

Signer-Base-Signaturen

Distributed Oblivious Transfer

**Seminararbeit am Lehrstuhl für Informatik I
der RWTH Aachen**

Betreuer: Dr. Walter Unger

Urs Enke
Stefan Rieger

11. August 2003

Signer-Base-Signaturen

Herkömmliche Signatur-Schemata bringen bei Diebstahl des Schlüssels eine Reihe von Problemen mit sich: Um Mißbrauch zu verhindern, muß der Schlüssel widerrufen werden sowie ein neuer verbreitet, außerdem sind Signaturen der Vergangenheit nicht mehr verifizierbar. Bei Signer-Base-Signaturen wird dies durch verteilte Speicherung und zeitlich begrenzte Gültigkeit des daher regelmäßig aufzufrischenden Schlüssels vermieden. Eine mögliche Umsetzung wird in dieser Seminararbeit vorgestellt sowie auf Korrektheit und Sicherheitsaspekte hin überprüft.

Distributed Oblivious Transfer

Der Distributed Oblivious Transfer ergänzt das Konzept verteilter Geheimnisse um die Geheimhaltung der Wahl der gewünschten Information (Oblivious Transfer) bei gleichzeitiger Gewährleistung eines gewissen Schutzes vor betrügerischen Koalitionen der Teilnehmer. Der Vorstellung eines dies realisierenden Protokolls folgt eine Analyse der zugrundeliegenden Sicherheitsparameter.

Inhaltsverzeichnis

1	Signer-Base-Signaturen	4
1.1	Idee	4
1.1.1	Probleme herkömmlicher Schemata	4
1.1.2	Ansatz der SBS	4
1.2	Protokoll	4
1.3	Algorithmen	5
1.3.1	Nomenklatur und Notation	5
1.3.2	Schlüssel-Generierung	6
1.3.3	Perioden-Update Base	6
1.3.4	Perioden-Update Signer	6
1.3.5	Zusatz-Update Base	7
1.3.6	Zusatz-Update Signer	7
1.3.7	Unterschreiben	7
1.3.8	Verifizieren	8
1.4	Sicherheit	9
1.4.1	Sinn der mathematischen Vorgehensweise	9
1.4.2	Mögliche Probleme	9
1.5	Zusammenfassung	10
2	Distributed Oblivious Transfer	11
2.1	Einführung	11
2.1.1	Verteilte Geheimnisse	11
2.1.2	Oblivious Transfer	11
2.1.3	DOT	12
2.1.4	Parameter	12
2.2	Implementierung	13
2.2.1	Initialisierung (seitens \mathcal{S})	13
2.2.2	Anfrage (seitens \mathcal{R})	13
2.2.3	Antwort und Geheimniserhalt	13
2.3	Mathematische Struktur	14
2.3.1	Wieso funktioniert das Verfahren?	14
2.3.2	Qualität des vorgestellten Verfahrens	15
2.4	Sicherheit	15
2.4.1	Exkurs: Entropie	15
2.5	Zusammenfassung	17
2.6	Ausblick	17

1 Signer-Base-Signaturen

1.1 Idee

1.1.1 Probleme herkömmlicher Schemata

Besteht bei Anwendung eines üblichen Signatur-Schemas der Verdacht, der zum Signieren nötige private Schlüssel sei Dritten bekannt geworden und daher die Fälschung von Signaturen möglich, bleibt dem Besitzer des Schlüssels i.d.R. nichts anderes übrig, als einen Mißbrauch durch offiziellen Widerruf des kompletten Paares Private Key / Public Key einzugrenzen zu versuchen. Hierbei ist nicht nur der mit dem Widerruf und der anschließenden Verbreitung eines neuen öffentlichen Schlüssels einhergehende Aufwand zu bemängeln, sondern auch die Tatsache, daß fortan alle jemals mit dem alten Schlüssel erstellte Signaturen nicht als vom seinem „rechtmäßigen“ Besitzer kreiert verifizierbar und somit u.U. bedeutungslos sind.

1.1.2 Ansatz der SBS

Der Wunsch nach einem robusteren System liegt daher nahe. Die sinnvollen Forderungen lassen sich auf folgende beiden Punkte zusammenfassen:

- *Forward Security*: Die Integrität von Signaturen der Vergangenheit muß nicht angezweifelt werden, d.h. der Besitz eines aktuellen Schlüssels ermöglicht keine Erzeugung von für vergangene Zeitperioden gültig gewesenen Unterschriften.
- *Key Insulation*: Mißbrauch eines gestohlenen Schlüssels (also das Leisten von Unterschriften) ist auf einen bestimmten Zeitraum beschränkt.

Wären beide Aspekte gegeben, bliebe der durch ein Bekanntwerden des privaten Schlüssels potentiell verursachte Schaden auf die Gegenwart begrenzt. Hierzu ist es nötig, die privaten Schlüssel mit einer Art „Verfallsdatum“ zu versehen. Dies zu erreichen, ohne auch das zur Verifikation benötigte öffentliche Gegenstück des Schlüssels regelmäßig ändern zu müssen, ist die wesentliche Eigenschaft der in diesem Kapitel betrachteten *Signer-Base-Signaturen* (vorgestellt in [1]).

Grundidee hierbei ist, wie der Name nahelegt, die Aufteilung der signierenden Seite in den die Unterschriften leistenden *Signer* sowie eine sogenannte *Base*, die sich in regelmäßigen Abständen gegenseitig aktualisieren. Das zugrundeliegende Protokoll ist so gestaltet, daß ein isoliertes Bekanntwerden der Daten von Signer *oder* Base noch keine Fortschreibung des privaten Schlüssels ermöglicht, aber auch daß ein „komplettierender“ zweiter Diebstahl des jeweils anderen Teilschlüssels zwecklos bleibt, solange nur in der Zwischenzeit eines der erwähnten Updates vollzogen wurde.

Diese Vorgehensweise verhindert also in gewissem Rahmen die Fortschreibbarkeit eines gestohlenen Schlüssels und erfüllt daher die Bedingung der *Key Insulation*. Forward Security wiederum wird dadurch gewährleistet, daß für die Updates *One-Way*-Funktionen benutzt werden, die eine Berechnung alter Schlüssel aus aktuellen praktisch verhindert.

1.2 Protokoll

Wesentlicher Teil des Verfahrens sind neben den in regelmäßigen Zeitabständen durchzuführenden *Perioden-Updates* in beliebiger Anzahl einstreubare *Zusatz-Updates*, die z.B. sofort bei Bekanntwerden des Diebstahls eines der Teilschlüssel durchgeführt werden können, um weitergehenden Schaden im Falle eines kurzfristig folgenden Diebstahls des jeweils anderen Teilschlüssels abzuwenden.

Eine solche Situation könnte wie folgt aussehen: In Zeitperiode 1 stehle der Dieb den Schlüssel des Signers, er kann daher in dieser Periode Unterschriften ausstellen, die von jedem, der Zugriff auf den dazugehörigen öffentlichen Schlüssel hat, als korrekt verifiziert werden können. Wird nun ein Zusatz-Update vollzogen und stiehlt derselbe Dieb anschließend auch den Schlüssel der Base, so kann er zwar bis zum Ende der Periode 1 noch gültige Signaturen ausstellen, nicht jedoch das zum Periodenwechsel von Signer und Base durchgeführte Perioden-Update korrekt nachvollziehen und auch einen für Periode 2 gültigen Schlüssel generieren.

Ein vor dem Zusatz-Update auch den anderen Schlüssel erlangender Dieb könnte hingegen in allen folgenden Zeitperioden gültige Signaturen erzeugen, daher sollte nach bekanntgewordenem Teil-Diebstahl sofort ein Zusatz-Update vollzogen werden.

Das reguläre, Eingriffe möglicher Gegner nicht berücksichtigende und daher auf nicht essentielle Zusatz-Updates verzichtende, SBS-Protokoll läuft wie folgt ab:

1. Generierung der Schlüssel von Signer und Base sowie des öffentlichen für eine bestimmte Anzahl von Zeitperioden
2. Publikation des öffentlichen Schlüssels
3. In jeder Zeitperiode
 - a) Durchführung eines Perioden-Updates
 - b) Durchführung eines initialisierenden Zusatz-Updates
 - c) ggf. Leistung von aktuell gültigen Signaturen seitens des Signers
 - d) evtl. Durchführung weiterer Zusatz-Updates

1.3 Algorithmen

1.3.1 Nomenklatur und Notation

Bei der nachfolgenden Beschreibung der an SBS beteiligten Algorithmen werden durchgehend die hier aufgeführten Bezeichnungen verwendet:

PK Öffentlicher Schlüssel

BK Privater Schlüssel der Base

SK Privater Schlüssel des Signers

PM Perioden-Update-Message

ZM Zusatz-Update-Message

m zu verschlüsselnde Nachricht

v Ergebnis der Verifikation

k, l Sicherheitsparameter

(t, s) (Periode, zugehörige Signatur)

Da die Schlüssel sich nach jedem Update ändern, sind sie mit von der Anzahl durchgeführter Perioden- und Zusatz-Updates bestimmten Indizes ausgezeichnet:

$K_{t,z} :=$ Schlüssel K in Periode t nach z Zusatz-Updates in t

Dabei gilt: $t.z < t'.z' \Leftrightarrow t < t'$ oder $(t = t'$ und $z < z')$

Beispiel:

⋮	
$K_{3.4}$	↔ Zusatz-Update
$K_{3.5}$	↔ Perioden-Update
$K_{4.0}$	↔ Zusatz-Update
$K_{4.1}$	
⋮	

K_t sei aus Gründen der Übersichtlichkeit synonym zu $K_{t.z}$ verwendet.

1.3.2 Schlüssel-Generierung

Unter Vorgabe der Sicherheitsparameter k und l sowie der Anzahl T der Zeitperioden, über die hinweg Unterschriften möglich sein sollen, liefert der Algorithmus A_{GEN} die Ausgangswerte für die Teilschlüssel von Signer (SK_0) und Base (BK_0) sowie den öffentlichen Schlüssel PK .

A_{GEN} : $(k, l, T) \mapsto (SK_0, KB_0, PK)$
 Wähle Primzahlen $p_{1,2}$ der Länge $(\lceil \frac{k}{2} \rceil - 1)$ Bit, mit $2p_i - 1$ ebenfalls prim
 $n := (2p_1 - 1) \cdot (2p_2 - 1)$
 Wähle Primzahlen $2^l(1 + \frac{i-1}{T}) \leq e_i < 2^l(1 + \frac{i}{T})$, $i = 1, \dots, T$
 Wähle $s_0, b_0 \in \mathbb{Z}_n^*$
 $PK := (s_0 \cdot b_0)^{-e_1 \dots e_T} \bmod n$
 return $((0, s_0, \emptyset), (0, b_0), PK)$ (n, T, e_i seien bekannt)

Beispiel:

$A_{GEN}(8, 8, 4) \mapsto (SK_0, KB_0, PK)$
 $p_1 = 3, p_2 = 7 \Rightarrow n = 65$
 $e_1 = 257, e_2 = 331, e_3 = 389, e_4 = 449$
 $s_0 = 6, b_0 = 12$
 $PK = (6 \cdot 12)^{-257 \cdot 331 \cdot 389 \cdot 449} \bmod 65 = 7$
 return $(SK_0 = (0, 6, \emptyset), BK_0 = (0, 12), PK = 7)$

1.3.3 Perioden-Update Base

Aus dem alten Base-Teilschlüssel BK_t wird neben dessen neuer Version BK_{t+1} für die Zeitperiode $t + 1$ auch die später den Signer-Teilschlüssel aktualisierende Perioden-Update-Message PM_t erzeugt.

A_{PUB} : $BK_t \mapsto (BK_{t+1}, PM_t)$
 Sei $BK_t = (t, b_t)$
 $b_{t+1} := b_t^{e_{t+1}} \bmod n$
 $PM_t := b_t^{e_{t+2} \dots e_T} \bmod n$
 return $((t + 1, b_{t+1}), PM_t)$

Beispiel:

$A_{PUB}(BK_0) \mapsto (BK_1, PM_0)$
 $BK_0 = (0, 12)$
 $b_1 = b_0^{e_1} \bmod 65 = 12^{257} \bmod 65 = 12$
 $PM_0 = b_0^{e_2 \cdot e_3 \cdot e_4} \bmod n = 12^{331 \cdot 389 \cdot 449} \bmod 65 = 38$
 return $(BK_1 = (1, 12), PM_0 = 38)$

1.3.4 Perioden-Update Signer

Mit Hilfe der Perioden-Update-Message PM_t wird aus dem alten Signer-Teilschlüssel SK_t der zur nächsten Zeitperiode gehörige Teilschlüssel SK_{t+1} erzeugt.

A_{PUS} : $(SK_t, PM_t) \mapsto SK_{t+1}$
 Sei $SK_t = (t, s_t, S_t)$

$$s_{t+1} := s_t^{e_{t+1}} \bmod n$$

$$S_{t+1} := s_t^{e_{t+2} \cdots e_T} \cdot PM_t \bmod n = (s_t \cdot b_t)^{e_{t+2} \cdots e_T} \bmod n$$

$$\text{return } (t+1, s_{t+1}, S_{t+1})$$
Beispiel:

$$A_{PUS}(SK_0, PM_1) \mapsto SK_1$$

$$SK_0 = (0, 6, \emptyset), PM_0 = 38$$

$$s_1 = s_0^{e_1} \bmod 65 = 6^{257} \bmod 65 = 41$$

$$S_1 = s_0^{331 \cdot 389 \cdot 449} \cdot 38 \bmod 65 = 6^{57812791} \cdot 38 \bmod 65 = 58$$

$$\text{return } SK_1 = (1, 41, 58)$$

1.3.5 Zusatz-Update Base

Analog dem Perioden-Update wird aus dem alten Base-Teilschlüssel $BK_{t,z}$ neben dessen neuer Version $BK_{t,(z+1)}$ auch die anschließend den Signer-Teilschlüssel aktualisierende Zusatz-Update-Message $ZM_{t,z}$ erzeugt.

$$A_{ZUB}: BK_{t,z} \mapsto (BK_{t,(z+1)}, ZM_{t,z})$$

$$\text{Sei } BK_{t,z} = (t, b_t)$$

$$\text{Wähle } ZM_{t,z} \in \mathbb{Z}_n^*$$

$$b'_t := b_t \cdot ZM_{t,z}^{-1} \bmod n$$

$$\text{return } ((t, b'_t), ZM_{t,z})$$
Beispiel:

$$A_{ZUB}(BK_{1,0}) \mapsto (BK_{1,1}, ZM_{1,0})$$

$$BK_{1,0} = (1, 12)$$

$$ZM_{1,0} = 17$$

$$b'_1 = b_1 \cdot 17^{-1} \bmod 65 = 12 \cdot 23 \bmod 65 = 16$$

$$\text{return } (BK_{1,1} = (1, 16), ZM_{1,0} = 17)$$

1.3.6 Zusatz-Update Signer

Mit Hilfe der Zusatz-Update-Message $ZM_{t,z}$ wird aus dem Signer-Teilschlüssel $SK_{t,z}$ eine neue, aber für die gleiche Zeitperiode taugliche, Version $SK_{t,(z+1)}$ erzeugt.

$$A_{ZUS}: (SK_{t,z}, ZM_{t,z}) \mapsto SK_{t,(z+1)}$$

$$\text{Sei } SK_{t,z} = (t, s_t, S_t)$$

$$s'_t := s_t \cdot ZM_{t,z} \bmod n$$

$$\text{return } (t, s'_t, S_t)$$
Beispiel:

$$A_{ZUS}(SK_{1,0}, ZM_{1,0}) \mapsto SK_{1,1}$$

$$SK_{1,0} = (1, 41, 58)$$

$$s'_1 = s_1 \cdot 17 \bmod 65 = 41 \cdot 17 \bmod 65 = 47$$

$$\text{return } (1, 47, 58)$$

1.3.7 Unterschreiben

Unter Verwendung des aktuellen Signer-Teilschlüssels SK_t wird zu einer Nachricht m die zugehörige Signatur (ein 4-Tupel) erzeugt.

$$A_{SIG}: (m, SK_t) \mapsto (z, \sigma, t, e_t)$$

$$\text{Sei } SK_t = (t, s_t, S_t)$$

$$\text{Wähle } x \in \mathbb{Z}_n^*$$

$$y := x^{e_t} \bmod n$$

$$\sigma := H(t, e_t, y, m) \text{ wobei } H: \{0, 1\}^* \rightarrow \{0, 1\}^l \text{ Hash-Funktion}$$

$$z := x \cdot S_t^\sigma \bmod n$$

$$\text{return } (z, \sigma, t, e_t)$$

Beispiel:
 $A_{SIG}(3, SK_1) \mapsto \text{Signatur}$

 Sei $SK_1 = (1, 47, 58)$
 $x = 63 \Rightarrow y = 63^{e_t} \bmod 65 = 63^{257} \bmod 65 = 33$
 $\sigma = H(1, 257, 33, 3) = H(110000000110000111) = 215 \quad (H \hat{=} \bmod 251)$
 $z = 63 \cdot 58^{215} \bmod 65 = 56$

 return $(56, 215, 1, 257)$
1.3.8 Verifizieren

Durch Verrechnung des bekannten öffentlichen Schlüssels PK , der als signiert zu verifizierenden Nachricht m sowie des vom Signer erhaltenen Signatur-Tupel wird die Echtheit letzterer geprüft und im Falle eines positiven Ergebnisses eine 1 ausgegeben (ansonsten 0).

 $A_{VER}(m, PK, (z, \sigma, t, e_t)) \rightarrow \{0, 1\}$

 if $e_t \notin [2^l, 2^l(1 + \frac{1}{T})] \setminus 2\mathbb{Z}$ then return(0)

 if $z \bmod n = 0$ then return(0)

 $y' := z^{e_t} \cdot PK^\sigma \bmod n$

 if $\sigma = H(t, e_t, y', m)$ then return(1) else return(0)
Beispiel:
 $A_{VER}(3, 7, (56, 215, 1, 257)) \rightarrow \{0, 1\}$

ok

ok

 $y' := 56^{257} \cdot 7^{215} \bmod 65 = 33$
 $H(1, 257, 33, 3) = H(110000000110000111) = 215 = \sigma \Rightarrow \text{ok}$
Beweis der Korrektheit der Verifikation

Der Verifizierer erkennt eine Unterschrift dann als gültig an, wenn die von ihm und vom Signer berechneten Hash-Werte $H(t, e_t, y', m)$ und $H(t, e_t, y, m)$ übereinstimmen. Im Folgenden wird gezeigt, daß bereits $y' = y$ gilt und daher im Falle einer gültigen Unterschrift der Verifizierer zu einer Bestätigung gelangt. Gemäß der Definition einer Hash-Funktion wäre es einem Betrüger praktisch unmöglich, ein anderes y' zu finden, für das der Test ebenfalls positiv endet.

$$PK = \frac{1}{(s_0 \cdot b_0)^{e_1 \dots e_T}} \bmod n$$

$$\begin{aligned} \text{Signer:} \quad y &= x^{e_t} \bmod n \\ z &= x \cdot S_t^\sigma \bmod n \end{aligned}$$

$$\begin{aligned} \text{Verifizierer:} \quad y' &= z_t^e \cdot PK^\sigma \bmod n \\ &= \frac{(x \cdot S_t^\sigma)^{e_t}}{(s_0 \cdot b_0)^{e_1 \dots e_T \cdot \sigma}} \bmod n \\ &= x^{e_t} \cdot \frac{S_t^{\sigma \cdot e_t}}{(s_T \cdot b_T)^\sigma} \bmod n \\ &= x^{e_t} \cdot \frac{(s_{t-1} \cdot b_{t-1})^{e_t+1 \dots e_T \cdot \sigma \cdot e_t}}{(s_T \cdot b_T)^\sigma} \bmod n \\ &= x^{e_t} \cdot \frac{(s_T \cdot b_T)^\sigma}{(s_T \cdot b_T)^\sigma} \bmod n \\ &= x^{e_t} \bmod n = y \end{aligned}$$

1.4 Sicherheit

1.4.1 Sinn der mathematischen Vorgehensweise

Forward Security

Wie im letzten Abschnitt ersichtlich war, ergeben sich die neuen Teilschlüssel aus den alten durch Potenzierung, also durch Anwendung einer One-Way-Funktion (siehe 1.1.2):

$$b_t := b_{t-1}^{e_t} \bmod n \quad s_t := s_{t-1}^{e_t} \bmod n$$

Die Rekonstruktion alter Schlüssel ist daher so schwer wie das Problem, einen diskreten Logarithmus zu berechnen; die Forderung nach *Forward Security* kann also als erfüllt betrachtet werden.

Key Insulation

Signiert werden kann nur unter Verwendung des aktuellen Base-Schlüssels. Steht dieser einem Dieb des Signer-Teilschlüssels nicht zur Verfügung, ist ihm eine Fortschreibung des gestohlenen nicht möglich, da er folgende Multiplikation aus $APUS$ nicht durchführen kann:

$$S_t := (s_t \cdot b_t)^{e_{t+2} \cdots e_T} \bmod n$$

Eine Fortschreibung des Signierschlüssels durch Potenzierung mit den e_i ist nicht möglich: Bei der Verifikation wird S_t mit e_t potenziert, dieser Exponent darf also nicht bereits in S_t enthalten sein, um ein korrektes Auskürzen zu ermöglichen. Eine Fortschreibung erforderte also wie die Rekonstruktion die Berechnung des diskreten Logarithmus, um einmal „in S_t enthaltene“ Exponenten wieder „entfernen“ zu können.

Verifikations-Invarianz der Zusatz-Updates

Die Zusatz-Updates der Teilschlüssel „heben sich gegenseitig auf“; das zum Leisten einer Unterschrift nötige Produkt der beiden wird hingegen unbrauchbar, wenn auch nur einer der beiden Teilschlüssel älter ist als der andere:

$$b'_t := b_t \cdot ZM_{t,z}^{-1} \bmod n \quad s'_t := s_t \cdot ZM_{t,z} \bmod n$$

1.4.2 Mögliche Probleme

Wie bei jedem Kommunikation involvierenden Protokoll ist auch im Falle der Signer-Base-Signaturen die Möglichkeit des sabotierenden Eingriffs eines Dritten durch gefälschte Nachrichten in Betracht zu ziehen.

Fälschbar sind im beschriebenen Protokoll die zum Update der Teilschlüssel benötigten Perioden- wie Zusatz-Messages. Im ersteren Fall wird der Teilschlüssel des Signers sofort unbrauchbar; der Schaden bestünde also nicht in der etwaigen Möglichkeit gefälschter Signaturen, sondern im zum Aufbau einer neuen Instanz des Signatur-Systems nötigen Aufwand. Ähnliches gilt für eine Fälschung der Zusatz-Message, nur daß die Unbrauchbarkeit erst beim nächsten Perioden-Update offenbar wird.

Wenn auch der Signer keine Backups alter Schlüssel aufbewahren sollte (er würde die Unfälschbarkeit alter Signaturen gefährden), so könnte er allerdings zumindest bei Perioden-Updates durchaus den jeweils aufzufrischenden Teilschlüssel so lange intakt lassen, bis er sich von der Funktionsfähigkeit des neuen überzeugt hat.

In keinem der beschriebenen Fälle jedoch gewinnt der Saboteur relevante Informationen: Die Rekonstruktion alter Schlüssel ist nach wie vor schwer, und ohne Kenntnis des echten Teilschlüssels der Base kann er auch keine zukünftigen Schlüssel errechnen.

Ein unzweifelhaft existierendes Risiko besteht jedoch in der Möglichkeit des Abfangens *echter* Update-Messages: Da das SBS-System darauf aufbaut, durch Updates das Erlangen zusammenpassender Teilschlüssel für feindlich gesinnte Dritte zu erschweren, ist es essenziell, daß eben jene Messages geheim

bleiben. Diesem Problem muß daher durch Nutzung geeignet gesicherter Kommunikationsverbindungen begegnet werden.

1.5 Zusammenfassung

Herkömmliche Signatur-Verfahren bringen einen erheblichen Aufwand für den Fall des Verlusts des privaten Schlüssels mit sich: Nicht nur ist ein neuer öffentlicher Schlüssel zu verbreiten, der wesentliche Nachteil besteht in der mit dem Diebstahl einhergehenden verschwindenden Aussagekraft jeglicher in der Vergangenheit mit dem alten Schlüssel geleisteter Unterschriften.

Signer-Base-Signaturen vermeiden dieses Problem, indem sie die Gültigkeit von Schlüsseln zeitlich begrenzen und die zu deren Fortschreibung nötige Information auf zwei Stellen, eben den Signer und die Base, verteilen: Gestohlene Schlüssel können daher maximal in der aktuellen Zeitperiode zum Unterschreiben eingesetzt werden.

Der Aufbau der vorgestellten Algorithmen zum Update der Teilschlüssel garantiert, daß eine Errechnung alter Schlüssel komplexitätstechnisch unpraktikabel und eine Fortschreibung für zukünftige Perioden nur im Falle der gleichzeitigen Erlangung beider Teilschlüssel möglich ist.

2 Distributed Oblivious Transfer

2.1 Einführung

Bekannte, aber in der Regel unabhängig voneinander betrachtete Konzepte der Kryptographie stellen *Verteilte Geheimnisse* sowie der sogenannte *Oblivious Transfer* dar: Ersteres bezeichnet ein System, bei dem Informationen auf mehrere Stellen verteilt sind und nur durch deren Kooperation rekonstruiert werden können, letzteres die Übertragung genau einer Information aus einer größeren Menge, ohne daß der Informierende erfährt, welche gewählt und übertragen wurde.

2.1.1 Verteilte Geheimnisse

Verteilte Geheimnisse können beispielsweise Anwendung finden, wenn garantiert werden soll, daß die Unterschrift unter einen Vertrag nur dann erfolgen kann, wenn eine bestimmte Anzahl Mitglieder des betreffenden Entscheidungsgremiums diesem zustimmen und daher ihre Teil-Geheimnisse beisteuern, die zusammen eine Signatur zum vorliegenden Dokument ergeben. (Hierbei darf durch Kooperation natürlich nur die passende Signatur, nicht aber der Schlüssel selbst bekannt werden, weil sonst fortan einer der Beteiligten alleine unterschreiben könnte.)

Als Beispiel eines alleine der gemeinsamen Rekonstruktion eines Geheimnisses dienenden Verfahrens sei folgendes, von Shamir [3] entwickelte Protokoll angegeben:

s Geheimnis, m Anzahl der Server, r Anzahl der zu befragenden Server

- Wähle eine Primzahl $p > m$
- Wähle Polynom $f(x)$ mit Werten in \mathbb{Z}_p vom Grad $r - 1$ mit $f(0) = s$
- Jeder der Server i bekommt den Wert $f(i)$
- Empfänger fragt r Server nach ihren Funktionswerten
- Er interpoliert aus den Wertepaaren $(i, f(i))$ das Polynom $f(x)$
- ...und berechnet das Geheimnis als Wert $f(0)$

2.1.2 Oblivious Transfer

Verfahren des Oblivious Transfer finden Anwendung, wenn einem Interessenten an einer Information gelegen ist, aber ebenfalls daran, daß nicht einmal seine Informationsquelle erfährt, welches der verfügbaren Geheimnisse nun übermittelt wurde.

Ein dies realisierendes, auf der Verfügbarkeit eines kommutativen Verschlüsselungssystems aufbauendes, Protokoll (aus [2]) ist folgendes:

E_S, D_S Ver-/Entschlüsselungsfunktion Sender

E_R, D_R Ver-/Entschlüsselungsfunktion Empfänger

s_1, \dots, s_n Geheimnisse

- Empfänger wählt und übermittelt Zufallszahlen z_1, \dots, z_n
- Sender antwortet mit $y_i := E_S(s_i \oplus z_i)$

- Empfänger berechnet und verschickt $x = E_R(y_\sigma)$ für ein σ seiner Wahl
- Sender sendet $x' = D_S(x)$ an den Empfänger
- Empfänger ermittelt $D_R(x') = D_R(D_S(x)) = D_R(D_S(E_R(y_\sigma))) = D_S(y_\sigma) = D_S(E_S(s_\sigma \oplus z_\sigma)) = s_\sigma \oplus z_\sigma$
- ...und bestimmt $s_\sigma = s_\sigma \oplus z_\sigma \oplus z_\sigma$

2.1.3 DOT

Das in diesem Kapitel vorzustellende Protokoll (aus [4]) vereinigt die beiden oben vorgestellten Konzepte: Gegeben sind Geheimnisse s_0, \dots, s_{n-1} und einen \mathcal{R} genannten *Empfänger*, der an einem bestimmten Geheimnis s_σ interessiert ist. Die Geheimnisse wurden von einem *Sender* \mathcal{S} auf diverse *Server* S_i verteilt, von denen \mathcal{R} eine Mindestanzahl kontaktieren muß, um s_σ zu erfahren.

Die sinnvollerweise an ein Protokoll des *Distributed Oblivious Transfer (DOT)* gestellten Forderungen lauten wie folgt:

- Aus den Informationen von r Servern kann der Empfänger s_σ rekonstruieren
- \mathcal{R} darf nur s_σ und keine anderen Geheimnisse erfahren
- Um σ zu ermitteln, müssen mindestens t Server zusammenarbeiten
- \mathcal{R} muß sich mit mehr als l Servern zusammentun, um über s_σ hinaus weitere Geheimnisse zu erhalten

Ein diese Forderungen erfüllendes Protokoll kann daher kompakt mit (r/m) -DOT- $(1/n)$ bezeichnet werden. Sein, von einer konkreten Implementierung unabhängiger, genereller Ablauf sieht wie folgt aus:

- Sender \mathcal{S} schickt jedem Server S_i ein „Programm“ P_i
- Empfänger \mathcal{R} schickt r der Server je eine Anfrage q_i
- Jeder angesprochene Server S_i reagiert mit Antwort a_i
- \mathcal{R} setzt das ausgewählte Geheimnis aus a_1, \dots, a_r zusammen

2.1.4 Parameter

Bevor im nächsten Abschnitt die konkrete Implementierung eines (r/m) -DOT- $(1/n)$ -Protokolls vorgestellt wird, seien an dieser Stelle noch einmal die vorkommenden Variablen aufgelistet und exemplarisch kohärente, später auch für das die Implementierung illustrierende Beispiel verwendete, Werte genannt:

- n Geheimnisse aus \mathbb{Z}_p (davon Nr. σ gewünscht)
- m Server (davon r mindestens zu befragende)
- Sicherheitsparameter:
erst t Server können σ aufdecken, erst $l+1$ zusammen mit dem Empfänger alle Geheimnisse ermitteln
- $r \geq t + l$

Beispiel

- $n = 4$, $p = 7$, $(s_i) = (6, 1, 2, 0)$, $\sigma = 2$ (d.h. gewünscht ist $s_2 = 2$)
- $m = 7$, $r = 5$
- $t = 2$, $l = 3$

2.2 Implementierung

2.2.1 Initialisierung (seitens \mathcal{S})

- Generiere Polynome $B_0(x), \dots, B_{n-1}(x)$, wobei

$$\begin{aligned} \deg(B_0) &= r - 1, & B_0(0) &= s_0 \\ \deg(B_j) &= l, & B_0(0) + B_j(0) &= s_j \text{ für } j = 1, \dots, n - 1 \end{aligned}$$

- Generiere Polynom $Q(x, y_1, \dots, y_{n-1}) = B_0(x) + \sum_{j=1}^{n-1} B_j(x) \cdot y_j$
- Sende Polynom $Q(i, y_1, \dots, y_{n-1})$ an alle Server S_i

Beispiel

- $B_0 = 3x^4 + 2x^2 + x + 6$
 $B_1 = 2x^3 + 6x + 2$
 $B_2 = x^3 + x^2 + 3x + 3$
 $B_3 = 5x^3 + 1$
- $Q(x, y_1, y_2, y_3) = 3x^4 + 2x^2 + x + 6$
 $+ y_1(2x^3 + 6x + 2) + y_2(x^3 + x^2 + 3x + 3) + y_3(5x^3 + 1)$
- $Q(1, y_1, y_2, y_3) = 3y_1 + y_2 + 6y_3 + 5$
 $Q(2, y_1, y_2, y_3) = 2y_1 + 6y_3 + 1$
 $Q(3, y_1, y_2, y_3) = 4y_1 + 6y_2 + 3y_3 + 4$
 $Q(4, y_1, y_2, y_3) = 4y_2 + 6y_3 + 5$
 $Q(5, y_1, y_2, y_3) = 2y_1 + 3y_3 + 4$
 $Q(6, y_1, y_2, y_3) = y_1 + 3y_3 + 3$
 $Q(7, y_1, y_2, y_3) = 2y_1 + 3y_2 + y_3 + 6$

2.2.2 Anfrage (seitens \mathcal{R})

- Generiere Polynome $D_1(x), \dots, D_{n-1}(x)$ vom Grad $t - 1$, wobei
 $D_\sigma(0) = 1$ und $D_{\neq\sigma}(0) = 0$ (falls $\sigma = 0$, dann $D_j(0) = 0$ für alle j)
- Sende den gewählten r Servern S_i die Anfrage $q_i = (D_1(i), \dots, D_{n-1}(i))$

Beispiel

- $D_1 = x$
 $D_2 = 2x + 1$
 $D_3 = 5x$
- $q_1 = (1, 3, 5), q_2 = (2, 5, 3), q_4 = (4, 2, 6)$
 $q_5 = (5, 4, 4), q_6 = (6, 6, 2)$

2.2.3 Antwort und Geheimniserhalt

- Die r Server S_i schicken $a_i = Q(i, D_1(i), \dots, D_{n-1}(i))$ an Empfänger \mathcal{R}
- \mathcal{R} interpoliert ein Polynom $V(x)$ aus den Wertepaaren (i, a_i) und erhält das gewünschte Geheimnis als $V(0)$

Beispiel

- $a_1 = Q(1, 1, 3, 5) = 6$
 $a_2 = Q(2, 2, 5, 3) = 2$
 $a_4 = Q(4, 4, 2, 6) = 0$
 $a_5 = Q(5, 5, 4, 4) = 5$
 $a_6 = Q(6, 6, 6, 2) = 1$
- Zu interpolieren: $(1, 6)$ $(2, 2)$ $(4, 0)$ $(5, 5)$ $(6, 1)$
(z.B. mittels Newton-Verfahren)

Interpolation (hier: Newton-Methode)

$$\begin{array}{r|l}
1 & 6 \\
2 & 2 \\
4 & 0 \\
5 & 5 \\
6 & 1
\end{array}
\begin{array}{l}
\frac{2-6}{2-1} = 3 \\
\frac{0-2}{0-2} = 6 \\
\frac{5-0}{5-0} = 5 \\
\frac{1-5}{6-5} = 3
\end{array}
\begin{array}{l}
\frac{6-3}{4-1} = 1 \\
\frac{5-6}{5-6} = 2 \\
\frac{3-5}{3-5} = 6
\end{array}
\begin{array}{l}
\frac{2-1}{5-1} = 2 \\
\frac{6-2}{6-2} = 1
\end{array}
\begin{array}{l}
\frac{1-2}{6-1} = 4
\end{array}$$

$$\begin{aligned}
V(x) &= 6 \\
&+ 3(x-1) \\
&+ 1(x-1)(x-2) \\
&+ 2(x-1)(x-2)(x-4) \\
&+ 4(x-1)(x-2)(x-4)(x-5) \\
&= 4x^4 - 4x^3 + x^2 - 4x + 2 \quad \Rightarrow \quad V(0) = 2 = s_\sigma
\end{aligned}$$

2.3 Mathematische Struktur**2.3.1 Wieso funktioniert das Verfahren?**

Um nachvollziehen zu können, wieso das vorgestellte Verfahren erfolgreich das gewünschte Geheimnis übermittelt, sei zunächst noch einmal an das vom Sender generierte Polynom erinnert:

$$\begin{aligned}
Q(x, y_1, y_2, y_3) &= 3x^4 + 2x^2 + x + 6 \\
&+ y_1(2x^3 + 6x + 2) \\
&+ y_2(x^3 + x^2 + 3x + 3) \\
&+ y_3(5x^3 + 1)
\end{aligned}$$

Die wesentlichen Schritte des Protokolls lauten nun wie folgt:

- Der Sender stattet jeden Server S_i mit $Q(i, \dots)$ aus
- Der Empfänger sendet den befragten Servern Tripel $(i, 2i + 1, 5i)$ (entsprechend der von ihm generierten Polynome D_j)

Wenn nun jeder Server S_i dem Empfänger den Wert $Q(i, i, 2i + 1, 5i)$ schickt, kann dies jeweils als Polynom $Q'(i)$ aufgefaßt werden, da nun alle ehemaligen Parameter des Ausgangs-Polynoms $Q(x, y_1, y_2, y_3)$ durch die jeweilige Servernummer i ausgedrückt werden.

Eine Interpolation von $Q'(i)$ an der Stelle 0 liefert daher den gleichen Wert wie $Q(0, 0, 1, 0)$. Eine Betrachtung des Ausgangs-Polynoms für $i = 0$ offenbart, wieso die Interpolation genau das gewünschte Geheimnis s_σ liefert:

$$\begin{aligned}
Q(x, y_1, y_2, y_3) &= 6 \\
&+ y_1 \cdot 2 \\
&+ y_2 \cdot 3 \\
&+ y_3 \cdot 1
\end{aligned}$$

Einerseits waren die Funktionswerte an der Stelle $x = 0$ jener Teilpolynome, zu denen die y_j die Koeffizienten darstellen, genau so gewählt, daß die Addition einer von ihnen zum Wert (an der gleichen Stelle) des ersten Summanden (hier: 6) das jeweilige Geheimnis ergibt. Andererseits hat der Empfänger die Koeffizienten y_j durch Wahl der Polynome D_j so bestimmt, daß genau das zum gewünschten Geheimnis passende y_j den Wert 1 besitzt. Damit hat sozusagen der Empfänger bestimmt, welche Teilpolynome an der Stelle $i = 0$ relevant werden, während der Sender dafür gesorgt hat, daß die Funktionswerte dieser Polynome an der gleichen Stelle auch das gewünschte Geheimnis ergeben:

$$Q'(0) = Q(0, 0, 1, 0) = s_2$$

2.3.2 Qualität des vorgestellten Verfahrens

Geheimhaltung der Wahl des Empfängers

Da der Empfänger Polynome vom Grad $t - 1$ generiert, müssen mindestens t Server zur Aufdeckung von σ (mittels Interpolation) zusammenarbeiten, was genau der an das Verfahren gestellten Forderung entspricht.

Kein Bekanntwerden anderer Geheimnisse

Der Sender generiert ein Polynom vom Grad $r - 1$ ($\geq l$), so daß selbst l zusammenarbeitende Server keine zusätzlichen Geheimnisse erfahren – auch dieser Wert entspricht der diesbezüglich geforderten Schranke.

Forderung in kompletter Schärfe notwendig?

Nachdem das vorliegende Protokoll offenbar die verlangten Bedingungen erfüllt, stellt sich die Frage, ob dies auch ohne strikte Einhaltung der im Protokoll verankerten Ungleichung $r \geq t + l$ möglich gewesen wäre. Daß dem nicht so ist, soll im nächsten Abschnitt gezeigt werden.

2.4 Sicherheit

2.4.1 Exkurs: Entropie

Zum Beweis der Unmöglichkeit von die gestellten Forderungen erfüllenden, aber die Ungleichung $r \geq t + l$ verletzenden Protokollen greifen wir auf das Konzept der *Entropie* zurück. Die Entropie $H(X)$ einer Zufallsvariablen X ist zunächst definiert als

$$H(X) := - \sum_{x \in X} P_X(x) \log P_X(x) \geq 0$$

Sie kann verstanden werden als ein Maß für die Gleichverteilung von X . Da Zufallsvariablen für uns aber nur Mittel zum Zweck sind, stelle man sich Entropie im Folgenden als Maß für Informationsgehalt vor.

Neben obiger Definition benötigen wir auch den Begriff der *Bedingten Entropie* $H(X|Y)$ von X unter Y :

$$H(X|Y) := - \sum_{y \in Y} \sum_{x \in X} P_Y(y) P_{X|Y}(x|y) \log P_{X|Y}(x|y) \geq 0$$

Analog zur vorangegangenen intuitiven Interpretation handelt es sich hierbei um den Informationsgehalt von X bei bekanntem Y . Eine detailliertere Beschreibung findet sich in [5].

Beweisstrategie

Wie aber soll nun der angesprochene Beweis funktionieren? Die Unmöglichkeit zulässiger DOT-Protokolle ohne Beachtung von $r \geq t + l$ ist bewiesen, wenn gezeigt werden kann, daß unabhängig von der konkreten

Implementierung eine Nichterfüllung der Ungleichung den Erfolg einer Koalition des Empfängers mit nur l Servern impliziert – im Gegensatz zur Voraussetzung, daß dies frühestens bei $l + 1$ Komplizen möglich sein soll.

Definition von Zufallsvariablen

Um aber das Konzept der Entropie auf den vorliegenden Fall anwenden zu können, muß dieser zunächst mittels Zufallsvariablen modelliert werden:

- Q_i Anfrage an Server i
- A_i Antwort des Servers i
- R Programm des Empfängers
- P_i Programm des Servers i
- C_i Kommunikationsprotokoll des Servers i
- W Wahl einer Geheimnisfolge (s_1, \dots, s_n)
- T Wahl eines Index $\sigma \in \{1, \dots, n\}$

Beispiel: Umformulierung der Anforderungen

Bevor wir zum eigentlichen Beweis kommen, sei die intendierte Bedeutung und Verwendung der oben definierten Zufallsvariablen anhand Umformulierungen der an DOT-Protokolle gestellten Anforderungen illustriert:

- $t - 1$ Server können die Wahl σ nicht aufdecken:

$$H(T \mid P_1 \dots P_{t-1} C_1 \dots C_{t-1}) = H(T)$$
- Der Empfänger erfährt auf regulärem Wege nur ein Geheimnis:

$$H(W \setminus W_T \mid R C_1 \dots C_r W_T T) = H(W \setminus W_T)$$
- Kooperation mit l Servern verschafft ihm keine zusätzlichen Geheimnisse:

$$H(W \mid R C_1 \dots C_l P_1 \dots P_l T) = H(W)$$
- ...auch dann nicht, wenn er bereits ein Geheimnis erfahren hat:

$$H(W \setminus W_T \mid R C_1 \dots C_r P_1 \dots P_l T W_T) = H(W \setminus W_T)$$

Beweis der Notwendigkeit der Forderung $r \geq t + l$

Behauptung: Gilt $r < t + l$, kann der Empfänger zusammen mit l Servern alle Geheimnisse erfahren.

Beweis: Sei $r = l + t - 1$ („bestmögliches“ r), dann gilt

$$\begin{aligned}
 & H(W \mid P_1 \dots P_l Q_{l+1} \dots Q_r A_{l+1} \dots A_r R) \\
 \leq & H(W_1 \dots W_n \mid P_1 \dots P_l Q_{l+1} \dots Q_r A_{l+1} \dots A_r R) \\
 = & \sum_{i=1}^n H(W_i \mid W_1 \dots W_{i-1} P_1 \dots P_l Q_{l+1} \dots Q_r A_{l+1} \dots A_r R) \\
 \leq & \sum_{i=1}^n H(W_i \mid P_1 \dots P_l Q_{l+1} \dots Q_r A_{l+1} \dots A_r R) \\
 = & \sum_{i=1}^n H(W_i \mid P_1 \dots P_l Q_1 \dots Q_r A_1 \dots A_r R) \\
 \leq & \sum_{i=1}^n \underbrace{H(W_i \mid Q_1 \dots Q_r A_1 \dots A_r R)}_{=0} = 0
 \end{aligned}$$

Offenbar ist die bedingte Entropie der die Geheimnisse modellierenden Zufallsvariable gleich 0, wenn das Programm R des Empfängers, die Programme P_i von l Servern sowie die „regulären“ Antworten A_i

der zu einer Gesamtzahl von r noch fehlenden restlichen Servern bekannt sind. Die Behauptung ist somit gezeigt.

2.5 Zusammenfassung

Ziel des *Distributed Oblivious Transfer* ist die Vereinigung zweier kryptographischer Konzepte, um die Übermittlung exakt eines aus einer Menge von Geheimnissen zu ermöglichen

- wenn eine Mindestanzahl von Stellen befragt wird, auf die die Geheimnisse verteilt sind und
- wobei diesen Stellen verborgen bleibt, welches Geheimnis eigentlich übertragen wurde.

Das dies realisierende vorgestellte Protokoll kodiert die Geheimnisse durch Polynome, die Dekodierung erfolgt entsprechend durch Interpolation im Zuge des Verfahrens erhaltener Wertepaare.

Die Art der jeweils übermittelten Informationen verhindert die Aufdeckung der Geheimniswahl; durch Wahl des Grades der verwendeten Polynome kann die Zahl der mindestens zu befragenden Stellen (Server) bestimmt werden. Durch Anpassung der Sicherheitsparameter entsprechend einer als notwendig bewiesenen Ungleichung wird garantiert, daß kleine Koalitionen betrügerischer Teilnehmer zwecklos sind und das Verfahren diesbezüglich als robust anzusehen ist.

2.6 Ausblick

Bei der vorgestellten Variante des Distributed Oblivious Transfer sind die Mengen der zu befragenden oder nicht erfolgreich zusammenarbeiten sollenden Server schlicht durch Mindest- bzw. Maximalwerte ihrer Anzahl bestimmt. Diese Vorgehensweise paßt nicht unbedingt zu realen Situationen, in denen der Empfänger verschiedenen konkreten Servern unterschiedliches Vertrauen entgegenbringt: Es mag z.B. solche geben, die seinem Empfinden nach sogar durchaus erfahren dürften, welches Geheimnis er erfahren möchte, während einer gewissen Gruppe von Servern selbst bei kompletter Zusammenarbeit keine Erkenntnisse über jegliche Geheimnisse zugestanden werden sollen.

So könnte man drei Mengen von Servern wie folgt definieren:

Γ Qualifizierte Server (zur Anfrage seitens des Empfängers geeignet)

Δ_1 Verbotene Server (dürfen im Zuge des Verfahrens die Wahl des Empfängers nicht erfahren)

Δ_2 Korrupte Server (dürfen auch bei Kooperation mit dem Empfänger keine zusätzlichen Geheimnisse erhalten)

Eine Zusammenstellung solcher Mengen wird als *General Access Structure* bezeichnet. In [4] wird ein diese Klassifizierung von Servern respektierendes Protokoll vorgestellt sowie folgende bei dessen Anwendung zu berücksichtigende notwendige Bedingung für die beteiligten Mengen aufgeführt:

$$\Gamma \cap (\Delta_1 * \Delta_2) = \emptyset$$

Hierbei steht $*$ für die die Anwendung der elementweisen Vereinigung auf die beteiligten Mengen. Die genannte Gleichung spielt bei General Access Structures die Rolle der Ungleichung, die im vorangegangenen Kapitel als für das vorgestellte (r/m) -DOT- $(1/n)$ vorauszusetzen gezeigt wurde.

Literaturverzeichnis

- [1] G. Itkis, L. Reyzin: *SiBIR: Signer-Base Intrusion-Resilient Signatures*, CRYPTO 2002, LNCS 2442, pp. 499-514
- [2] W. Unger: *Algorithmische Kryptographie*, Vorlesung im WS 02/03 an der RWTH Aachen
- [3] A. Shamir: *How To Share A Secret*, Communications of the ACM 1979, 22(11), pp. 612-613
- [4] V. Nikov, S. Nikova, B. Preneel, J. Vandewalle: *On Unconditionally Secure Distributed Oblivious Transfer*, INDOCRYPT 2002, LNCS 2551, pp. 395-408
- [5] C. Blundo, P. D'Arco, A. De Santis, D. Stinson: *New Results on Unconditionally Secure Distributed Oblivious Transfer*, SAC 2002, LNCS 2595, pp. 291-309