INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ein Verfahren zur passwortlosen Authentifizierung

Mirko Oleszuk

Bachelorarbeit

Beginn der Arbeit:	16. Juli 2013
Abgabe der Arbeit:	16. Oktober 2013
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Jörg Rothe

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 16. Oktober 2013

_____

Mirko Oleszuk

Zusammenfassung

Die vorliegende Arbeit stellt ein Verfahren zur passwortlosen Authentifizierung von Benutzern einer Webanwendung vor.

Die Grundlage für die Authentifizierung bildet das asymmetrische RSA-Kryptosystem. Aufgrund der Eigenschaft der Asymmetrie besteht eine eindeutige Zuordnung zwischen verschlüsselten Daten und dem Schlüsselpaar, mit dem die Daten verschlüsselt wurden. Passwörter lassen sich bei der Authentifizierung in Webanwendungen vermeiden, da eine Webanwendung einen Benutzer anhand seines öffentlichen Schlüssel durch signierte Noncen oder verschlüsselte Cookies authentifizieren kann.

Diese Arbeit zeigt zudem Schwächen der passwortbezogenen Authentifizierung auf und analysiert bereits vorhandene Alternativen auf Ihre Sicherheit und Einsatzfähigkeit.

Die Beschreibung der Entwicklung des Verfahrens wird von Erläuterungen zu einigen Angriffstechniken begleitet und die jeweiligen Designentscheidungen und Weiterentwicklungen dadurch begründet.

Die Implementierung des Verfahrens wird anhand eines Clientplugins für den Browser Chromium und anhand eines Serverplugins für die Blogsoftware Wordpress beschrieben. Es werden die einzelnen Kommunikationsschritte ausführlich dargestellt. Die Plugins wurden generisch geschrieben, sodass eine Portierung auf andere Systeme keinen großen Aufwand bedeutet.

In der Evaluation der Arbeit wird gezeigt, dass die Laufzeit des Verfahrens, mit einem 2048 Bit langen Schlüssel, ca. 13 Sekunden beträgt. Damit ist das Verfahren sicher und zugleich performant.

Somit ist das im Rahmen dieser Arbeit entwickelte Verfahren zur passwortlosen Authentifizierung eine sichere Alternative zur klassischen Authentifizierung via Passwort. Es funktioniert sicher über eine unverschlüsselte Verbindung, sodass es in eingeschränkten Umgebungen eingesetzt werden kann. Der Benutzer benötigt zur Installation keine besonderen Rechte und auch keine andere Software als den Browser und die Webanwendung selbst.

Inhaltsverzeichnis

1 Einleitung

Täglich wird man im Internet nach Benutzernamen und Passwörtern gefragt, um sich damit zu authentifizieren. Häufig werden einfache und kurze Passwörter gewählt, da diese leichter zu merken und schneller einzugeben sind. Dabei wird aus Bequemlichkeit das Risiko eingegangen, dass ein Angreifer ein solches unsicheres Passwort beispielsweise durch Ausprobieren erraten kann. Oft wird zudem dasselbe Passwort für mehrere Webanwendungen gleichzeitig benutzt, da es nicht komfortabel ist, sich viele Passwörter zu merken.

Programme zur Verwaltung von Passwörtern helfen einem Benutzer, den Überblick über viele verschiedene Passwörter zu behalten. Doch auch diese Hilfsprogramme helfen nicht gegen Phishing-Angriffe, die den Benutzer täuschen und das Passwort erfragen.

Viele Webanwendungen speichern zudem Passwörter im Klartext oder nur schwach verschlüsselt ab und immer wieder erfährt man von Angriffen, bei denen Kriminelle Zugangsdaten der Benutzer kopiert haben. So geschah es zum Beispiel im Oktober 2013 bei Adobe [spi13], im Monat davor bei Vodafone [hei13] und bereits vielen anderen zuvor.

Deswegen ist ein Ziel dieser Arbeit, ein Authentifizierungsverfahren zu entwickeln, welches nicht durch einen Einbruch auf Seiten des Servers kompromittiert werden kann.

Vorhandene Alternativen wie zum Beispiel SSL-Client-Zertifikate bieten zwar eine sichere Authentifizierung an, allerdings müssen die Zertifikate von einer sogenannten Trusted Authority signiert sein, da sonst Warnmeldungen des Browsers eingeblendet werden. Solch ein signiertes Zertifikat ist in der Regel nicht kostenlos erhältlich und somit für kleine oder nicht-kommerzielle Webanwendungen ungeeignet. Zusätzlich muss das Zertifikat mit einer Software erstellt werden, die nicht zum Browser gehört. Die Installation des Zertifikats benötigt zudem einen Administrationszugang zum Server der Webanwendung.

Das Ziel dieser Arbeit ist demzufolge, eine Lösung zu entwickeln, die eine sichere Authentifizierung ohne Passwort ermöglicht. Zudem soll das Authentifizierungsverfahren für jeden Betreiber einer Webanwendung auch in eingeschränkten Umgebungen ohne großen Aufwand zur Verfügung stehen.

Der Rest der Arbeit ist wie folgt gegliedert: In Kapitel 2 werden verwandte Arbeiten vorgestellt und analysiert. In Kapitel 3 folgen die theoretischen Grundlagen, sodass in Kapitel 4 die Idee des Verfahrens und seine Entwicklung erläutert werden kann. In Kapitel 5 wird die Implementierung des Verfahrens erläutert. Das Verfahren und seine Implementierung werden in Kapitel 6 hinsichtlich Performance, Sicherheit und Bedienkomfort untersucht. In Kapitel 7 wird ein Fazit gezogen und ein Ausblick auf zukünftige Entwicklungen gegeben.

2 Verwandte Arbeiten

In diesem Kapitel werden einige Plugins vorgestellt, die sich mit Verschlüsselung und Authentifizierung im Browser beschäftigen. Die Plugins werden hinsichtlich ihrer Ziele untersucht und es wird geprüft, ob sie für diese Arbeit verwendet werden können.

2.1 FireGPG

FireGPG [Cuo10] ist ein Plugin, das für den Webbrowser Firefox bis ins Jahr 2010 entwickelt wurde. Es kann Texte im Webbrowser verschlüsseln, signieren, entschlüsseln und verifizieren. Somit kann FireGPG für die Erstellung von verschlüsselten E-Mails bei Webmailanbietern und für die Erstellung und Verifikation von beispielsweise signierten Forenbeiträgen verwendet werden. Das Plugin funktioniert nur, wenn GnuPG als native Anwendung installiert ist und dient dann als graphischer Wrapper für dessen Funktionen.

In der Standardkonfiguration ist die Arbeitsweise von FireGPG nicht sicher, da man erst den Text in der Webseite schreiben und danach verschlüsseln soll. Während des Schreibens kann ein JavaScript der Webseite den Text jedoch mitlesen und verändern. Details zu diesem Angriff kann man in [web12a] nachlesen. Über die Einstellungen kann man dieses Verhalten ändern und die Ver- und Entschlüsselung in ein separates Fenster des Browsers verlegen, sodass kein externes JavaScript auf den Klartext zugreifen kann.

2.2 WebPG

WebPG [Huf10b] ist ein Plugin für die Webbrowser Firefox und Chrome. Es kann als Nachfolger von FireGPG betrachtet werden, da WebPG auf der Basis von FireGPG weiterentwickelt wurde. WebPG stellt die Funktionen von GnuPG dem Browser über eine mitgelieferte Bibliothek zur Verfügung und ruft nicht wie FireGPG GnuPG direkt auf. WebPG ist mit der selben Methode angreifbar, mit der auch FireGPG angreifbar ist.

Im September 2013 gab das Team hinter dem Chromium-Projekt in [Sch13] bekannt, dass die NPAPI, ein Interface über das WebPG die Bibliotheken anspricht, im Jahr 2014 aus dem Browser entfernt wird. Demzufolge wird WebPG dann nicht mehr funktionieren.

WebPG wird aktiv weiterentwickelt, sodass man erwarten kann, dass die Entwickler eine Lösung für die Einstellung der NPAPI finden werden.

2.3 Monkeysphere

Monkeysphere [web12b] ist ein Plugin für den Webbrowser Firefox und seine Derivate. Das Ziel des Projekts ist es, eine dezentrale Serverauthentifizierung zu ermöglichen. Das Plugin baut auf einer via SSL gesicherten Verbindung auf und meldet sich erst, wenn das Serverzertifikat die Gültigkeitsprüfungen des Browsers nicht besteht. Das Serverzertifikat wird dann mit Hilfe des Web of Trusts (Netz des Vertrauens) überprüft. Dabei wird

die Gültigkeit von Schlüsseln durch gegenseitig signierte Schlüssel und Vertrauen überprüft.

2.4 gpgAuth

gpgAuth [Huf10a] ist ein Entwurf eines Authentifizierungsverfahren auf der Basis von OpenPGP. Das Ziel des Verfahrens ist eine gegenseitige Authentifizierung von Client und Server. Praktisch werden dazu signierte Noncen versendet.

Das beschriebene Verfahren ist allerdings anfällig für DoS-Angriffe, da es vorsieht, dass der Server, ohne weitere Überprüfung, die empfangenen Daten des Clients mit dessen öffentlichem Schlüssel verifiziert. Danach generiert der Server eine Nonce und signiert diese mit seinem privaten Schlüssel.

Zum jetzigen Zeitpunkt gibt es keine funktionierende Implementierung des Verfahrens.

2.5 Fazit

Wir sehen, dass keines der oben beschriebenen Plugins eine passwortlose Authentifizierung via HTTP ermöglicht. Monkeysphere arbeitet mit HTTPS und löst somit nicht unser Authentifizierungsproblem, da es unser Ziel ist, eine sichere Authentifizierung via HTTP anzubieten. FireGPG und WebPG zeigen uns, dass Verschlüsselung im Browser möglich ist, aber insbesondere zeigen diese Plugins, dass es triviale Angriffe gibt und dass man nicht nur das Plugin gegen fremden Zugriff schützen muss, sondern auch die Arbeitsweise so absichern muss, dass die Daten des Benutzers stets geschützt sind.

Die vorgestellten Plugins sind entweder nicht mehr aktuell oder verfolgen andere Ziele, die somit das Problem der passwortlosen Authentifizierung nicht lösen, sodass ein neues Verfahren notwendig ist.

3 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen, die im Rahmen dieser Arbeit verwendet werden, erläutert und anhand von Beispielen erklärt.

3.1 RSA-Kryptosystem

RSA [Rot08] ist ein Kryptosystem, mit dem Daten verschlüsselt und signiert werden können. Der Name setzt sich aus den Anfangsbuchstaben der Nachnamen der drei Entwickler Ronald Rivest, Adi Shamir und Leonard Adleman zusammen. Es arbeitet asymmetrisch, das heißt jeder Benutzer dieses Systems besitzt ein Schlüsselpaar, das aus einem öffentlichen und einem privaten Schlüssel besteht. Es gehört daher zur Gruppe der Public-Key-Kryptosysteme.

Wenn die Benutzerin Alice Daten sicher an Bob schicken möchte, so verschlüsselt sie diese mit dem öffentlichen Schlüssel von Bob. Nur der Besitzer des dazugehörigen privaten Schlüssels – also im besten Fall nur Bob selbst – kann diese verschlüsselten Daten wieder entschlüsseln.

Mit dem öffentlichen Schlüssel von Bob ist jeder nun in der Lage ihm verschlüsselte Daten zu schicken.

Die Identität des Absenders lässt sich mit einer digitalen Signatur nachweisen. Wenn Alice also sicherstellen möchte, dass bestimmte Daten ihr zugeordnet werden sollen, so kann sie diese vor dem Versenden mit ihrem privaten Schlüssel signieren.

Jeder kann nun mit dem öffentlichen Schlüssel von Alice die signierten Daten verifizieren und überprüfen, ob genau diese Daten von Alice signiert und nicht von Dritten verändert wurden.

3.1.1 Schlüsselgenerierung

Für die Schlüsselgenerierung werden zwei verschiedene Primzahlen p und q gewählt. Wir wählen für die folgende Beispielrechnung die Primzahlen $p = 5$ und $q = 11$ aus. Es wird das Produkt der Primzahlen n und $\phi(n)$ berechnet.

$$n = p \cdot q = 5 \cdot 11 = 55$$
$$\phi(n) = (p-1) \cdot (q-1) = 40$$

Nun wird der Exponent $e \in \mathbb{N}$ gewählt für den

$$1 < e < \phi(n) \text{ und } ggT(e, \phi(n)) = 1$$

gilt. Das bedeutet, e darf keine Primfaktoren mit $\phi(n)$ gemeinsam haben. Wir wählen für dieses Beispiel $e = 3$ aus. Das Zahlenpaar $(n, e) = (55, 3)$ bildet nun bereits den öffentlichen Schlüssel.

Der private Schlüssel d lässt sich wie folgt berechnen

$$1 < d < \phi(n) \text{ und } e \cdot d \equiv 1 \mod \phi(n).$$

Wir wählen 27 für dieses Beispiel aus, sodass der private Schlüssel $d = 27$ lautet.

3.1.2 Verschlüsselung mit RSA

Für die Verschlüsselung von Nachrichten ist es notwendig, die Buchstaben in Zahlen zu übersetzen. Zur Vereinfachung werden wir hier die Stelle des Buchstabens im Alphabet als ihren Zahlenwert betrachten, also $A = 1$, $B = 2$ usw. Die Ver- und Entschlüsselungsfunktionen sind wie folgt definiert.

$$E(m) = m^e \mod n \qquad \text{(Verschlüsselung)}$$
$$D(c) = c^d \mod n \qquad \text{(Entschlüsselung)}$$

Der Text „DTH“ wird mit dem öffentlichen Schlüssel $(n, e) = (55, 3)$ wie folgt verschlüsselt.

$$E(4) = 4^3 \mod 55 = 9 \mathrel{\widehat{=}} I$$
$$E(20) = 20^3 \mod 55 = 25 \mathrel{\widehat{=}} Y$$
$$E(8) = 8^3 \mod 55 = 17 \mathrel{\widehat{=}} Q$$

Der Geheimtext „IYQ“ kann nun an den Empfänger übertragen werden. Ohne den zugehörigen privaten Schlüssel lässt sich der Geheimtext nicht in den ursprünglichen Klartext übersetzen. Mit dem privaten Schlüssel $d = 27$ und mit n, aus dem öffentlichen Schlüssel $(n, e) = (55, 3)$, entschlüsselt sich der Text wie folgt.

$$D(9) = 9^{27} \mod 55 = 4 \mathrel{\widehat{=}} D$$
$$D(25) = 25^{27} \mod 55 = 20 \mathrel{\widehat{=}} T$$
$$D(17) = 17^{27} \mod 55 = 8 \mathrel{\widehat{=}} H$$

Das Signieren von Daten erfolgt analog, jedoch mit dem Unterschied, dass jeweils in der Formel zur Ver- und Entschlüsselung die Zahlen e und d vertauscht werden. Die Verschlüsselungsformel wird mit d anstelle von e zum Signieren und die Entschlüsselungsformel wird mit e anstelle von d zum Verifizieren benutzt.

3.1.3 Eindeutigkeit der Verschlüsselung

Wie wir im vorherigen Abschnitt gesehen haben, lässt sich der Geheimtext nur mit dem zugehörigen privaten Schlüssel wieder entschlüsseln. Die Eigenschaft der eindeutigen Zuordnung von dem privaten Schlüssel zum öffentlichen Schlüssel machen wir uns in dieser Arbeit zunutze.

3.1.4 Das Faktorisierungsproblem

Die Sicherheit von RSA beruht auf der Schwierigkeit des Faktorisierungsproblem. Das Faktorisierungsproblem beschreibt die Zerlegung einer Zahl in ihre Primfaktoren. Da es noch keinen Algorithmus gibt, der Primfaktoren gegebener Zahlen effizient berechnen kann, wird RSA als sicher angesehen.

Der Aufwand für das Brechen eines Schlüssels mit der Länge von 1024 Bit wurde 2003 in [ST03] untersucht und für theoretisch möglich befunden, falls man die dafür nötige Hardware, im Wert von ca. 10 Millionen US-Dollar, zur Verfügung hat und diese ca. ein Jahr lang rechnen lassen kann. Demzufolge sollte man nur Schlüssel verwenden, die länger als 1024 Bit sind.

3.2 Proof-of-Work-System

Ein Proof-of-Work (PoW) System ist eine Maßnahme gegen Denial-of-Service (DoS) Angriffe. Das Ziel eines DoS-Angriffs ist die Überlastung eines Dienstes, in dessen Folge der Dienst nicht mehr für legitime Benutzer erreichbar ist. Ein PoW-System stellt sicher, dass ein Benutzer des Dienstes bzw. ein Angreifer genügend Ressourcen eingesetzt hat, bevor der Dienst Ressourcen einsetzt. Es gibt viele verschiedene Implementierungen von PoW-Systemen. Die Gemeinsamkeit von allen besteht aber darin, dass dem Benutzer des Dienstes eine Aufgabe gestellt wird, deren Lösung intensive Berechnungen benötigt. Die Überprüfung der Lösung benötigt hingegen nur sehr wenige Berechnungen.

3.2.1 Time Lock Puzzle

Eine Variante eines PoW-Systems nennt sich Time Lock Puzzle (TLP) [Riv99]. Dabei wählt der Server zufällig zwei verschiedene Primzahlen p und q. Das Produkt der Primzahlen n und eine Zeitslotvariable t, die die Schwierigkeit des Puzzles bestimmt, werden dem Client übergeben. Da dem Client die Primfaktoren p und q nicht bekannt sind, muss dieser sequentiell $2^{2^t} \mod n$ für die Lösung berechnen. Beginnend mit $i = 0$ berechnet der Client $W(i)$ für jedes i, für das gilt $0 \leq i < t$.

$$\begin{aligned} W(0) &= 2 \\ W(i+1) &= (W(i)^2) \mod n \qquad \text{(für } i > 0\text{)} \end{aligned}$$

Der Server hingegen kennt die Primfaktoren und kann das Puzzle effizienter lösen, da

$$\begin{aligned} n &= p \cdot q \\ \phi(n) &= (p-1) \cdot (q-1) \\ s &= 2^{2^t \mod \phi(n)} \mod n \end{aligned}$$

gilt.

Beispiel

Wir wählen die Primzahlen $p = 5$, $q = 11$ und die Zeitslotvariable $t = 3$ als Beispiel für den Server aus. Dann ist

$$\begin{aligned} n &= p \cdot q = 55 \\ \phi(n) &= (p-1) \cdot (q-1) = 40 \\ s &= 2^{2^t \mod \phi(n)} \mod n \\ s &= 2^{2^3 \mod 40} \mod 55 \\ s &= 2^8 \mod 55 \\ s &= 36. \end{aligned}$$

Dem Client werden die Parameter $n = 55$ und $t = 3$ übergeben, sodass dieser für die Lösung

$$\begin{aligned} W(0) &= 2 \\ W(i+1) &= W(i)^2 \mod n \\ W(1) &= W(0)^2 \mod 55 = 2^2 \mod 55 = 4 && \text{(i = 0)} \\ W(2) &= W(1)^2 \mod 55 = 4^2 \mod 55 = 16 && \text{(i = 1)} \\ W(3) &= W(2)^2 \mod 55 = 16^2 \mod 55 = 36 && \text{(i = 2)} \end{aligned}$$

berechnen muss.

Wir sehen hier, dass der Server die Anzahl der Berechnungsdurchläufe und somit die Laufzeit der Berechnung beim Client anhand der Zeitslotvariable t bestimmen kann. In diesem Beispiel ist $t = 3$ gewählt, um die Berechnung zu zeigen. Für Implementierungen muss jedoch ein größeres t gewählt werden, für das $2^t \gg \phi(n)$ gelten muss, damit ein asymmetrischer Rechenaufwand entsteht. Die Laufzeit bei der Erstellung des Puzzles bleibt hingegen bei wachsendem t konstant.

Ein Time Lock Puzzle ist demnach eine effektive Maßnahme gegen DoS-Angriffe.

3.3 Nonce

Eine Nonce („Number used only once“) [KR12] ist eine Zahl, die einen sicherheitsrelevanten Vorgang legitimiert. Die Parameter des Vorgangs werden in die Nonce eingebunden, sodass die Nonce nur für genau den Vorgang, für den sie erstellt wurde, verwendet werden kann. Eine Nonce schützt eine Anwendung auch vor der doppelten Ausführung einer Aktion. So tritt beispielsweise bei einer Bank-Überweisung sofort ein finanzieller Schaden auf, falls diese, fehlerhafterweise, zweimal ausgeführt wird. In der Lebenszeit eines Protokolls wird eine Nonce nur einmal verwendet. Eine Nonce sollte zudem zufällig und nicht vorhersagbar sein.

3.4 HTTP

Das Hypertext Transfer Protocol (HTTP) [FGM+99] ist ein Protokoll, das der Übertragung von Daten dient. Es wird hauptsächlich zur Übertragung von Webseiten vom Server zum Client benutzt. Die Datenübertragung geschieht dabei unverschlüsselt. Die verschlüsselte Übertragung wird Hypertext Transfer Protocol Secure (HTTPS) [Res00] genannt und ist via SSL/TLS möglich. Praktisch ist das nur mit signierten Zertifikaten sinnvoll, die in der Regel kostenpflichtig von Zertifizierungsstellen ausgegeben werden, da sonst der Browser Warnmeldungen ausgibt.

3.4.1 Cookie

Ein Cookie [KM97] ist eine Information einer Webanwendung, die zwischen dem Browser des Benutzers und dem Server der Webanwendung ausgetauscht wird. Die Information wird vom Browser verwaltet und in einer Datei gespeichert, die als Cookie bezeichnet wird. Eine Webanwendung kann beispielsweise Informationen zur Identität eines Benutzers oder benutzerspezifische Einstellungen dort speichern. Diese Informationen werden bei jeder Anfrage des Benutzers an die Webanwendung übertragen. Da HTTP selbst keine Informationen über frühere Anfragen von Benutzern speichert, sind Cookies nötig, um Benutzer zu identifizieren.

4 Der Public-Key-Web-Login

In diesem Kapitel wird das im Rahmen dieser Arbeit entwickelte Verfahren zur passwortlosen Authentifizierung namens Public-Key-Web-Login (PKWL) vorgestellt.

4.1 Idee und Ziel des Verfahrens

Die Idee hinter dem Verfahren basiert auf der eindeutigen Zuordnung von verschlüsselten oder signierten Daten zu einem Schlüssel und dessen Besitzer. Da verschlüsselte oder signierte Daten nur mit dem entsprechenden Schlüssel entschlüsselt bzw. verifiziert werden können, lässt sich damit eine eindeutige Zuordnung zu einem Benutzer schaffen. Dies gilt nur unter der Voraussetzung, dass der private Schlüssel geheim gehalten wird.

Das Ziel des Verfahrens ist, eine passwortlose Authentifizierung für Webanwendungen mit der Hilfe von Public-Key-Kryptographie zu schaffen. Daher lautet der Name des Verfahrens Public-Key-Web-Login.

4.2 PKWL 1.0 - Signierte Nachrichten

In der ersten Version dieses Authentifizierungsverfahren schickt der Client zwei Nachrichten an die Webanwendung. Die erste Nachricht enthält den Benutzernamen im Klartext. Damit identifiziert die Webanwendung das Benutzerkonto und holt sich den gespeicherten öffentlichen Schlüssel des Benutzers. Dieser Schlüssel wird für die zweite Nachricht benötigt, da diese den signierten Benutzernamen enthält. Die Webanwendung entschlüsselt die signierten Daten und vergleicht den entschlüsselten Benutzernamen mit dem im Klartext gesendeten Benutzernamen.

Bei Gleichheit hat sich der Benutzer authentifiziert. Da ein Angreifer die Kommunikation aber belauschen und aufzeichnen könnte, ist diese Authentifizierung nicht sicher. Ein Angreifer wäre in der Lage, der Webanwendung dieselben Daten, die auch ein legitimer Benutzer senden würde, zu übermitteln, und sich damit zu authentifizieren. Dieser Angriffstyp wird Playback- oder Replay-Angriff genannt.

4.3 PKWL 1.1 - Signierte Noncen

Um das Problem der vorherigen Version zu lösen, kann man eine Nonce einführen. Die Nonce wird von der Webanwendung erstellt und vom Client signiert. Da die Nonce in der Lebenszeit des Verfahrens nur einmal verwendet wird, ist ein Playback-Angriff nicht möglich.

Sicher ist diese Version des Verfahrens aber ebenfalls nicht, da ein Angreifer sich als die Webanwendung ausgeben und dem Client eine manipulierte Nonce zuschicken könnte, um diese signieren zu lassen. Eine manipulierte Nonce könnte beispielsweise der Hash einer Nachricht sein, wie sie bei OpenPGP [CDF+07] verwendet wird. Ein Angreifer könnte damit eine signierte Nachricht im Namen des Besitzers des privaten Schlüssels schreiben, ohne den Schlüssel zu besitzen.

Ein weiteres Sicherheitsproblem ist ein Denial-of-Service (DoS) Angriff. Ein Angreifer kann die Webanwendung mit Anfragen überhäufen, sodass diese legitimen Benutzern nicht mehr antworten kann.

4.4 PKWL 2.0 - Proof-of-Work-System mit Signaturen

Die Lösung des Problems von manipulierten Noncen geht mit der Lösung des Problems von DoS-Angriffen einher. Wir betrachten zunächst die Lösung für DoS-Angriffe.

Wie in Abschnitt 3.2 beschrieben, lassen sich DoS-Angriffe mit einem Proof-of-Work (PoW) System effizient eindämmen.

Das Verfahren wird daher, um das in Abschnitt 3.2.1 beschriebene Time Lock Puzzle (TLP), erweitert, sodass die Webanwendung nun bei einer eingehenden Anfrage des Clients ein TLP erzeugt und dem Client die Aufgabenstellung übermittelt.

Die Lösung des TLP, die der Client berechnet, ist in unserer Implementierung eine Dezimalzahl. Diese Dezimalzahl wird mit dem privaten Schlüssel signiert und dient als Nonce.

Das Problem eines manipulierten Hashwerts, der signiert wird, ist damit gelöst, da keine Hashfunktion bekannt ist, die eine ca. 20-stellige Dezimalzahl als Hashwert einer Nachricht generiert. Falls doch, so könnte man durch Einfügen eines Salt-Werts die Wahrscheinlichkeit einer Kollision reduzieren.

Nehmen wir an, dass dieses Problem nun gelöst sei, so können wir daraus schlussfolgern, dass die Webanwendung den Client nun authentifizieren kann und dabei vor DoS-Angriffen geschützt ist.

Es gibt aber noch eine weitere Angriffsmethode, die davon abhängig ist, wie die Webanwendung die Authentifizierung des Clients verwaltet. Die meisten Webanwendungen setzen dafür Cookies ein. Im letzten Schritt des Verfahrens wird dieser Cookie im Klartext an den Client geschickt und kann somit von jedem, der sich zwischen dem Client und dem Server der Webanwendung befindet, gelesen werden. Dieser Angriffstyp heißt Man-in-the-middle-Angriff.

Diese Version des Verfahrens ist via HTTPS dennoch sicher, da solch eine Verbindung gegen Man-in-the-middle-Angriffe geschützt ist.

4.5 PKWL 3.0 - Verschlüsselte Cookies

Ein Ziel unseres Verfahrens ist, ohne SSL auszukommen, daher folgt in diesem Abschnitt eine Lösung, die Man-in-the-middle-Angriffe erfolgreich abwehrt und somit eine sichere Authentifizierung des Clients via HTTP bietet.

Die Lösung für das Problem des Cookies im Klartext ist einfach. Wir verschlüsseln den Cookie mit dem öffentlichen Schlüssel des Benutzers. Somit kann nur der Besitzer des privaten Schlüssels diesen entschlüsseln.

Diese Änderung führt dazu, dass der Client die Lösung des TLP nicht mehr signieren muss, da die Authentifizierung des Clients nun durch die erfolgreiche Entschlüsselung

des Cookies gegeben ist. Wir könnten den Client auch weiterhin, das TLP signieren lassen, jedoch würde das nicht die Sicherheit steigern, sondern lediglich die Dauer des Verfahrens erhöhen.

In den folgenden Abschnitten werden die einzelnen Schritte des Verfahrens detailliert erklärt.

4.5.1 Vorbereitung

Damit das Verfahren vollständig ablaufen kann, sind einige client- und serverseitige Vorbereitungen zu treffen.

Der Client muss den Benutzernamen und den privaten Schlüssel sowie den Hostnamen des Servers der Webanwendung, bei der der Client authentifiziert werden soll, kennen.

Die Webanwendung muss den öffentlichen Schlüssel des Benutzers gespeichert haben.

4.5.2 Ablauf

Das PKWL-Verfahren besteht aus fünf Schritten. Die einzelnen Schritte laufen, wie in Abbildung 1 zu sehen, nacheinander jeweils abwechselnd auf der Seite des Clients und des Servers ab.

Das Clientplugin führt die Schritte eins (Authanfrage), drei (Puzzlelösung) und fünf (Cookieentschlüsselung) aus. Die Schritte zwei (Puzzlegenerierung) und vier (Puzzleverifizierung und Cookieverschlüsselung) führt das Serverplugin aus. Es folgen nun die Schritte des Verfahrens in chronologischer Reihenfolge.

1. Authanfrage

Das Verfahren beginnt mit einer Anfrage an die Webanwendung zur Authentifizierung des Clients. Dazu wird der Benutzername an die Webanwendung übertragen.

2. Puzzlegenerierung

Das Serverplugin prüft, ob zu dem empfangenen Benutzernamen ein Benutzerkonto existiert und ob ein öffentlicher Schlüssel hinterlegt ist. Falls dem so ist, so wird geprüft, ob ein Time Lock Puzzle für diesen Benutzernamen vorhanden ist.

Falls kein TLP für diesen Benutzer vorhanden ist, so erstellt das Serverplugin ein neues und speichert die Aufgabenstellung und die Lösung des TLP zusammen mit dem Benutzernamen und dem aktuellen Zeitstempel ab. Danach wird die Aufgabenstellung des TLP dem Client zugeschickt.

Falls für diesen Benutzernamen bereits ein TLP vorhanden ist, so wird die Gültigkeit dessen anhand des gespeicherten Zeitstempels und des aktuellen Zeitstempels berechnet. Falls es noch gültig ist, so wird das gespeicherte TLP an den Client ausgeliefert.

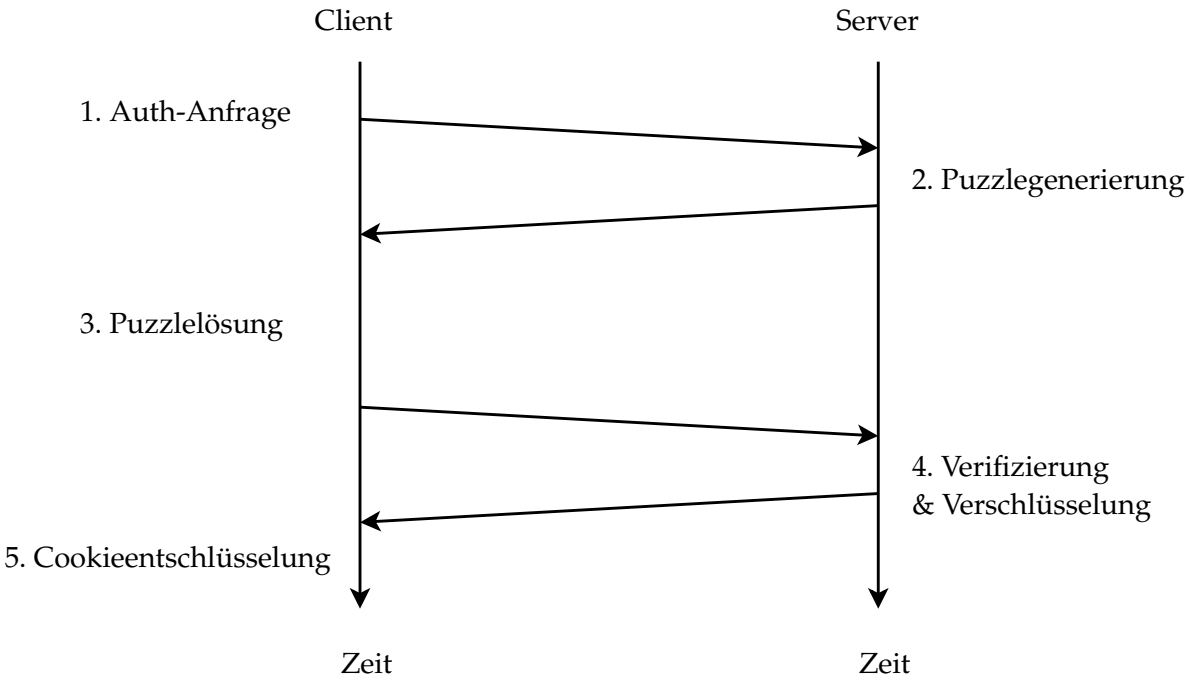

Abbildung 1: Schematische Darstellung des Verfahrens.

Falls also mehrere Anfragen mit demselben Benutzernamen die Webanwendung innerhalb der Lebenszeit eines Time Lock Puzzles erreichen, so bewirkt nur die erste Anfrage die Erstellung eines TLP. Jede spätere Anfrage, die innerhalb der Lebenszeit des TLP die Webanwendung erreicht, empfängt das für diesen Benutzernamen gespeicherte TLP.

Diese Vorgehensweise verhindert, dass ein Angreifer einen Authentifizierungsvorgang eines Clients stören kann, da ein gültiges TLP nicht durch eine erneute Anfrage überschrieben wird und auch das Speichermedium (beispielsweise die Datenbank) nicht mit Time Lock Puzzles geflutet wird.

3. Puzzlelösung

Der Client berechnet die Lösung für das TLP und schickt diese zusammen mit seinem Benutzernamen an die Webanwendung.

4. Puzzleverifizierung und Cookieverschlüsselung

Das Serverplugin vergleicht die empfangene Lösung mit der gespeicherten Lösung. Falls der Client das Puzzle erfolgreich gelöst hat, ist der Proof-of-Work sichergestellt.

Daraufhin wird ein Cookie für den Benutzer erstellt. Der Cookie wird mit dem öffentlichen Schlüssel des Benutzers verschlüsselt, damit nur der legitime Benutzer den Cookie entschlüsseln kann. Der verschlüsselte Cookie wird schließlich an den Client geschickt.

Das gerade gelöste TLP wird nicht aus der Datenbank gelöscht, da dieses auch von einem Angreifer gelöst werden kann. Ein Löschen würde eine legitime Anfrage eines Benutzers, die nur kurze Zeit nach einer Anfrage eines Angreifers einging oder von einem Client, der weniger Rechenleistung zur Verfügung hat und das TLP langsamer löst, blockieren.

Es werden lediglich alle abgelaufenen Time Lock Puzzles aus der Datenbank gelöscht.

5. Cookieentschlüsselung

Der Client entschlüsselt die empfangenen Daten mit seinem privaten Schlüssel und speichert diese in einem Cookie ab. Ab diesem Zeitpunkt kann sich der Client gegenüber der Webanwendung mit dem Cookie authentifizieren.

5 Implementierung

In diesem Kapitel wird die Implementierung des Public-Key-Web-Logins (PKWL) beschrieben.

Die Implementierung spaltet sich in das clientseitige Browserplugin und in das serverseitige Plugin für die Webanwendung auf.

Das Clientplugin wurde für den Browser Chromium und das Serverplugin für die Blogsoftware Wordpress entwickelt. Die Wahl, für welche konkrete Umgebung die Software entwickelt werden sollte, fiel auf diese beiden Projekte, da beide in Ihrem Bereich weltweit zu den Marktführern im Hinblick auf die Anzahl ihrer Benutzer gehören (bei Chromium zusammen mit dem Derivat Chrome). Zudem wird von beiden Umgebungen ein Application-Programming-Interface (API) angeboten, das die Einbindung von Plugins ermöglicht.

Beide Plugins sind generisch geschrieben, sodass eine Anpassung für andere Browser und Webanwendungen keinen großen Aufwand erfordert, da beispielsweise der Browser Firefox einen ähnlichen Aufbau für Plugins verwendet.

5.1 Serverplugin

Das Serverplugin bildet die Schnittstelle zwischen dem Clientplugin und der Webanwendung. Es kommuniziert mit dem Clientplugin und der Datenbank der Webanwendung, um den Benutzer zu authentifizieren. Dieser Abschnitt erläutert die Einbindung in die Webanwendung Wordpress sowie die Kommunikation mit dem Clientplugin.

5.1.1 Datenbankschema

Das Serverplugin muss den öffentlichen Schlüssel des Benutzers für die Verschlüsselung der Authentifizerungs-Cookies, eine endliche Menge von Primzahlen für die Erstellung eines Time Lock Puzzles (TLP) und die Parameter des TLP speichern können. Die Webanwendung Wordpress benutzt für ihre Daten das relationale Datenbanksystem MySQL zu dem auch Plugins einen Zugang haben, sodass es sich anbietet dieses Datenbanksystem für die Speicherung der oben genannten Daten zu benutzen.

Um die dritte Normalform zu wahren, würde man die Tabelle der Benutzer bei der Installation um eine Spalte für den öffentlichen Schlüssel erweitern. Das ist möglich, aber die Entwickler von Wordpress empfehlen die Benutzung von sogenannten Meta-Keys. Meta-Keys dienen der Speicherung von benutzerbezogenen Daten und werden in einer eigenen Tabelle „usermeta“ gespeichert, die eine fortlaufende ID für jeden Meta-Key, einen Fremdschlüssel zum Benutzerkonto sowie die eigentlichen Daten, bestehend aus Name und Wert, speichern. Die 3NF bleibt dadurch erhalten und die Datenbankstruktur von Wordpress wird nicht verändert. Ein weiterer Vorteil ist, dass in der Meta-Keys-Tabelle nur Einträge für Benutzer des Plugins erstellt werden und keine leeren Einträgt angelegt werden, wie es bei einer zusätzlichen Spalte in der Benutzertabelle der Fall wäre.

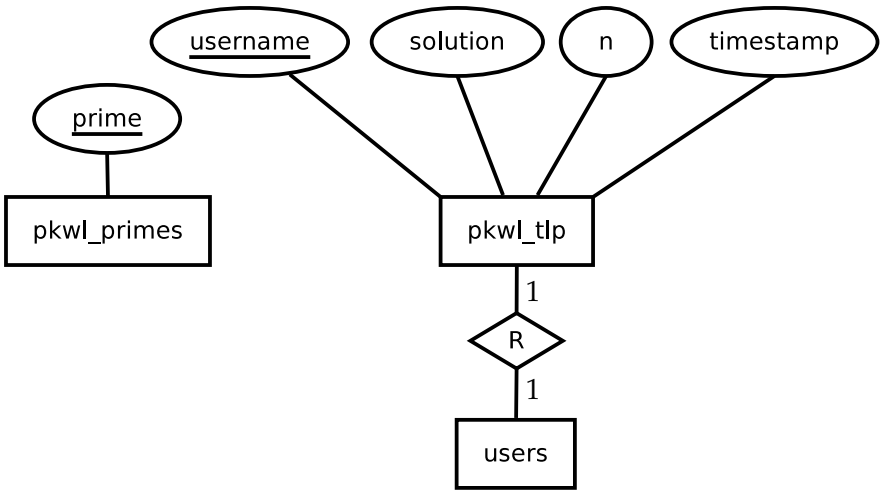

Abbildung 2: Die Datenbankstruktur des Serverplugins im ER-Modell. Die Entität „users“ gehört zur Datenbankstruktur von Wordpress.

5.1.2 Installation

Die Installation des Serverplugins erfolgt über das Administrations-Backend von Wordpress. Das Serverplugin wird dort über den Browser hochgeladen und lässt sich sofort aktivieren. Bei der Aktivierung erstellt das Serverplugin alle nötigen Datenbanktabellen. Zusätzlich sucht das Serverplugin innerhalb weniger Sekunden 1000 Primzahlen für das Time Lock Puzzle und speichert diese in der Datenbank ab. Der Administrator der Webanwendung kann weitere Primzahlen hinzufügen lassen. Eine größere Menge von Primzahlen führt zu einer größeren Menge möglicher Time Lock Puzzles.

Die Meta-Key-Einträge werden für jeden Benutzer einzeln über die Wordpress-API angelegt, wenn dieser seinen öffentlichen Schlüssel einträgt.

Es ist keine manuelle serverseitige Konfiguration nötig. Bei der Deinstallation, die über das Backend von Wordpress möglich ist, werden die Datenbanktabellen des Plugins entfernt.

5.1.3 Lauschen auf Auth-Anfragen

Über die Wordpress-API lässt sich eine Funktion des Plugins auswählen, die bei jedem Seitenaufruf ausgeführt wird. Diese Funktion prüft dann, ob die HTTP-POST-Parameter $pkwl\_action$ und $pkwl\_username$ gesetzt sind und erstellt entweder ein Time Lock Puzzle oder falls zusätzlich der Wert $pkwl\_solution$ gesetzt ist, prüft es die empfangene Puzzlelösung.

Falls eine Auth-Anfrage eingeht – die Parameter $pkwl\_action$ und $pkwl\_username$ sind dabei gesetzt, nicht aber $pkwl\_solution$ – wird zunächst überprüft, ob der empfangene Benutzername zu einem gültigen Benutzerkonto gehört und ob dieses Benutzerkonto einen öffentlichen Schlüssel hinterlegt hat.

Falls bereits ein Time Lock Puzzle für diesen Benutzer vorhanden ist, so wird dieses dem Client zugeschickt. Da der Benutzername in der Tabelle der Time Lock Puzzles der Primärschlüssel ist, gibt es pro Benutzer immer nur genau ein TLP. Falls keines vorhanden ist oder ein bereits erstelltes TLP älter als 60 Sekunden ist, so wird ein neues TLP, wie in Kapitel 3.2 beschrieben, erstellt.

Für die Zeitslotvariable gilt $t = 50000$. Dieser Wert wurde so gewählt, da damit keine unnötig große Verzögerung beim Ablauf des Verfahrens entsteht, aber ein genügend großer Rechenaufwand beim Client bzw. Angreifer für die Berechnung der Lösung nötig ist, sodass ein Angreifer deutlich mehr Aufwand betreiben muss als der Server.

Die Parameter des TLP werden zu einem String konkateniert und, wie in Abbildung 4 zu sehen, mit einem Semikolon getrennt und anschließend dem Client zugeschickt.

5.1.4 Lauschen auf Puzzlelösungen

Falls der Parameter $pkwl\_solution$ gesetzt ist, wird der vierte Schritt des Verfahrens, die Puzzleverifizierung und Cookieverschlüsselung, ausgeführt.

In der Tabelle der Time Lock Puzzles wird anhand des Benutzernamens das zugehörige Puzzle gesucht und die gespeicherte Lösung mit der übermittelten verglichen. Zusätzlich wird anhand des Zeitstempels die Gültigkeit überprüft. Falls die Puzzlelösung gültig ist, so wird über die Wordpress-API der Authentifizierungs-Cookie erstellt.

Normalerweise werden Cookies im Klartext über HTTP an den Client geschickt, worauf der Browser den Cookie speichert. Wir verschlüsseln allerdings den Inhalt des Cookies mit dem öffentlichen Schlüssel des Benutzers, damit niemand den Cookie mitlesen und sich nur der Besitzer des privaten Schlüssels mit diesem Cookie authentifizieren kann.

Die einzelnen Parameter des Cookies werden zu einem String konkateniert und jeweils durch ein Komma getrennt. Falls es mehrere Cookies gibt (bei Wordpress werden drei Cookies zur Authentifizierung gesetzt), so werden diese einzelnen Cookiestrings ebenfalls konkateniert und, wie in Abbildung 3 zu sehen, durch ein Semikolon getrennt.

```
1 Cookie1,Wert1,Ablaufdatum1;Cookie2,Wert2,Ablaufdatum2;
```

Abbildung 3: Der Cookiestring als Klartext

Dieser String wird dann mit dem öffentlichen Schlüssel des Benutzers verschlüsselt.

Die Verschlüsselung funktioniert im Detail so, dass dieser String nun in mehrere Teile zerlegt wird. Die Länge jedes Teils wird durch die Größe von n bestimmt. Ein RSA-Schlüssel, der beispielsweise 2048 Bit lang ist, kann nur maximal 2048 Bit Daten auf einmal verschlüsseln. Jedes Zeichen eines Strings benötigt 8 Bit Speicher, daher darf der zu verschlüsselnde String maximal $2048 \div 8 = 256$ Zeichen lang sein. Ein längerer String muss demnach in mehrere Teilstrings mit maximal 256 Zeichen zerlegt werden, die dann jeweils einzeln verschlüsselt werden müssen.

Die Länge der Teilstrings lässt sich in unserer Implementierung anhand des gespeicherten öffentlichen Schlüssel einfach berechnen. Der gespeicherte Schlüssel ist in hexadezi-

maler Darstellung als String abgespeichert, daher reicht es die Länge dieses Strings ohne führendes „0x“ durch zwei zu dividieren, um die maximale Länge der Teilstrings zu erhalten, die mit diesem Schlüssel verschlüsselt werden können. Dadurch unterstützt der Server nicht nur eine bestimmte Schlüssellänge, sondern jede.

Die einzelnen Teilstrings des Cookieklartexts werden mit dem öffentlichen Schlüssel des Benutzers verschlüsselt. Danach werden die Geheimtexte ebenfalls konkateniert, durch Semikolons getrennt und dem Client zugeschickt.

Nach dem Verschlüsseln folgt noch eine Anfrage an die Datenbank, alle abgelaufenen Time Lock Puzzles zu löschen. Das aktuell benutzte TLP wird noch nicht gelöscht, da es einerseits noch gültig ist und andererseits von jedem gelöst werden kann, also auch von einem Angreifer. Dadurch, dass es nicht gelöscht wird, kann es noch innerhalb seiner Lebenszeit von dem legitimen Benutzer gelöst werden.

5.2 Clientplugin

Das Clientplugin verwaltet die Schlüssel und Benutzernamen der Webanwendungen. Es ermöglicht den Import und die Generierung neuer Schlüssel. Um den Benutzer zu authentifizieren, kommuniziert es mit dem Serverplugin. In diesem Abschnitt werden das eingesetzte Datenbanksystem und die Kommunikation mit dem Server erläutert.

5.2.1 Datenbanksystem IndexedDB

Für die Speicherung der Schlüssel, der Benutzernamen und der Zuordnung zu den Webanwendungen bietet sich ein Datenbanksystem an. Die HTML5 Spezifikation stellt die IndexedDB [W3C13a] als clientseitiges Datenbanksystem zur Verfügung. Dabei handelt es sich um eine Key-Value-Datenbank. Die Unterschiede zu einem relationalen Datenbanksystem wie MySQL bestehen darin, dass keine Verknüpfungen von Tabellen möglich sind und es keine starren Spalten gibt, die nur einen bestimmten Datentyp speichern können. Stattdessen speichert die IndexedDB Objekte ab. Ein Objekt kann beliebig viele verschiedene Datentypen enthalten, beispielsweise einen Integer und ein Array von Strings. Daher spricht man hier auch nicht von Tabellen, sondern von sogenannten „Object-Stores“. Der Zugriff erfolgt über ein asynchrones Application-Programming-Interface (API), das der Browser bereitstellt. Die Datensätze können anhand des Keys oder beliebiger Indizes gesucht werden.

Für das PKWL-Verfahren ist es notwendig die Schlüssel, die Benutzernamen und die Hostnamen der Webanwendungen zu speichern. Die Zahlen der Schlüssel werden in hexadezimaler Darstellung als String gespeichert, da JavaScript Zahlen – in der Größenordnung wie sie für RSA benötigt werden – nicht nativ verarbeiten kann. Um dieses Problem bei der Ver- und Entschlüsselung zu lösen, wird eine BigInteger-Library eingesetzt. Die Hostnamen der Webanwendungen und die Benutzernamen sind ebenfalls Strings.

Der Key-Wert von jedem Datensatz ist ein Integer, der vom Datenbanksystem verwaltet wird und für jeden Datensatz im Object-Store um eins erhöht wird. Der Object-Store „hosts“ speichert den Hostnamen der Webanwendung, den Benutzernamen und den Key-Wert des Schlüssels ab. Die Schlüssel sind in dem „keys“-Object-Store abgelegt. Ein

Object-Store	Key	Value
hosts	Key	{hostname, username, $keys.Key$}
keys	Key	{name, n, e, d, p, q}

Tabelle 1: Datenbankstruktur des Clientplugins

Schlüssel besteht aus einem Namen, der nur für eine einfache Identifizierung benötigt wird und aus den Zahlen n, e, d, deren Bedeutung man in Kapitel 3.1 nachlesen kann. Wir speichern zusätzlich p und q ab, weil zukünftige Implementierungen diese Zahlen für einen schnelleren Entschlüsselungsalgorithmus nutzen können (siehe Kapitel 7.1).

Zusätzlich wird ein Index auf den Werten von $hostname$ in der „hosts“-Tabelle verwaltet, damit beim Öffnen des Plugins schnell angezeigt werden kann, ob es für eine Webanwendung einen Eintrag gibt.

Im Gegensatz zu relationalen Datenbanksystemen ist es nicht möglich, Object-Stores über gemeinsame Attribute bzw. Spalten zu verknüpfen. Zum Start des Verfahrens sind somit zwei Datenbankabfragen nötig, da die erste Abfrage anhand des Hostnamens der Webanwendung und des Benutzernamens in dem „hosts“-Object-Store den Key-Wert des benötigten Schlüssels sucht und bei einem erfolgreichen Treffer anhand dieses Key-Werts den benötigten Schlüssel selbst in dem „keys“-Object-Store sucht.

Diese zwei Datenbankabfragen sind nötig, da wir Datensätze aus zwei verschiedenen Object-Stores verknüpfen wollen. Eine andere Datenbankstruktur, in der man nur einen Object-Store benötigt, der als $Value$ Hostnamen, Benutzernamen und Attribute des Schlüssels speichert, wäre eine mögliche Alternative. Diese Alternative hat jedoch den Nachteil der mehrfachen Speicherung desselben Schlüssels.

In der Evaluation wird man zudem sehen, dass die zweite Datenbankabfrage keinen Einfluss auf die Performance nimmt, sodass diese Entscheidung gerechtfertigt ist.

5.2.2 Kommunikation mit dem Serverplugin

Die Kommunikation mit dem Serverplugin findet über HTTP statt. Über die API vom Browser Chromium erfragt das Clientplugin den Hostnamen vom geöffneten Fenster bzw. Tab. Der Hostname dient als Zieladresse für die Kommunikation.

Somit sieht die Auth-Anfrage des Benutzers „Heinrich“ an den Server von „www.uni-duesseldorf.de“ beispielsweise so aus:

```
1 POST www.uni-duesseldorf.de HTTP/1.1
2 Host: <IP-Adresse von Heinrichs Computer>
3 pkwl_action: auth
4 pkwl_username: Heinrich
```

Die Antwort der Webanwendung ist wie folgt dargestellt und enthält die Parametern für das Time Lock Puzzle.

```
1 HTTP/1.1 200 OK
2 Server: www.uni-duesseldorf.de
3 Date: ...
4 ...
5
6 10349185566243082007;500000
```

Abbildung 4: Die Antwort des Servers auf eine Auth-Anfrage

Der Client empfängt die Antwort und liest daraus die Parameter n und t aus, die durch ein Semikolon getrennt sind und für das Time Lock Puzzle benötigt werden. Die Berechnung der Lösung des Time Lock Puzzles findet wie in Kapitel 3.2 beschrieben statt.

Nachdem die Lösung berechnet wurde, wird diese über ein neues XMLHttpRequest-Objekt zum Server geschickt.

```
1 POST www.uni-duesseldorf.de HTTP/1.1
2 Host: <IP-Adresse von Heinrichs Computer>
3 pkwl_action: auth
4 pkwl_username: Heinrich
5 pkwl_solution: 5026783858892303621
```

Der Server verifiziert die Lösung und verschlüsselt den Cookie. Das Clientplugin empfängt somit einen Geheimtext, der sich mit dem privaten Schlüssel des Benutzers in einen Klartext übersetzen lässt.

Die Entschlüsselung des Geheimtexts sowie das Parsen der einzelnen Cookies und deren Parametern geschieht analog wie in Abschnitt 5.1.4 beschrieben

Das Speichern der Cookies übernimmt der Chromium Browser, der sich über sein Application-Programming-Interface ansteuern lässt.

Nachdem die Cookies gespeichert wurden, lässt das Clientplugin noch den Tab bzw. das Fenster der Webanwendung neu laden und schließt das Fenster des Clientplugins, sodass der Benutzer sofort sieht, dass er nun authentifiziert mit der Webanwendung arbeiten kann.

5.2.3 Verwaltung der Schlüssel

Das Clientplugin verfügt über die Möglichkeit, dem Benutzer RSA-Schlüssel zu generieren. Die Generierung übernimmt die forge-Library [Dig13], wobei man die Schlüssellänge in Bit und den öffentlichen Exponenten e als Parameter übergeben kann. Die Schlüssellänge ist in unserer Implementierung aus Gründen der Performance und Sicherheit auf 2048 Bit festgesetzt und für den öffentlichen Exponenten gilt $e = 0\text{x}10001$. e wurde so gewählt, da andere Implementierungen, wie beispielsweise OpenSSL [Ope13], ebenfalls diesen Wert benutzen und weil durch einen konstanten Wert die Dauer der Verschlüsselung konstant bleibt und nicht abhängig von der Schlüssellänge ist.

Zusätzlich lassen sich auch RSA-Schlüssel importieren. Der Import ist bisher nur durch die Eingabe der Schlüsselzahlen n und d möglich. Hierbei wird ebenfalls $e = 0\text{x}10001$ benutzt.

5.2.4 Verwaltung der Webanwendungen

Über das Schlüsselsymbol lässt sich das Clientplugin für Chromium öffnen. Falls für den aktuellen Hostnamen kein Benutzername gespeichert ist, wird die Ansicht in Abbildung 5 dargestellt. Dort lässt sich der Benutzername einer Anwendung eintragen und ein Schlüssel auswählen bzw. generieren. Falls ein oder mehrere Benutzernamen vorhanden sind, so wird eine andere Ansicht angezeigt, die nur ein Auswahlmenü der verfügbaren Benutzernamen und einen Login-Button anzeigt, sodass ein Benutzer für eine Authentifizierung nur mit einem Klick das Clientplugin öffnen und mit dem zweiten Klick den Login-Button bestätigen muss.

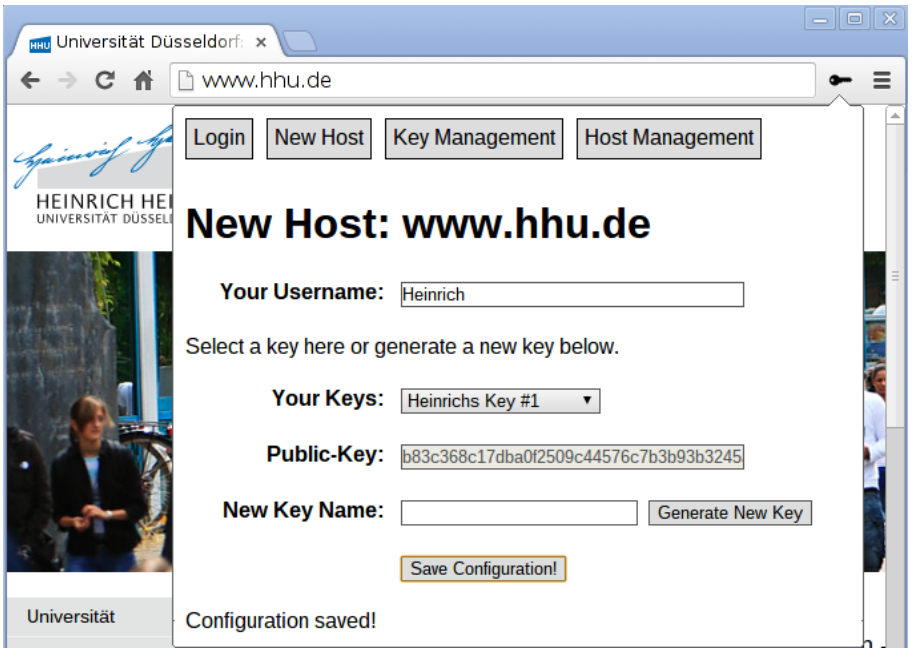

Abbildung 5: Die Benutzeroberfläche zur Verwaltung der Webanwendungen

6 Evaluation

In diesem Kapitel wird die Performance des implementierten Verfahrens analysiert. Es werden verschiedene Angriffe auf das Verfahren diskutiert und der Installations- und Bedienungsaufwand beschrieben.

6.1 Sicherheit per Design

Das Verfahren wurde so entwickelt, dass mögliche Angriffe bereits durch das Design des Verfahrens selbst wirkungslos sind oder bestmöglich abgewehrt werden können.

So ist es beispielsweise nicht möglich, das Verfahren durch einen Replay-Angriff zu brechen. Ein Angreifer kann alle fünf Schritte des Verfahrens aufzeichnen und abspielen. Beim Abspielen einer aufgezeichneten Auth-Anfrage wird lediglich ein Time Lock Puzzle angefordert oder beim Abspielen einer aufgezeichneten Puzzlelösung wird der Server, falls die aufgezeichnete Puzzlelösung noch gültig ist, einen Cookie erstellen und diesen verschlüsseln. Der Angreifer kann diesen aber nicht entschlüsseln, da er nicht den privaten Schlüssel besitzt.

Aufgrund der Verschlüsselung des Cookies mit dem öffentlichen Schlüssel des Benutzers ist es einem „Man-in-the-Middle“ nicht möglich den Cookie zu entschlüsseln, solange das Faktorisierungsproblem nicht effizient lösbar ist. Falls es doch klappen sollte, so müsste die Entschlüsselung innerhalb der Lebenszeit des Cookies von zwei Wochen passieren. Wie in Abschnitt 3.1.4 bereits erwähnt, wird die Verwendung von längeren Schlüssel als 1024 Bit empfohlen, damit eine zeitnahe Entschlüsselung unmöglich ist.

Weitere Angriffe auf das Verfahren, insbesondere über den Chromium Browser, sind nicht Bestandteil dieser Arbeit, dennoch sind die Plugindaten, also die Schlüssel, die Benutzernamen und die Hostnamen der Webanwendungen, genauso sicher wie alle Daten auf der Festplatte des Benutzers. Das Plugin ist von anderen Plugins des Browsers und jeglichen JavaScripts von anderen Webanwendungen abgeschottet, sodass keine Cross-Site-Scripting- und Cross-Site-Request-Forgery-Angriffe möglich sind.

6.2 Analyse der Performance

Um die Performance des Verfahrens beurteilen zu können, wurden mehrere hundert Messungen auf einem handelsüblichen Computer durchgeführt. Die Eckdaten des Testrechners sind in Tabelle 2 aufgeschlüsselt. Der Testrechner war zugleich Client und Server. Dadurch haben wir Testergebnisse erhalten, die frei von Schwankungen in der Übertragungsdauer sind, da keine Übertragungen über das Internet stattfanden, sondern lediglich innerhalb des Testrechners. Dies ist möglich, da die Schritte des Verfahrens jeweils abwechselnd auf Seite des Clients und des Server stattfinden.

Hardware	
Prozessor	AMD E-450 mit 2x 1,65 GHz
Arbeitsspeicher	4 GB
Festplatte	HDD via SATA
Software	
Betriebssystem	Arch Linux 64 Bit Kernel 3.11
Chromium Version	30.0.1599.66 (225456)
Blink Engine	537.36 (@158213)
JavaScript Engine	V8 3.20.17.13
Wordpress Version	3.6.1
MySQL Version	5.5.30
PHP Version	5.5.4
Apache Version	2.2.25

Tabelle 2: Eckdaten des Testcomputers

6.2.1 Definition der Messpunkte

Insgesamt wurden sechs Zeitpunkte gemessen. Dabei wurde die Zeit, die das Clientplugin braucht, um anhand des ausgewählten Benutzernamen und des Hostnamen der Webanwendung den Schlüssel aus der Datenbank auszuwählen, gemessen. Diese Messung ist interessant, da hiermit die Performance der noch experimentellen IndexedDB analysiert werden kann.

Die zweite Messung gibt an, wie lange der Server für die Überprüfung des Benutzernamens und die Generierung eines Time Lock Puzzles braucht. Darauf folgt die Zeitmessung für das Lösen des Time Lock Puzzles. Diese beiden Zeitmessungen sind interessant, um den DoS-Schutz bewerten zu können.

Die vierte Zeitmessung gibt die Dauer der Erstellung der Cookies und deren Verschlüsselung an. Schließlich folgen die Zeitmessungen der Cookieentschlüsselung und wie lange der Chromium Browser zum Setzen der Cookies braucht.

6.2.2 Ablauf der Zeitmessungen

Es wurden je 100 Messungen mit je zwei Schlüsseln der Länge 1024 und 2048 Bit durchgeführt und je 20 mit zwei Schlüsseln der Länge 4096 Bit. Die kleinere Anzahl von Messungen bei letzterem Schlüssel erklärt sich durch die hohe Laufzeit des Verfahrens bei dieser Schlüssellänge und dadurch, dass die Messungen wie erwartet abliefen, sodass diese wahrscheinlich keine neuen Erkenntnisse gebracht hätten.

Obwohl wir in Abschnitt 3.1.4 gesehen haben, dass 1024 Bit Schlüssel nicht mehr verwendet werden sollten, wurden diese dennoch in die Zeitmessung einbezogen, da die Performance interessant ist. Ebenfalls wurden Schlüssel mit einer Länge von 4096 Bit getestet, um einen Vergleich der Performance anstellen zu können und um zu sehen, wie lange das Verfahren dauert, falls Schlüssel der Länge 2048 Bit nicht mehr sicher sind.

6.2.3 Auswertung der Messergebnisse

Die Messergebnisse sind in Abbildung 6 dargestellt. Auf den ersten Blick erkennt man deutlich, dass ein längerer Schlüssel die Laufzeit des Verfahrens erhöht. Ein genauerer Blick lässt uns erkennen, dass jedoch nur die Laufzeit der Cookieentschlüsselung angestiegen ist. Die Dauer der restlichen Schritte bleibt konstant. Man erkennt an den Fehlerbalken, dass es sehr geringe Abweichungen der einzelnen Messungen gab. Die größten Abweichungen sieht man an dem Messpunkt an dem das Time Lock Puzzle gelöst wurde. Dies lässt sich durch die zufällig gewählten Primzahlen erklären.

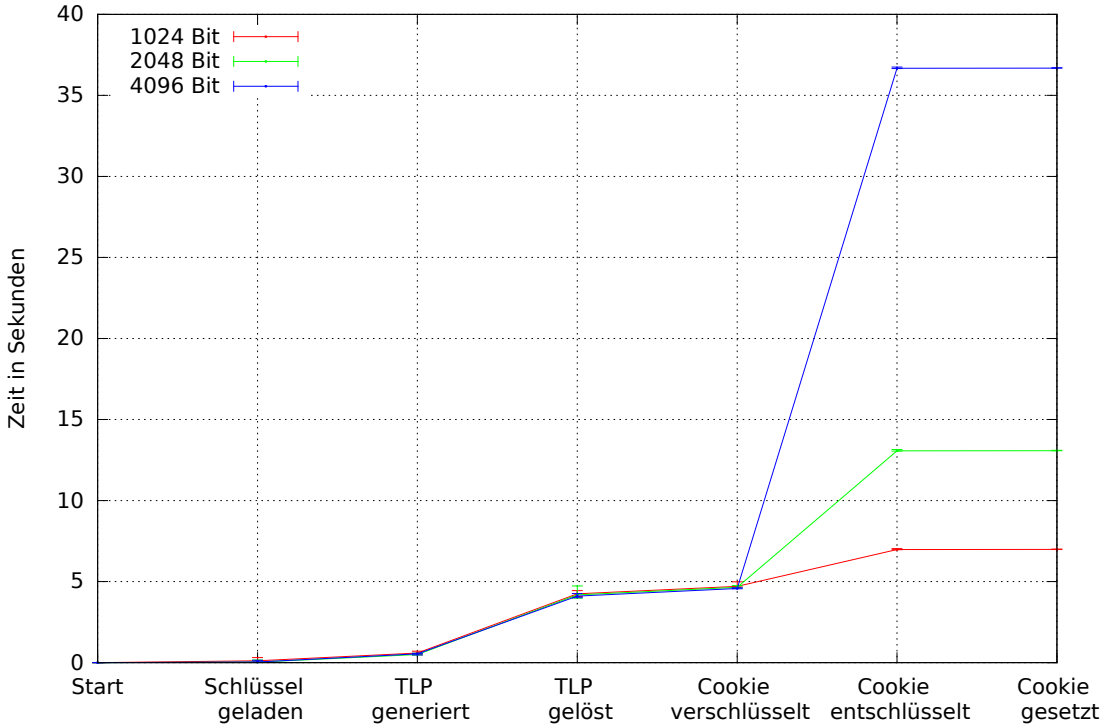

Abbildung 6: Vergleich der Performance von verschiedenen Schlüssellängen.

Denial-of-Service-Angriffe sind auf die Schritte zwei und vier des Servers ebenfalls nicht wirkungsvoll, da wir in Abbildung 6 sehen, dass die Erstellung eines Time Lock Puzzles und das Verschlüsseln der Cookies gleich performant ist. Ein DoS-Angriff ist auf den Schritt der Cookieverschlüsselung auch nur mit einem gültigen und gelösten Time Lock Puzzle möglich. Da die Lebenszeit der Puzzles kurz ist, verringert sich somit die Angriffsfläche erheblich.

Das implementierte Time Lock Puzzle ist somit eine effektive Maßnahme gegen Denial-of-Service-Angriffe.

6.3 Komfort

Der serverseitige Installationsaufwand wurde bereits kurz in Abschnitt 5.1.2 erwähnt. Da keinerlei Konfiguration nötig ist und kein Administrationszugang zum Server bestehen muss, lässt sich das PKWL-Verfahren auch in eingeschränkten Umgebungen nutzen. Die einzige Voraussetzung des implementierten Plugins ist ein installiertes Wordpress.

Der clientseitige Aufwand ist ebenfalls gering, da auch hier lediglich die Plugindatei über das Menü des Browsers ausgewählt werden muss. Die interne Datenbankstruktur wird automatisch angelegt und in wenigen Sekunden lassen sich Schlüssel generieren. Webanwendungen kann man komfortabel über die graphische Oberfläche hinzufügen und Benutzernamen und Schlüssel zuordnen.

Nach der Einrichtung muss der Benutzer nur noch zwei Mausklicks ausführen, um sich bei einer Webanwendung zu authentifizieren.

Das PKWL-Verfahren bietet demnach Komfort und Sicherheit bei der Authentifizierung.

7 Fazit

In dieser Arbeit wird ein Verfahren zur passwortlosen Authentifizierung entwickelt.

Die Problematik einer passwortbezogenen Authentifizierung wird dargestellt und vorhandene Alternativen für unzureichend befunden, da sie die Probleme nicht lösen oder nur sehr aufwändig einzurichten und kostenpflichtig sind.

Diese Arbeit zeigt eine Lösung mit Hilfe des RSA-Kryptosystems, sodass eine funktionierende, sichere und passwortlose Authentifizierung gewährleistet werden kann. Die Lösung wird als Browserplugin für Chromium und als Serverplugin für Wordpress entwickelt. Sie lässt sich aber ohne großen Aufwand auf andere Browser oder Webanwendungen portieren.

Durch den Einsatz von asymmetrischer Kryptographie kann es zu Einbußen der Performance kommen. Es wurden daher Benchmarks durchgeführt, die zeigen, dass der Server konstante Zeit rechnet und dass die Rechenzeit des Clients von der Länge seines Schlüssels abhängig ist.

Zusätzlich wird gezeigt, dass das implementierte Proof-of-Work-System eine effektive Maßnahme gegen DoS-Angriffe ist.

Insgesamt ist das im Rahmen dieser Arbeit entwickelte Verfahren zur passwortlosen Authentifizierung eine einfache, komfortable und sichere Alternative zur klassischen Authentifizierung via Passwort.

7.1 Ausblick

Das Verfahren lässt sich an einigen Stellen verbessern.

Eine Verbesserung, die die Performance betrifft, ist die Implementierung der Entschlüsselung des verschlüsselten Cookies mit Hilfe des chinesischen Restsatzes. Der Server kann außerdem automatisch an das Clientplugin melden, dass der Server das PKWL-Verfahren unterstützt, sodass der Benutzer dann komplett automatisch authentifiziert werden kann, wenn dieser das möchte.

Das World Wide Web Consortium (W3C) plant die Einführung einer Web Cryptography API [W3C13b]. Damit werden keine JavaScript Bibliotheken mehr nötig sein und man kann auf native Funktionen zurückgreifen, wodurch die Performance der Ver- und Entschlüsselung steigen kann.

Das Time Lock Puzzle könnte man optimieren, indem man die Zeitslotvariable und Lebenszeit eines TLP dynamisch wählt, sodass diese Werte mit der Auslastung des Servers steigen und fallen, damit der Client nicht immer eine intensive Berechnung durchführen muss.

Literatur

[CDF+07] CALLAS, J. ; DONNERHACKE, L. ; FINNEY, H. ; SHAW, D. ; THAYER, R.: *OpenPGP Message Format*. http://tools.ietf.org/html/rfc4880. Version: November 2007

[Cuo10] CUONY, Maximilien: *FireGPG discontinued*. http://blog.getfiregpg.org/2010/06/07/firegpg-discontinued/. Version: Juni 2010

[Dig13] DIGITAL BAZAAR: *Forge*. https://github.com/digitalbazaar/forge. Version: September 2013

[FGM+99] FIELDING, R. ; GETTYS, J. ; MOGUL, J. ; FRYSTYK, H. ; MASINTER, L. ; LEACH, P. ; BERNERS-LEE, T.: *Hypertext Transfer Protocol – HTTP/1.1*. http://tools.ietf.org/html/rfc2616. Version: Juni 1999

[hei13] HEISE.DE: *Insider-Angriff: Bankdaten von zwei Millionen Vodafone-Kunden entwendet*. http://heise.de/-1955090. Version: September 2013

[Huf10a] HUFF, Kyle: *About gpgAuth*. https://curetheitch.com/projects/gpgauth/. Version: Dezember 2010

[Huf10b] HUFF, Kyle: *ChangeLog*. https://github.com/firegpg/firegpg/blob/master/ChangeLog. Version: Juni 2010

[KM97] KRISTOL, D. ; MONTULLI, L.: *HTTP State Management Mechanism*. http://tools.ietf.org/html/rfc2109. Version: Februar 1997

[KR12] KUROSE, James F. ; ROSS, Keith W.: *Computernetzwerke*. München : Pearson Verlag, 2012

[Ope13] OPENSSL: *OpenSSL: The Open Source toolkit for SSL/TLS*. http://www.openssl.org/. Version: Oktober 2013

[Res00] RESCORLA, E.: *HTTP Over TLS*. https://tools.ietf.org/html/rfc2818. Version: Mai 2000

[Riv99] RIVEST, Ronald L.: *Description of the LCS35 Time Capsule Crypto-Puzzle*. http://people.csail.mit.edu/rivest/lcs35-puzzle-description.txt. Version: April 1999

[Rot08] ROTHE, Jörg: *Komplexitätstheore und Kryptologie*. Berlin : Springer Verlag, 2008

[Sch13] SCHUH, Justin: *Saying Goodbye to Our Old Friend NPAPI*. http://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html. Version: September 2013

[spi13] SPIEGEL.DE: *Kriminelle erbeuten Daten von 2,9 Millionen Adobe-Kunden*. http://spon.de/ad24h. Version: Oktober 2013

[ST03] SHAMIR, Adi ; TROMER, Eran: On the Cost of Factoring RSA-1024. In: *RSA CryptoBytes* 6 (2003), S. 10–19

[W3C13a] W3C: *Indexed Database API*. `http://www.w3.org/TR/IndexedDB/`. Version: Juli 2013

[W3C13b] W3C: *Web Cryptography API*. `http://www.w3.org/TR/WebCryptoAPI`. Version: Juni 2013

[web12a] *FireGPG susceptible to devastating attacks*. `https://tails.boum.org/doc/encryption_and_privacy/FireGPG_susceptible_to_devastating_attacks/`. Version: Dezember 2012

[web12b] *Why should you be interested in the Monkeysphere?* `http://web.monkeysphere.info/why/#index1h2`. Version: Juli 2012

Abbildungsverzeichnis

Tabellenverzeichnis