die digitale Signatur

Überblick

Einleitung und Überblick

Der RSA-Algorithmus

Signatur nach Feige, Fiat und Shamir

DSA und DSS

One-Time-Signaturen

Signaturen mit zusätzlichen Funktionen

Praxis

Ausblick

Definition

Eine digitale Unterschrift bindet eine Nachricht an eine Einheit.

Es ist möglich festzustellen ob Daten verändert wurden und wer signiert hat.

Beispiel: Eine eMail kann darauf geprüft werden, ob der Absender stimmt und ob ein Angreifer diese verändert hat.

Was kann die digitale Signatur?

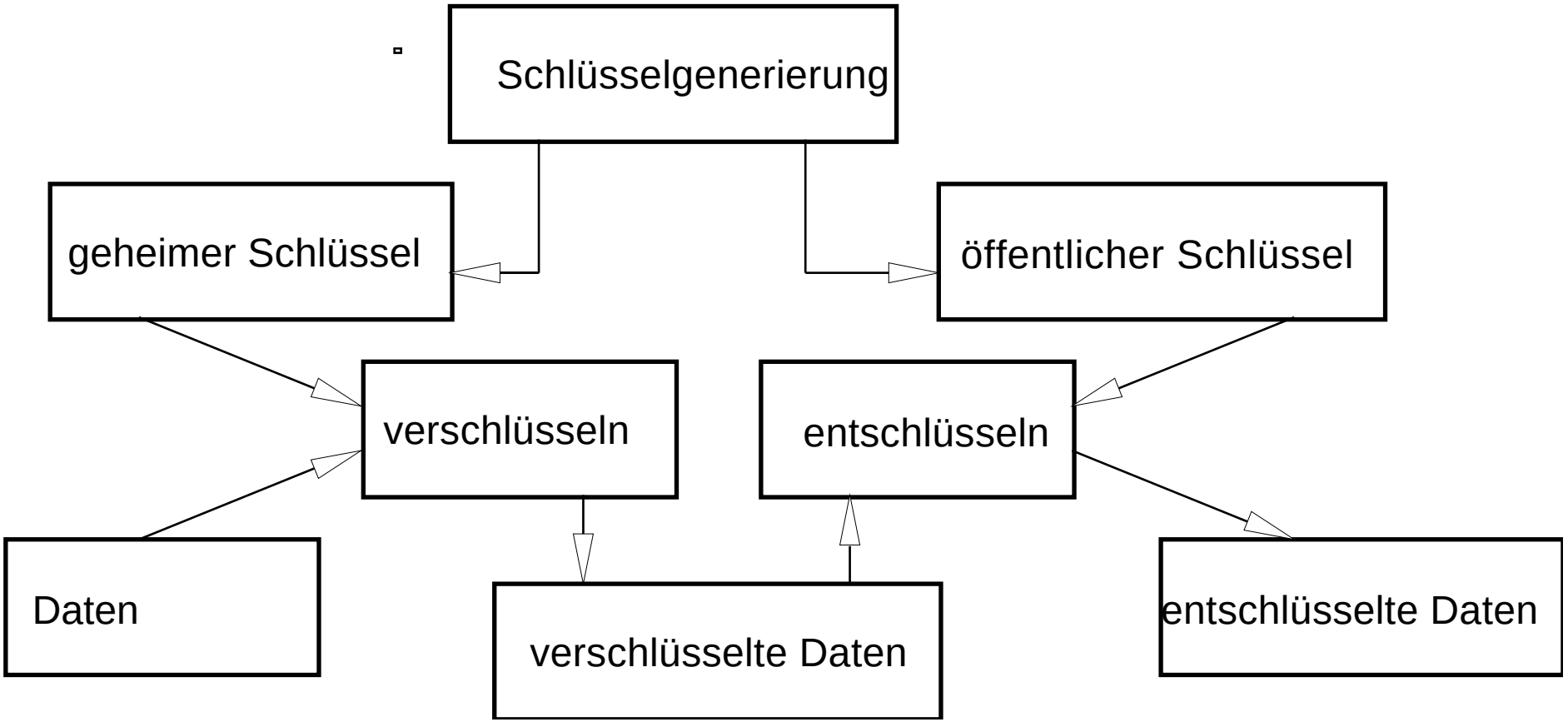
Die digitale Signatur kann:

+den Absender mit dem öffentlichen Schlüssel überprüfen
+Veränderungen an den Daten feststellten

Die digitale Signatur kann mit Hilfe einer dritten Partei:

+eine Identifikationsmöglichkeit bieten
+ein Netz der Vertrauenswürdigkeit spannen
+eine Aussage über den Zeitpunk des Signieren geben

Public-Key-Prinzip



Public-Key-Prinzip

Jedes Publik-Key-Verschlüsselungs-System bietet indirekt die Möglichkeit zur digitalen Signatur.

Denn: eine Nachricht, die sich mit dem öffentlichen Schlüssel von Person A entschlüsseln läßt, wurde mit dem geheimen Schlüssel von A verschlüsselt. Ist das Entschlüsseln der Nachricht nicht möglich, dann hat entweder Person A nicht unterschrieben, oder die Nachricht wurder verändert.

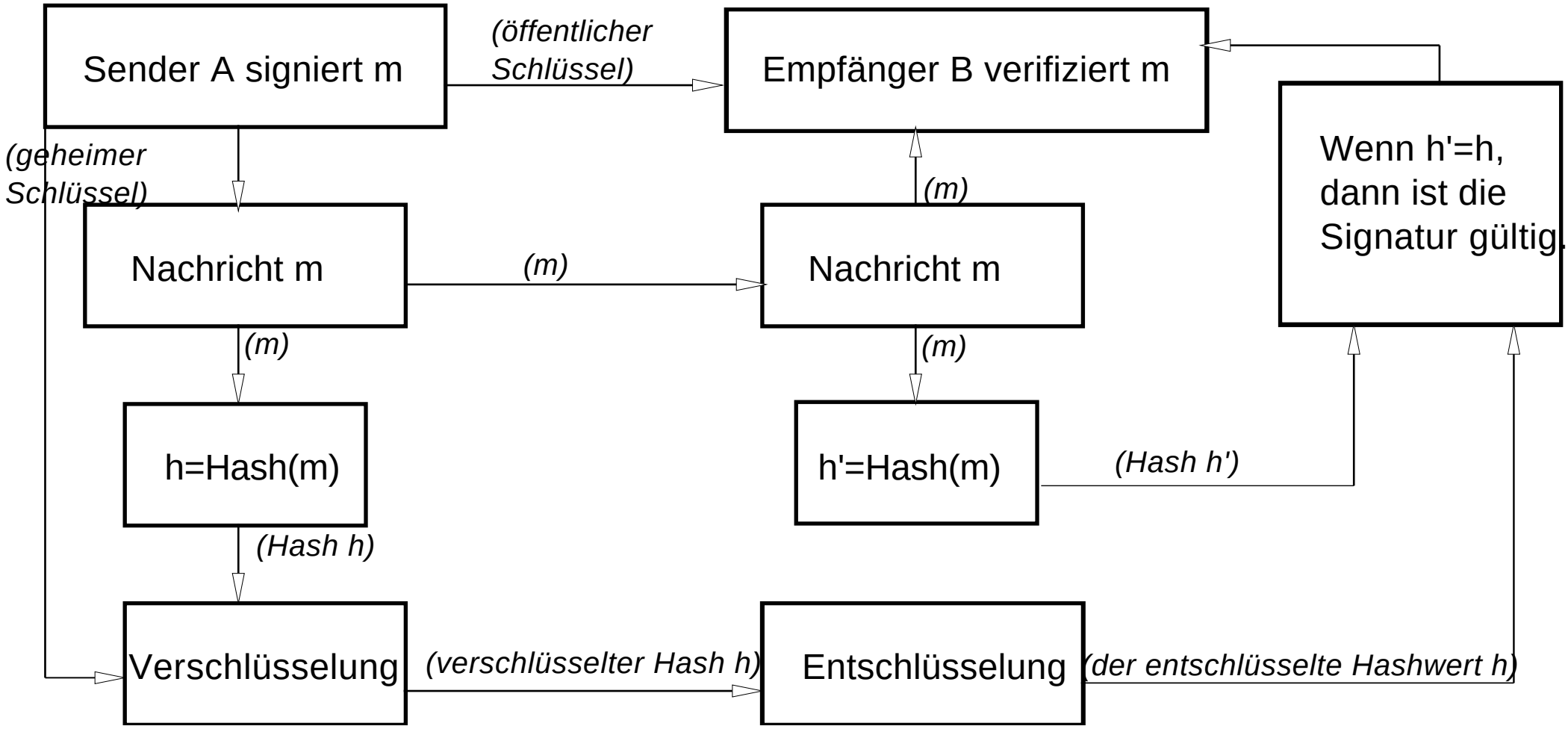
Wenn auch Leute ohne den öffentlichen Schlüssel die Nachricht lesen sollen, muss diese zusätzlich im Klartext mitgeschickt werden.
Dieses Problem wird durch „hashing“ gelößt.

Hashfunktionen

Oft wollen/können nicht alle Empfänger die Signatur einer Nachricht überprüfen.
Damit die Nachricht nicht zweimal (verschlüsselt und im Klartext) übertragen werden muss, verschlüsselt man nicht die ganze Nachricht, sondern eine Prüfsumme der Daten.
Diese wird durch eine Hash-Funktion erstellt, die folgende Eigenschaften haben muss:

1. Sie muss Eingaben von beliebiger Länge akzeptieren
2. Sie muss Ausgabem von fester Länge erzeugen
3. Man darf aus der Ausgabe nicht auf die Eingabe schliessen können

Signatur mit Public-Key und einer Hash-Funktion



RSA

nach Ideen von Diffie und Hellman

1977 vorgestellt

am MIT von Rivest, Shamir und Adleman entwickelt

basiert auf dem Problem große Zahlen zu faktorisieren

Sicherheit von RSA

mit den bekannten Methoden ist es sehr Zeitaufwendig große Zahlen zu faktorisieren

Gefahr durch den Einsatz von viel Rechenleistung
(kann durch vergrösserung des Schlüssels wirksam verhindert werden)

grössere Gefahr besteht darin, dass ein effektiver Algorithmus gefunden wird, der das Problem mit relativ wenig Aufwand lößt.

RSA Schlüssel Erzeugung - Algorithmus mit Beispiel

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

$p = 3$ $q = 17$
$n = 3 * 17 = 51$, $\phi = (3-1) * (17-1) = 32$
$e = 3$ (3 und 32 sind relativ prim/teilerfremd).
$d = 11$ (weil: $3 * 11 = 33$, $33/32 = 1 \mod 1$).
geheimer Schlüssel: $(51, 11)$
öffentlicher Schlüssel:$(51, 3)$

RSA Signatur - Algorithmus und Beispiel

1. Berechne $\tilde{m} = R(m)$, eine Integer Zahl im Bereich $[0, n-1]$.
2. Berechne $s = \tilde{m}^d \mod n$.
3. s ist die Signatur für die Nachricht m.

wir wählen $m =$„abc“ wobei gilt: $a = 1$, $b = 2$, $c = 3$.

$a = 1$ $\quad 1^{11} = 1 = 0 * 51 + 1$

$b = 2$ $\quad 2^{11} = 2048 = 40 * 51 + 8$

$c = 3$ $\quad 3^{11} = 177147 = 3473 * 51 + 24$

s=$(1, 8, 24)$

RSA Verifikation - Algorithmus und Beispiel

1. Berechne $\tilde{m} = s^e \mod n$.
2. Prüfe ob $\tilde{m} \in M : R$; wenn nicht verwerfe die Daten.
3. Stelle $m = R^{-1}\tilde{m}$ her.

$1^3 = 1 = 0 * 51 + 1$
$8^3 = 512 = 10 * 51 + 2$
$24^3 = 13824 = 271 * 51 + 3$
$m = (1, 2, 3) =$„abc“

Probleme von RSA

Das generieren von Schlüsseln benötigt viel Rechenleistung
(bei Implementation auf z.B. Chipkarten wird der Schlüssel extern generiert.)

Es ist nicht bewiesen, dass RSA sicher ist.

Falls RSA sicher sein sollte, bleibt das Problem der Implementation.
(Angriff auf die schwächste Stelle im System)

Feige-Fiat-Shamir

Ist ein modifiziertes Identifikations-Protokoll, wobei die für der Verifizierer zufällig gestellte Aufgabe durch eine Hash-Funktion ersetzt wird.

Feige-Fiat-Shamir-Schlüsselerzeugung

1. $n = p * q$, wobei p,q geheime, zufällige Primzahlen sind.
2. Wähle ein zufälliges k und verschieden zufällige Zahlen $s_1, s_2, ..., s_k \in \mathbb{Z}^*$.
3. $v_j = s_j^{-2} \mod n$, $1 \leq j \leq k$
4. Der öffentliche Schlüssel von A ist der k-Tupel $(v_1, v_2, ..., v_k)$ und n; der geheime Schlüssel von A ist der k-Tupel$(s_1, s_2, ..., s_k)$.

Feige-Fiat-Shamir Signatur

1. Wähle eine Zufallszahl r, $1 \leq r \leq n-1$.
2. $u = r^2 \mod n$
3. $e = (e_1, e_2, ..., e_k) = h(m||u)$; mit $e_i \in \{0, 1\}$
4. $s = r * \prod_{j=1}^{k} s_j^{e_j} \mod n$
5. Die Signatur für die Nachricht m ist (e, s).

Feige-Fiat-Shamir Verifikation

1. $w = s^2 * \prod_{j=1}^{k} * v_j^{e_j} \mod n$
2. $e' = h(m||w)$
3. Wenn $e = e'$ ist die Signatur zu akzeptieren.

DSA und DSS

Steht für Digital-Signature-Algorithm und ist eine Variante des El-Gama-Systems.

Wurde 1991 amerikanischen (NIST) entwickelt.

Seit 1994 dortiger Standard (FIPS 186), auch DSS (Digital Signature Standard) genannt.

Als Hash-Funktion wird explizit SHA-1 vorgeschrieben.

DSA Schlüssergenerierung

1. Wähle Primzahl q, wobei $2^{159} < q < 2^{160}$.
2. Wähle t mit $0 \leq t \leq 8$ und wähle Primzahl p mit $2^{511+64t} \leq p \leq 2^{512+64t}$ mit der Eigenschaft, dass q $(p-1)$ teilt.
3. Wiederhole bis $\alpha \neq 1$: suche $g \in \mathbb{Z}_p^*$ und berechne $\alpha = g^{(p-1)/q} \mod p$.
4. Suche Zufallszahl a mit $1 \leq a \leq q-1$.
5. $y = a^{\alpha} \mod p$
6. Der öffentliche Schlüssel ist (p, q, α, y), der geheime a.

$q = 3 \wedge p = 7$ mit q teilt $(p-1)$
für $g = 3$ gilt $\alpha = 2$
$a = 2$
$y = \alpha^a = 4$
Der öffentliche Schlüssel ist (p, q, α, y), der Private ist a.

DSA Signatur

1. Wähle eine geheime Zufallszahl k, mit $0 < k < q$.
2. $r = (\alpha^k \mod p) \mod q$
3. Berechne $k^{-1}(\mod q)$.
4. $s = k^{-1}\{h(m) + ar\} \mod q$
5. Die Signatur für m ist (r, s).

Um es einfach zu machen: $m = 12 \wedge h(m) = 12$
$k = 2$
$r = \alpha^k \mod q = 1$
$k^{-1}(\mod q) = 4$
$k^{-1} \mod 3 = 1$
$s = 1 * (12 + 2 * 1) \mod 3 = 2$
Die Signatur für m ist (r, s)

DSA Verifikation

1. Prüfe, ob $0 < r < q \wedge 0 < s < q$: wenn nicht, ist die Signatur zu verwerfen.
2. $w = s^{-1} \mod q$
3. $u_1 = w * h(m) \mod q$; $u_2 = rw \mod q$
4. $v = (\alpha^{u_1} y^{u_2} \mod p) \mod q$
5. Nur wenn $v = r$ ist die Signatur zu aktzeptiren.

$w = 4 \mod 3 = 1$

$u_1 = 0$

$u_2 = 1$

$v = 1 = r$

da $v = r$ wird die Signatur akzeptiert.

One-Time-Signaturen

Nur eine Nachricht darf signiert werden,
sonst kann die Signatur gefälscht werden.

Vorteil: sehr schnell

Rabin One-Time Verfahren

Einer der ersten Ansätze für digitale Signatur.

Als erstes wird die Authenzititität des Absenders überprüft.
Dann die Nachricht auf Veränderungen.

Aufgrund der nötigen Interaktion zwischen Absender und Empfänger nicht Praxistauglich.

Signaturen mit zusätzlichen Funktionen

Einige Probleme sind mit den bisher besprochenen Signatur-Methoden nicht lösbar.

- Datenschutz

- Ableugnen von Signaturen

- Erkennen und Beweise von (erfolgreichen) Fälschungen der Signatur

Eine TTP ist für die Lösung des letzte Problems von Nöten.

Blindes Signieren

Notwendig, wenn der Signierende nicht wissen soll, was er Unterschreibt.

Beispielsweise eine Firma, die ihre Mitarbeiter als diese Identifiziert aber deren Privatsphäre achtet.
Oder ein Sohn mit einem schlechten Zeugnis und einem schlechtgelauten Vater.

Hashing der Nachricht ist nicht sicher genug, da der Angreifer theoretisch alle Nachrichten hashen und die Daten vergleichten könnte.

Lösung: Blinde Signatur nach Chaum
Zum verschleiern der Nachricht wird eine Zufallszahl k verwendet.

Blinde Signatur nach Chaum

Der öffentliche Schlüssel von B ist (n, e) der private (n, d).

A möchte eine "blinde Signatur" von B:

1. A berechnet $m^* = mk^e \mod n$ wobei $0 \leq k \leq n-1$ und schickt m^* an B.
2. Blinding: B berechnet $s^* = (m^*)^d \mod n$ und schickt s^* an A.
3. Signieren: A berechnet $s = k^{-1}s^* \mod n$.
4. Unblinding: s ist die Signatur von m durch B.

Verbindliches Signieren

A kann nur dann beweisen, dass eine Signatur s zu einer Nachricht m von B kommt, wenn B bei der Verifikation mitwirkt.

Gegen die Möglichkeit, dass B sich weigert die Signatur zu bestätigen, bewusst Fehler bei der Verifikation macht oder die Signatur gefälscht ist, gibt es das „Disavowal-Protokoll“.

Verbindliches-Signieren Schlüsselgenerierung

1. Wähle $p = 2q + 1$, wobei p,q Primzahlen sind.
2. Wähle zufällig $\beta \in \mathbb{Z}_p^*$ und berechne $\alpha = \beta^{(p-1)/q} \mod p$ bis $\alpha \neq 1$.
3. Wähle zufällig $a \in \{1, 2, ..., q-1\}$ und berechne $y = \alpha^a \mod p$.
4. Der öffentliche Schlüssel ist (p, α, y), der private ist a.

Verbindliches-Signieren Signatur und Verifikation

1. $s = m^a \mod p$
2. s ist die Signatur zu m.

A möchte die Signatur von B überprüfen.

1. A wählt zufällig geheime $x_1, x_2 \in \{1, 2, ..., q-1\}$.
2. A berechnet $z = s^{x_1} y^{x_2} \mod p$ und sendet z zu B.
3. B berechnet $w = (z)^{a^{-1}} \mod p$ und sendet w zu A.
4. A berechnet $w' = m^{x_1} \alpha^{x_2} \mod p$.
5. Wenn $w' = w$, akzeptiert A die Signatur.

Verbindliches-Signieren Disavowal

A kann feststellen, ob die Signatur s zur Nachricht $m$von B kommt, ob B versucht die Signatur zu leugnen oder ob die Signatur gefälscht wurde.

1. A wählt zufällig geheime $x_1, x_2 \in \{1, 2, ..., q-1\}$.
2. A berechnet $z = s^{x_1} y^{x_2} \mod p$ und sendet z zu B.
3. B berechnet $w = (z)^{a^{-1}} \mod p$ und sendet w zu A.
4. A berechnet $w' = m^{x_1} \alpha^{x_2} \mod p$.
5. Wenn $w' = w$, akzeptiert A die Signatur und das Protokoll stoppt.
6. A wählt zufällig geheime $x'_1, x'_2 \in \{1, 2, ..., q-1\}$
7. A berechnet $z' = s^{x'_1} y^{x'_2} \mod p$ und sendet z' zu B.
8. B berechnet $v = (z')^{a^{-1}} \mod p$ und sendet v zu A.
9. A berechnet $v' = m^{x'_1} \alpha^{x'_2} \mod p$.
10. Wenn $v = v'$, akzeptiert A die Signatur und das Protokoll stoppt.
11. A berechnet $c = (w\alpha^{x_2})^{x'_1} \mod p$ und $c' = (v\alpha^{-x'_2})^{x_1} \mod p$. Wenn $c = c'$ ist s eine Fälschung, sonst versucht B die Signatur zu leugnen.

Fail-Stop-Signaturen

Eine Fail-Stop-Signatur gibt bei Fälschung der Signatur die Möglichkeit dies auch zu Beweisen.

Die Schlüsselgenerierung teilen sich Absender und eine TTP.

Fail-Stop Schlüsselgenerierung

TTP:

1. Wähle Primzahlen p und q, so dass $(p-1)$ von q geteilt wird.
2. Wiederhole bis $\alpha \neq 1$: wähle zufällig $g \in \mathbb{Z}_p^*$ und berechne $\alpha = g^{(p-1)/q}$.
3. Wähle zufällig und geheim a, $1 \leq a \leq q-1$ und berechne $\beta = \alpha^a \mod p$.
4. Schicke (p, q, α, β) an A.

A:

5. Wähle geheime Zufallszahlen x_1, x_2, y_1, y_2 aus dem Intervall $[0, q-1]$.
6. $\beta_1 = \alpha^{x_1}\beta^{x_2}$ $\beta_2 = \alpha^{y_1}\beta^{y_2} \mod p$
7. Der öffentliche Schlüssel ist $(\beta_1, \beta_2, p, q, \alpha, \beta)$, der private $\bar{x} = (x_1, x_2, y_1, y_2)$

Fail-Stop Signatur und Verifikation

Signatur:

1. $s_{1,m} = x_1 + my_1 \mod q$ und $s_{2,m} = x_2 + my_2$ wobei $m \in [0, q-1]$
2. Die Signatur der Nachricht m ist $(s_{1,m}, s_{2,m})$.

Verifikation:

1. $v_1 = \beta_1 \beta_2^m \mod p$ und $v_2 = \alpha^{s_{1,m}} \beta^{s_{2,m}} \mod p$
2. Die Signatur ist valid, wenn $v_1 = v_2$.

Fail-Stop Fälschungsbeweis

Es wird versucht, aus der vermeintlich gefälschten Signatur, a zu errechnen. Wenn dies gelingt, ist das ein Beweis für eine Fälschung, da nur die TTP dieses kennen darf.

1. s' sei die gefälschte Signatur für m. Solange $s = s'$: berechne s für m
2. $a = (s_{1,m} - s'_{1,m}) * (s_{2,m} - s'_{2,m})^{-1} \mod q$

Praxis

Trustcenter

GnuPG

Reale Probleme

Literatur

A. Menezes, P. van Orschot, S. Vanstone
— Handbook of Applied Cryptography
(http://www.cacr.math.uwaterloo.ca/hac)

verschiedene Autoren
— deutsche Übersetzeung der RSA-Crypto FAQ
(http://www.iks-jena.de/mitarb/lutz/security/cryptfaq/)

William Stalling
— Sicherheit im Datennetz

Chaos Computer Club
— verschiedenste Publikationen
(https://www.ccc.de)