RUHR-UNIVERSITÄT BOCHUM

Privacy-Preserving Authentication

3 – Interaktive Beweissysteme, Sigma Protokolle und Zero-Knowledge

WS 2015/2016

Sven Schäge,
Fakultät für Mathematik,
Ruhr-Universität Bochum

Übersicht

Organisation der Vorlesung

- Einleitung
- Kurze Einführung in wichtige kryptografische Bausteine
  - Hash Funktionen
  - Digitale Signaturen
  - Verschlüsselungsverfahren (Public Key Encryption)
  - Commitmentverfahren
- Interaktive Beweissysteme
- (Nicht-interaktive) Zero-Knowledge Beweissysteme
- Ringsignaturen
- Gruppensignaturen
- Direct Anonymous Attestation
- Anonymous Credential Systems
- Weitere Anwendungen

Übersicht

Relation, Sprache, Statement, Witness

- Eine Relation über X, Y ist eine Menge von Paaren $R \subseteq X \times Y$.
- Wir betrachten Relationen, zu denen es Algorithmen gibt, die gegeben z, effizient erkennen können, ob $z \in X$ oder $z \in Y$.
- Wir definieren Funktion (Prädikat) $R(\cdot, \cdot) : X \times Y \to \{0, 1\}$ wie folgt:

$$R(x, y) = 1 \iff (x, y) \in R.$$

- Wir nennen $y \in Y$ Statement. Falls $(x, y) \in R$, so nennen wir x Witness (Zeuge) (für y).
- Sprache L_R über Relation R ist die Menge aller Statements, für die ein Witness existiert:

$$L_R = \{y | \exists x \in X \text{mit} (x, y) \in R\} \subseteq Y.$$

Interaktives Beweissystem

- Intuitiv ist ein interaktives Beweissystem für R ein Protokoll, mit dem ein Prover P einen Verifier V davon überzeugen kann, dass es für y einen Witness gibt.
- Sei $R \subseteq X \times Y$ eine Relation. Ein interaktives Beweissystem (interactive proof system IP) für Sprache $L_R = \{y | \exists x \in X \text{ mit } R(x, y) = 1)\} \subseteq Y$ mit $2q + 1$ Zügen ist ein Tupel von Algorithmen (IP.Pgen, IP.KGen, $\{\mathrm{V}_i\}_{i \in [q+1]}, \{\mathrm{P}_i\}_{i \in [q+1]}$).
- IP.Pgen(1^κ) generiert öffentliche Parameter pp für Relation R. Diese werden an P und V gegeben.
- IP.KGen(pp) generiert ein Paar $(x, y) \leftarrow_\$ R$. Das Statement y wird verwendet als öffentlicher Schlüssel $\mathrm{pk} = y$ und veröffentlicht. Der Witness x wird verwendet als Secret Key $\mathrm{sk} = x$ und vom Prover geheim gehalten.

Interaktives Beweissystem

- Algorithmus P_i für $i \in [1; q+1]$ berechnet eine öffentliche Ausgabe p_i und eine private Ausgabe (Zustand) pst_i. Eingaben sind sk, pk, pst_{i-1} und die öffentliche Ausgabe von V_{i-1}, v_{i-1}. Es gilt $pst_0 = v_0 := \varepsilon$.
- Algorithmus V_i für $i \in [1; q+1]$ berechnet eine öffentliche v_i und eine private Ausgabe (Zustand) vst_i. Eingaben sind pk, vst_{i-1} und die öffentliche Ausgabe von P_i, p_i. Es gilt $vst_0 := \varepsilon$. V_{q+1} ist deterministisch und gibt 1 aus (akzeptiert) wenn V_{q+1} davon überzeugt ist, dass pk $\in L_R$ gilt.
- Wir verwenden P kurz für $\{\mathsf{P}_i\}_{i\in[1;q+1]}$ und V kurz für $\{\mathsf{V}_i\}_{i\in[q+1]}$. Wir identifizieren die Partei, welche die Algorithmen P (V) aufruft oft mit P (V).

Interaktives Beweissystem, Notation

- Alle Algorithmen V_i, P_i sind zustandslos. Werden Zustandsinformationen benötigt, so werden diese in den Variablen vst_i und pst_i abgelegt und an den nächsten Algorithmus V_{i+1}, P_{i+1} weitergegeben, bspws. das bisherige Kommunikationstranskript.
- Alternativ können wir uns *einen* zustandsbehafteten Algorithmus V (P) vorstellen, der intern die V_i (P_i) aufruft, die Zustände vst_i (pst_i) speichert und sie stets richtig weitergibt.
- Bis auf V_{q+1} können alle Algorithmen V_i, P_i probabilistisch sein. Falls erforderlich, machen wir dies explizit. Dann nehmen die Algorithmen zusätzlich zu allen anderen Eingaben die zufälligen Werte vr_i oder pr_i als Eingabe. In diesem Fall können wir die Algorithmen V_i, P_i als deterministisch betrachten.
- Wir nutzen $vst_{q+1} =< \mathrm{P}(\mathrm{pp}, sk, pk), \mathrm{V}(\mathrm{pp}, pk) >$, um zu beschreiben, dass das Protokoll zwischen den Parteien P und V ausgeführt wird und V im letzten Schritt vst_{q+1} ausgibt.

Vollständigkeit/Completeness

- Wir sagen IP hat perfect completeness (vollständig) wenn der Verifier immer akzeptiert, falls alle Algorithmen ehrlich ausgeführt wurden.

$$\Pr\left[\begin{array}{c} 1 =< \mathsf{P}(\mathsf{pp}, sk, pk), \mathsf{V}(\mathsf{pp}, pk) >; \\ (sk, pk) \leftarrow_{\$} \mathsf{IP.KGen}(\mathsf{pp}),\ \mathsf{pp} \leftarrow_{\$} \mathsf{IP.Pgen}(1^{\kappa}) \end{array}\right] = 1.$$

Soundness und Proof-of-Membership

- Soundness schützt den Verifier vor unehrlichen Provern.
- Betrachten wir das folgende Sicherheitsspiel:
  - Der Angreifer $\mathcal{A}$ erhält $\mathsf{pp} \leftarrow_\$ \mathsf{IP.Pgen}(1^\kappa)$.
  - Der Angreifer gibt $pk \in Y$ aus.
  - Der Angreifer kommuniziert mit dem Challenger. Dabei spielt der Challenger die Rolle des Verifiers und der Angreifer die Rolle das Provers. Der Angreifer darf beliebig von den Protokollvorgaben abweichen, der Challenger folgt den Algorithmen für den Verifier.
  - Am Ende gibt der Challenger vst_{q+1} aus. Der Angreifer gewinnt wenn $vst_{q+1} = 1$ und $pk \in Y \setminus L_R$.
- Der Vorteil des Angreifers ist seine Erfolgswkt. in diesem Spiel zu gewinnen:

$$\begin{aligned} & \Pr\left[\begin{array}{c} 1 =< \mathcal{A}(\mathsf{pp}, sk, pk), \mathsf{V}(\mathsf{pp}, pk) >; \\ pk \leftarrow_\$ \mathcal{A}(\mathsf{pp}),\ pk \in Y \setminus L_R, \mathsf{pp} \leftarrow_\$ \mathsf{IP.Pgen}(1^\kappa) \end{array}\right] \\ = \quad & \mathsf{Adv}^{\mathrm{sound}}_{\mathcal{A},\mathsf{IP}}(\kappa). \end{aligned}$$

Soundness und Proof-of-Membership

- Im Allgemeinen muss (lediglich) gelten $\mathrm{Adv}^{\mathrm{sound}}_{\mathcal{A},\mathsf{IP}}(\kappa) \leq 1 - \mathrm{negl}(\kappa)$. Meist definiert man einfach $\mathrm{Adv}^{\mathrm{sound}}_{\mathcal{A},\mathsf{IP}}(\kappa) \leq 1/2$.
- Durch t-fache Wiederholung kann der Vorteil des Angreifers dann auf $(1/2)^t$ reduziert werden.
- Intuitiv beweist dies, dass y tatsächlich zur Sprache L_R gehört. Wir sprechen auch von einem *Proof-of-Membership*.
- Falls der Angreifer im Soundness-Spiel unbeschränkt ist, so nenne wir IP (im engeren Sinne) ein interaktives Beweissystem.
- Falls der Angreifer im Soundness-Spiel eine PPT ist, so nenne wir IP ein Argumentsystem.
- Neben dieser allgemeinen Definition von Soundness gibt es noch eine weitere wichtige Soundness-Definition, die wir etwas später kennen lernen werden.

Übersicht

- Sigma-Protokolle sind eine wichtige Klasse von Sicherheitsprotokollen.
- Die interaktive Phase bei Sigma-Protokollen besteht aus drei Zügen.
- Zuerst sendet der Prover eine Nachricht $a \in A$ (oft auch mit t bezeichnet) an den Verifier. Diese Nachricht wird manchmal *Commitment* genannt.
- Anschließend antwortet der Verifier mit Nachricht $c \in C$. Diese Nachricht wird oft *Challenge* genannt. Meist wählt der Verifier c dazu zufällig aus einer Menge C.
- Die letzte Nachricht wird meist mit $s \in S$ oder z bezeichnet und üblicherweise *Response* oder *Solution* genannt.
- Wir nutzen $(a, c, s) \leftarrow_{\$} < \mathsf{P}, \mathsf{V} >$ um das Transkript zu bezeichnen, dass in einer Kommunikation zwischen P und V entstand.
- Zur besseren Übersicht machen wir die Eingaben pp, pk in die Algorithmen meist nicht mehr explizit.

Beispiel: Sigma-DDH. Proof-of-Membership.

- Seien, p, q prim, $S \subset \mathbb{Z}_p^*$ mit $|S| = q, q|p-1$ und g, h mit $<g>=<h>= S$. pp $:= (g, h, p, q)$.

Sigma-DDH: Proof-of-Members. für $L_R = \{(g', h') | \exists \alpha : (g', h') = (g^\alpha, h^\alpha)\}$

P		V
$sk = x \leftarrow_\$ \mathbb{Z}_q, pk = (pk_1, pk_2) = (g^x, h^x)$		
$r \leftarrow_\$ \mathbb{Z}_q, a = (a_1, a_2) = (g^r, h^r)$	$\xrightarrow{a}$	
	$\xleftarrow{c}$	$c \leftarrow_\$ C$
$s = cx + r$	$\xrightarrow{s}$	
		$pk_1^c a_1 \stackrel{?}{=} g^s \wedge pk_2^c a_2 \stackrel{?}{=} h^s$

Completeness and Soundness

- Perfect completeness:

$$\begin{aligned} pk_1^c a_1 &= (g^x)^c g^r = g^{xc+r} = g^s, \\ pk_2^c a_2 &= (h^x)^c h^r = h^{xc+r} = h^s. \end{aligned}$$

- Statistical Soundness: es gibt stets $x, v, r, w \in \mathbb{Z}_q$ mit $pk_1 = g^x$, $pk_2 = h^{x+v}$, $a_1 = g^r$ und $a_2 = h^{r+w}$. Betrachten wir nun die zugehörigen Verifikationsgleichungen, die erfüllt sein müssen:

$$\begin{aligned} pk_1^c a_1 &= (g^x)^c g^r = g^{xc+r} = g^s, \\ pk_2^c a_2 &= (h^{x+v})^c h^{r+w} = h^{(x+v)c+r+w} = h^s. \end{aligned}$$

- Einsetzen ergibt: $(x+v)c + r + w = xc + r$ woraus $vc + w = 0$ folgt.
- D.h. für jedes $v \neq 0$ gibt es nur einen Wert c, der beide Verifikationsgleichungen erfüllbar macht. c wird aber zufällig gezogen nachdem v und w festgelegt wurden. Ein passendes $c = -w/v$ wird daher nur mit Wkt. $|C| = 1/q$ gezogen.

Übersicht

Special Soundness

- Special Soundness schützt den Verifier ebenfalls vor unehrlichen Provern.
- Die Frage ob $y \in L_R$ tritt allerdings dabei eher in den Hintergrund. Insbesondere kann sie trivial beantwortbar sein.
- Special Soundness garantiert, dass das Protokoll einen sogenannten Proof-of-Knowledge implementiert. Das bedeutet, dass der Verifier sich sicher sein kann, dass der Prover den/einen Witness tatsächlich kennt. Notation:

$$PoK\{(\alpha) : (\alpha, y) \in R\} = PoK\{(\alpha) : \alpha \text{ ist secret key zu } pk\}.$$

- Wir sagen IP hat Special Soundness wenn es einen effizienten Algorithmus – den Extraktor Ext – gibt, der für zwei akzeptierende Transkripte mit gleichem Commitment a, c, s, c', s' den Secret Key berechnen kann:

$$\mathsf{sk} \leftarrow \mathsf{Ext}(\mathsf{pp}, \mathsf{pk}, a, c, s, c', s').$$

Proof-of-Knowledge

- Es bleibt zu klären, wie man an zwei gültige Transkripte mit gleichem a kommt, um Ext laufen zu lassen.
- Wir nehmen dazu in einem Gedankenexperiment an, dass es einen effizienten Prover P^* gibt, der den Verifier mit nicht-vernachlässigbarer Wkt. ϵ überzeugen kann.
- Wir beschreiben nun einen effizienten Extraktor $(\text{Ext}^*)^{P^*}(\text{pp}, pk)$, der mit P^* kommuniziert und zwei valide Transkripte mit gleichem a generiert. Wir stellen uns dazu vor, dass der Ext^* den effizienten P^* auf einem Computersystem laufen lässt. Wir nutzen nun aus, dass wir uns P^* als ein Paar von zustandslosen effizienten, deterministischen Algorithmen P_1^*, P_2^* vorstellen können. Nur P_2^* hängt von c ab.
- Die zufälligen Münzen, die P_1^*, P_2^* verbrauchen $(pr_1 \leftarrow\!\!\$\, PR_1, pr_2 \leftarrow\!\!\$\, PR_2)$, werden von Ext^* bereitgestellt.

Proof-of-Knowledge

- Ext* zieht $pr_1 \leftarrow_\$ PR_1$, lässt $P_1^*(pk, pr_1)$ laufen und erhält a und den Zustand pst_1 zurück .
- Ext* zieht $c \leftarrow_\$ C$, $pr_2 \leftarrow_\$ PR_2$ und ruft $s \leftarrow_\$ P_2^*(pk, pst_1, c, pr_2)$ auf.
- Ext* zieht $c' \leftarrow_\$ C$, $pr_2' \leftarrow_\$ PR_2$ und ruft $s' \leftarrow_\$ P_2^*(pk, pst_1, c', pr_2')$ auf. Wir können uns gleichermaßen vorstellen, das Ext* den zustandsbehafteten P^* bis zum Punkt nach der Ausgabe von a *zurückspult* (rewinding) und noch einmal mit frischen Werten laufen lässt.

Erfolgswkt. von Ext^*

- Sei $E = PR_1$ und $F = C \times PR_2$.
- Nehmen wir an, $P^*(pk)$ ist erfolgreich mit Wkt. ϵ. Sei $G \subset E \times F$ der Anteil aller "'guten"' Paare (e, f), also derjenigen Paare, die den Verifier akzeptieren lassen. Es gilt dann $|G| \geq \epsilon |E \times F|$.

Lemma (Splitting Lemma)

Sei k_e (für beliebiges, festes e) die Anzahl aller $f \in F$, so dass $(e, f) \in G$. Definiere $G' = \{(e, f) \in G; k_e \geq \epsilon/2|F|\}$ als die Menge aller "'super-guten"' Paare. Dann gilt $|G'| \geq \epsilon/2|E \times F|$.

- Mit Wkt. $\epsilon/2$ liegt $(pr_1, (c, pr_2))$ in G'. Mit Wkt. $\epsilon/2$ liegt $(pr_1, (c', pr_2'))$ in G'. Mit Wkt. $1/|C|$ gilt $c' \neq c$.
- Daher gilt: $Pr[(a, c, s, c', s') \leftarrow_\$ (\mathrm{Ext}^*)P^*(pk)] = \epsilon^2/4 - 1/|C|$.
- Für exponentiell großes C gilt: special soundness $\implies$ PoK.

Beweis Splitting Lemma

Proof.

Nehmen wir an $|G'|/|E \times F| < \epsilon/2$. Dann gilt:

$$|G| = |G'| + |G \setminus G'| < \epsilon/2|E \times F| + |G \setminus G'|.$$

Für jedes $(e, f) \in G \setminus G'$ gilt außerdem $k_e \leq \epsilon/2|F|$. Es gibt aber nur $|E|$ verschiedene e. Damit gilt: $|G \setminus G'| \leq \epsilon/2|E||F|$. Es gilt damit

$$|G| = |G'| + |G \setminus G'| < \epsilon/2|E \times F| + \epsilon/2|E||F|,$$

was $|G| \geq \epsilon/2|E \times F|$ widerspricht. Es muss also $|G'|/|E \times F| \geq \epsilon/2$. □

Proof-of-Knowledge of DLOG – Schnorr's Protokoll

- Seien p, q prim, $G_q \subset \mathbb{Z}_p^*$ mit $|G_q| = q, q|p-1$ und g mit $<g>= G_q$. $\text{pp} := (g, p, q)$.

Schnorrs Protokoll: $PoK\{(\alpha) : g^x = pk\}$

P		V
$sk = x \leftarrow_\$ \mathbb{Z}_q, pk = g^x,$		
$r \leftarrow_\$ \mathbb{Z}_q, a = g^r$	$\xrightarrow{a}$	
	$\xleftarrow{c}$	$c \leftarrow_\$ C$
$s = cx + r$	$\xrightarrow{s}$	
		$pk^c a \stackrel{?}{=} g^s$

Completeness and Soundness

- Perfect completeness:

$$\mathrm{pk}^c a \quad = \quad (g^x)^c g^r = g^{xc+r} = g^s.$$

- Special Soundness: gegeben zwei valide Transkripte (a, c, s, c', s') so können wir einfach nach a auflösen:

$$\mathrm{pk}^c / g^s = \mathrm{pk}^{c'} / g^{s'} \equiv \mathrm{pk}^{c-c'} = g^{s-s'}$$

  . Da $c \neq c'$ erhalten wir $\mathrm{sk} = (s - s')/(c - c')$.
- Die Sicherheit des Protokolls kann (im Bestenfall) nur im Sinne von computational security halten: Ein Angreifer, der den DLOG von pk bezüglich g berechnen kann, kann den Verifier immer akzeptieren lassen. Special Soundness muss dies wiederspiegeln.
- Special Soundness zeigt, das man jeden erfolgreichen Angreifer P^* verwenden kann um den DLOG von h bezüglich g berechnen kann. Dazu muss P^* einfach auf $pk := h$ laufen gelassen werden.

Übersicht

Zero-Knowledge Beweise/Argumente

- Soundness und Special Soundness schützen den Verifier vor betrügerischen Provern.
- Die Zero-Knowledge Eigenschaft eines Protokolls schützt den Prover vor neugierigen aber sich ehrlich verhaltenden Verifiern V* (HVZK: Honest Verifier Zero-Knowledge) (oder passiven Zuhörern), die Informationen über sk gewinnen möchten.
- Die Kernidee ist, nachzuweisen, dass der Angreifer durch Betrachtung von validen Transkripten keine neuen Informationen über sk lernen kann.
- Technisch umgesetzt wird dies durch die Angabe eines effizienten Simulators Sim, der ohne Kenntnis von sk Transkripte produzieren kann, die von validen Kommunikationstranskripten nicht unterschieden werden können.

Zero-Knowledge Beweise/Argumente

- Wir fordern dass die Verteilungen computationally close, statistically close oder perfectly close sind.
- Alles, was der Angreifer also durch Betrachtung von validen Transkripten lernen könnte, hätte er sich durch Aufruf von Sim selbst beibringen können – auch ohne sk. Die Betrachtung von Transkripten liefert dem Angreifer daher keine neuen Informationen.

Ununterscheidbarkeit von Verteilungen

- Wir sagen, ein Sigma-Protokoll Σ bietet computational HVZK wenn für jeden PPT Unterscheider D gilt:

$$\left| \begin{array}{r} \Pr\left[1 \leftarrow_{\$} \mathsf{D}(\mathsf{pp}, pk, (a, c, s)); (a, c, s) \leftarrow_{\$} \mathsf{Sim}(\mathsf{pp}, pk)\right] \\ - \Pr\left[1 \leftarrow_{\$} \mathsf{D}(\mathsf{pp}, pk, (a, c, s)); (a, c, s) \leftarrow_{\$} < \mathsf{P}(\mathsf{pp}, \mathsf{sk}, \mathsf{pk}), \mathsf{V}^* >\right] \end{array} \right| \leq \mathsf{Adv}^{\mathrm{hvzk}}_{\mathsf{D},\Sigma}(\kappa) = \mathsf{negl}(\kappa)$$

- Wir sagen, ein Sigma-Protokoll bietet statistical HVZK wenn D unbeschränkt sein darf, aber nur auf $\mathsf{poly}(\kappa)$-viele Samples der Verteilungen von $\mathsf{Sim}(\mathsf{pp}, pk)$ und $< \mathsf{P}(\mathsf{pp}, \mathsf{sk}, \mathsf{pk}), \mathsf{V}^* >$ zugreifen kann. Alternativ heißt das, dass $\mathsf{Adv}^{\mathrm{hvzk}}_{\mathsf{D},\Sigma}(\kappa)$ statistisch klein ist.
- Wir sagen, ein Sigma-Protokoll bietet perfect HVZK wenn D unbeschränkt ist und auf exponentiell viele Samples Zugreifen kann: $\mathsf{Adv}^{\mathrm{hvzk}}_{\mathsf{D},\Sigma}(\kappa) = 0$.

Varianten von HVZK

- Wir sagen, ein Sigma-Protokoll Σ bietet Special HVZK (SHVZK), wenn es HVZK bietet und zusätzlich gilt, dass der Simulator für beliebiges gegebenes, zufälliges $c \in C$ passendes a, s erzeugen kann so dass (a, c, s) nicht von einem echten Transkript (a', c', s') mit Challenge $c' = c$ unterscheidbar ist.
- Wir sagen ein Sigma-Protokoll Σ bietet Strong Special HVZK, wenn es SHVZK bietet und zusätzlich gilt, dass der Simulator bei der Erzeugung von a, s den Wert s zufällig wählt: $s \leftarrow\!\!\!\!\!\!\$\ S$.

Sigma*-Protokolle

- Wir können nun eine wichtige Klasse von Sigma-Protokollen definieren.
- Wir sagen, ein Sigma-Protokoll Σ ist ein Sigma*-Protokoll, wenn es Perfect Completeness, Special Soundness, und Special HVZK bietet.
- Wir nenne Sigma* Protokolle auch (etwas ungenau) Zero-Knowledge Proofs-of-Knowledge. und kennzeichen dies in dem wir die Notation für Proofs-of-Knowledge: $PoK = \{(\alpha) : (\alpha, y) \in R\}$ anpassen: $ZKPoK = \{(\alpha) : (\alpha, y) \in R\}$.

Beispiele für Sigma*-Protokolle

- Das Schnorr Protokoll ist ein Sigma*-Protokolle.
- Wir haben bereits gesehen, dass das Schnorr Protokoll Perfect Completeness und Special Soundness bietet. Es bleibt SHVZK zu zeigen:
- Dazu können wir einen effizienten Simulator Sim angeben: Sim erhält $c \in C$, wählt $s \leftarrow_{\$} \mathbb{Z}_q$ zufällig und berechnet $a = pk^c / g^s$.
- Es bleibt zu zeigen, dass die Verteilung von (a, c, s) ununterscheidbar von der echter Transkripte (a', c', s') mit $c' = c$ ist.
- Für festes c wird in der simulierten Verteilung s zufällig gewählt. Es gilt $s = c * x + r$ und damit ist $r = (s - c * x)$ und $a = g^r$ ebenfalls zufällig verteilt. Dies entspricht der echten Verteilung wo zuerst $a = g^r$ berechnet wird.

Beispiele für Sigma*-Protokolle

- Das Sigma-DDH ist ein Sigma*-Protokolle.
- Wir müssen noch nachweisen, dass Sigma-DDH Special Soundness und SHVZK bietet.
- Special Soundness: gegeben a, c, s, c', s' können wir einfach $x = (s - s')/(c - c')$ berechnen. Insbesondere muss gelten $g^x = pk_1$ und $h^x = pk_2$.
- SHVZK:
  Sim erhält $c \in C$, wählt $s \leftarrow_\$ \mathbb{Z}_q$ zufällig und berechnet $a_1 = pk_1^c/g^s$ und $a_2 = pk_1^c/h^s$.
- Es bleibt zu zeigen, dass die Verteilung von (a, c, s) ununterscheidbar von der echter Transkripte (a', c', s') mit $c' = c$ ist. Hier können wir die gleichen Argumente anführen wie für das Schnorr Protokoll.

Komposition von Sigma*-Protokolle

- Eine nützliche Eigenschaft von Sigma* Protokollen ist, dass sie unter Konjunktion und Disjunktion abgeschlossen sind. Aus Ihnen können beliebige monotone (Boolsche) Ausdrücke konstruiert werden (solche, die nur aus Oder und Und Verknüpfungen bestehen).
- Seien im Folgenden zwei Sigma* Protokolle

$$ZKPoK_1 = \{(\alpha_1) : (\alpha_1, y_1) \in R_1\}$$
$$ZKPoK_2 = \{(\alpha_2) : (\alpha_2, y_2) \in R_2\}$$

mit Transkripten

$$(a_1, c_1, s_1)(a_2, c_2, s_2)$$

gegeben.

Konjunktion von Sigma*-Protokolle

- Die Konjunktion der zwei Sigma* Protokolle

$$ZKPoK = \{(\alpha_1, \alpha_2) : (\alpha_1, y_1) \in R_1 \wedge (\alpha_2, y_2) \in R_2\}$$

besteht aus dem Transkript (a, c, s) mit

$$a = (a_1, a_2) \text{ und } s = (s_1, s_2).$$

- Alle Werte werden wie in den Teilprotokollen berechnet. Dabei wird jedoch c als Challenge in beiden Protkollen verwendet.

Beweis Konjunktion von Sigma* Protokollen

- Perfect Completeness folgt unmittelbar aus den entsprechenden Eigenschaften der Einzelprotokolle.
- Special Soundness folgt aus den Einzeleigenschaften. Wir können durch Rewinding zwei gültige Transkripte (a, c, s, c', s') mit gemeinsamen $a = (a_1, a_2)$ erhalten. Da $s = (s_1, s_2)$ und $s' = (s'_1, s'_2)$ erhalten wir auch jeweils zwei Transkripte für die Teilprotokolle: (a_1, c, s_1, c', s'_1) und (a_2, c, s_2, c', s'_2). Hier können wir nun die Special Soundness der Einzelprotokolle ausnutzen und α_1 und α_2 berechnen.
- SHVZK können wir zeigen in dem wir, gegeben c, die Simulatoren für $ZKPoK_1$ und $ZKPoK_2$ getrennt aufrufen. Hier nutzen wir essentiell aus, dass bei Special Soundness c an den Simulator gegeben wird. Nach Annahme ist jedes der so erhaltenen Transkripte vom jeweiligen echten Teilprotokoll ununterscheidbar. Damit ist auch das Gesamttranskript ununterscheidbar.

Disjunktion von Sigma*-Protokolle

- Die Disjunktion der zwei Sigma* Protokolle

$$ZKPoK = \{(\alpha_i, i \in \{1,2\}) : (\alpha_1, y_1) \in R_1 \vee (\alpha_2, y_2) \in R_2\}$$

  besteht aus dem Transkript (a, c, s) mit

$$a = (a_1, a_2) \text{ und } s = (s_1, c_1, s_2).$$

- Sei $c_2 := c \; XOR \; c_1$. Dann ist (a_1, c_1, s_1) und (a_2, c_2, s_2) ein gültiges Transkript für das jeweilige Teilprotokoll mit der zusätzlichen Bedingung $c = c_1 \; XOR \; c_2$, die in *ZKPoK* überprüft werden muss.
- Nehmen wir an, der Prover kennt α_i. Dann kann er (a_1, a_2) folgendermaßen generieren: erst wird c_j mit $j = 3 - i$ zufällig gewählt und der SHVZK Simulator für $ZKPoK_j$ aufgerufen um a_j, s_j zu berechnen. Anschließend wird a_i wie in $ZKPoK_i$ generiert.
- Nachdem er c erhalten hat, berechnet der Prover die Challenge $c_2 := c_1 \; XOR \; c$ und nutzt α_i um wie im Teilprotokoll $ZKPoK_i$ definiert s_i passend zu a_i, c_i zu berechnen.

Beweis Disjunktion von Sigma* Protokollen

- Perfect Completeness folgt unmittelbar aus den entsprechenden Eigenschaften der Einzelprotokolle.
- Special Soundness folgt aus den Einzeleigenschaften. Wir können durch Rewinding zwei gültige Transkripte (a, c, s, c', s') mit gemeinsamen $a = (a_1, a_2)$ erhalten. Da $s = (s_1, c_1, s_2)$ und $s' = (s'_1, c'_1, s'_2)$ erhalten wir auch jeweils zwei Transkripte für die Teilprotokolle: $(a_1, c_1, s_1, c'_1, s'_1)$ und $(a_2, c_2, s_2, c'_2, s'_2)$. Es gilt $c' = c'_1 \; XOR \; c'_2 \neq c = c_1 \; XOR \; c_2$ also für mindestens ein $i \in \{1, 2\}$ dass $c'_i \neq c_i$. Für dieses i können wir dann, die Special Soundness von $ZKPoK_i$ ausnutzend, α_i berechnen.
- SHVZK können wir zeigen in dem wir, gegeben c, den Wert c_1 zufällige wählen, $c_2 := c_1 \; XOR \; c_2$ setzen und die Simulatoren für $ZKPoK_1$ mit c_1 und $ZKPoK_2$ mit c_2 getrennt aufrufen. Da jedes der so erhaltenen Transkripte vom jeweiligen echten (Teil-)Protokoll ununterscheidbar ist, gilt das auch für das Gesamttranskript.