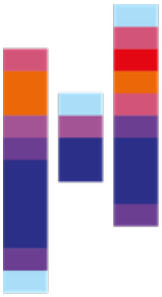

Hochschule
Flensburg
University of
Applied Sciences

Wissenschaftliche Ausarbeitung

PHOABE: Policy-Hidden Outsourced Attribute Based Encryption

Florian Hansen
Hochschule Flensburg

6. Juni 2020

1 Einleitung

Diese wissenschaftliche Ausarbeitung wird im Rahmen der Hochschulveranstaltung *Hot-Topics in der IT-Security* angefertigt und handelt über den sicheren Datenaustausch zwischen Internet-of-Things-Geräten. Besonders wird hierbei auf die Attribute Based Encryption (ABE) eingegangen, welche ein relativ junges Forschungsgebiet der Kryptographie darstellt. Es existieren zwar einige ABE-Schemata, diese sind jedoch für leistungsschwache Geräte eher ungeeignet.

Zu Beginn widmet sich die Ausarbeitung den grundlegenden mathematischen Hintergründen, die benötigt werden, um die hier vorgestellten Schemata verstehen zu können. Dazu gehört vor allem die Definitionen von Zugriffsregeln bzw. -strukturen. In der Praxis verwendet man beispielsweise gerne Zugriffsstrukturen in Form eines binären Baumes, um eine oder mehrere Zugriffsregeln zu definieren. In den kryptografischen Algorithmen findet man jedoch häufig die Beschreibung von Zugriffsregeln durch sogenannte *Linear-Secret-Sharing-Schemes* in Form einer Matrix. Dies stellt vor allem eine sehr viel elegantere Darstellung von Zugriffsregeln dar, als binäre Bäume. Da ein Baum jedoch wesentlich besser nachvollzogen werden kann, wird eine Zugriffsstruktur erst in Form eines Baumes definiert und dann später in eine Matrix umgewandelt.

Diese Ausarbeitung beschäftigt sich hauptsächlich mit der Arbeit von [1], welche ein neues attribut-basiertes Verschlüsselungsschema vorstellt: *PHOABE*. Dieses Schema widmet sich bekannten Problemen der Attribut-basierten Verschlüsselung, wie zum Beispiel das Verstecken der Zugriffsregelungen (*Access-Policies*), die sensitive Daten enthalten können. Zusätzlich fordert das Schema eine Auslagerung des Entschlüsselungsvorganges, welches aufgrund von bilinearen Abbildungen relativ rechenintensiv ist. Hierbei sollen sogenannte *Semi-trusted Cloud Server* den aufwändigen Teil der Berechnung übernehmen, damit auch leistungsschwache Geräte solche attribut-basierte Verschlüsselungen durchführen können.

2 Problembeschreibung

Das *Internet der Dinge*, *Internet of Things* (IoT), hat die Aufgabe, verschiedenste Hardwarekomponenten miteinander zu verbinden und damit einen Datenaustausch zu ermöglichen. Ein gutes Beispiel hierfür sind Smartphones und Sensoren, die ständig miteinander kommunizieren müssen bzw. sollen. Grundsätzlich können IoT-Anwendungen aufgrund ihrer Kommunikationsfähigkeit in zwei Bereiche eingeteilt werden. In Einzelanwendungen benötigen man lediglich eine Autorität während es auch Anwendungsfälle gibt, die über unterschiedliche Bereiche bzw. Domänen kommunizieren müssen. Diese besitzen damit mehrere Autoritäten [1].

Beide Klassen von IoT haben eins gemeinsam. Sie müssen einen sicheren Austausch von Daten und die Sicherung der Privatsphäre gewährleisten. Hierfür werden Daten verschlüsselt und mithilfe von Zugriffskontrollmechanismen nur bestimmten Parteien zur Verfügung gestellt. Genau deshalb ist die attribut-basierte Verschlüsselung ein attraktiver Kandidat, da sie Zugriffskontrolle (*Access-Control*) und Verschlüsselung miteinander verknüpft. Ein Problem dabei exisitiert dennoch. Dieses Verfahren verwendet bilineare Abbildungen, was vor allem in der Entschlüsselung viel Rechenaufwand bedeutet. Für leistungsstarke Desktop-Rechner mag dies weniger eine Rolle spielen, ist jedoch bezogen auf IoT ein großes Problem, denn dort kommunizieren in der Regel Geräte mit sehr eingeschränkten Ressourcen miteinander. Zudem erhöht sich der Rechenaufwand proportional zur Anzahl der Attribute. Das Problem liegt also darin, Daten sicher und effizient zwischen ressourcenarmen Geräten auszutauschen, ohne die Privatsphäre der Benutzer zu verletzen.

3 Motivation

Die attribut-basierte Verschlüsselung ist, wie in der Problembeschreibung bereits erwähnt, ineffizienter, je mehr Attribute verwendet werden. Besonders die Entschlüsselung wird durch die bilinearen Operationen extrem rechen-

aufwändig, was zur Folge hat, dass dieses Schema auf ressourcenärmeren Geräten und damit auf IoT nur bedingt anwendbar ist. Eine wichtige Frage, die sich also stellt ist, wie man ABE für eben solche Anwendungsfälle anwendbar gestalten kann.

In weiterführenden Arbeiten, so auch der von [4], wird ein neues Konzept in Verbindung mit attribut-basierter Verschlüsselung vorgestellt, um diese effizienter zu gestalten. Für die kostspielige Entschlüsselung ist nun nicht mehr alleine der jeweilige Benutzer zuständig, sondern eine weitere Instanz. Ein halb vertrauenswürdiger Cloud-Server (*semi-trusted cloud server*, STCS), welcher den rechenaufwändigen Teil der Entschlüsselung übernehmen soll. Halb vertrauenswürdig bedeutet, dass wir dem Server vertrauen, dass dieser uns die gewünschten Resultate liefert, jedoch selbst versucht private Daten auszulesen. Der STCS soll lediglich einen Teil der Entschlüsselung übernehmen. Anschließend wird das Ergebnis vom Benutzer selbst entschlüsselt. So soll garantiert werden, dass die rechenaufwändige Arbeit vom STCS und nicht vom Benutzer übernommen wird und trotzdem keinerlei Informationen über die eigentliche Nachricht an dem Server preisgegeben wird.

Eine weitere Überlegung von [1] ist folgende. Was ist, wenn der STCS das teilweise entschlüsselte Chiffrat fälscht? Es muss also zusätzlich die Möglichkeit bestehen, dass das Ergebnis der Entschlüsselung des STCS' überprüft werden kann. Einige vergangene Arbeiten haben sich dem Problem gewidmet, jedoch keine effiziente Lösung geboten oder sind inpraktikabel für IoT, da sie sich auf einer einzigen Autorität stützen. Da IoT jedoch hauptsächlich über mehrere Domänen kommuniziert, ist dies keine praktikable Lösung. Die Motivation hinter [1] ist also, ein ausgelagertes, verifizierbares multi-authority ABE-Schema zu entwerfen.

4 Grundlagen

4.1 Zugriffsregeln

Grundsätzlich werden Regeln für den Datenzugriff durch zwei Formate repräsentiert. Zum Einen durch boolsche Funktionen und zum Anderen durch sogenannte Linear Secret Sharing Schemes (LSSS). In diesem Abschnitt sollen beide Repräsentationsmöglichkeiten eingeführt werden.

Definition 1 (Zugriffsstrukturen [5]). *Sei* $\{P_1, P_2, ..., P_n\}$ *die Menge der beteiligten Parteien. Eine Sammlung* $\mathbb{A} \subseteq 2^{\{P_1,P_2,...,P_n\}}$ *ist monoton, wenn* $\forall B, C. \quad B \in \mathbb{A} \quad \wedge \quad B \subseteq C \implies C \in \mathbb{A}$. *Eine Zugriffsstruktur ist damit eine Sammlung* $\mathbb{A}$ *von nicht-leeren Untermengen von dem Universum* $\{P_1, P_2, ...P_n\}$. *Alle Mengen* $A \in \mathbb{A}$ *werden als authorisierte Mengen während die nicht in* $\mathbb{A}$ *vertetenden Mengen als unauthorisierte bezeichnet werden.*

Definition 1 kann dabei so interpretiert werden, als dass alle Obermengen jedes Elements $B \in \mathbb{A}$ verteten sein müssen. Im Folgenden soll ein Beispiel dies verdeutlichen.

Beispiel. *Sei* $\mathbb{U} = \{1, 2, 3, 4\}$ *ein Universum und* $\mathbb{A} \subseteq 2^{\mathbb{U}}$ *eine Zugriffsstruktur.*

Die Zugriffsstruktur $\mathbb{A} = \{\{1, 2\}, \{3, 4\}\}$ *ist nicht monoton, da das Element* $\{1, 2, 3\}$ *nicht vorhanden ist.*

Die Zugriffsstruktur $\mathbb{A} = \{\{3, 4\}, \{1, 3, 4\}, \{1, 2, 3, 4\}\}$ *ist monoton, da alle Obermengen der Elemente von* $\mathbb{A}$ *vorhanden sind.*

Definition 2 (Linear Secret-Sharing Schemes [5]). *Ein Linear Secret-Sharing Scheme (LSSS) über eine Menge von Parteien P ist linear, wenn*

1. *Die Anteile (shares) für jede Partei einen Vektor* $\vec{v} \in \mathbb{Z}_p^{n+1}$ *formen.*
2. *Eine Matrix M existiert, die l Zeilen und* $n + 1$ *Spalten enthält. Jede i-te Zeile aus M mit* $i \in \{1, ..., l\}$ *ist dann mit einer Partei* $x_i \in P$ *beschriftet. Wenn wir den Spaltenvektor* $v = (s, r_1, r_2, ..., r_n)$ *betrachten,*

wobei $s \in \mathbb{Z}_p$ das zu teilende Geheimnis ist und $r_1, ..., r_n \in \mathbb{Z}_p$ zufällig gewählt werden, dann ist Mv ein Vektor bestehend aus l Anteilen des Geheimisses s.

Beispiel. *Gegeben sei eine LSSS-Matrix $M \in \mathbb{Z}_2^{l \times n}$ und die Abbilding $\rho : \mathbb{N}^+ \times \mathbb{Z}_p^{l \times n} \to \mathbb{Z}_p^n$, welche die i-te Zeile der Matrix M zurückgibt. Zudem werden den Zeilen Parteien P_i zugewiesen, sodass $\rho(i, M)$ den Anteil der jeweiligen Partei liefert.*

$$M = \begin{array}{c} P_2 \\ P_2 \\ P_1 \\ P_3 \\ P_4 \end{array} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Betrachten wir die Kombination $\{P_2, P_4\}$. Mithilfe der Abbildung ρ erhalten wir die dementsprechenden Zeilen.

$$\begin{aligned} \rho(0, M) &= \vec{v_1}_{P_2} = (1, 1, 0, 1) \\ \rho(1, M) &= \vec{v_2}_{P_2} = (0, 1, 1, 0) \\ \rho(4, M) &= \vec{v}_{P_4} = (0, 0, 1, 1) \end{aligned}$$

Um die Authentizität zu prüfen, muss die Summe der Zeilenvektoren $e = (1, 0, 0, 0)$ ergeben. Wir berechnen nun also die Summe der Vektoren

$$\vec{v_1}_{P_2} + \vec{v_2}_{P_2} + \vec{v}_{P_4} = (1, 2, 2, 2) \equiv (1, 0, 0, 0) \mod 2$$

und sehen, dass die Kombination $\{P_2, P_4\}$ authorisiert ist.

4.2 Bilineare Abbildungen

Definition 3 (Bilineare Abbildungen [5])**.** *Sei $\mathbb{P}$ die Menge aller Primzahlen und $\mathbb{G}$, $\mathbb{G}_T$ zwei multiplikative zyklische Gruppen mit einer Ordnung $p \in \mathbb{P}$. Sei g ein Generator von $\mathbb{G}$ und $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ eine bilineare Abbildung mit folgenden Eigenschaften.*

1. *Bilinearität: $\forall u, v \in \mathbb{G}, \ \ \forall a, b \in \mathbb{Z}_p. \ \ e(u^a, v^b) = e(u, v)^{ab}$*
2. *Nicht-Entartung: $e(g, g) \neq 1$*
3. *Effizient Berechenbar: Die Gruppenoperation von $\mathbb{G}$ und die bilineare Abbildung e sind effizient berechenbar.*

Beispiel. *Sei $(G, +)$ eine Gruppe und $x, y, z \in G$. Sei $(G_T, *)$ eine multiplikative Gruppe und $e : G \times G \to G_T$ eine bilineare Abbilding. Dann gilt*
$e(3x, y) = e(x + x + x, y) = e(x, y) * e(x, y) * e(x, y) = e(x, y)^3 = e(x, 3y)$.

5 Attribute Based Encryption (ABE)

Im Gegensatz zur klassischen Public-Key-Verschlüsselung zielt die attributbasierte Verschlüsselung (ABE) darauf ab, ein Chiffrat für mehrere anstatt für nur einen Benutzer zu erzeugen. Genauer ausgedrückt ist das Ziel eines ABE-Schemas die Einhaltung der Anonymität eines Benutzers, da dieser nur die benötigten Zugriffsbeschränkungen besitzen muss, um bestimmte Daten zu entschlüsseln. Dafür werden die privaten Schlüssel und Chiffrate der Benuzer (Parteien) als Zugriffsregeln bzw. Attribute ausgelegt. Ferner ist ein Benutzer in der Lage das Chiffrat mithilfe von Attributen oder festgelegten Regeln zu entschlüsseln, sodass jeder Benutzer mit den richtigen Attributen Zugriff auf die Daten hat. Die Attribute bzw. Zugriffsregelungen werden dabei von einer sogenannten Attribut-Autorität verteilt. Grundsätzlich werden ABE-Schemata in zwei unterschiedliche Kategorien eingeteilt [1].

5.1 Key-Policy ABE

KP-ABE-Schemata verwenden Zugriffsregeln als private Schlüssel und berechnen ein Chiffrat basierend auf ausgewählte Attribute. Als Beispiel soll die Zugriffsregel $A \wedge B$ als Schlüssel dienen. Zudem wird ein Chiffrat mit dem Attribut $\{A\}$ erzeugt. Versucht man nun das Chiffrat zu entschlüsseln, so schlägt dies fehl, da die Zugriffsregel nicht erfüllt wird. Werden hingegen die Attribute $\{A, B\}$ verwendet, wird die Regel eingehalten und das Chiffrat kann entschlüsselt werden. Hieraus wird ein mögliches Problem sichtbar, denn mit einem solchen Schema lässt sich nicht kontrollieren welche Benutzer Zugriff besitzen sollen. Stattdessen erhalten alle Benutzer Zugriff, denen eine entsprechende Zugriffsregel zugewiesen wurde [2].

Ein Key-Policy-ABE-Schema besteht aus insgesamt vier Algorithmen [3].

1. $\mathit{Setup}\,(n) \rightarrow (PK, SK_M)$: Dieser Algorithmus nimmt als Eingabe den Sicherheitsparameter n und liefert als Ergebnis eine Menge bestehend aus öffentlichen Parametern PK, sowie einen Master-Secret-Key SK_M.

2. $\mathit{KeyGen}\,(\mathbb{A}, SK_M) \rightarrow SK$: Als Eingabe nimmt dieser Algorithmus die Zugriffsstruktur $\mathbb{A}$ und den Master-Secret-Key SK_M. Es wird ein privater Schlüssel SK zurückgegeben.

3. $\mathit{Enc}\,(m, A, PK) \rightarrow c$: Der Verschlüsselungsalgorithmus nimmt als Eingabe die Nachricht m, eine nicht-leere Menge von Attributen A und die öffentlichen Parameter PK. Als Ausgabe wird ein Chiffrat c erzeugt.

4. $\mathit{Dec}\,(c, SK) \rightarrow \{m, \perp\}$: Dieser Algorithmus nimmt als Eingabe ein Chiffrat c und den privaten Schlüssel SK. Als Ausgabe wird die Nachricht m oder ein Fehler (dargestellt als $\perp$) erzeugt.

5.2 Ciphertext-Policy ABE

CP-ABE verwendet im Gegensatz zu KP-ABE Attribute als private Schlüssel [2]. Zudem werden Chiffrate mithilfe von Zugriffsregeln erzeugt, also genau

engegengetzt zu dem Vorgehen bei KP-ABE. Nehmen wir an, wir erzeugen einen Schlüssel für Benutzer 1 bestehend aus den Attributen $\{A, B\}$ und einen Schlüssel für Benutzer 2 bestehend aus den Attributen $\{B\}$. Bei einem Chiffrat, welches unter der Zugriffsregel $A \wedge B$ erstellt wurde, ist Benutzer 1 dazu in der Lage, dieses zu entschlüsseln. Benutzer 2 jedoch nicht. Bei diesem Verfahren ist man also in der Lage zu entscheiden, welche Daten für welchen Benutzer zur Verfügung stehen sollen bzw. welche Attribute vorausgesetzt werden, um Geheimtexte zu entschlüsseln.

Ein Ciphertext-Policy ABE-Schema besteht aus insgesamt vier Algorithmen [6].

1. $Setup\,(n) \rightarrow (PK, SK_M)$: Dieser Algorithmus nimmt als Eingabe den Sicherheitsparameter n und liefert als Ergebnis eine Menge bestehend aus öffentlichen Parametern PK, sowie einen Master-Secret-Key SK_M.

2. $KeyGen\,(A, SK_M) \rightarrow SK$: Als Eingabe nimmt dieser Algorithmus eine Menge von Attributen A und den Master-Secret-Key SK_M. Es wird ein privater Schlüssel SK zurückgegeben.

3. $Enc\,(m, \mathbb{A}, PK) \rightarrow c$: Der Verschlüsselungsalgorithmus nimmt als Eingabe die Nachricht m, eine Zugriffsstruktur $\mathbb{A}$ und die öffentlichen Parameter PK. Als Ausgabe wird ein Chiffrat c erzeugt.

4. $Dec\,(c, SK) \rightarrow \{m, \perp\}$: Dieser Algorithmus nimmt als Eingabe ein Chiffrat c und den privaten Schlüssel SK. Als Ausgabe wird die Nachricht m oder ein Fehler (dargestellt als $\perp$) erzeugt.

5.3 Single- und Multi-Authority

Die Erzeugung der privaten Schlüssel kann in einem ABE-Schema auf zwei verschiedenen Arten durchgeführt werden. Zum einen existiert das Modell einer zentralen Autorität, welche für die Generierung der Schlüssel für alle Benutzer zuständig ist. Man bezeichnet diese ABE-Schemata als *central-* bzw. *single-autority*. Da in diesem Verfahren eine einzelne Autorität die Schlüssel

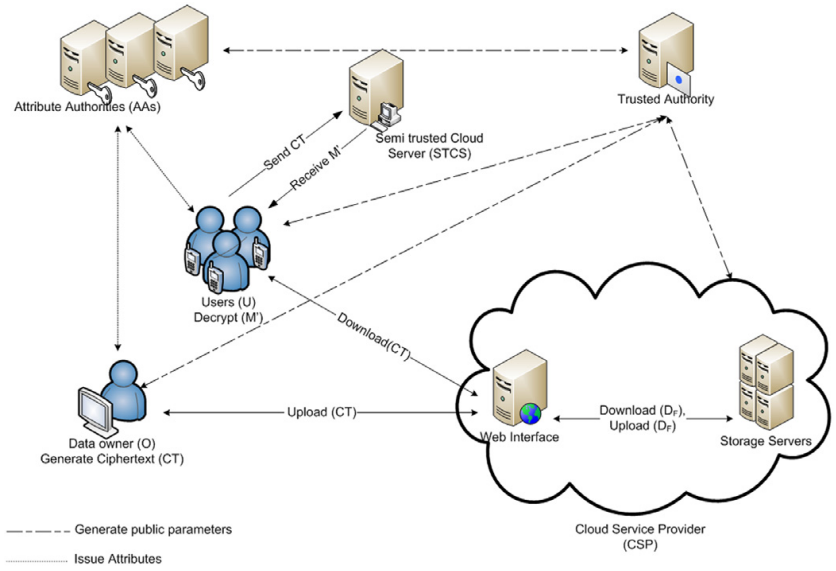

Abbildung 1: Architektur von PHOABE [1]

bereitstellt, muss dieser vertraut werden. Dies bringt natürlich ein gewisses Risiko mit sich, denn eine zentrale Autorität besitzt sämtliche Schlüssel der Parteien und kann dementsprechend alle Geheimtexte entschlüsseln.

Um dieses Sicherheitsrisiko zu umgehen, werden sogenannte *multi-authority* Schemata eingesetzt. Anstatt einer zentralen erzeugen und verwalten mehrere Autoritäten die privaten Schlüssel. Eine einzelne Autorität ist damit nicht in der Lage einen Geheimtext zu entschlüsseln, da ihr nicht alle Bestandteile des benötigten Schlüssels zur Verfügung stehen. Zudem ist dieses Verfahren entlastend für sämtliche Autoritäten, da die Verwaltung der Schlüssel durch mehrere Instanzen realisiert wird.

6 Policy-Hidden Outsourced ABE

Es wird nun ein Überblick zu *PHOABE* (Policy-Hidden Outsourced Attribute Based Encryption) gegeben, einem Schema, welches aufgrund seiner Architektur für attribut-basierte Verschlüsselung auf IoT-Geräten interessant ist. Wie der Name bereits andeutet, handelt es sich hierbei um ein

Schema, welches die Geheimhaltung der Privatsphäre durch Zugriffsstrukturen gewährleistet (Policy-Hidden). Zudem wird ein STCS verwendet, um den Entschlüsselungsprozess teilweise auszulagern (Outsourced). Der Grund hierfür ist, dass IoT-Geräte oft leistungsschwach sind und die Entschlüsselung hingegen sehr kostspielig ist, weshalb viele andere ABE-Schemata nicht praktikabel sind. Da ein STCS verwendet wird, stellt PHOABE zudem sicher, dass die Integrität der Informationen gewährleistet ist. Der Grund für die Sicherstellung ist, dass angenommen wird, dass der Server selbst korrupt sein und versuchen könnte, private Daten auszuspähen, uns jedoch trotzdem valide Ergebnisse erzeugt. Kurz zusammengefasst weist PHOABE folgende Eigenschaften auf.

- Ausgelagert: Die Entschlüsselung wird teilweise durch einen Server (STCS) übernommen.
- Vertraulichkeit: Eine Änderung des Geheimtextes bewirkt keine sinnvolle Änderung der verschlüsselten Nachricht.
- Integrität: Es kann überprüft werden, ob die Resultate des STCS gefälscht wurden.
- Geheimhaltung der Privatsphäre: Zugriffsstrukturen garantieren die Sicherung der Privatsphäre.

6.1 Algorithmen

PHOABE besteht aus insgesamt sieben Algorithmen [1]. Es werden folgende Symbole für die Darstellung der Komponenten des Schemas verwendet.

1. **Setup** $(\lambda) \rightarrow PP$**.** Der Setup-Algorithmus nimmt als Eingabe einen Sicherheitsparameter λ, liefert die öffentlichen Parameter PP und wird von einer *Central Trusted Authority* (CTA) ausgeführt.
   1.1. Definiere die multiplikativen Gruppen $\mathbb{G}_1$, $\mathbb{G}_T$ der Ordnung p.
   1.2. Definiere die bilinearen Abbildung $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$.

1.3. Setze den Generator g, sodass $\langle g \rangle = \mathbb{G}_1$.

1.4. Definiere die Hashfunktion $H' : \{0,1\}^* \to \mathbb{Z}_p$.

1.5. Definiere die Hashfunktion $H_0 : \{M\} \to \{0,1\}^{n_{H_0}}$. Die Ausgabe von H_0 besitzt dabei die feste Länge n_{H_0}.

1.6. Definiere die Hashfunktion $H_1 : \{M\} \to \{0,1\}^*$.

1.7. Definiere die Hashfunktion $H_2 : \{0,1\}^* \to \{0,1\}^{n_{H_2}}$. Die Ausgabe von H_2 besitzt dabei die feste Länge n_{H_2}.

1.8. **return** $(\mathbb{G}_1, \mathbb{G}_T, p, H', H_0, H_1, H_2, e, g, e(g,g))$.

2. **Setup**$_{auth}\,(PP) \to \left(sk_{AA_j}, pk_{AA_j}\right)$**.** Bei N Autoritäten führt die Autorität AA_j mit $j \in N$ diesen Algorithmus aus, um einen geheimen Schlüssel sk_{AA_j} und einen öffentlichen Schlüssel pk_{AA_j} zu generieren. Als Eingabe nimmt der Algorithmus die öffentlichen Umgebungsparameter PP. Im Folgenden sei die Menge der von der Attribute-Authority AA_j bereitgestellten Attribute durch S_{AA_j} dargestellt.

   2.1. Wähle zwei zufällige Zahlen $\alpha_i, t_i \in \mathbb{Z}_p^*$ für jedes Attribut $i \in S_{AA_j}$.

   2.2. Wähle eine zufällige Zahl $y \in \mathbb{Z}_p^*$.

   2.3. Setze den geheimen Schlüssel als $sk_{AA_j} = \left(\left\{\alpha_i, t_i \mid i \in S_{AA_j}\right\}, y\right)$.

   2.4. Berechne den öffentlichen Schlüssel $pk_{AA_j} = \left(\left\{g^{\alpha_i}, g^{t_i} \mid i \in S_{AA_j}\right\}, g^y\right)$.

   2.5. **return** $\left(sk_{AA_j}, pk_{AA_j}\right)$.

3. **Encrypt** $(PP, M_{pk}, m, \Phi) \to C$**.** Dieser Algorithmus verschlüsselt die Nachricht m unter Angabe einer Menge der öffentlichen Schlüssel $M_{pk} = \left\{pk_{AA_j} \mid j \in N\right\}$ und wird vom Datenbesitzer ausgeführt. Jeder dieser Schlüssel wird von einer *Attribute Authority* bereitgestellt. Zudem wird eine Zugriffsregel Φ verwendet. Als Ausgabe wird ein Geheimtext C erzeugt.

   3.1. Wähle eine zufällige Zahl $x \in \mathbb{Z}_p^*$

3.2. Berechne für jedes Attribut a_i den Wert $q_i = e((g^y)^x, H'(a_i))$, wobei $a_i \in \Phi$ ein Attribut aus der Zugriffsstruktur Φ ist und $|\{a_i\}| = F$.

3.3. Konvertiere Φ mithilfe $\{q_i\}_{i \in F}$ in eine LSSS-Matrix $\left(A^{n \times l}, \rho\right)$, wobei F die Anzahl der Attribute innerhalb der Zugriffsstruktur Φ ist. Die Umwandlung einer Zugriffsstruktur in eine LSSS-Matrix wird dabei als gegeben (Black-Box) angesehen.

3.4. Wähle eine zufällige Zahl $s \in \mathbb{Z}_p$.

3.5. Wähle $\left\{p_i \mid p_i \in \mathbb{Z}_p^*\ ,\ 0 \leq i < F\right\}$.

3.6. Berechne λ_i, ω_i sodass $\lambda_i = \vec{A_i} \cdot \vec{v}$ und $\omega_i = \vec{A_i} \cdot \vec{\tau}$, wobei $\vec{v} = (s, v_2, ..., v_l) \in \mathbb{Z}_p^l$ und $\vec{\tau} = (0, \tau_2, ...\tau_l) \in \mathbb{Z}_p^l$ zwei zufällige Vektoren aus dem l-dimensionalen Raum von $\mathbb{Z}_p$ sind. Das bedeutet im Konkreten, dass ein zufälliger Vektor $\vec{v}$ erstellt wird und der ersten Komponente s zugewiesen wird. Bei $\vec{\tau}$ wird auch ein zufälliger Vektor erzeugt, mit dem Unterschied, dass der ersten Komponente von $\vec{\tau}$ der Wert 0 zugewiesen wird. Die Verknüpfung zweier Vektoren mit $\cdot$ stellt dabei das Skalarprodukt dar.

3.7. Berechne $CT_{ABE} = (h, C_0, C_{1,r}, C_{2,r}, C_{3,r})$ für jede Zeile r aus der LSSS-Matrix A mit $h = g^x$, $C_0 = R \cdot e(g,g)^s$, $C_{1,r} = g^{\lambda_{\rho(i)}} \cdot g^{\alpha_{\rho(i)} p_i}$, $C_{2,r} = g^{p_i}$ und $C_{3,r} = g^{\omega_{\rho(i)}} \cdot g^{t_{\rho(i)} p_i}$. Zur Erinnerung: $\rho(i)$ gibt das Attribut a zurück, welches der Zeile i der LSSS-Matrix zugeordnet wurde. Dementsprechend sind $\lambda_{\rho(i)}$ und $\omega_{\rho(i)}$ zufällige Werte speziell für ein Attribut a.

3.8. Wähle eine zufällige Nachricht $R \in \mathbb{G}_T$ aus der Zielgruppe und berechne $R_0 = H_0(R)$ und den symmetrischen Schlüssel $K_{sym} = H_1(R)$. Der Wert R_0 hat dabei eine beliebige während K_{sym} eine feste Länge besitzt.

3.9. Führe eine symmetrische Verschlüsselung der Nachricht m durch, sodass $CT_{sym} = encrypt_{sym}\,(K_{sym}, m)$ und berechne den Verifizierungsschlüssel $V = H_2(R_0 || CT_{sym})$, wobei $||$ die Werte von R_0 und CT_{sym} miteinander verkettet.

3.10. **return** $C = (CT_{ABE}, CT_{sym}, V)$.

4. **Keygen** $\left(PP, sk_{AA_j}, pk_{AA_j}, GID, S_{j,GID}\right) \rightarrow sk_{j,GID}$**.** Dieser Algorithmus erzeugt einen privaten Schlüssel $sk_{j,GID}$ und wird von einer *Attribute Authority* AA_j ausgeführt. Der Schlüssel wird für einen Benutzer erzeugt und steht in Verbindung mit den Attributen $S_{j,GID} = \left\{a_1, ..., a_{n_j}\right\}$ der AA_j. Dabei stellt $a \in S_{j,GID}$ ein Attribut und n_j die Anzahl der Attribute von AA_j dar. GID ist die Identität vom Benutzer.

   4.1. Zur Erinnerung: $sk_{AA_j} = \left(\{\alpha_i, t_i\}_{i \in S_{AA_j}}, y\right)$.

   4.2. Berechne $K_{1,i} = g^{\alpha_i} \cdot H'(GID)^{t_i}$ für jedes Attribut $i \in S_{j,GID}$.

   4.3. Berechne $K_{2,i} = H'(i)^y$ für jedes Attribut $i \in S_{j,GID}$.

   4.4. **return** $sk_{j,GID} = \{K_{1,i}, K_{2,i} \mid i \in S_{j,GID}\}$.

5. **Transform** $(PP, M_{sk}, (A, \rho), C) \rightarrow T$**.** Dieser Algorithmus wird von einem Benutzer ausgeführt. Dieser besitzt eine Menge von Attributen S_{GID} und die dazugehörigen Schlüssel $M_{sk} = \{sk_{j,GID} \mid j \in N\}$ bei N Attribute-Authorities. Die Schlüssel wurden vorher durch diese für den Benutzer ausgestellt (siehe Punkt 4). Als Eingabe nimmt der Algorithmus zusätzlich die öffentlichen Parameter PP, die Zugriffsregel (A, ρ) als LSSS-Matrix und den Geheimtext C. Als Ausgabe wird eine Menge von transformierten Schlüsseln M_{tk} und das Geheimnis tsk_{GID} für den Benutzer mit der Identität GID erzeugt.

   5.1. Berechne $\forall i \in S_{j,GID} \ : \ q'_i = e(h, H'(i)^y)$. Dabei stellt $S_{j,GID}$ die Menge aller Attribute von der Attribute-Authority AA_j für den Benutzer mit der Identität GID dar.

   5.2. Konstruiere S'_{GID}, indem jedes Attribut i durch das entsprechende q'_i ersetzt wird.

   5.3. Finde alle für die Entschlüsselung benötigten Attribute $L' = \{\rho(A_i) \mid 0 \leq i < n\} \cap S'_{GID}$, wobei A_i eine Zeile und n die Anzahl der Zeilen der LSSS-Matrix $A^{n \times l}$ darstellt.

   5.4. Wähle ein zufälliges $z = tsk_{GID}$, $z \in \mathbb{Z}_p^*$.

5.5. Berechne $tpk_{j,GID} = \left(\left\{K_{1,i}^{1/z} \mid i \in L'\right\}, g^{1/z}, H(GID)^{1/z}\right)$ für alle Attribute-Authorities $\{AA_j \mid 0 \leq j < N\}$.

5.6. Setze die Menge der öffentlichen Transformationsschlüssel $M_{tpk} = \{tpk_{j,GID} \mid 0 \leq j < N\}$

5.7. **return** $T = (M_{tpk},\ tsk_{GID})$.

6. **Decrypt**$_{out}\,(PP, M_{tpk}, (A, \rho), C) \rightarrow m'$**.** Dieser Algorithmus wird vom STCS ausgeführt und erzeugt die teilweise entschlüsselte Nachricht m'. Als Eingabe nimmt der Algorithmus die öffentlichen Parameter PP, eine Menge von Transformationsschlüssel $M_{tpk} = \{tpk_{j,GID} \mid 0 \leq j < N\}$, die LSSS-Matrix (A, ρ) und den Geheimtext C.

6.1. Berechne für alle Zeilen der Matrix A die Werte

$$R_i = \frac{e(g^{1/z}, C_{1,i}) \cdot e(H'(GID)^{1/z}, C_{3,i})}{e(g^{\alpha_i/z} \cdot H'(GID)^{t_i/z}, C_{2,i})}$$

$$= (e(g,g)^{\lambda_i} \cdot e(H'(GID), g)^{\omega_i})^{1/z}$$

6.2. Wähle eine Menge $\{c_i \mid 0 \leq i < n\}$, sodass folgende Eigenschaft erfüllt wird.

$$\sum_{i=0}^{n-1} c_i \cdot A_i = (1, 0, ..., 0)^{\mathrm{T}}$$

Dabei sei erwähnt, dass n die Anzahl der Zeilen und A_i die i-te Zeile der LSSS-Matrix $A^{n \times l}$ sind. Demnach ist das obere Resultat ein Vektor aus dem l-dimensionalen Raum $\mathbb{Z}_p^l$.

6.3. **return** $m' = \prod_{i=0}^{n-1} {R_i}^{c_i} = e(g,g)^{\frac{s}{z}}$.

7. **Decrypt** $(m', tsk_{GID}) \rightarrow m$**.** Dieser Algorithmus wird vom Benutzer ausgeführt und nimmt als Eingabe die teilweise entschlüsselte Nachricht m' und den geheimen Transformationsschlüssel tsk_{GID}. Als Ausgabe wird die Nachricht m erzeugt.

7.1. Berechne im ersten Schritt den Wert R. Besonders hierbei ist, dass lediglich einmal Potenziert werden muss, ohne eine einziges

Pairing berechnen zu müssen.

$$R = \frac{C_0}{(m')^{tsk_{GID}}} = \frac{C_0}{\left(e(g,g)^{\frac{s}{z}}\right)^z} = \frac{C_0}{e(g,g)^s}$$

7.2. Berechne $R_0 = H_0(R)$ und prüfe, ob $H_2(R_0 \mid\mid CT_{sym})$ einen Fehler $\perp$ liefert. Falls ja, beende den Algorithmus mit einem Fehler. Ansonsten fahre mit dem Berechnen des symmetrischen Schlüssels fort.

7.3. Berechne den symmetrischen Schlüssel $K_{sym} = H_1(R)$.

7.4. **return** $m = decrypt_{sym}\,(K_{sym}, CT_{sym})$.

Symbol	Beschreibung
STCS	Semi Trusted Cloud Server
CTA	Central Trusted Authority
λ	Sicherheitsparameter
AA_j	Die j-te Attribute Authority
PP	Public Parameter
sk_{AA_j}	Der geheime Schlüssel der j-ten Attribute Authority
pk_{AA_j}	Der öffentliche Schlüssel der j-ten Attribute Authority
M_{pk}	Menge der öffentlichen Schlüssel aller Attribute Authorities
A	Zugriffsstruktur in LSSS-Matrix-Darstellung
ρ	Abbildung zum Auslesen einer Zeile von A
Φ	Eine Zugriffsstruktur
C	Der aus der Nachricht erzeugte Geheimtext
GID	Die Identität eines Benutzers
$S_{j,GID}$	Menge von Attributen der j-ten Attibute Authority für den Benutzer mit der Identität GID
$sk_{j,GID}$	Der geheime Schlüssel der j-ten Attribute Authority für den Benutzer mit der Identität GID
M_{sk}	Menge von geheimen Schlüsseln $sk_{j,GID}$ der j-ten Attribute Authority
M_{tk}	Menge der Transformationsschlüssel $tk_{j,GID} \in M_{tk}$
$tpk_{j,GID}$	Öffentlicher Transformationsschlüssel der j-ten Attribute Authority für den Benutzer mit der Identität GID
$tsk_{j,GID}$	Geheimer Transformationsschlüssel der j-ten Attribute Authority für den Benutzer mit der Identität GID
m	Die Nachricht
m'	Die teilweise entschlüsselte Nachricht vom STCS

7 Performance Analyse

Schlüsselgröße. Laut [1] beträgt die Speicherkapazität für einen Schlüssel eines Benutzers $2|S|$, wobei $|S|$ die Anzahl der für den Benutzer zugewiesenen Attribute darstellt. Um dies nachzuvollziehen, müssen wir unsschauen den Keygen-Algorithmus einmal näher anschauen. Im ersten Schritt berechnet dieser für jedes Attribut $i \in S$ einen Wert $K_{1,i}$ und einen Wert $K_{2,i}$, wobei S die Menge aller Attribute darstellt, die dem Benutzer zugewiesen wurden. Da sich der Schlüssel aus den beiden resultierenden Mengen zusammensetzt, besitzt dieser die Größe $|S| + |S| = 2|S|$.

Größe des Transformationsschlüssels. Die Größe eines Transformationsschlüssels beträgt $2|S| + 3$. Auch hier schauen wir genauer auf den Transform-Algoritmus, um dies nachzuvollziehen. Zuerst wird festgestellt, welche Attribute für eine Entschlüsselung benötigt werden. Dies entspricht der Menge S, die alle Attribute für den Benutzer enthält. Zudem wird ein zufälliges $z \in \mathbb{Z}_p$ gewählt, welches einen zusätzlichen Teil des Transformationsschlüssels darstellt. Anschließend wird für jedes Attribut $i \in S$ der Wert $K_{1,i}^{1/z}$ berechnet. Als weitere Teile des Schlüssels werden die beiden Werte $g^{1/z}$ und $H(GID)^{1/z}$ berechnet. Die Größe des gesamten Schlüssels beträgt damit $1 + |S| + 1 + 1 = |S| + 3$.

Geheimtextgröße. Im Encrypt-Algorithmus wird der Geheimtext aus insgesamt drei Teilen gebildet. Die Größe dessen bildet sich daher ebenfalls aus genau drei Teilen und lässt sich als $\texttt{size}\,(CT) = \text{size}(CT_{ABE}) + \text{size}(CT_{sym}) + \text{size}(V)$ beschreiben. Dabei stellt $CT_{ABE} = (h, C_0, C_{1,i}, C_{2,i}, C_{3,i})$ den attribut-basierten Teil und $CT_{sym} = \text{encrypt}_{sym}\,(K_{sym}, m)$ die symmetrisch verschlüsselte Nachricht dar. Außerdem gibt der Algorithmus einen Verifizierungsschlüssel V zurück. Die Größe $\texttt{size}\,(V)$ beträgt damit 1. Um $\texttt{size}\,(CT_{ABE})$ zu bestimmen, müssen wir uns die einzelnen Komponenten genauer anschauen. Die Komponenten h und C_0 benötigen jeweils einen Speicherplatz während $C_{1,i}$, $C_{2,i}$ und $C_{3,i}$ für jedes Attribut innerhalb der Zugriffsstruktur berechnet werden und damit insgesamt drei mal der Anzahl

an Attributen groß sind. Die Größe $\texttt{size}\,(CT_{ABE})$ beträgt somit $2+3n$, wobei n die Anzahl aller Attribute, die in dem Schema vorkommen, darstellt. Zuletzt muss noch die Größe von $\texttt{size}\,(CT_{sym})$ bestimmt werden. Da CT_{sym} lediglich die symmetrische Verschlüsselung einer Nachricht darstellt, beträgt der benötigte Speicherplatz zum Speichern dieser Variable 1. Der gesamte Geheimtext, der sich aus all den Teilen zusammensetzt, benötigt daher $\texttt{size}\,(CT) = 2+3n+1+1 = 4+3n$ Speichereinheiten.

Kosten der Verschlüsselung. Es soll nun auf die Komplexität der Verschlüsselung einer Nachricht eingegangen werden. Beginnen wir mit der Berechnung aller $q_i = e\,((g^y)^x, H'(a_i))$ zum Rekonstruieren der LSSS-Matrix. Hier wird für jedes q_i eine Potenzierung in $\mathbb{G}_1$ und einmal der Hashfunktion über a_i ausgeführt. Da für jedes Attribut a_i innerhalb der Zugriffsstruktur ein q_i berechnet wird, ergibt sich folgende Komplexität für diesen Teil: $n \cdot E_1 + n \cdot \mathcal{O}(H) + n \cdot \tau_P = n(E_1 + \mathcal{O}(H) + \tau_P)$. Darauf folgend wird $CT_{ABE} = (h, C_0, C_{1,i}, C_{2,i}, C_{3,i})$ berechnet. Die Komplexität von CT_{ABE} wird also durch seine Komponenten bestimmt, die nun genauer untersucht werden.

$$
\begin{aligned}
h &= g^a && \Longrightarrow\ E_1 \\
C_0 &= R \cdot e(g,g)^s && \Longrightarrow\ E_T \\
C_{1,i} &= g^{\lambda_{\rho(i)}} g^{\alpha_{\rho(i)} p_i} && \Longrightarrow\ n \cdot 2E_1 \\
C_{2,i} &= g^{p_i} && \Longrightarrow\ n \cdot E_1 \\
C_{3,i} &= g^{t_{\rho(i)} p_i} g^{\omega_i} && \Longrightarrow\ n \cdot 2E_1
\end{aligned}
$$

Zudem wird eine Hashfunktion H für die Berechnung von R_0, K_{sym} und V ausgeführt. Insgesamt erhalten wir damit eine Komplexität von

$$
\begin{aligned}
\mathcal{O}\,(encrypt) &= \mathcal{O}\,(\{q_i \mid 0 \leq i < n\}) + \mathcal{O}\,(CT_{ABE}) + 3\mathcal{O}\,(H) \\
&= n(E_1 + \mathcal{O}(H) + \tau_P) + E_1 + 5n \cdot E_1 + E_T + 3\mathcal{O}(H).
\end{aligned}
$$

Da h und alle q_i lediglich einmal initial berechnet werden, müssen diese nicht bei jeder Verschlüsselung erneut bestimmt werden. Der Aufwand der Berech-

nungen beträgt damit $\mathcal{O}(encrypt) = 5n \cdot E_1 + E_T + 3\mathcal{O}(H)$.

Kosten der Entschlüsselung (Benutzer).

Kosten der Entschlüsselung (STCS).

Literatur

[1] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, and R. Attia. PHOABE: Securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted IoT. *Computer Networks*, (133):141–156, 2018.

[2] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption, 2007.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309, 2006. https://eprint.iacr.org/2006/309.

[4] M. Green, S. Hohenberger, and B. Waters. Outsourcing the Decryption of ABE Ciphertexts. *USENIX Security Symposium*, 2011.

[5] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. Cryptology ePrint Archive, Report 2007/323, 2007. https://eprint.iacr.org/2007/323.

[6] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Cryptology ePrint Archive, Report 2008/290, 2008. https://eprint.iacr.org/2008/290.