeit am selben Tag Geburtstag haben. Der Grund für diese erstaunlich grosse Wahrscheinlichkeit ist, dass mit 23 Personen insgesamt 253 verschiedene Paarungen möglich sind und nur bei einer davon die Geburtstage übereinstimmen müssen. In einer Gruppe von zwei Personen beträgt die Wahrscheinlichkeit, dass beide am gleichen Tag Geburtstag haben weniger als ein Prozent, was überproportional weniger ist.

Diese Asymmetrie wird vom BiBa Signaturschema ausgenutzt, wobei Personen durch Bälle und Tage durch Körbe ersetzt werden. Der Name „BiBa" steht für „bins and balls" (englisch für „Körbe und Bälle"). Dem Signierer steht eine grosse Anzahl Bälle zur Verfügung. Mittels einer Hash-Funktion werden die Bälle zufällig in Körbe verteilt. Eine Kollision von mehreren Bällen in einem Korb bildet dann die Signatur. Da ein Angreifer nur wenige Bälle kennt, ist es für ihn sehr schwierig eine Kollision zu erzeugen und damit eine Signatur zu erstellen.

Perrig [Per01] nennt die Bälle „*SEAL*s". Dies ist die Abkürzung für „**SE**lf **A**uthenticating va**L**ues" oder auf Deutsch „selbst authentifizierende Werte". Ein SEAL ist eine zufällige Zahl

[1] Eine Funktion $f : X \to Y$ ist eine Einwegfunktion, falls $f(x)$ für alle $x \in X$ einfach zu berechnen ist und es für ein zufälliges $y \in \mathrm{Im}(f)$ schwierig ist ein $x \in X$ zu finden, für das gilt $f(x) = y$. Eine Einwegfunktion $f : X \to Y$ hat eine Trapdoor, falls es mit einer Zusatzinformation (*Trapdoor-Information*) einfach wird für alle $y \in \mathrm{Im}(f)$ ein $x \in X$ zu finden, für das gilt $y = f(x)$. [MVO97]

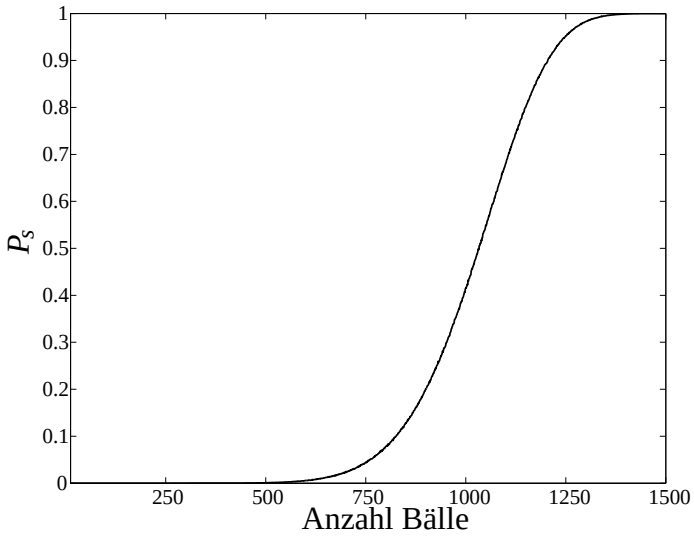

Abbildung 2.1.: Wahrscheinlichkeit einer erfolgreichen BiBa Signatur unter Verwendung einer gewissen Anzahl Bälle für $k = 12$ und $n = 222$ (durch eine Simulation über 100'000 Runden approximiert)

von b_S Bit Länge. Dem Signierer einer Nachricht m stehen t im Voraus erzeugte und voneinander verschiedene SEALs $S_1, S_2, \ldots, S_t$ zur Verfügung. Sie seien vorerst nur ihm bekannt. Sei $F : \{0,1\}^{b_S} \times \{0,1\}^{b_K} \to \{0,1\}^{b_S}$ eine Pseudo-Zufallsfunktion (PRF) [OGM86]. In $F_x(y)$ entspricht der Selektor x einem SEAL mit Bitlänge b_S und y dem Argument mit Bitlänge b_K. Das *Commitment* eines SEALs S_i ist definiert als $F_{S_i}(0)$ und die Commitments sämtlicher SEALs bilden den Public-Key, der den Verifizierern einer Nachricht bekannt ist und den sie dem Signierer authentisch zuordnen können. Die Funktion F ist eine Einwegfunktion und dient der Authentifikation von SEALs. Sie kann z.B. mit einer Hash-Funktion oder anderen Kryptoprimitiven implementiert werden.

Um eine Nachricht m zu signieren, wird zuerst deren Hash $h = H(m)$ berechnet, wobei H eine Hash-Funktion aus dem Random-Oracle-Modell [BR93] ist. Sei $G_h : \{0,1\}^{b_S} \to [0, n-1]$ eine Instanz einer Hash-Funktionsfamilie aus dem Random-Oracle-Modell mit Indikator h. Nun wird für jeden der SEALs $S_1, \ldots, S_t$ die Funktion G_h ausgewertet, was dem Verteilen der Bälle auf n Körbe entspricht. Findet sich in einem Korb eine k-fach-Kollision von SEALs $G_h(S_{x_1}) = \ldots = G_h(S_{x_k})$, so bildet $\langle S_{x_1}, \ldots, S_{x_k} \rangle$ die Signatur der Nachricht m und wird mit derselben mitgeschickt. Es werden also k SEALs pro Signatur freigegeben.

Die Signatur $\langle S_{x_1}, \ldots, S_{x_k} \rangle$ einer Nachricht m wird verifiziert, indem zuerst der Hash der Nachricht $h = H(m)$ berechnet wird und überprüft wird, ob alle SEALs paarweise voneinander verschieden sind und alle miteinander unter G_h kollidieren: $G_h(S_{x_1}) = \ldots = G_h(S_{x_k})$. Zusätzlich muss geprüft werden, ob die SEALs authentisch sind, d. h. ob für jeden SEAL S_{x_i} einer Signatur das Commitment $F_{S_{x_i}}(0)$ im Public-Key des Signierers vorhanden sind.

Die Wahrscheinlichkeit P_s mit einer gewissen Anzahl Bällen mindestens in einem von 222 Kör-

ben eine 12-fach-Kollision zu finden, ist in Abbildung 2.1 dargestellt. Die Parameter entsprechen denen von Perrig [Per01]. Da die Berechnung dieser Wahrscheinlichkeit die exzessive Anwendung von Inklusion/Exklusion erfordert, wurde sie mittels einer Simulation über 100'000 Runden approximiert. Es ist zu erkennen, dass ein Signierer mit einer grossen Anzahl Bällen mit grosser Wahrscheinlichkeit eine Signatur erstellen kann. Auf der anderen Seite ist es schwierig eine Kollision zu finden, falls nur wenige Bälle zur Verfügung stehen, wie das für einen Angreifer der Fall ist. Da P_s mit der Anzahl der Bälle zunimmt, darf nur eine kleine Menge von SEALs freigegeben werden, um die Chancen für das erfolgreiche Fälschen einer Signatur nicht zu gross werden zu lassen.

Da die Anzahl dem Signierer bekannte SEALs t direkten Einfluss auf die Public-Key-Grösse hat, legt Perrig [Per01] die Wahrscheinlichkeit für den Signierer eine Signatur zu finden auf 50% fest und passt für gegebene k und t die Anzahl Körbe n an. Um sicherzustellen, dass jede Nachricht signiert werden kann, wird der Hash der Nachricht neu mit $h = H(m||c)$ berechnet. Dabei ist c ein Zählwert und $m||c$ steht für die Konkatenation von m mit c. Ist es für ein h nicht möglich eine Signatur zu erstellen, so wird der Zählerwert um eins erhöht und die Signierung erneut versucht. Für $P_s = 0.5$ sind im Mittel zwei Versuche nötig um eine Signatur zu erstellen. Diese besteht neu aus den verwendeten SEALs sowie dem Zählerwert c.

Sei P_f die Wahrscheinlichkeit für einen Angreifer in einem Versuch eine gültige Signatur für einen gefälschten Datenblock zu finden. Für den Fall, dass dem Angreifer r SEALs zur Verfügung stehen, leitet Perrig [Per01] eine obere Grenze für P_f her:

$$P_f \leq \frac{\binom{r}{k}(n-1)^{r-k}}{n^{r-1}}$$

Für kleine r ist dies eine enge obere Grenze und kann daher gut verwendet werden, um die Sicherheit der BiBa Signatur abzuschätzen. Sei z.B. $n = 222$, $k = 12$ und $r = k$, so ist die Wahrscheinlichkeit für einen erfolgreichen Angriff mit einem Versuch $P_f \leq 2^{-85.73}$. Wird k erhöht und n durch eine gleichbleibende Erfolgswahrscheinlichkeit für den Signierer entsprechend angepasst, so nimm P_f weiter ab und die Sicherheit damit zu.

Ein Gegner kann SEALs aus zwei Quellen erhalten. Zum einen kann er SEALs sammeln, die mit Signaturen freigegeben werden und zum anderen kann er versuchen mittels Brute-Force gültige SEALs für die Commitments $F_{S_i}(0)$ zu finden. Die Anzahl freigegebenen SEALs kann der Signierer direkt bestimmen. Die Sicherheit der Commitments kann durch die Wahl der Funktion F und die Grösse der SEALs b_S beeinflusst werden. Die Funktion F darf keine Möglichkeiten bieten mit geringerem Aufwand als mit Raten für ein zufälliges Commitment $F_{S_i}(0)$ einen SEAL S_j zu finden, für den gilt $F_{S_j}(0) = F_{S_i}(0)$. Sie muss also *Preimage* resistent sein [MVO97]. b_S muss genügend gross sein, damit der Gegner keine gültigen SEALs erraten und nicht für jeden möglichen SEAL die Commitments im Voraus berechnen kann.

Da für die Verifikation einer Signatur nur $2k+1$ Hash-Funktionen berechnet werden müssen, ist das Biba Signaturschema für die Empfänger sehr effizient. Für eine Signatur werden jedoch $(t+1)/P_s$ Auswertungen benötigt. Mit den Zeiten zur Berechnung von Hash-Operationen und herkömmlichen Signaturen aus den Tabellen A.2 und A.1 im Anhang lässt sich die Effizienz

der BiBa Signatur abschätzen. Für den Fall $k = 12$, $t = 1024$ und die Verwendung von MD5 ergibt sich ein Zeitaufwand von 52 μs für eine Verifikation, was 21 mal schneller ist als die Verifikation einer RSA Signatur bei einer Schlüsselgrösse von 1024 bit. Das Erstellen einer BiBa Signatur dürfte hingegen 2.9 ms benötigen, was noch sechs mal schneller ist als das Signieren mit RSA. Wählen wir die Grösse der SEALs willkürlich als $b_S = 64$ bit, so ergibt sich eine Signaturgrösse von 96 B, was geringfügig weniger ist, als die Signaturgrösse bei RSA von 128 B. Der Public-Key wäre in diesem Setting bei BiBa 8 KB gross.

2.2.1. Powerball

Mitzenmacher und Perrig ([MP02]) schlagen ein neues und auf BiBa basierendes Signaturschema namens *Powerball* vor, welches kleinere Signaturgrössen bei gleicher Sicherheit erlaubt. In diesem Abschnitt wird Powerball kurz erläutert.

Bei BiBa wird eine Signatur durch k Bälle verkörpert, welche alle auf den gleichen Korb abgebildet wurden. Bei Powerball befinden sich k' Bälle einer Signatur nicht im selben Korb, sondern in Körben, welche einem bestimmtem Muster entsprechen. Ein Signierer verfügt über t solcher Muster M_i mit $1 \leq i \leq t$. Jedes Muster legt k' Körbe fest: $M_i = \langle b_1, \ldots, b_{k'} \rangle$. Zum Erstellen einer Signatur werden wie bei BiBa die t Bälle mit der von $h = H(m||c)$ selektierten Funktion G_h in die n Körbe verteilt. Als zweiter Schritt muss ein *vollständiges Muster* gesucht werden, dessen Körbe mindestens einen Ball enthalten. Falls ein Korb in einem Muster β mal vorkommt, so muss er mindestens β Bälle enthalten. Wird ein vollständiges Muster gefunden, so bilden die k' Bälle aus den entsprechenden Körben zusammen mit dem Muster und dem Zähler c die Signatur. Falls kein vollständiges Muster gefunden wurde, wird c um eins erhöht und die Signierung erneut versucht.

Zur Verifikation einer Signatur wird wie bisher geprüft, ob die Bälle paarweise voneinander verschieden und authentisch sind und ob sie durch die Funktion G_h auf die durch das Muster spezifizierten Körbe abgebildet werden. Zudem muss die Authentizität des Musters M_i durch die Existenz von dessen Commitment $F_{M_i}(0)$ im Public-Key bestätigt werden.

Um den Public-Key durch die Commitments der t Muster nicht zu verdoppeln, benutzen Mitzenmacher und Perrig [MP02] die Bälle als Commitments für die Muster: $S_i = F_{M_i}(0)$. Wegen dieser Doppelfunktionalität nennen sie die Bälle daher Powerballs. Das Commitment eines Balles authentisiert sowohl einen Ball als auch ein Muster: $F_{S_i}(0) = F_{F_{M_i}(0)}(0)$. Durch das Muster M_i einer Signatur erhält ein Gegner einen zusätzlichen Ball. Um seine Erfolgschancen nicht unnötig zu erhöhen, darf dieser Ball zum Erstellen einer Signatur nicht verwendet werden.

Da ein Muster auch einen SEAL darstellt, entspricht bei Powerball in einer Signatur die Anzahl SEALs nicht der Anzahl Bällen. Die Anzahl Bälle wird mit k' und die Anzahl SEALs mit $k = k' + 1$ bezeichnet. Bei BiBa und dem später vorgestellten HORS gilt $k = k'$ und es wird zwischen SEALs und Bällen nicht unterschieden.

k'	n	k'	n
5	3862	11	1407
6	2905	12	1295
7	2347	13	1204
8	1986	14	1125
9	1734	15	1061
10	1548	16	1007

Tabelle 2.1.: Anzahl Körbe n bei k' Bällen pro Signatur für Powerball mit $P_s = 0.5$. Werte für $k' \in [9,13]$ von [MP02], die restlichen Werte durch Simulation erhalten.

Wenn ein Angreifer nur ein Muster und k' Bälle kennt, hat er $k'!$ Möglichkeiten diese Bälle in die Körbe des Musters zu verteilen, falls die Körbe voneinander verschieden sind. Die Wahrscheinlichkeit eine Signatur in einem Versuch fälschen zu können beträgt demnach:

$$P_f \leq \frac{k'!}{n^k}$$

Falls t SEALs für mehr als eine Signatur verwendet wird und ein Angreifer gültigen Bälle und Muster von γ Signaturen kennt, kann seine Erfolgschance abgeschätzt werden durch:

$$P_f \leq \frac{\gamma \cdot \binom{(k'+1)\cdot\gamma-1}{k'} \cdot k'!}{n^{k'}}$$

Tabelle 2.1 bestimmt die Anzahl Körbe n für eine gewisse Anzahl Bälle pro Signatur k', so dass ein Signierer im ersten Versuch mit einer Wahrscheinlichkeit von $P_s = 0.5$ eine Signatur erstellen kann.

Obwohl Powerball praktisch den selben Rechenaufwand benötigt wie BiBa, kann mit der gleichen Signaturgrösse ein höheres Sicherheitsniveau erreicht werden.

2.2.2. HORS

Das BiBa oder Powerball Signaturschema ist zwar für den Empfänger sehr effizient, der Sender muss jedoch $(t+1)/P_s$ Hash-Funktionen auswerten. Reyzin und Reyzin [RR02] beschreiben ein Signaturschema welches sie *HORS* nennen und welches das Erstellen einer Signatur stark vereinfacht. Dieses Signaturschema wird in diesem Abschnitt vorgestellt.

Wie bei BiBa bilden t SEALs $S_1, S_2, \ldots, S_t$ den Private-Key und deren Commitments den Public-Key. Sei H wieder eine Hash-Funktion aus dem Random-Oracle-Modell. Um eine Nachricht m zu signieren wird wieder zuerst deren Hash $h = H(m)$ berechnet. Dieser wird in k je $\log t$ bit lange Substrings $i_1, \ldots, i_k$ zerschnitten, welche als positive ganze Zahlen interpretiert werden und als Indizes für die signaturbildenden SEALs $S_{i_1}, \ldots, S_{i_k}$ dienen. Mit Hilfe der Hash-Funktion wird also eine zufällige Untermenge aller SEALs ausgewählt und

als Signatur verwendet (HORS: „**H**ash to **O**btain **R**andom **S**ubset“). Bei der Verifikation wird genau gleich verfahren, nur muss geprüft werden, ob $F_{S_j}(0)$ für $1 \leq j \leq k$ dem Commitment mit Index i_j im Public-Key entspricht.

Die Sicherheit von HORS beruht auf der für einen Angreifer mit r SEALs geringen Wahrscheinlichkeit, durch Ziehen mit Zurücklegen in k Versuchen immer eines von diesen r SEALs zu erwischen, wobei sich t SEALs in der Urne befinden. Die Wahrscheinlichkeit eine Signatur fälschen zu können beträgt demnach:

$$P_f = \left(\frac{r}{t}\right)^k$$

Vergleichen wir HORS mit BiBa, so stellen wir fest, dass die Signierung deutlich und die Verifikation leicht effizienter wurde. Anstelle von $(t+1)/P_s$ ist nun zum Erstellen einer Signatur nur noch eine einzige Hash-Funktion zu berechnen. Insbesondere ist es mit HORS immer möglich auf den ersten Versuch eine Signatur zu erstellen, da man nicht darauf angewiesen ist, dass eine Kollision auftritt. Mit BiBa ist die Signierung nur mit Wahrscheinlichkeit P_s erfolgreich. Bei der Verifikation reduziert sich der Aufwand von $2k+1$ auf $k+1$ Berechnungen. Wie später gezeigt wird, ist die Sicherheit von HORS mit der von BiBa vergleichbar. HORS ist also ein Signaturschema, welches Ähnlichkeiten mit BiBa hat, jedoch deutlich effizienter ist und trotzdem ein ähnliches Sicherheitsniveau bietet.

2.3. BiBa Broadcast-Authentifizierungsprotokoll

Die Sicherheit der vorgestellten Signaturschemata beruht darauf, dass dem Signierer viel mehr SEALs zur Verfügung stehen als einem Angreifer. Aus diesem Grund dürfen nicht zu viele SEALs freigegeben werden, was jedoch für die fortlaufende Erstellung von Signaturen nötig ist. Ein Public/Private-Schlüsselpaar kann nur einmal verwendet werden, um r/k Signaturen zu erstellen (*One-Time-Signaturen*). Um diese Limitierung zu beheben, könnte der gesamte Public-Key ständig erneuert werden, was aber wegen dessen Grösse nicht praktikabel ist. Eine weitere Möglichkeit wäre das Übertragen von einem neuen Commitment pro verwendetem SEAL. So könnte der Sender unbegrenzt Signaturen erstellen, es müssten aber pro Signatur doppelt so viele Daten übertragen werden. Perrig [Per01] stellt zur Lösung des Problems das BiBa Broadcast-Authentifizierungsprotokoll sowie zwei Erweiterungen dazu vor, welche in diesem Abschnitt zusammengefasst werden.

Das Prinzip des BiBa Broadcast-Authentifizierungsprotokolls basiert auf *Hash-Ketten*. Die Funktion F wird verwendet, um mehrere *SEAL-Ketten* zu erstellen, welche von einer mit einer Pseudo-Zufallsfunktion F' erzeugten *Salt-Kette* abhängig sind. In diesem Absatz wird deren Erzeugung und danach deren Verwendung beschrieben. Sei l die Länge der Salt-Kette, ein Salt K_j eine Zahl mit b_K Bit Länge und $1 \leq j \leq l$ und die PRF $F' : \{0,1\}^{b_K} \times \{0,1\}^{b_K} \rightarrow \{0,1\}^{b_K}$. Der Sender wählt zuerst das Ende K_l der Salt-Kette zufällig aus und berechnet die weiteren Salts rekursiv: $K_j = F'_{K_{j+1}}(0)$ mit $1 \leq j < l$. Es ist also einfach der Kette in absteigender

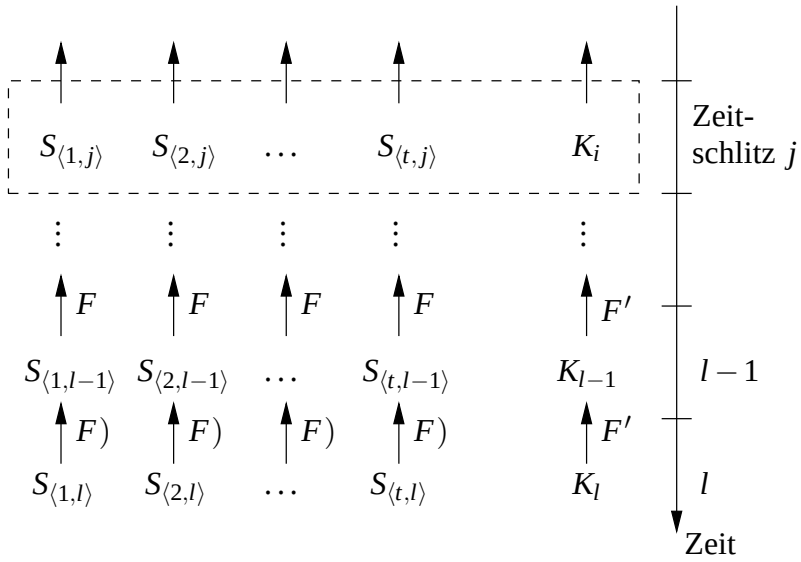

Abbildung 2.2.: Hash-Ketten im BiBa Broadcast-Authentifizierungsprotokoll. Im Zeitschlitz j sind die SEALs $S_{\langle i,j\rangle}$ und das Salt K_j aktiv. Den Ketten kann in Pfeilrichtung gefolgt werden. Die Graphik wurde mit Genehmigung von [Per01] kopiert und geringfügig angepasst.

Richtung zu folgen, in die aufsteigende Richtung ist wegen der Einwegeigenschaft von F' nicht möglich. Als zweites werden t SEAL-Ketten erzeugt. Sei $S_{\langle i,j\rangle}$ ein SEAL in der SEAL-Kette i auf Position j mit $1 \leq i \leq t$ und $1 \leq j \leq l$. Es werden wieder zuerst die Enden der SEAL-Ketten $S_{\langle i,l\rangle}$ zufällig gewählt und die restlichen SEALs rekursiv berechnet. Diese hängen nun aber vom vorhergehenden SEAL in der Kette als auch vom Salt vorherigen Stufe ab: $S_{\langle i,j\rangle} = F_{S_{\langle i,j+1\rangle}}(K_{j+1})$ für $1 \leq j < l$. Die Abhängigkeit vom Salt erhöht die Komplexität zur Vorausberechnung sämtlicher SEALs, wodurch deren Grösse reduziert werden kann.

Für die Anwendung des BiBa Broadcast-Authentisierungsprotokolls unterteilt der Sender die Zeit in Zeitschlitze gleicher Dauer. In einem Zeitschlitz j sind jeweils die SEALs $S_{\langle i,j\rangle}$ sowie das Salt K_j aktiv, was in Abbildung 2.2 veranschaulicht wird. SEALs werden wie zuvor nur veröffentlicht, wenn sie Teil einer Signatur sind. Zu Beginn des nächsten Zeitschlitzes publiziert der Sender das neue Salt K_{j+1} und die alten SEALs werden nicht mehr verwendet.

Der Empfänger verifiziert zu Beginn des Zeitschlitzes j, ob das neue Salt authentisch ist, d.h. ob gilt $K_{j-1} = F'_{K_j}(0)$. SEALs können authentifiziert werden, indem deren SEAL-Ketten mit $F_{S_{\langle i,j'\rangle}}(K_{j'}) = S_{\langle i,j'-1\rangle}$ gefolgt wird, bis ein authentischer SEAL gefunden wurde. Da in einem Zeitschlitz nicht alle SEALs verwendet werden, muss ein SEAL im Mittel über t/r Zeitschlitze zurückverfolgt werden. Ein neuer Empfänger erhält über einen authentischen Kanal alle SEALs des vorherigen Zeitschlitzes sowie das aktuelle Salt.

Ein SEAL $S_{\langle i,j\rangle}$ eines Zeitschlitzes j ist nur gültig solange sein Zeitschlitz j aktiv ist. Dies erfordert, dass Sender und Empfänger zeitlich synchronisiert und über den Paketzeitplan informiert sind. Falls ein Angreifer Pakete auf dem Weg zum Empfänger verzögern könnte,

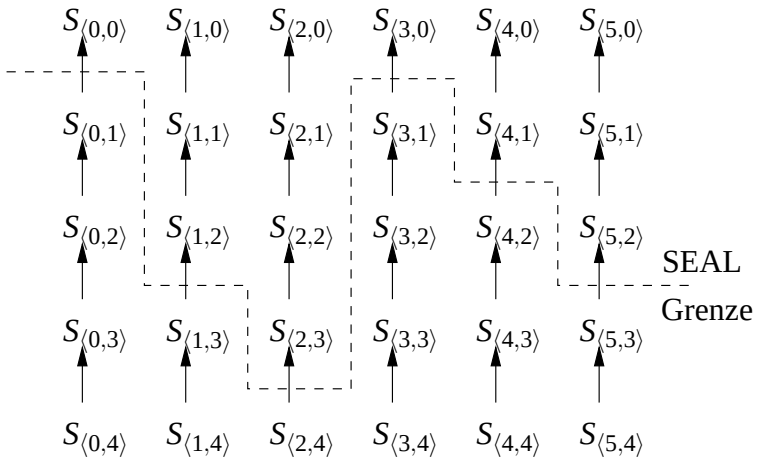

Abbildung 2.3.: SEAL-Grenze in den Erweiterungen zum BiBa Broadcast-Authentifizierungsprotokoll. Oberhalb der Grenze befinden sich die Commitments der aktiven SEALs. Die Graphik wurde mit Genehmigung von [Per01] kopiert und geringfügig angepasst.

wäre es ihm möglich Signaturen zu sammeln, bis er eine genügende Anzahl von SEALs zum Erstellen einer gefälschten Signatur zur Verfügung hat. Da er auch fortlaufend neue SEALs vom Sender erhält, könnte er sogar kontinuierlich Pakete fälschen. Die Authentizität von Signaturen, welche zu spät bei einem Empfänger eintriffen, kann demnach nicht gewährleistet werden.

Ein Empfänger muss sicherstellen können, dass ein Gegner maximal r SEALs eines aktiven Zeitschlitzes zur Verfügung hat. Angenommen die zeitliche Synchronisation sei mit einem Fehler von δ behaftet und die Zeit für den Transport sei vernachlässigbar, dann ist dies erfüllt, falls der Sender in der Zeit δ maximal $\lfloor r/k \rfloor$ Nachrichten signiert. Falls die Framerate f grösser ist, so kann der Sender mehrere Signaturinstanzen abwechselnd benutzen, um trotzdem die erforderliche Anzahl Signaturen erstellen zu können.

Da ein Grossteil von SEALs nie verwendet wird, ist der Verifikationsaufwand durch das Zurückverfolgen der SEAL-Ketten gross. Dies könnte durch das Veröffentlichen sämtlicher inaktiv gewordenen SEALs vermieden werden, der Datenoverhead wäre dabei jedoch gross. Perrig [Per01] löst das Problem in den Erweiterungen A und B, welche sämtliche SEALs verwenden. Diese Erweiterungen werden nachfolgend vorgestellt.

2.3.1. Erweiterung A

In beiden Erweiterungen wird auf die Salt-Kette und das Konzept der Zeitschlitze verzichtet. Die Konstruktion von SEAL-Ketten und die Authentifizierung von SEALs erfolgt durch $S_{\langle i,j-1\rangle} = F_{S_{\langle i,j\rangle}}(0)$. Der Sender verwendet zum Signieren jeweils SEALs, welche in den SEAL-Ketten direkt auf bereits verwendete SEALs folgen. Zwischen bereits verwendeten und noch

nicht verwendeten SEALs verläuft also eine Grenze, welche ständig verschoben wird. Ein bereits verwendeter SEAL direkt oberhalb der Grenze dient als Commitment eines neuen SEALs direkt unterhalb der Grenze. Die *SEAL-Grenze* ist in Abbildung 2.3 dargestellt. Da in Erweiterung A im Gegensatz zur Basisversion vorausgesetzt wird, dass keine Paketverluste auftreten, ist es auch den Empfängern möglich die Grenze zu verfolgen. Würden Paketverluste auftreten, so könnten die Empfänger die Grenze für die verloren gegangenen SEALs nicht aktualisieren und die Authentizität zukünftiger SEALs könnte nicht mehr garantiert werden.

2.3.2. Erweiterung B

Die Erweiterung B ist der Erweiterung A ähnlich, erlaubt jedoch Paketverluste. Dies wird erreicht, indem mit der Signatur Information über die aktuelle SEAL-Grenze übermittelt wird. Dadurch können alle SEALs authentifiziert werden, auch wenn Paketverluste auftreten. In diesem Fall müssen die SEALs einer SEAL-Kette so weit zurückverfolgt werden, wie SEALs in dieser Kette hintereinander verloren gingen.

Mit einer absoluten Codierung wird die Position sämtlicher SEALs direkt oberhalb der SEAL-Grenze in jedem Paket mitgeschickt. Es ist klar, dass dadurch deutlich mehr Daten übertragen werden müssen. Mit einer relativen Codierung werden nur die Veränderungen der SEAL-Grenze relativ zu einer dem Sender und Empfänger bekannten Grenze kommuniziert, wodurch sich der zusätzliche Datenaufwand reduzieren lässt. Dies könnten z.B. die Indizes der in den letzten Signaturen verwendeten SEAL-Ketten sein. Auf jeden Fall muss ein Empfänger mindestens ein Paket alle $\lfloor r/k \rfloor$ Pakete erhalten. Ansonsten ist es möglich, dass ein Angreifer während einer Folge von Paketverlusten mehr als r SEALs sammeln kann, was eine Verletzung der Sicherheit darstellt.

3. Design

3.1. Anforderungen

Die Anforderungen an ein Livestreaming-Protokoll wie Pulsar umfassen:

- **Der vom Protokoll verursachte Datenoverhead in der Übertragung muss klein sein.** Die zur Verfügung stehende und möglicherweise knappe Bandbreite soll für Inhalte genutzt werden können und nicht für Daten, die dem Benutzer keinen direkten Mehrwert bringen.
- **Die Frames müssen ab deren Erzeugung schnell bei den Clients zur Anzeige bereit sein.** Bereits kleinere Verzögerungen können sich negativ bemerkbar machen. Einem Zuschauer, der mit Pulsar ein Penaltyschiessen der Fussball EM verfolgt, können seine Nachbarn den Spass schnell verderben, falls sie die Bilder im herkömmlichen Fernsehen einige Sekunden früher erhalten. Neben der Minimierung des Delays ist auch das Einhalten der Abspiel-Deadline zentral. Übermittelte Frames sind nutzlos, falls sie nach dem Abspielzeitpunkt bei den Clients eintreffen.
- **Rechenzeit und Speicher dürfen nur wenig in Anspruch genommen werden.** Dies ermöglicht auch leistungsschwachen Teilnehmern einen Stream zu verfolgen und leistungsstarken Teilnehmern die Ressourcen ihres Systems anderweitig zu nutzen oder den Energiebedarf der CPU tief zu halten.

Das Pulsar Protokoll erfüllt diese Anforderungen gut. Der Datenoverhead ist gering und die Abspiel-Deadline kann sehr gut eingehalten werden. Der Delay ist wegen des mehrsekundigen Empfangspuffers für das Fussballbeispiel bereits zu gross. Für die meisten Anwendungen dürfte er jedoch ausreichend klein sein. Locher et al. [LMSW07] emulieren mit der realen Codebasis des Pulsar Clients 100’000 Peers auf einem einzigen Computer mit 4 GB RAM. Mit Ressourcen wird also sehr sparsam umgegangen.

Es ist klar, dass die Einführung eines Integritätsprotokolls nicht ohne zusätzliche Kosten einhergeht. Diese müssen jedoch genügend klein sein, damit die Anforderungen an das Pulsar Protokoll weiterhin erfüllt werden können. An das Integritätsprotokoll stellen sich zudem spezifische Anforderungen:

- **Nur authentische Frames dürfen wiedergegeben werden.** Würde es z.B. einem Angreifer gelingen ein Key-Frame zu fälschen, so würde das die Wiedergabe vorübergehend stark stören.

- **Nur authentische Frames dürfen weitergeleitet werden.** Dies ist insbesondere in der Push-Phase von grosser Bedeutung, da dort jedes Frame möglichst viele Teilnehmer erreichen soll. Von einem Angreifer gefälschte Frames würden sonst im schlimmsten Fall im ganzen Netzwerk verteilt und würden dieses unnötig belasten. Es wäre dann weniger Bandbreite für die Übermittlung von legitimen Frames vorhanden. In der Pull-Phase sind die Auswirkungen kleiner, da dort ein gefälschtes Frame weniger Peers erreicht.
- **Ankommende Frames müssen sofort authentifiziert werden können.** In der Push-Phase ist es wichtig ankommende Frames sofort weiterzuschicken, da sich eine Verzögerung auf eine grosse Anzahl Clients auswirkt und sich über die Anzahl Hops aufaddiert.
- **Die Verfügbarkeit darf nicht beeinträchtigt werden.** Das Ziel der Senderauthentifizierung ist die Verfügbarkeit der Daten zu steigern. Ein Angreifer könnte jedoch versuchen, Teilnehmer des Netzwerkes so zu manipulieren, dass sie authentische Frames nicht als solche erkennen. Diese und weitere Angriffsmöglichkeiten auf die Verfügbarkeit des Dienstes oder Teilen davon müssen vermieden werden.

3.2. Angreifer

Die Anforderungen verlangen, dass nur authentische Frames wiedergegeben oder weitergeleitet werden dürfen. Ein Angreifer hat jedoch immer eine verschwindend kleine Chance durch Zufall eine gültige Signatur zu finden. Die Anforderung muss daher soweit aufgeweicht werden, dass es einem starken Angreifer nur mit einer sehr kleinen Wahrscheinlichkeit möglich sein darf ein Frame zu fälschen. Diese Fälschungswahrscheinlichkeit lege ich auf eins zu einer Million fest. Mit einer Framerate von $f = 36\,\mathrm{s}^{-1}$ bedeutet das, dass ein starker Angreifer im Mittel in $7\frac{3}{4}\,\mathrm{h}$ ein Frame fälschen kann. Das Protokoll muss demnach so entworfen werden, dass es dadurch nicht allzufest gestört werden kann.

Zwischen dem Sicherheitsniveau und der Effizienz des Authentifizierungsprotokolls besteht ein Zielkonflikt. Ein hohes Sicherheitsniveau würde unter anderem bedeuten: grosse SEALs, viele SEALs pro Signatur, viele SEALs die dem Signierer und wenige SEALs die dem Gegner zur Verfügung stehen. Dies hat negative Einflüsse auf die Signaturgrösse, die Public-Key-Grösse und insbesondere bei BiBa und Powerball auf den Aufwand zum Erstellen einer Signatur. Es ist daher wichtig die Stärke eines Gegners abzuschätzen, um das Sicherheitsniveau vernünftig tief zu halten.

Um eine Signatur zu fälschen, kann ein Angreifer versuchen mit den SEALs, die er kennt, eine gültige Signatur zu finden. Für jeden Versuch muss er mindestens eine Hash-Funktion auswerten. Um seine Chancen zu vergrössern kann er zudem versuchen durch Raten oder Vorausberechnung weitere gültige SEALs zu finden. Auch dazu ist die Berechnung einer Hash-Funktion pro Versuch oder vorausberechnetes Commitment notwendig. Die Anzahl Hash-Funktionen, die ein Gegner pro Sekunde auswerten kann, gibt daher Auskunft über dessen Möglichkeiten.

Aufgrund der Messungen in Tabelle A.2 im Anhang kann ein heute halbwegs moderner Desktop-Computer schätzungsweise 10^6 Hash-Funktionen pro Sekunde auswerten. Ich gehe davon aus, dass ein starker Angreifer über eine äquivalente Leistung von höchstens 10^6 solcher Geräte verfügt. Möglichkeiten eine solche Rechenleistung zu erreichen umfassen Supercomputer, Cluster, dedizierte Hardware und Bot-Netze. Als Beispiele seien hier ein im Jahr 2005 ausgehobenes Bot-Netz mit $1.5 \cdot 10^6$ Computern[1] sowie das Desktop Grid Computing Projekt Boinc mit $2.6 \cdot 10^5$ aktiven Teilnehmern[2] erwähnt. Ein starker Angreifer benötigt also mindestens

$$T_e = 10^{-12}\,\mathrm{s}$$

pro berechnete Hash-Funktion. Diese Annahme dürfte die heutigen Möglichkeiten eines starken Gegners gut abschätzen. Künftige Entwicklungen müssen jedoch mitverfolgt und das Sicherheitsniveau nötigenfalls angepasst werden.

3.3. Authentifizierungsprotokoll

Grundsätzlich wird für das BiBa Broadcast-Authentifizierungsprotokoll gefordert, dass der Sender und die Empfänger bezüglich des Paketzeitplanes synchronisiert sind. Nur so kann garantiert werden, dass einem Angreifer maximal r aktive SEALs bekannt sind. Sei T_d die Zeit von der Emission einer Signatur bis diese den letzten Empfänger erreicht hat. Diese Sendezeit T_d ist im Pulsar Protokoll in der Grössenordnung von wenigen Sekunden und ist im Vergleich zum Fehler der zeitlichen Synchronisation dominant. Die Sicherheit kann gewährleistet werden, wenn in der Zeitspanne T_d maximal $\lfloor r/k \rfloor$ Signaturen ausgestellt werden, wobei k die Anzahl SEALs pro Signatur bezeichnet. Diese Limitierung erfordert die Verwendung von mehreren Signaturinstanzen, um trotzdem die nötige Anzahl Signaturen ausstellen zu können. Bezeichnet n_i die Anzahl Instanzen, so sind

$$n_i = \left\lceil T_d \cdot f \cdot \frac{k}{r} \right\rceil$$

Instanzen nötig. Jede Instanz braucht einen eigenen Public-Key, wodurch deren Anzahl die Grösse des gesamten Public-Keys proportional beeinflusst. Die Anzahl Instanzen ist daher möglichst klein zu wählen.

3.3.1. Basisversion

Die Basisversion des Authentifizierungsprotokolls hat gegenüber dessen Erweiterungen den Vorteil, dass Paketverluste kein Problem darstellen und dass die SEALs etwas kleiner sein können.

[1] `http://www.techweb.com/wire/security/172303160`
[2] `http://www.boincstats.com/stats/project_graph.php?pr=bo`

Der Sender benutzt hierbei für die Signaturen eines Zeitschlitzes j sämtliche t SEALs $S_{\langle i,j \rangle}$ sowie das Salt K_j. Veröffentlicht werden jedoch lediglich die r SEALs, welche für die Signaturen nötig sind. Dies hat zur Folge, dass den Empfängern im Mittel t/r SEALs in jeder SEAL-Kette fehlen. Für die Authentifizierung sämtlicher SEALs einer Signatur sind daher durchschnittlich $t \cdot k/r$ Hash-Funktionsauswertungen nötig. Des Weiteren müssen die Ketten im Vergleich zu den Erweiterungen A und B bei gleicher Lebensdauer des Public-Keys bedeutend länger sein. Die Erzeugung und Speicherung ist entsprechend aufwändiger. Um Speicher zu sparen wäre es möglich anstelle der ganzen Ketten nur Zwischenwerte abzuspeichern. Vor der Verwendung müssten die SEALs dann von den Zwischenwerten abgeleitet werden, wodurch sich der Generierungsaufwand indirekt verdoppeln würde.

Powerball hat in Kombination mit der Basisversion des Authentifizierungsprotokolls weitere Nachteile. Die Idee von Powerball ist, dass ein Ball das Commitment eines Musters bildet. Dies bedeutet, dass für das Muster eine ganze Stufe von SEALs verwendet werden muss. Die Länge der Ketten müsste doppelt so gross sein und damit auch die oben erwähnten senderseitigen Probleme. Alternativ dazu könnte das Powerball-Prinzip aufgegeben und für Bälle und Muster verschiedene Ketten verwendet werden, wobei für die Ketten der Muster die Erweiterung A oder B verwendet würde. Der Nachteil dieser Variante wäre die Verdoppelung des ohnehin schon grossen Public-Keys sowie die Aufgabe der Resistenz gegen Paketverluste.

3.3.2. Erweiterung A

Die Erweiterung A des Authentifizierungsprotokolls behebt die Nachteile der Basisversion, benötigt aber etwas grössere SEALs und beruht auf der Annahme, dass keine Signaturverluste auftreten. Es ist den Empfängern also möglich über sämtliche bereits verwendete SEALs Bescheid zu wissen, d.h. die SEAL-Grenze mitverfolgen zu können.

Falls Paketverluste auftreten würden, könnte die SEAL-Grenze nicht aktualisiert werden. Die verloren gegangenen SEALS könnten von einem Gegner zur Erzeugung einer gefälschten Signatur miteinbezogen werden, was die Sicherheit schwächen würde. Verloren gegangene SEALs müssen daher für ungültig erklärt werden, was erreicht werden kann, wenn nach einem erkannten Signaturverlust die gesamte SEAL-Grenze aktualisiert wird.

Das Pulsar Protokoll ist äusserst zuverlässig, wodurch die Erweiterung A eingesetzt werden kann. Aufgrund des grossen Public-Keys dürfen Paketverluste aber nur äusserst selten bis gar nicht auftreten. Es ist daher wünschenswert, dass eine geringe Anzahl Paketverluste toleriert werden kann. Dafür kann die Erweiterung B des BiBa Broadcast-Authentifizierungsprotokolls beigezogen werden. Mit den später vorgestellten Index-Updates existiert zudem noch eine weitere Möglichkeit die SEAL-Grenze nach einem Verlust korrekt zu aktualisieren.

3.3.3. Erweiterung B

Mit der Erweiterung B ist es möglich eine gewisse Robustheit gegen Paketverluste zu erreichen. Um n_l aufeinanderfolgende Paketverluste zu erlauben, können mit jeder Signatur die Indizes der in den letzten n_l Signaturen verwendeten SEAL-Ketten mitgeschickt werden. Falls also n_l Signaturen verloren gehen, kann mit der nächsten Signatur die SEAL-Grenze korrekt aktualisiert werden und die vom Angreifer gesammelten SEALs werden ungültig. Diese Robustheit hat den Preis von

$$b_l = n_l \cdot k \cdot \log t$$

zusätzlichen Bits pro Signatur. Des Weiteren muss ein Angreifer

$$r \geq (n_l + 1) \cdot k$$

SEALs kennen dürfen. Ansonsten ist es ihm eher möglich mit den $n_l \cdot k$ dem Empfänger unbekannten SEALs und den darauffolgenden k dem Empfänger noch nicht bekannten SEALs eine Signatur zu fälschen. Wegen der Kosten ist also keine beliebige Robustheit gegen Paketverluste erreichbar.

3.4. SEALs

Die Sicherheit der im Theorieteil vorgestellten Signaturschemata basiert auf der Annahme, dass ein Gegner nur wenige aktive SEALs kennt. Dieser kann versuchen, zu den maximal r mit Signaturen veröffentlichter SEALs, weitere zu finden. Das festgelegte Sicherheitslevel verlangt, dass es einem starken Angreifer mit einer nicht grösseren Wahrscheinlichkeit als 10^{-6} möglich sein darf eine Signatur zu fälschen. Das Finden eines zusätzlichen aktiven SEALs erhöht zwar die Wahrscheinlichkeit eine Signatur fälschen zu können, ist jedoch unproblematisch, falls es nur selten möglich ist und und trotz der neu gefundenen SEALs die Sicherheit nicht zu stark geschwächt wird. Seien die Ereignisse

$$\begin{aligned} A &:= \text{Gegner kann eine Signatur fälschen} \\ B = i &:= \text{Gegner kennt } i \text{ zusätzliche aktive SEALs} \end{aligned}$$

dann muss gelten

$$\begin{aligned} P[A] &= \sum_{i=0}^{t} P[A \cap B = i] \\ &= \sum_{i=0}^{t} P[A \mid B = i] \cdot P[B = i] \\ &< 10^{-6} \end{aligned}$$

Die Wahrscheinlichkeit $P[B = i]$ ist abhängig von der Länge der SEALs und der Dauer, während der ein Gegner SEALs suchen kann. $P[A \mid B = i]$ ist abhängig vom verwendeten Signaturschema und der Gültigkeitsdauer der veröffentlichten SEALs.

Da die Funktion F nicht direkt invertiert werden kann, muss ein Angreifer um unveröffentlichte aktive SEALs zu finden für beliebige SEALs deren Commitment vorausberechnen. Die Anzahl SEALs, die ein Gegner in der Zeit T_c vorausberechnen kann ist T_c/T_e. Insgesamt gibt es 2^{b_S} verschiedene SEALs und t davon sind jeweils aktiv und werden von den Empfängern als authentisch akzeptiert. Es wird hierbei davon ausgegangen, dass die Implementation von F kollisionsresistent[3] ist. Um herauszufinden wie wahrscheinlich es ist, dass ein Gegner nach T_d eine gewisse Anzahl SEALs kennt, kann die hypergeometrische Wahrscheinlichkeitsverteilung herangezogen werden. Aufgrund der geringen Erfolgswahrscheinlichkeit, wird sie in den nachfolgenden Berechnungen durch die Poisson-Verteilung approximiert.

3.4.1. Basisvariante

Die Basisvariante des Authentifizierungsprotokolls schützt die SEALs eines Zeitschlitzes mit einem Salt. Dieses muss bei der Verifikation eines SEALs in die Berechnung miteinbezogen werden. Da das Salt am Ende eines Zeitschlitzes wechselt, verlieren alle vorausberechneten SEALs ihre Gültigkeit. Einem Angreifer steht also nur die Dauer eines Zeitschlitzes zur Verfügung, um SEALs für ein bestimmtes Salt vorauszuberechnen. Falls nur eine Signatur pro Zeitschlitz ausgestellt wird entspricht diese Dauer der Sendezeit T_d.

Die Wahrscheinlichkeit, dass ein Gegner nach $T_c = T_d = 1\,\mathrm{s}$ Vorausberechnungszeit eine gewisse Anzahl aktiver SEALs finden kann, ist für die SEAL-Längen von 48 und 56 bit in Tabelle 3.1(a) dargestellt. Mit den etwas kürzeren SEALs ist es bedeutend wahrscheinlicher, dass ein Gegner aktive SEALs findet. Dies würde erfordern, dass die Signaturen besser geschützt werden müssten. Wären anstelle von 6 dafür 7 SEALs pro Signatur nötig, so ginge der Vorteil der kürzeren SEALs bereits wieder verloren. Man könnte argumentieren, dass ein Angreifer nicht eine volle Sekunde rechnen wird, da ihm dann keine Zeit zum Finden einer Signatur mehr bleibt. Berechnet er aber nur halb so lange, ist es nicht viel unwahrscheinlicher, dass er einige SEALs finden kann. Für die Basisversion sind daher SEALs von $b_S = 56\,\mathrm{bit}$ Länge nötig.

Anstatt SEALs für ein bestimmtes Salt vorauszuberechnen, kann ein Angreifer auch SEALs für zufällige Salts vorausberechnen. Darin ist er zeitlich nicht beschränkt. Um die Komplexität eines solchen Angriffs genügend hoch zu halten, müssen die Salts ausreichend lang sein. Tabelle 3.1(c) zeigt die Wahrscheinlichkeiten, dass nach 50 Jahren Vorausberechnung eine gewisse Anzahl aktiver SEALs bekannt sind. Sind ein SEAL und ein Salt zusammen 88 bit lang, sind die Wahrscheinlichkeiten etwas kleiner, als wenn die SEALs von 56 bit Länge für ein bestimmtes Salt angegriffen werden, was keine weiteren Sicherheitsmassnahmen erfordert. Die Salts müssen also 32 bit lang sein.

[3] Eine Funktion $f : X \to Y$ ist kollisionsresistent, falls es „rechnerisch unmöglich“ ist zwei voneinander verschiedene $x, x' \in X$ zu finden, für die gilt $f(x) = f(x')$. [MVO97]

$T_c = 1\,\mathrm{s}$	$b_S = 56\,\mathrm{bit}$	$b_S = 48\,\mathrm{bit}$
$P[B=0]$	0.9859	0.0263
$P[B=1]$	0.0140	0.0957
$P[B=2]$	$9.9549 \cdot 10^{-5}$	0.1741
$P[B=3]$	$4.7156 \cdot 10^{-7}$	0.2111
$P[B \geq 4]$	$1.6801 \cdot 10^{-9}$	0.4928

(a) Vorausberechnung SEALs Basisversion

$T_c = 1\,\mathrm{h}$	$b_S = 72\,\mathrm{bit}$	$b_S = 64\,\mathrm{bit}$
$P[B=0]$	0.9992	0.8189
$P[B=1]$	$7.8002 \cdot 10^{-4}$	0.1636
$P[B=2]$	$3.0445 \cdot 10^{-7}$	0.0164
$P[B=3]$	$7.9221 \cdot 10^{-11}$	0.0011
$P[B \geq 4]$	$1.5422 \cdot 10^{-14}$	$5.6666 \cdot 10^{-5}$

(b) Vorausberechnung SEALs Erweiterungen A/B

$T_c = 50\,\mathrm{a}$	$b_S + b_K = 96\,\mathrm{bit}$	$b_S + b_K = 88\,\mathrm{bit}$	$b_S = 80\,\mathrm{bit}$
$P[B=0]$	1.0000	0.9948	0.2630
$P[B=1]$	$2.0379 \cdot 10^{-5}$	0.0052	0.3513
$P[B=2]$	$2.0766 \cdot 10^{-10}$	$1.3539 \cdot 10^{-5}$	0.2346
$P[B=3]$	$1.4107 \cdot 10^{-15}$	$2.3545 \cdot 10^{-8}$	0.1044
$P[B \geq 4]$	$5.3901 \cdot 10^{-17}$	$3.0741 \cdot 10^{-11}$	0.0467

(c) Vorausberechnung SEALs unabhängig von Salts

Tabelle 3.1.: Wahrscheinlichkeit nach T_c Vorausberechnungszeit bei verschiedenen Bitlängen $B = i$ aktive SEALs zu kennen.

3.4.2. Erweiterungen A/B

Perrig [Per01] hat die SEAL-Ketten in den Erweiterungen A und B des Authentifizierungsprotokolles nicht mit Salts geschützt. Um die SEAL-Längen auch hier zu reduzieren, werden zuerst Salts eingeführt.

Sei $K_{j'}$ ein Salt mit Bitlänge b_K welches zur Zeit t mit $T_K \cdot j' \leq t < T_K \cdot (j'+1)$ gültig ist. T_K bezeichnet dabei die Dauer der Gültigkeit. Für zwei zum Zeitpunkt t gültigen SEALs $S_{\langle i,j \rangle}$ und $S_{\langle i,j+1 \rangle}$ einer Kette i auf Positionen j und $j+1$ gelte $S_{\langle i,j \rangle} = F_{S_{\langle i,j+1 \rangle}}(K_{j'})$. Die SEALs sind also vom Salt K_j abhängig. K_j wird kurz vor Beginn seiner Gültigkeit auf eine authentische Art veröffentlicht.

Da die SEAL-Grenze nur um tatsächlich verwendete SEALs wachsen soll, ist im Vergleich zur Basisversion des BiBa Broadcast-Authentifizierungsprotokolls der Zustand der SEAL-Ketten beim Wechsel des Salts nicht im Voraus bekannt. Es gibt zwei Möglichkeiten dieses Problem zu adressieren. Zum einen kann beim Wechsel des Salts der Public-Key erneuert werden, was zusätzlichen Datenoverhead in der Übertragung verursacht. Zum anderen kann der für T_K gültige Abschnitt der SEAL-Ketten so gross gewählt werden, dass er für diesen Zeitraum sicher ausreichend SEALs zur Verfügung stellt. Für das letzte SEAL einer Periode j' auf Position j in der Kette gelte in diesem Fall $S_{\langle i,j \rangle} = F_{S_{\langle i,j+1 \rangle}}(K_{j'+1})$. Dieser zweite Ansatz führt jedoch wieder SEALs ein, welche nicht verwendet werden. Der erste SEAL jeder Kette in der Periode $j'+1$ muss über mehrere Schritte bis zum letzten verwendeten SEAL in der Vorperiode j' zurückverfolgt werden. Dieser Ansatz hat Ähnlichkeiten mit der Basisversion, die Zurückverfolgung ist jedoch nur nach T_K nötig und nicht bei jeder Signatur. Auch die anderen im Zusammenhang mit der Basisversion diskutierten Nachteile, währen nicht so stark ausgeprägt. Je kürzer die Lebensdauer des Salts T_K ist, umso grösser sind die Gemeinsamkeiten mit der Basisversion.

Da die SEAL-Ketten auf jeden Fall einmal ihr Ende erreichen und erneuert werden müssen, wird die Variante mit der Kompletterneuerung gegenüber dem Aufschliessen in den Ketten bevorzugt. Mit genügend grossem T_K fällt die Redistribution des Public-Keys nicht ins Gewicht und es entsteht kein Mehraufwand zur Verifikation von SEALs an der Grenze.

Für den Fall, dass ein Gegner $T_K = 1\,\text{h}$ Zeit hat um SEALs vorauszuberechnen, gibt uns Tabelle 3.1(b) Auskunft über die Wahrscheinlichkeit, dass er eine gewisse Anzahl aktive SEALs finden kann. Mit einer SEAL-Länge von $b_s = 64\,\text{bit}$ ist die Wahrscheinlichkeit, dass der Gegner ein aktives SEAL kennt relativ gross und kann zusätzliche SEAL zur Verstärkung des Schutzes erfordern. Dies muss jedoch im Einzelfall beurteilt werden. Eine Länge von $b_S = 72\,\text{bit}$ bietet hingegen eine gute Sicherheit. Die minimale Länge des Salts lässt sich wie zuvor mit $88\,\text{bit} - b_S$ ermitteln. Da es aber nur nach T_K übertragen werden muss, kann es gut auch etwas grösser gewählt werden. $b_K = 32\,\text{bit}$ wäre eine angemessene Wahl.

3.4.3. Vergleich

Mit SEAL-Längen von 56 bit für die Basisversion und 72 bit für die Erweiterungen des Authentifizierungsprotokolls nimmt die Wahrscheinlichkeit zusätzliche aktive SEALs zu finden mit der Zunahme derer Anzahl sehr schnell ab. Die Schwächung der Signatur dadurch dürfte einiges geringer sein. Die Wahrscheinlichkeit für $B \geq 4$ sind vernachlässigbar und auch die anderen Wahrscheinlichkeiten sind klein genug um das Sicherheitslevel nicht massiv zu schwächen. Im weiteren Verlauf gehe ich davon aus, dass

$$P[A] \approx P[A \mid B = 0] \leq 10^{-6}$$

als Sicherheitsdefinition ausreicht. Diese Annahme wird in Abschnitt 3.7 noch genauer überprüft. Für die Erweiterungen könnte auch $b_S = 64\,\text{bit}$ bereits genügen, da die Sicherheit der Signatur mit jedem SEAL stufenweise zunimmt und daher oft noch etwas Reserve beinhaltet. $b_S = 48\,\text{bit}$ für die Basisversion wird hingegen kaum eine zulässige Länge darstellen.

Bezüglich der Grösse lässt sich feststellen, dass die SEALs für die Basisversion ein oder zwei Bytes kleiner sind als die SEALs für die Erweiterungen. Dafür müssen zusätzlich mit jeder Signatur 32 bit für das aktuelle Salt übermittelt werden. Für eine geringe Anzahl SEALs pro Signatur, sind die Signaturgrössen bei der Basisversion nicht viel kleiner als mit 64 bit bei den Erweiterungen. Mit $k = 4$ SEALs sind sie identisch, mit $k = 6$ sind die letzteren nur gerade 4% grösser und 8% grösser mit $k = 10$. Mit $b_S = 72\,\text{bit}$ für SEALs der Erweiterungen ist der Unterschied 17% bei $k = 6$ und 22% bei $k = 10$. Die Basisversion ist also eher vorteilhafter für eine grössere Anzahl SEALs pro Signatur.

3.5. Signaturschema

Die kleinstmögliche Signaturgrösse für ein festgelegtes Sicherheitslevel kann erreicht werden, wenn der Gegner maximal die Anzahl SEALs einer einzigen Signatur kennen darf, d.h. wenn gilt $r = k$. Wenn zusätzlich Resistenz gegen maximal n_l aufeinanderfolgende Paketverluste gefordert wird, muss gelten $r = (n_l + 1) \cdot k$.

Um die Signaturschemata BiBa/Powerball und HORS miteinander vergleichen zu können, muss berücksichtigt werden, dass ein Angreifer pro Versuch eine Signatur zu fälschen bei BiBa und Powerball mehrmals G_h und einmal H aufrufen muss und bei HORS nur einmal H. H ist eine Hash-Funktion, welche auf grösseren Inputs operiert als G_h. SHA-1 und MD5 operieren auf Blockgrössen von 512 bit ([MVO97]). Falls sowohl H als auch G_h mit einer dieser Funktionen implementiert wird und der Input für H 512 B und für G_h 64 B beträgt, so ist die Berechnung von H acht mal aufwändiger als die Berechnung von G_h. Diese Annahme ist gerechtfertigt, da ein Videoframe mindestens 512 B gross ist und die SEALs kleiner sind als 64 B.

Bei einem Angriff auf BiBa berechnet ein Gegner zuerst $H(m')$ und wirft dann zwei Bälle. Wenn diese nicht im selben Korb landen, kann er den Versuch abbrechen, falls er nur die Bälle einer einzigen Signatur kennt. Kennt er gültige Bälle von $n_l + 1$ Signaturen, wird er mindestens $(2 + k \cdot n_l + 8) \cdot T_e$ für einen Fälschungsversuch aufwenden. Bei Powerball dauert ein Versuch mindestens $(1 + k \cdot n_l + 8) \cdot T_e$ und bei HORS $8T_e$. Es ist erforderlich, dass die Anzahl nötiger Versuche um eine Signatur fälschen zu können viel grösser ist, als die Anzahl Versuche, die ein starker Gegner machen kann, bis die ihm bekannten SEALs deren Gültigkeit verlieren. Ohne Paketverluste werden sie nach T_d ungültig, mit tolerierten Paketverlusten spätestens nach $T_d + \max(T_d, n_l/f)$. Einem starken Gegner ist es z.B. möglich $T_d/(8T_e)$ Versuche zu machen eine HORS-Signatur zu fälschen, wenn keine Paketverluste erlaubt sind. Pro Versuch ist er mit einer Wahrscheinlichkeit von $P_f(r)$ erfolgreich. Es handelt sich um eine grosse Anzahl Versuche mir sehr geringen Erfolgswahrscheinlichkeiten. Die Wahrscheinlichkeit, dass es insgesamt möglich ist eine Signatur zu fälschen, kann darum mit der Poisson-Verteilung berechnet werden. Für das besagte Beispiel mit HORS muss gelten

$$P[A \mid B = 0] = 1 - e^{-np} = 1 - e^{-\frac{T_d \cdot P_f(r)}{8T_e}} \leq 10^{-6}$$

Wir erinnern uns, dass für die Wahrscheinlichkeit P_f eine Signatur in einem Versuch fälschen zu können bei BiBa gilt

$$P_f \leq \frac{\binom{r}{k}(n-1)^{r-k}}{n^{r-1}}$$

bei Powerball (mit k' der Anzahl Bällen pro Signatur)

$$P_f \leq \frac{(n_l + 1) \cdot \binom{(k'+1)\cdot(n_l+1)-1}{k'} \cdot k'!}{n^{k'}}$$

n_l	$T_d = 0.5\,\mathrm{s}$	$T_d = 2\,\mathrm{s}$	$T_d = 6\,\mathrm{s}$
0	$(7, 63\,\mathrm{B})$	$(8, 72\,\mathrm{B})$	$(8, 72\,\mathrm{B})$
1	$(10, 103\,\mathrm{B})$	$(11, 113\,\mathrm{B})$	$(11, 113\,\mathrm{B})$
2	$(12, 138\,\mathrm{B})$	$(13, 150\,\mathrm{B})$	$(13, 150\,\mathrm{B})$
3	$(15, 192\,\mathrm{B})$	$(15, 192\,\mathrm{B})$	$(16, 204\,\mathrm{B})$
4	$(17, 238\,\mathrm{B})$	$(19, 266\,\mathrm{B})$	$(20, 280\,\mathrm{B})$

(a) BiBa

n_l	$T_d = 0.5\,\mathrm{s}$	$T_d = 2\,\mathrm{s}$	$T_d = 6\,\mathrm{s}$
0	$(7, 63\,\mathrm{B})$	$(7, 63\,\mathrm{B})$	$(7, 63\,\mathrm{B})$
1	$(9, 93\,\mathrm{B})$	$(10, 103\,\mathrm{B})$	$(10, 103\,\mathrm{B})$
2	$(11, 127\,\mathrm{B})$	$(12, 138\,\mathrm{B})$	$(12, 138\,\mathrm{B})$
3	$(13, 166\,\mathrm{B})$	$(14, 179\,\mathrm{B})$	$(14, 179\,\mathrm{B})$
4	$(15, 210\,\mathrm{B})$	$(16, 224\,\mathrm{B})$	$(17, 238\,\mathrm{B})$

(b) Powerball

n_l	$T_d = 0.5\,\mathrm{s}$	$T_d = 2\,\mathrm{s}$	$T_d = 6\,\mathrm{s}$
0	$(8, 72\,\mathrm{B})$	$(9, 81\,\mathrm{B})$	$(9, 81\,\mathrm{B})$
1	$(11, 113\,\mathrm{B})$	$(11, 113\,\mathrm{B})$	$(11, 113\,\mathrm{B})$
2	$(12, 138\,\mathrm{B})$	$(13, 150\,\mathrm{B})$	$(13, 150\,\mathrm{B})$
3	$(14, 179\,\mathrm{B})$	$(15, 192\,\mathrm{B})$	$(15, 192\,\mathrm{B})$
4	$(16, 224\,\mathrm{B})$	$(16, 224\,\mathrm{B})$	$(17, 238\,\mathrm{B})$

(c) HORS

Tabelle 3.2.: (k, B_s): k SEALs pro Signatur und Signaturgrössen B_s mit max. n_l aufeinanderfolgenden Paketverlusten

und bei HORS

$$P_f = \left(\frac{r}{t}\right)^k$$

Zur Bestimmung der Anzahl SEALs pro Signatur kann diese solange erhöht werden, bis die Sicherheitsbedingung eingehalten wird. Die Anzahl Körbe n für BiBa und Powerball ist abhängig von der Anzahl SEALs pro Signatur und wird von Perrig [Per01] und Tabelle 2.1 vorgegeben. Die Signaturgrösse B_s in Byte setzt sich aus den SEALs sowie der Information über die SEAL-Grenze zur Toleranz von Paketverlusten zusammen:

$$B_s = \left\lceil k \cdot \frac{b_S + n_l \cdot \log t}{8} \right\rceil$$

In Tabelle 3.2 sind für verschiedene Auslieferungszeiten T_d und Paketverlusttoleranzen n_l die Anzahl SEALs pro Signatur k und die resultierenden Signaturgrössen B_s für BiBa, Powerball und HORS angegeben. Die Anzahl der SEALs, welche dem Signierer zur Verfügung stehen, beträgt jeweils $t = 1024$. Die kleinsten Signaturen lassen sich mit Powerball erreichen. Die Signaturen sind bei BiBa mit einer geringen Anzahl aufeinanderfolgenden Paketverlusten nur

einen SEAL grösser. Für $(T_d = 0.5\,\text{s}, n_l = 0)$ sind sie sogar gleich gross. Mit einer grösseren Anzahl erlaubter Paketverluste macht es sich jedoch deutlich bemerkbar und es können bis zu drei SEALs eingespart werden. Im Vergleich zu HORS hat Powerball ebenfalls kleinere Signaturen. Für wenige tolerierte Paketverluste macht der Unterschied ein oder zwei SEALs aus. Mit der zunehmenden Anzahl erlaubter Paketverluste werden die Unterschiede jedoch kleiner.

Da die Toleranz für n_l aufeinanderfolgende Paketverluste mit sich bringt, dass ein Angreifer $r = (n_l + 1) \cdot k$ SEALs kennen kann, muss die Fälschungssicherheit erhöht werden. Die Toleranz von Paketverlusten hat daher einen hohen Preis. Für $T_d = 0.5\,\text{s}$ werden die Signaturen beinahe 50% grösser, falls ein einziger Paketverlust erlaubt ist. Der Einfluss der Auslieferungszeit T_d hingegen ist eher gering. Obwohl $T_d = 6\,\text{s}$ 12 mal grösser ist als $T_d = 0.5$, ist für kleine Verlusttoleranzen höchstens ein zusätzlicher SEAL pro Signatur erforderlich. T_d bestimmt die Anzahl Fälschungsversuche, die ein Angreifer machen kann, sowie die Zeit nach der eine Signatur als verspätet gilt. Eine grössere Sicherheit hat also nur einen kleinen Preis, falls der Einfluss von T_d auf den Public-Key vernachlässigt wird.

BiBa kann als weiterer Kandidat ausgeschlossen werden, da Powerball bei gleichem Aufwand und gleicher Signaturgrösse eine höhere Sicherheit bietet. Die Signaturen von HORS sind zwar etwas grösser als die von Powerball, der Signierungsaufwand ist jedoch deutlich kleiner, weshalb HORS weiterhin als Kandidat verbleibt.

3.5.1. Optimierung HORS

HORSU Die zur Signierung zu verwendenden SEALs sind bei HORS zufällig, wodurch einzelne SEALs auch mehrfach vorkommen können. Es kann gefordert werden, dass die SEALs voneinander verschieden sein müssen. Diese Modifikation von HORS nenne ich *HORSU* („HORS of **U**nique SEALs“). Eine Signatur wie bis anhin wäre nicht mehr immer möglich. Ähnlich wie bei BiBa kann ein Zähler eingeführt werden, welcher eine erfolgreiche Signatur garantiert. Der Hash einer Nachricht kann dann mit $h = H(m||c)$ berechnet werden, wobei der Zähler c zu Beginn auf 0 gesetzt ist. Sind die daraus abgeleiteten SEAL-Indizes nicht paarweise voneinander verschieden, so wird der Zähler um eins erhöht und der Hash neu berechnet, bis eine gültige Signatur gefunden wurde. Der Erwartungswert der Anzahl mit $k = 9$ nötigen Signierungsversuche ist 1.04, was keinen nennenswerten Zusatzaufwand darstellt. Im Vergleich zu BiBa muss der Zähler nicht übermittelt werden. Jeder Empfänger beginnt ebenfalls mit $c = 0$ und authentifiziert die SEALs mit dem ersten gültigen Satz an Indizes. Die Wahrscheinlichkeit eine Signatur in einem Versuch fälschen zu können entspricht für HORSU

$$P_f = \frac{\binom{r}{k}}{\binom{t}{k}}$$

Tabelle 3.3 zeigt die Anzahl nötiger SEALs pro Signatur für HORS, HORSU und den später eingeführten HORSO. Gegenüber HORS kann HORSU die Anzahl SEALs pro Signatur um einen SEAL reduzieren. Für $T_d \in \{0.5, 1\}$ und keinen erlaubten Paketverlusten wird HORSU

n_l	HORS	HORSU	HORSO
0	9	8	6
1	11	10	8
2	13	12	9
3	15	14	10
4	16	16	11

Tabelle 3.3.: Anzahl SEALs pro Signatur für HORS und dessen Optimierungen mit $T_d = 2\,\mathrm{s}$ und max. n_l aufeinanderfolgenden Paketverlusten. HORSU verwendet eindeutige Indizes und HORSO ordnet diese zusätzlich.

mit $k = 7$ sogar gleich effizient wie Powerball. Da diese Optimierung praktisch keine Kosten verursacht aber die Signaturgrössen verringert, wird HORS uninteressant und HORSU an dessen Stelle weiterverfolgt.

HORSO Um die Komplexität für einen Angreifer weiter zu erhöhen, kann eine Ordnung in die SEALs einer Signatur gebracht werden, so dass jeder SEAL S_j mit $1 \leq j \leq k$ an einen bestimmten Substring i_j des Hashes h gebunden wird. Dies kann z.B. erreicht werden, wenn als j-tes SEAL einer Signatur nicht das erste SEAL unterhalb der SEAL-Grenze der Kette i_j verwendet wird, sondern das j-te SEAL unterhalb der SEAL-Grenze. HORSU mit einer solch erzwungenen Ordnung nenne ich *HORSO* („HORS of **O**rdered SEALs“). Kennt ein Gegner nur die SEALs einer Signatur, so kann er auch dann keine Signatur fälschen, wenn er zwar eine Nachricht findet, deren Hash die selben Substrings i_j ergibt, diese jedoch in einer anderen Reihenfolge sind. Er kennt nur einen SEAL, welcher durch k-fache Anwendung von F authentifiziert werden kann. Dieser muss daher zwingend aus der durch den Substring i_k spezifizierten Kette stammen. Ein Empfänger authentifiziert einen SEAL S_j einer Signatur, in dem er diesen in der entsprechenden SEAL-Kette über j Schritte zurückverfolgt. Es müssen zur Verifikation also insgesamt $\frac{k\cdot(k+1)}{2}$ Hash-Funktionen ausgewertet werden, wobei zuvor k nötig waren. Die Wahrscheinlichkeit für einen erfolgreichen Fälschungsversuch ist

$$P_f = \frac{\prod_{i=0}^{k-1}(n_l + 1 + i \cdot n_l)}{\binom{t}{k} \cdot k!}$$

Mit HORSO können die bisher kleinsten Signaturen erzeugt werden. Nur gerade $k = 6$ SEALs pro Signatur sind ausrechend, falls keine Paketverluste toleriert werden. Gegenüber HORSU können ein bis fünf SEALs eingespart werden. Die erzwungene Ordnung der SEALs hat also eine grosse Auswirkung auf die Sicherheit. Da pro Signatur bis zu k SEALs einer SEAL-Kette verbraucht werden, ist HORSO für die Basisversion des Authentifizierungsprotokolls weniger sinnvoll. Dessen Nachteile würden um Faktor k zunehmen.

Auch bei BiBa oder Powerball wäre es denkbar, zusätzliche Sicherheitsinformation in die SEAL-Ketten zu codieren. So könnte z.B. bei BiBa der Index des Korbes mit den k Bällen dazu verwendet werden, um die zu verwendenden Positionen in den SEAL-Ketten anzugeben.

Auch in diesem Falle würden dem Sender sämtliche Möglichkeiten offen stehen, ein Angreifer wäre jedoch stark eingeschränkt. Hier besteht viel Spielraum und kleinere Signaturen sind zu erwarten. In dieser Arbeit werden diese Möglichkeiten jedoch nicht weiter untersucht.

3.5.2. Weitere Ideen zur Reduktion der Signaturgrösse

Zur Reduktion der Signaturgrössen werden in diesem Unterabschnitt weitere Optionen geprüft. Für die Berechnungen gelte jeweils $T_d = 1\,\text{s}$ und $n_l = 0$.

Signierwahrscheinlichkeit Bei Powerball bestimmt die Wahrscheinlichkeit P_s, mit der ein Signierer im ersten Versuch eine Signatur finden kann, die Anzahl Körbe n. Diese Wahrscheinlichkeit war bisher als 50% festgelegt. Das Erzeugen einer Signatur dauert damit ca. 2.9 ms. Wenn diese Erfolgswahrscheinlichkeit reduziert wird, kann die Anzahl Körbe auf Kosten des Signierungsaufwandes vergrössert werden und das Sicherheitsniveau steigt. Mit $P_s = 0.1$ ergibt sich noch keine Auswirkung auf die Anzahl SEALs pro Signatur. Mit $P_s = 0.05$ kann hingegen die Signaturgrösse um einen SEAL auf $k = 6$ verringert werden. Das Erzeugen einer Signatur dauert dann aber ca. 29 ms, was sogar deutlich länger ist als die Zeit zum Signieren mit RSA. Da die Grösse nur um einen SEAL verkleinert werden kann, wird die Reduktion der Erfolgswahrscheinlichkeit nicht in Betracht gezogen.

Viele SEAL-Ketten Das Sicherheitslevel hängt von t der Anzahl SEALs des Signierers ab, welche bisher als $t = 1024$ gewählt wurde. Bei HORS ist der Einfluss offensichtlich. Bei Powerball führt ein grösseres t zu einer grösseren Anzahl Körbe und erhöht dadurch die Sicherheit. Die Kosten dafür sind ein grösserer Public-Key und bei Powerball ein höherer Signierungsaufwand. Powerball kommt mit $t = 2048$ SEAL-Ketten mit $k = 6$ SEALs pro Signatur aus, was einem weniger entspricht als mit $t = 1024$. Bei HORSU reicht $k = 6$ mit $t = 2120$ und $k = 5$ mit $t = 6845$. HORSO kann mit $t = 2629$ einen SEAL einsparen und es reichen $k = 5$ SEALs pro Signatur. Mit einer Verzehnfachung des Public-Keys auf $t = 10240$ ergibt sich bei keinem Signaturschema eine weitere Reduktion. Da der Public-Key ohnehin schon sehr gross ist, sollte die Erhöhung von t nur in Ausnahmefällen in Betracht gezogen werden. Vorher sollte z.B. geprüft werden, ob nicht HORSO anstelle von Powerball oder HORSU eingesetzt werden kann, ohne die Anzahl SEAL-Ketten zu vergrössern.

Weniger SEALs Wird einfach dreist mit einem SEAL weniger signiert, sinkt die Wahrscheinlichkeit für eine erfolgreiche Fälschung bei Powerball auf $1.4 \cdot 10^{-5}$ und bei HORSU auf $7.9 \cdot 10^{-5}$. Dies verfehlt das angestrebte Sicherheitslevel von 10^{-6}.

3.6. Framebündelung

Sämtliche im letzten Abschnitt entworfenen Signaturen sind für Pulsar zu gross und sprengen dessen Effizienz. Zur Reduktion der Signaturgrösse pro Frame kann anstelle der einzelnen Frames eine Liste signiert werden, welche die Hashes von n_f Frames enthält. Beim Sender muss dafür ein Zwischenspeicher eingeführt werden. Für n_f sendebereite Frames $\langle m_1, \ldots, m_{n_f} \rangle$ wird jeweils ein Hash $h'_i = H'(m_i)$ mit $1 \leq i \leq n_f$ berechnet. $H'(m_i)$ ist eine Hash-Funktion aus dem Random-Oracle-Modell, die ein Frame auf einen $b_{h'}$ langen Bitstring abbildet. Diese Liste von Hash-Werten $m = h'_1 || \ldots || h'_{n_f}$ wird signiert und vor den eigentlichen Frames an die Empfänger verteilt. Diese authentifizieren die Liste und können später ein Frame m_i authentifizieren, indem sie prüfen, ob $h'(m_i)$ in der Hash-Liste an Position i steht. Da es wichtig ist, dass die Liste vor den Frames bei den Empfängern eintrifft, ist eine Voraussendezeit T_v nötig. Die Länge des Zwischenspeichers entspricht demnach $T_v + n_f / f$. Für das Authentifizierungsprotokoll sind mit Framebündelung nur Verluste von Signaturen von Bedeutung. Verlorene Frames können die Wiedergabequalität vermindern, haben jedoch keinen Einfluss auf das Aktualisieren der SEAL-Grenze.

Für die Sicherheit der Hash-Werte muss gelten, dass ein starker Gegner nicht innert deren Gültigkeit $T_v + T_d + \frac{n_f}{f}$ für einen Hash $h'_i = H'(m_i)$ eines Frames m_i ein weiteres Frame m_j mit $H'(m_j) = H'(m_i)$ und $m_j \neq m_i$ finden kann. Er darf auch nicht innert mehrerer Jahre für beliebige Frames deren Hashes vorausberechnen und dadurch mit zu grosser Wahrscheinlichkeit für einen zufälligen Hash h'_i ein Frame m_i mit $H'(m_i) = h'_i$ kennen. Wie bei den SEALs in Abschnitt 3.4 kann ein Salt verwendet werden, um die Hash-Länge $b_{h'}$ klein zu halten. Dieses zufällige Salt K' mit Bitlänge $b_{K'}$ habe die selbe Gültigkeitsdauer wie die Hash-Liste selbst und wird zusammen mit ihr übermittelt. Die Funktion $H'(m_i)$ wird durch $H'(m_i || K')$ ersetzt. Ein starker Gegner braucht für deren Auswertung $8T_e$. Es stellen sich folgende Bedingungen:

$$1 - e^{-np} = 1 - e^{-\frac{T_v + T_d + \frac{n_f}{f}}{8T_e} \cdot \frac{1}{2^{b_{h'}}}} \leq 10^{-6}$$

$$1 - e^{-np} = 1 - e^{-\frac{50\,\mathrm{a}}{8T_e} \cdot \frac{1}{2^{b_{h'} + b_{K'}}}} \leq 10^{-6}$$

Mit $T_v = 2\,\mathrm{s}$, $T_d = 6\,\mathrm{s}$, $n_f = 36$ und $f = 36\,\mathrm{s}^{-1}$ muss die Länge der Hash-Werte auf ganze Bytes gerundet $b_{h'} = 64\,\mathrm{bit}$ betragen. Auch die Wahl von etwas kleineren Werten resultiert in der selben Grösse. Für die Salts reichen $b_{K'} = 24\,\mathrm{bit}$.

Mit Framebündelung wird die Rate, mit der Signaturen erstellt werden, um Faktor n_f reduziert. Ein sehr angenehmer Effekt davon ist die Reduktion der Anzahl nötigen Signaturinstanzen und damit der Grösse des Public-Keys:

$$n_i = \left\lceil T_d \cdot \frac{f}{n_f} \cdot \frac{k}{r} \right\rceil = \left\lceil T_d \cdot \frac{f}{n_f} \cdot \frac{1}{n_l + 1} \right\rceil$$

Die Grösse einer Signatur setzt sich aus den SEALs, der Information zur Toleranz von Signa-

n_f	$n_l = 0$		$n_l = 1$		$n_l = 2$	
1	63 B	(63.0 B)	93 B	(93.0 B)	127 B	(127 B)
9	138 B	(15.3 B)	168 B	(18.7 B)	202 B	(22.4 B)
18	210 B	(11.7 B)	240 B	(13.3 B)	274 B	(15.2 B)
27	282 B	(10.4 B)	312 B	(11.6 B)	346 B	(12.8 B)
36	354 B	(9.8 B)	384 B	(10.7 B)	418 B	(11.6 B)

(a) Powerball mit $k = (7,9,11)$ für $n_l = (0,1,2)$

n_f	$n_l = 0$		$n_l = 1$		$n_l = 2$	
1	63 B	(63.0 B)	103 B	(103.0 B)	138 B	(138 B)
9	138 B	(15.3 B)	178 B	(19.8 B)	213 B	(23.7 B)
18	210 B	(11.7 B)	250 B	(13.9 B)	285 B	(15.8 B)
27	282 B	(10.4 B)	322 B	(11.9 B)	357 B	(13.2 B)
36	354 B	(9.8 B)	394 B	(10.9 B)	429 B	(11.9 B)

(b) HORSU mit $k = (7,10,12)$ für $n_l = (0,1,2)$

n_f	$n_l = 0$		$n_l = 1$		$n_l = 2$	
1	54 B	(54.0 B)	82 B	(82.0 B)	104 B	(104 B)
9	129 B	(14.3 B)	157 B	(17.4 B)	179 B	(19.9 B)
18	201 B	(11.2 B)	229 B	(12.7 B)	251 B	(13.9 B)
27	273 B	(10.1 B)	301 B	(11.1 B)	323 B	(12.0 B)
36	345 B	(9.6 B)	373 B	(10.4 B)	395 B	(11.0 B)

(c) HORSO mit $k = (6,8,9)$ für $n_l = (0,1,2)$

Tabelle 3.4.: Signaturgrössen und in Klammern Signaturgrössen pro Frame mit n_f gebündelten Frames, max. n_l aufeinanderfolgenden Signaturverlusten und $T_d = 1\,\mathrm{s}$

turverlusten sowie neu der Hash-Werte der gebündelten Frames und dem Salt zusammen:

$$B_s = \left\lceil \frac{k \cdot (b_S + n_l \cdot \log t) + n_f \cdot b_{h'} + b_{K'}}{8} \right\rceil$$

Tabelle 3.4 zeigt die Signaturgrössen mit Framebündelung sowie die Signaturgrössen pro Frame für eine Auslieferungszeit von $T_d = 1\,\mathrm{s}$. Je mehr Frames gebündelt werden, umso kleiner ist die Signatur pro Frame, da die Grösse der SEALs unter den Frames aufgeteilt wird. Der Hash eines Frames in der Hash-Liste trägt jedoch mit 8 B einen wesentlichen Teil zur Signaturgrösse pro Frame bei und stellt für sie eine asymptotische untere Grenze dar. Die maximale Anzahl aufeinanderfolgender Signaturverluste n_l hat nach wie vor einen grossen Einfluss auf die Signaturgrösse. Mit der in den späteren Abschnitten 3.9 und 3.10 vorgeschlagenen Handhabung, ist keine Verlusttoleranz in den Signaturen nötig, weshalb nachfolgend nur der Fall $n_l = 0$ betrachtet wird.

Ohne Framebündelung ist die Signatur pro Frame zu gross und Bündelung schafft Abhilfe. Neben der Grösse pro Frame ist jedoch auch die totale Signaturgrösse von Bedeutung. Es ist wünschenswert, dass ein separates IP Paket für die Signatur vermieden wird und diese mit

Modus / SEAL-Länge	Powerball	HORSU	HORSO
Basisversion 56 bit	$1.3 \cdot 10^{-7}$	$6.0 \cdot 10^{-7}$	$1.1 \cdot 10^{-7}$
Basisversion 48 bit	$1.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$	0.48
Erweiterung 72 bit	$1.2 \cdot 10^{-7}$	$5.5 \cdot 10^{-7}$	$1.1 \cdot 10^{-7}$
Erweiterung 64 bit	$2.8 \cdot 10^{-7}$	$1.6 \cdot 10^{-6}$	$5.6 \cdot 10^{-5}$

Tabelle 3.5.: Wahrscheinlichkeit mit Vorausberechnung von SEALs eine Signatur fälschen zu können. Es gilt $T_d = 1\,\text{s}$ und $n_l = 0$.

einem Frame verschickt werden kann. Zusätzlichen Overhead durch die IP und UDP Header kann so vermieden werden. Die maximale Paketgrösse beträgt 1500 B. Das Videoframe nimmt 1328 B in Anspruch. Für einen IPv6- und einen UDP-Header werden 48 B gebraucht [PD03]. Es verbleiben 124 B, wovon noch einige Bytes für Pulsar gebraucht werden. Die Signatur sollte daher unter 100 B gross sein. Keine der analysierten Signaturen erfüllt mit Bündelung diese Bedingung. Um die Signaturgrösse weiter zu reduzieren, kann deshalb in der Hash-Liste anstelle von einem Hash pro Frame nur ein Hash für zwei Frames aufgeführt. Für einen Hash h'_i in der Hash-Liste gilt dann: $h'_i = H'(m_{2i}||m_{2i+1})$. Dies vermag den Anteil des Hashes pro Frame zu halbieren. Das Frame m_{2i} kann jedoch nur authentifiziert werden, wenn auch das Frame m_{2i+1} vorhanden ist. Geht ein Frame verloren, kann das ganze Framepaar nicht authentifiziert werden. Zudem entsteht ein kleiner Delay, da zuerst gewartet werden muss, bis das zweite Frame eingetroffen ist, bevor das erste weitergeschickt werden kann. Dieser Delay kann klein gehalten werden, wenn das Pulsar Protokoll derart verändert wird, dass zusammengehörende Frames auf dem selben Pfad verbreitet werden.

Es ist klar, dass mit dieser Modifikation n_f gerade sein sollte. Mit $n_f = 8$ entspricht die Signaturgrösse bei Powerball und HORSU 98 B und bei HORSO 89 B. Pro Frame sind das 12.3 B und 11.1 B. Für HORSO würde die Möglichkeit bestehen $n_f = 10$ Frames zu bündeln. Die Signatur wäre dann 97 B gross und somit immer noch unter den gewünschten 100 B. Pro Frame wäre der Anteil noch 9.7 B. Die Doppel-Hashes erlauben also kleine Signaturen mit einem einen geringen Signaturanteil pro Frame bei einer kleineren Verzögerung.

3.7. Überprüfung der SEAL-Längen

Bei der Festlegung der SEAL-Längen in Abschnitt 3.4 wurde die Annahme getroffen, dass es deutlich schwieriger sei einen zusätzlichen aktiven SEAL durch Vorausberechnung zu finden, als dass dadurch die Sicherheit geschwächt würde. Für die Festlegung der Anzahl SEALs pro Signatur wurde dann angenommen, dass ein Gegner keine weiteren aktiven SEALs kennt. In diesem Abschnitt werden diese Annahmen geprüft.

Tabelle 3.5 zeigt die totale Wahrscheinlichkeit mit Vorausberechnung von SEALs eine Signatur fälschen zu können. Wie bereits vermutet, ist eine SEAL-Länge von $b_S = 48\,\text{bit}$ für die

Schema / Modus	Bündelung	Signatur	pro Frame
Powerball Basisversion	$n_f = 8$	89 B	11.1 B
	$n_f = 10$	97 B	9.7 B
Powerball Erweiterungen	$n_f = 8$	92 B	11.5 B
	$n_f = 10$	100 B	10.0 B
HORSO Erweiterungen	$n_f = 8$	89 B	11.1 B
	$n_f = 10$	97 B	9.7 B

Tabelle 3.6.: Signaturgrössen für die interessanten Kombinationen von Authentifizierungsprotokoll und Signaturschema. Es gilt $T_d = 1\,\text{s}$ und $n_l = 0$.

Basisversion des Authentifizierungsprotokolls nicht ausreichend. Mit den „sicheren“ SEAL-Längen von 56 bit für die Basisversion und 72 bit für dessen Erweiterungen ist die Fälschungswahrscheinlichkeit überall unter 10^{-6} und die Annahme ein Gegner kenne keine weiteren SEALs damit gerechtfertigt. Beim Einsatz einer der Erweiterungen können mit Powerball sogar die kürzeren SEALs von $b_S = 64\,\text{bit}$ verwendet werden. Für HORSU ist dies streng genommen nicht erlaubt, könnte jedoch in Erwägung gezogen werden. Für HORSO sind hingegen SEALs von 72 bit Länge zwingend.

Da Powerball bei gleicher Anzahl SEALs pro Signaturgrösse eine höhere Sicherheit bietet und den Einsatz kürzerer SEALs erlaubt als dies mit HORSU möglich ist, wird es HORSU vorgezogen. Es besteht zwar ein Unterschied im Signierungsaufwand, dieser ist jedoch vertretbar. Aufgrund der negativen Effekte auf die Kettenlänge wird HORSO nur für die Erweiterungen des Authentifizierungsprotokolls analysiert. Tabelle 3.6 können die Signaturgrössen der verbleibenden Möglichkeiten entnommen werden. Für Powerball wurde erstmals die Grösse des Counters von einem Byte mitberücksichtigt.

Je nachdem ob die Signaturgrösse oder deren Anteil pro Frame wichtiger ist, kann in allen Fällen eine Bündelung von 8 oder 10 Frames gewählt werden. Es sei hier wichtiger mehr als ein Byte pro Frame einzusparen und es wird $n_f = 10$ gewählt. Die Signaturgrössen unterscheiden sich dann unabhängig von Signaturschema und Authentifizierungsprotokoll nicht gross voneinander. Powerball hat mit der Basisversion kleine SEALs, muss dafür das Salt der SEALs mitsenden. Mit den Erweiterungen hat HORSO weniger SEALs pro Signatur, dafür sind diese länger als die von Powerball. Der zusätzliche Aufwand für längere SEAL-Ketten mit der Basisversion lohnt sich also nur, wenn die SEALs einen grossen Teil der Signatur ausmachen und Powerball verwendet werden soll. Da HORSO pro Signatur mehr SEALs von den SEAL-Ketten benutzt und die Signaturgrösse beinahe identisch ist, fällt die Wahl auf Powerball. HORSO könnte natürlich auch gewählt werden, sollte jedoch zuerst mit weiteren Abänderungsmöglichkeiten von Powerball verglichen werden, wie dies in Unterabschnitt 3.5.1 angedeutet wurde.

n_f	$T_d = 0.5\,\text{s}$			$T_d = 6\,\text{s}$		
	$n_l = 0$	$n_l = 1$	$n_l = 2$	$n_l = 0$	$n_l = 1$	$n_l = 2$
1	18	9	6	216	108	72
9	2	1	1	24	12	8
18	1	1	1	12	6	4
27	1	1	1	8	4	3
36	1	1	1	6	3	2

Tabelle 3.7.: Anzahl Signaturinstanzen mit n_f gebündelten Frames, max. Auslieferzeit T_d und max. n_l aufeinanderfolgenden Signaturverlusten

3.8. Anzahl Signaturinstanzen

Wie bereits in Abschnitt 3.6 erwähnt, hat Framebündelung nicht nur Einfluss auf die Signaturgrösse pro Frame, sondern auch auf die Anzahl Signaturinstanzen. In Tabelle 3.7 wird dies deutlich. Je mehr gebündelt wird, je öfter eine Instanz wiederverwendet werden kann und je schneller die Signaturen bei den Empfängern eintreffen, desto weniger Instanzen sind nötig. Die Anzahl gebündelter Frames wurden bereits mit $n_f = 10$ und die maximale Anzahl aufeinanderfolgenden Signaturverlusten mit $n_l = 0$ festgelegt. Die maximale Auslieferungszeit einer Signatur wurde zwar oft als $T_d = 1\,\text{s}$ angenommen, wurde jedoch nie genau überprüft.

Mit dem Pulsar Protokoll werden laut Locher et al. [LMSW07] bei einer Netzwerkgrösse von 100'000 Teilnehmern sämtliche Frames innert 1.5 s ausgeliefert. Diese Verzögerung wächst logarithmisch zur Anzahl Peers. Nach der Push-Phase fehlende Frames werden in zufälliger Reihenfolge von den Nachbarn angefordert, sofern diese über die Frames verfügen. Wenn dabei Pakete mit Signaturen gegenüber normalen Paketen priorisiert werden, lassen sich diese schneller verteilen. Es ist davon auszugehen, dass Signaturen mit Priorisierung in einer Sekunde verbreitet werden können. Mit $T_d = 1\,\text{s}$ sind dann $n_i = 4$ Signaturinstanzen erforderlich. Auch mit $T_d = 1.1\,\text{s}$ würde die selbe Anzahl Instanzen ausreichen.

Wird der Public-Key mit einer $B_{RSA} = 128\,\text{B}$ grossen RSA Signatur versehen, ergibt sich dessen Grösse B_p zu:

$$B_p = n_i \cdot t \cdot \frac{b_S}{8} + \frac{b_K}{8} + B_{RSA} = 4 \cdot 1024 \cdot 8\,\text{B} + 4\,\text{B} + 128\,\text{B} \approx 32\,\text{KB}$$

Der Public-Key ist somit relativ gross. Eine stärkere Framebündelung könnte die Anzahl Signaturinstanzen reduzieren und der Public-Key könnte maximal um Faktor 4 kleiner gemacht werden. Da die Signaturgrösse jedoch nicht grösser sein soll als 100 B, kann dies nicht realisiert werden.

3.9. Index-Updates

Das Pulsar Protokoll ist grundsätzlich sehr zuverlässig und Paketverluste treten so gut wie nicht auf. In Einzelfällen könn jedoch Signaturen verloren gehen. In diesem Fall ist es wichtig, dass die SEAL-Grenze korrekt weiterverfolgen werden kann. Die Toleranz von einer geringen Anzahl Paketverlusten wurde bis anhin mit einer Erhöhung des Sicherheitslevels bezahlt, was sich auf die Anzahl SEALs ausgewirkt hat. Zudem wurden mit jeder Signatur die Indizes der von den vorherigen n_l Signaturen verwendeten SEAL-Ketten mitübertragen. Um mit einem geringeren Datenoverhead auszukommen, können diese Indizes periodisch mit einer herkömmlichen Signatur wie RSA verteilt werden.

Im Vergleich zur vorherigen Methode zum Tolerieren von Signaturverlusten kann die SEAL-Grenze nicht sofort korrigiert werden, sondern erst beim Eintreffen dieses *Index-Updates*. Der Ketten-Index jedes SEALs wird hingegen nur einmal übertragen. Ein weiterer entscheidender Vorteil ist, dass die Index-Updates auch unter den Clients authentisch ausgetauscht werden können, da die RSA Signatur ihre Gültigkeit auch über die Zeit beibehält. Dies kann zur dezentralen Initialisierung neuer Clients genutzt werden. Ein neuer Teilnehmer braucht eine alte SEAL-Grenze, die Index-Updates sowie die aktuelle SEAL-Grenze. Alles kann er von seinen Nachbarn erhalten. Die alte SEAL-Grenze und die Index-Updates können mit dem RSA Public-Key des Senders authentifiziert werden. Die aktuelle SEAL-Grenze kann dann durch Zurückverfolgen der Ketten über die in den Index-Updates angegebenen Schritte authentifiziert werden.

Wenn eine Signatur verlorengeht, so können nachfolgende Signaturen nicht mit grosser Sicherheit korrekt authentifiziert werden. Schlägt eine Verifikation nach einem Signaturverlust fehl, so weiss man nicht, ob das aufgrund einer erneuten Verwendung der betroffenen SEAL-Ketten passiert ist oder weil die Signatur gefälscht wurde. Falls eine Signatur nach einem Signaturverlust erfolgreich verifiziert werden kann, könnte sie mit zu grosser Wahrscheinlichkeit von einem Angreifer stammen. Beide Fälle stellen daher ein Problem dar. Streng genommen müssten nach einem Signaturverlust alle Signaturen und Frames zurückgewiesen werden, bis das nächste Index-Updates Klarheit schafft. Um die Wiedergabe durch einen Signaturverlust nicht zu stören, können die Videoframes trotzdem zur Anzeige gebracht werden. Sie sollten jedoch nicht mittels Pushing an andere Clients weiterverbreitet werden, was die Auswirkung auf den Teilnehmer beschränkt, welcher den Signaturverlust erlitten hat. Wenn die Verfügbarkeit der nicht authentifizierten Frames mit einem Hinweis auf die Nichtauthentizität den Nachbarn bekanntgemacht würde, so hätten diese trotzdem die Möglichkeit an die Frames zu kommen, falls sie sonst nicht verfügbar sind. Da ein Client als authentisch angekündigte Frames zuerst beziehen würde, ist das Risiko gefälschte Frames zu übertragen gering. Gefälschte Frames, welche nur als nicht authentisch angekündigt vorhanden sind oder bösartig als authentisch angekündigt werden, können weiterhin von Clients, welche keine Signaturverluste erlitten haben, korrekt erkannt werden. Dies ist ein angemessener Kompromiss zwischen hoher Integrität und hoher Verfügbarkeit.

Sei n_u die Anzahl Signaturen, die ein Index-Update bestätigt und B_{RSA} die Grösse einer RSA

n_u	T_u	B_u	Anteil
25	6.9 s	347 B	1.4 B
40	11.1 s	478 B	1.2 B
50	13.9 s	566 B	1.1 B
75	20.8 s	785 B	1.0 B
100	27.8 s	1003 B	1.0 B
150	41.7 s	1441 B	1.0 B
300	83.3 s	2753 B	0.9 B

Tabelle 3.8.: Index-Updates: Anzahl bestätigter Signaturen n_u, Zeit zwischen zwei Updates T_u, Update-Grösse B_u sowie deren Anteil pro Frame

Signatur in Byte, dann ergibt sich für ein Index-Update eine Bytegrösse B_u von:

$$B_u = \left\lceil \frac{n_u \cdot k \cdot \log t}{8} \right\rceil + B_{RSA}$$

Die Zeit T_u zwischen zwei Index-Updates beträgt:

$$T_u = \frac{n_u \cdot n_f}{f}$$

Mit der bisherigen Methode zum Schutz vor Signaturverlusten und $n_f = 9$ gebündelter Frames pro Signatur kostete die Toleranz von einem einzigen Signaturverlust in Folge etwa 3.5 B pro Frame. Mit den Index-Updates kann eine beliebige Anzahl Signaturen verloren gehen. Da die Signaturen jedoch nicht zusätzlich geschützt werden, muss hingegen eine Zeit mit erhöhtem Fälschungsrisiko in Kauf genommen werden. Eine untere Grenze für die Kosten pro Frame beträgt in Byte:

$$\frac{k \cdot \log t}{8 \cdot n_f} \approx 0.9\,\mathrm{B}$$

Wie in Tabelle 3.8 dargestellt, sind die Kosten pro Frame bereits mit $n_u = 40$ bestätigten Signaturen pro Index-Update nicht mehr allzu weit von den minimalen Kosten entfernt. Eine Verteilung der Index-Updates alle $T_d = 11.1\,\mathrm{s}$ ist günstig und begrenzt die maximale Zeit, in der ein Teilnehmer nach einem Signaturverlust die Signaturen nicht zuverlässig verifizieren kann, auf ein sinnvolles Mass. Die Index-Updates bieten also mit geringen Kosten eine sehr gute Toleranz gegen Paketverluste, eine angemessene Sicherheit und können zur dezentralen und authentischen Verteilung der aktuellen SEAL-Grenze an neue Teilnehmer genutzt werden.

3.10. Verspätete Signaturen

Wenn eine Signatur später als in T_d eintrifft, kennt ein Gegner bis zu diesem Zeitpunkt die SEALs einer weiteren Signatur der selben Instanz und seine Erfolgschancen nehmen zu. Aus diesem Grund wurde bisher gefordert, dass verspätete Signaturen einem Signaturverlust gleichzustellen und daher zu ignorieren sind. In diesem Abschnitt werden zuerst die Möglichkeiten des Gegners die Verteilung einer Signatur aktiv zu verzögern untersucht. Danach wird auf die Auswirkungen eingegangen, falls eine Signatur trotz Verspätung als gültig akzeptiert wird.

Die Zuverlässigkeit von Pulsar basiert auf den Pull-Operationen mit denen fehlende Datenblöcke von den Nachbarn angefordert werden können. Die Daten haben eine unzählige Anzahl mögliche Pfade, um ihr Ziel zu erreichen. Der kürzeste Pfad bestimmt die Auslieferungszeit und Verzögerungen auf anderen Pfaden haben daher keine Auswirkungen. Hat ein Gegner jedoch direkten Zugriff auf einen wichtigen Abschnitt des Internets, auf dem Pulsar Pakete versendet werden, kann er sämtliche Pakete verzögern oder gar löschen. Für einen Angreifer interessante Punkte wären unter anderem die Provider des Senders und der Empfänger, wichtige Backbones und unter Umständen lokale Netze beim Sender oder den Empfängern. Eine weitere Möglichkeit für einen Angreifer ist zu versuchen, sämtlichen Pulsar-Verkehr für einen oder mehrere Clients durch sich zu lenken, um den kürzesten Pfad zu kontrollieren. Er muss dazu alle Nachbarn stellen, welche einen Client oder eine Gruppe von Clients mit anderen Teilnehmern verbinden. Ein solcher Angriff wird als Eclipse-Angriff bezeichnet [SNDW06]. Gelingt ihm dies, so kann er wiederum Pakete beliebig löschen oder verzögern. Weiter kann auch das Überfluten eines Clients mit beliebigen Datenpaketen zu Verzögerungen von Paketen führen. Sämtliche Szenarien, welche es einem Angreifer ermöglichen Pakete beliebig zu verzögern oder deren Verbreitung zu verhindern, bieten ihm effektive Möglichkeiten den Betrieb zu stören. Das Ziel des Integritätsprotokolls ist, das Pulsar-Netzwerk vor Störungsversuchen durch Modifikation oder Erzeugung zu schützen. Es vermag nicht vor Verzögerungen zu schützen. Diese sind deshalb für das Authentifizierungsprotokoll nicht von Bedeutung und es wird davon ausgegangen, dass ein Angreifer die Zustellung von Signaturen nicht aktiv verzögern kann. Er interferiert insbesondere nur mit dem Pulsar Protokoll und nicht mit tieferen Layern. Für die zeitliche Synchronisation sowie die Übertragung des Public-Keys wurde dies zuvor schon implizit angenommen.

Da die Auslieferungszeit mit $T_d = 1\,\mathrm{s}$ nicht viel Reserve beinhaltet, kann es auch aus natürlichen Gründen zu Verspätung von Signaturen kommen. Wenn eine Signatur erst nach $2T_d = 2\,\mathrm{s}$ bei einem Empfänger eintrifft, kann ein starker Gegner mit einer Wahrscheinlichkeit von $2.4 \cdot 10^{-4}$ kurz vorher diese Signatur fälschen. Wenn sie erst nach $3T_d = 3\,\mathrm{s}$ eintrifft, beträgt die Wahrscheinlichkeit 0.04. Treffen diese Ereignisse selten ein, so können die Sicherheitsanforderungen trotzdem erfüllt werden, auch wenn verspätete Signaturen akzeptiert werden. Es darf eine von 240 Signaturen um T_d und eine von 40'000 um $2T_d$ verspätet sein. Wird dies eingehalten, so kann ein starker Angreifer weiterhin nur eine von 10^6 Signaturen fälschen. Zudem haben diejenigen Clients, welche die Signaturen verspätet erhalten, keinen so grossen Einfluss mehr. Sie befinden sich eher am Ende der Verteilkette und geben die Si-

gnatur nicht mehr an viele andere Teilnehmer weiter. Die Index-Updates würden zudem bei einer erfolgreichen Fälschung spätestens nach $T_u = 11.1\,\mathrm{s}$ den Fehler beheben und es ergäbe sich nur eine kurze Störung.

3.11. SEAL-Ketten

Die SEAL-Ketten haben eine begrenzte Länge l und müssen daher von Zeit zu Zeit erneuert werden. Für die Basisversion des Authentifizierungsprotokolles ist die Länge linear zur Lebensdauer des Public-Keys T_p:

$$l = \frac{T_p \cdot f}{n_i \cdot n_f}$$

Für die Erweiterungen des Authentifizierungsprotokolls hingegen lässt sich die minimal erforderliche Kettenlänge nicht einfach berechnen. Um die Länge l der t SEAL-Ketten bei BiBa oder HORS abzuschätzen, können die Erkenntnisse von Raab und Steger [RS98] herangezogen werden. Es müsste nur eine Konstante durch eine Simulation abgeschätzt werden und die Kettenlänge wäre für eine beliebige Public-Key-Lebenslänge, Anzahl SEALs pro Signatur, Anzahl Signaturinstanzen, Anzahl gebündelter Frames und Anzahl SEAL-Ketten bekannt. Bei Powerball werden für das Korbmuster jedoch zwei SEALs in der Kette benutzt und bei HORSO werden die k für eine Signatur benutzten Ketten noch unterschiedlicher beansprucht. Es handelt sich also um leicht andere Probleme und die Erkenntnisse von Raab und Steger können nicht verwendet werden. Die einfachste Möglichkeit ist die nötige Anzahl SEAL-Ketten durch Simulation zu erhalten oder die Ketten einfach ausreichend gross zu wählen. Letzteres verursacht jedoch einen unnötig grösseren Aufwand zum Erzeugen der Ketten. Mittels einer Simulation über 10^5 Public-Key Lebensdauern ergibt sich die minimal erforderliche Kettenlänge für die Parameter $T_p = 1\,\mathrm{h}$, $n_f = 10$, $n_i = 4$, $f = 36\,\mathrm{s}^{-1}$ für Powerball mit $k = 7$ zu 103 und für HORSO mit $k = 6$ zu 189.

Falls die SEAL-Ketten komplett abgespeichert werden, beträgt der Speicherbedarf für frisch erzeugte SEAL-Ketten beim Sender:

$$n_i \cdot l \cdot (t \cdot B_s\,[+B_K])$$

Für Powerball mit der Basisvariante entspricht dies 89 MB und mit den Erweiterungen 3 MB. Der enorme Speicherbedarf mit der Basisversion macht es offensichtlich, dass nur Zwischenwerte der SEAL-Ketten gespeichert werden können, wie dies in Unterabschnitt 3.3.1 erwähnt wurde.

Wenn der Public-Key seine Lebensdauer erreicht hat, muss er neu erzeugt und übertragen werden. Wenn dies wie in Abschnitt 3.4 nach $T_p = 1\,\mathrm{h}$ der Fall ist, ergibt sich mit $n_i = 4$ Signaturinstanzen ein geringer Overhead von $0.25\,\mathrm{B}$ pro Frame.

4. Ergebnis

Das entworfene Integritätsprotokoll erfüllt die von einem Einsatz für Pulsar gestellten Anforderungen gut. Die Signaturen sind inklusive der Hash-Listen 100 B gross und der gesamte Overhead pro Frame beträgt lediglich 11.5 B, was weniger als ein Prozent der Nutzdaten ausmacht. Eine Signatur kann in ca. 2.9 ms erstellt und in ca. 56 μs verifiziert werden. Falls nötig liesse sich mit dem Einsatz eines auf HORS basierenden Signaturschemas der Signierungsaufwand weiter reduzieren. Mit einer auf RSA basierenden Lösung liesse sich weder diese Signaturgrösse noch der geringe Aufwand zum Erstellen und Verifizieren der Signaturen erreichen.

Die Sicherheit von Signaturen erfordert eine gewisse Anzahl SEALs, deren Länge nicht beliebig klein gewählt werden kann. Die resultierende Signatur ist zu gross, um jedes Video-Frame einzeln zu signieren. Hash-Listen sind nötig, um die Signaturgrösse pro Frame zu verringern. Die Folge davon ist eine kurze Verzögerung zum Sammeln der Frames einer Hash-Liste sowie der Verteilung der Signatur vor den eigentlichen Frames. Die Signaturgrösse wird um die Hashes der Frames und ein Salt erhöht. Sie machen einen grossen Teil der Signatur aus. Damit die Signaturen zusammen mit eigentlichen Frames verschickt und dadurch zusätzliche Header eingespart werden können, sollte die Grösse der Signatur 100 B nicht überschreiten. Dies kann erreicht werden, wenn sich mehrere Frames einen Hash in der Hash-Liste teilen. Damit zusammengehörende Frames gleichzeitig eintreffen und dadurch keine Verzögerung bei der Authentifizierung entsteht, muss das Pulsar Protokoll geringfügig modifiziert werden.

Die Zuverlässigkeit des Pulsar Protokolls kommt dem Integritätsprotokoll zu Gute. Da Datenblöcke kaum verloren gehen, kann die Erweiterung A des BiBa Broadcast-Authentifizierungsprotokolls eingesetzt werden, was sowohl für Sender als auch Empfänger weniger Aufwand bedeutet. Die Signaturen sind im Vergleich zur Basisversion nur unbedeutend grösser. Da es einem Angreifer kaum möglich ist Pakete zu verzögern, kann auf eine genaue Überprüfung des Paketzeitplanes verzichtet werden. Für den Fall, dass doch einmal Pakete verloren gehen oder ein starker Angreifer verspätete Signaturen ausnutzen kann, stellen nach kurzer Zeit die Index-Updates den störungsfreien Betrieb sicher. Sie ermöglichen auch dezentrales Bootstrapping, was mit einer Publik-Key-Grösse von 32 KB sehr willkommen ist. Die Skalierbarkeit des Pulsar Protokolls wird durch das Integritätsprokokoll nicht beeinträchtigt.

Im Gegensatz zu RSA Signaturen haben die in dieser Arbeit diskutierten Signaturschemata diverse Parameter. Da sich viele davon gegenseitig beeinflussen, sind beim Entwurf des Integritätsprotokolls verschiedene Zielkonflikte deutlich geworden. Eine optimale Signatur, welche alle Wunschkriterien erfüllt, konnte daher nicht gefunden werden. Es besteht dafür

aber die Möglichkeit, die Signaturen je nach Gewichtung der festgelegten Ziele an eine spezifische Anwendung anzupassen. Diese Flexibilität führt jedoch auch zu einem grösseren Entwurfsaufwand. Die je nach Signaturschema und Anzahl SEALs pro Signatur unterschiedliche Mindestlänge von SEALs ist nur ein Beispiel dafür. Abschliessend stelle ich fest, dass sich die auf BiBa basierenden One-Time-Signaturen gut für die Senderauthentifizierung im Pulsar Protokoll eignen. Auch für weitere Livestreaming-Protokolle kann deren Einsatz aufgrund des geringen Signierungs- und Verifikationsaufwandes sinnvoll sein. Für Anwendungen, welche einen grösseren Berechnungsaufwand zulassen, sollten hingegen andere Lösungen wie QUARTZ, McEliece oder DSA geprüft werden.

In künftigen Arbeiten kann versucht werden die Signaturgrössen weiter zu reduzieren. Die in Abschnitt 3.5.1 gemachten Optimierungen bieten einen Anhaltspunkt. Es ist davon auszugehen, dass kleinere Signaturen gefunden werden können. Interessant ist, wie gross die Kosten dafür sein werden. Weiter kann die Anfälligkeit für andere Angriffsarten wie z.B. Eclipse-Attacken untersucht werden. Angriffe auf die Integrität sind nicht die einzigen Möglichkeiten die Verfügbarkeit zu schwächen.

Literaturverzeichnis

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73, New York, NY, USA, 1993. ACM Press.

[JM96] Heeralal Janwa and Oscar Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Des. Codes Cryptography*, 8(3):293–307, 1996.

[LMSW07] Thomas Locher, Remo Meier, Stefan Schmid, and Roger Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. In *DISC*, volume 4731 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2007.

[MP02] Michael Mitzenmacher and Adrian Perrig. Bounds and Improvements for BiBa Signature Schemes. Technical Report TR-02-02, Harvard University, Cambridge, Massachusetts, 2002.

[MVO97] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[OGM86] Shafi Goldwasser Oded Goldreich and Silvio Micali. How to Construct Random Functions. *J. ACM*, 33(4):792–807, 1986.

[PCG01a] Jacques Patarin, Nicolas Courtois, and Louis Goubin. FLASH, a Fast Multivariate Signature Algorithm. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 298–307, London, UK, 2001. Springer-Verlag.

[PCG01b] Jacques Patarin, Nicolas Courtois, and Louis Goubin. QUARTZ, 128-Bit Long Digital Signatures. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 282–297, London, UK, 2001. Springer-Verlag.

[PD03] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, San Francisco, CA, USA, 3rd edition edition, 2003.

[Per01] Adrian Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 28–37, Philadelphia PA, USA, Nov. 2001.

[RR02] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In *ACISP '02: Proceedings of the 7th Australian Conference on Information Security and Privacy*, pages 144–153, London, UK, 2002. Springer-Verlag.

[RS98] Martin Raab and Angelika Steger. “Balls into Bins” — A Simple and Tight Analysis. In *RANDOM ’98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170, London, UK, 1998. Springer-Verlag.

[SNDW06] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM 2006: 25th IEEE International Conference on Computer Communications*, pages 1–12, Barcelona, Spain, April 2006.

A. Anhang

A.1. Performance

Operation	RSA		DSA	
	1024 bit	768 bit	1024 bit	768 bit
Schlüsselerzeugung	770.1 ms	317.3 ms	11.3 ms	6.5 ms
Signierung	18.2 ms	8.1 ms	11.0 ms	6.4 ms
Verifikation	1.1 ms	0.6 ms	21.9 ms	12.9 ms

Tabelle A.1.: Performance einer RSA/DSA Operation auf einem Laptop mit einem Intel Pentium 4 (2.2 Ghz). Signierung und Verifikation wurden über 50'000 Datenblöcke gemittelt, Schlüsselerzeugung über 50 Schlüsselpaare.

L	MD5	SHA-1
20 B	1.4 μs	1.8 μs
503 B	9.2 μs	13.0 μs
1328 B	22.4 μs	33.4 μs

Tabelle A.2.: Performance einer MD5/SHA-1 Operation auf einem Laptop mit einem Intel Pentium 4 (2.2 Ghz). Die Datenblöcke hatten eine Länge von L Bytes. Gemittelt wurde über 10^6 Datenblöcke.