Kryptographische Analyse

Spezifikation Schlüsselgenerierungsdienst ePA (Version 1.4.1)

Prof.Dr. Marc Fischlin
Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

erstellt von:

Prof. Dr. Marc Fischlin
Kryptographie und Komplexitätstheorie
Technische Universität Darmstadt
www.cryptoplexity.de

21. Dezember 2021

Zusammenfassung

Im Folgenden wird eine kryptographische Analyse der *Spezifikation Schlüsselgenerierungsdienst ePA* der *gematik* in Version 1.4.1 vom 5.November 2020 [gem20b] vorgestellt. Mit Hilfe dieses kryptographischen Protokolls kann ein autorisierter Teilnehmer sich von einem Schlüsselgenerierungsdienst (SGD) einen Schlüssel ausstellen lassen, mit dem dann ein Akten- und Kontextschlüssel für Gesundheitsdaten verschlüsselt wird. Für eine erhöhte Sicherheit werden die Akten- und Kontextschlüssel unter zwei solcher Schlüssel von verschiedenen SGDen geschützt. Die abgeleiteten Schlüssel hängen dabei von individuellen Merkmalen des Teilnehmers ab und lassen sich wiederholt ausstellen.

Unsere Analyse zeigt, dass die einzelnen versichertenspezifischen Schlüssel kryptographisch sicher sind. Das Sicherheitsniveau ist dabei sehr hoch, der Angreifer erhält unter kryptographischen Annahmen keine Informationen über die Schlüssel. Dies gilt:

1. Sogar wenn der Angreifer mehrere gleichzeitig ablaufende Ausführung zwischen Teilnehmern und SGDe aktiv angreift, selbst als sogenannter Man-in-the-Middle-Angreifer.

2. Sogar wenn der Angreifer andere versichertenspezifische Schlüssel oder andere kryptographische Schlüssel korrumpiert.

Die verwendeten kryptographischen Komponenten zum Verschlüsseln, Signieren, Schlüsselableiten etc. im Protokoll entsprechen den aktuellen Empfehlungen der Wissenschaft und Behörden und bieten adäquaten Schutz. Die Parameter für die Verfahren wie Schlüssellängen stimmen ebenfalls mit aktuellen Empfehlungen überein und bieten ausreichend Schutz für die nächsten Jahre. Zusätzlich beruht unsere Analyse nur auf abstrakten Sicherheitseigenschaften der kryptographischen Verfahren wie der IND-CCA-Sicherheit des eingesetzen Verschlüsselungssystems, so dass unerwartete Schwächen der Bausteine durch den Einsatz anderer Verfahren mitigiert werden können: Erfüllen die neuen Verfahren die abstrakten Sicherheitseigenschaften, ist auch das Protokoll nach wie vor beweisbar sicher, da der Protollentwurf robust ist.

1 Protokollbeschreibung

Wir stellen im folgenden die relevanten Aspekte des Protokolls vor und stellen es in geeigneter Form für die Analyse dar.

1.1 Protokoll

Die zugrundeliegende Idee ist, dass sich der Client von jedem der zwei SGDe jeweils einen versichertenspezifischen 256-Bit-AES-Schlüssel k mit Hilfe seines zertifizierten Signaturschlüsselpaares $(\mathsf{sk}_C, \mathsf{pk}_C)$ ausstellen lassen kann. Jeder AES-Schlüssel wird dabei deterministisch vom jeweiligen SGD mit Hilfe eines langfristigen Ableitungsschlüssels ltkd für einen Ableitungsvektors generiert, in den ein Zufallswert RND, die Versichertennummer `KVNR`, der Bezeichner `Bez` für den Ableitungsschlüssel, und je nach Art der Anfrage optional eine Vertreternummer und eine Telematik-ID eingehen. Die beiden versichertenspezifischen Schlüssel werden dann genutzt, um den Akten- und Kontextschlüssel im System kaskadenhaft unter einem AEAD-Verschlüsselungsverfahren (*engl. authenticated encryption with associated data*) zu schützen, wobei jeweils der Ableitungsvektor als assoziierte Daten verwendet wird. Die Daten werden in einem Aktensystem abgelegt.

Grundlegende Operationen. Der jeweilige versichertenspezifische AES-Schlüssel wird durch drei Operationen von den SGDen an den Client übertragen:

`GetPublicKey`**:** Mit der `GetPublicKey`-Operation erhält der Client einen zertifizierten Schlüssel pk_S des Verschlüsselungssystems zum vertraulichen Übertragen von Daten an den SGD. Als Verschlüsselungssystem wird das sogenannte ECIES-Verfahren mit einer standardisierten elliptischen Kurve verwendet. Das zugehörige Schlüsselpaar $(\mathsf{sk}_S, \mathsf{pk}_S)$ wird spätestens alle 15 Minuten aktualisiert, und nicht mehr benötigte Schlüssel gelöscht. Der öffentliche Schlüssel pk_S wird dabei unter einem Zertifikat cert_S des SGDs signiert und vom Client überprüft. Ferner gibt es Ableitungsschlüssel ltkd von mindestens 512 Bits Entropie, die nach spätestens 6 Monaten aktualisiert werden. Diese Ableitungsschlüssel müssen anschließend so lange wie nötig vorgehalten werden, um spätere Anforderungen erneut zu beantworten. Das Unterprotokoll ist in Abbildung 1.1 angegeben.

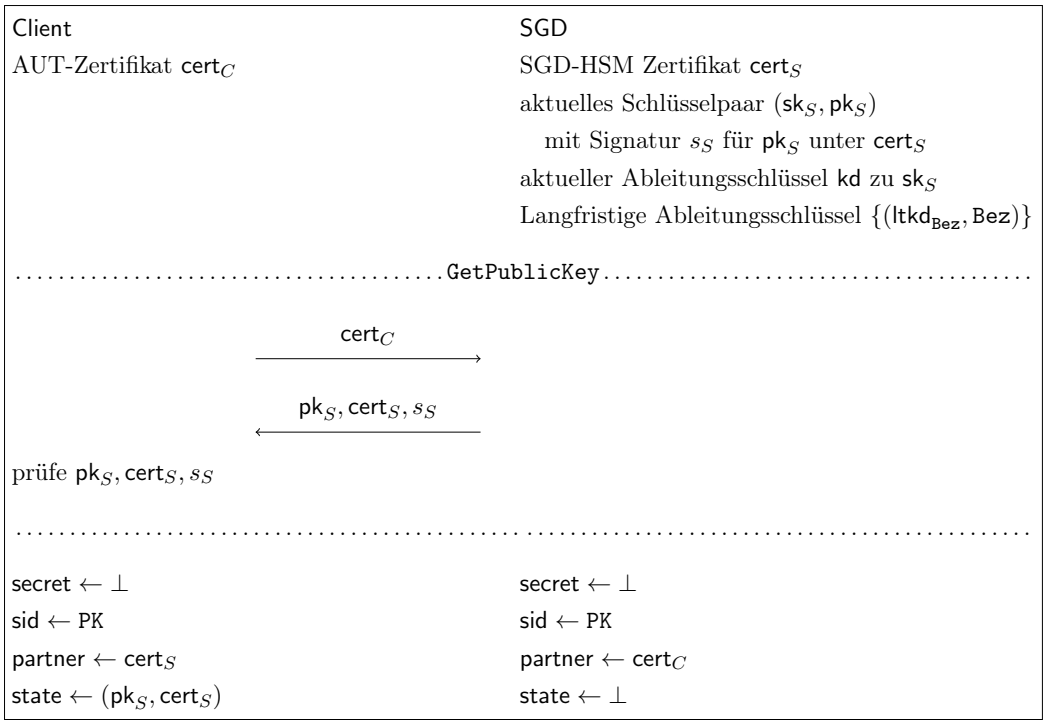

Abbildung 1.1: Operation `GetPublicKey` holt und prüft die öffentlichen ECIES-Schlüssel der beiden SGDe. Prüfen von Unterschriften schließt stets die Gültigkeitsprüfung des Zertifikats mit ein. Fällt die Prüfung negativ aus, fährt der Teilnehmer nicht fort. Das gemeinsame abgeleitete Geheimnis secret wird in diesem Unterprotokoll nicht gesetzt. Der Sitzungskennzeichner sid wird nur zur Vollständigkeit auf einen Konstante PK gesetzt. Die Variable partner identifiziert in der Analyse den Kommunikationspartner; state beschreibt Informationen, die an folgende Ausführungen weitergegeben werden.

`GetAuthenticationToken`: Mit der `GetAuthenticationToken`-Operation lässt sich der Client ein Authentisierungstoken AT vom jeweiligen SGD ausstellen. Das Authentisierungstoken ist das Resultat einer Schlüsselableitung des SGDs mit einem dem aktuellen Schlüsselpaar zugeordneten Schlüssel kd von mindestens 256 Bits Entropie unter dem Ableitungsvektor, der aus einem flüchtigen ECIES-Schlüssel epk_C des Clients mit Signatur unter dem AUT-Zertifikat cert_C besteht, sowie dem Zertifikat selbst und den Hash-Werten der beiden öffentlichen Schlüssel pk_S der SGDe besteht. Der Client schickt diese Daten verschlüsselt unter dem öffentlichen Schlüssel pk_S des jeweiligen SGD mit einer Challenge ch, die der SGD zusammen mit den Token AT zur Kontrolle zurückschickt, verschlüsselt unter dem Client-Schlüssel epk_C. Dieser Schlüssel kann für beide SGDe verwendet werden. Das Unterprotokoll ist in Abbildung 1.2 aufgeführt.

`KeyDerivation`: Mit der `KeyDerivation`-Operation kann der Client beim jeweiligen SGD ein aktuelles Authentisierungstoken gegen einen versichertenspezifischen AES-Schlüssel

"eintauschen". Dabei kann sich der Versicherte mit Versicherungsnummer `KVNR` einen Schlüssel für sich selbst,

Regel `r1:KVNR` oder `r1:`RND`|KVNR|Bez`,

oder für einen Berechtigungsempfänger (Vertreter mit Nummer `KVNR-Ver` oder Telematik-ID `T-ID`) einholen,

Regel `r2:KVNR-Ver/T-ID` oder `r2:`RND`|KVNR-Ver/T-ID|Bez`,

oder durch einen Vertreter mit `KVNR` für einen Berechtigten (mit Telematik-ID `T-ID`) für einen Inhaber `KVNR-Inh` anfragen bzw. der Berechtigte sich den Schlüssel ausstellen lassen,

Regel `r3:T-ID|KVNR-Inh` oder `r3:`RND`|KVNR-Inh|KVNR-Ver|T-ID|Bez`.

Der optionale Wert RND deutet dabei an, ob der Schlüssel neu inititalisert wird (und in dem Fall der SGD einen neuen Wert RND wählt und hinzufügt), oder ein früherer Wert RND verwendet werden. Dabei wird der alte Wert vom Dokumentensystem nach vorheriger Authentisierung an den Client gegeben. Der Bezeichner `Bez` beschreibt den entsprechenden langfristigen Ableitungsschlüssel des SGDs. Steht `Bez` nicht in der Anfrage, fügt der SGD den aktuellen Wert `Bez` an. Der SGD prüft, dass die Daten in der Ableitungsregel mit den entsprechenden Daten im übermittelten Zertifikat übereinstimmen.

Im Protokoll `KeyDerivation` selbst sendet der Client das aktuelle Authentisierungstoken AT mit einem zufälligen 256-Bit reqID und der Regel, verschlüsselt unter dem aktuellen öffentlichen Schlüssel pk_S des SGDs. Der SGD prüft die erhaltenen Daten und leitet den Schlüssel k mit dem Ableitungsvektor für den langfristigen Ableitungsschlüssel ltkd unter dem Bezeichner `Bez` ab. Der SGD sendet Schlüssel und Ableitungsvektor zusammen mit den Werten AT und reqID zurück an den Client, verschlüsselt mit dem flüchtigen Schlüssel epk_C des Client. Der Client kann dann, nachdem er die erhaltenen Daten entschlüsselt und geprüft hat, die Ableitungsvektoren aller SGDe als assoziierte Daten für die Verschlüsselungskaskade im Aktensystem ablegen. Das Unterprotokoll ist in Abbildung 1.3 dargestellt.

Schlüsselableitung. Für die Ableitung des versichertenspezifischen 256-Bit-AES-Schlüssel in einem SGD wird ein langfristiger Ableitungsschlüssel ltkd mit mindestens 512 Bits Entropie verwendet. Um spätere Anfragen mit dem gleichen Ableitungsschlüssel zu beantworten, müssen diese Schlüssel langfristig vorgehalten werden. Wir gehen bei der Protokollbeschreibung davon aus, dass ein SGD bereits eine Liste solcher Schlüssel $\mathsf{ltkd}_{\texttt{Bez}}$ mit Bezeichnern `Bez`

hält. Die Auswahl des richtigen Schlüssels erfolgt durch den Eintrag `Bez` in der Regel `rx`, die im `KeyDerivation`-Schritt übertragen wird. Der SGD verwirft, wenn dabei ein unbekannter Bezeichner verwendet wird. Dies ist in den Abbildungen nicht explizit aufgeführt.

Für den Ableitungsvektor bei der Schlüsselvektor erweitert der SGD die Regel `rx` zu $\texttt{rx}^{\text{ext}}$. Dabei wird eventuell ein zufälliger Wert `RND` und der aktuelle Bezeichner `Bez` eingefügt. Zusätzlich überprüft der SGD auch, dass die Identitäten wie `KVNR` mit den Einträgen im Zertifikat übereinstimmen. Wir gehen deshalb im Folgenden davon aus, dass auch nur solche Regeln übergeben werden.

1.2 Bemerkungen

Zur Vereinheitlichung der Darstellung für kryptographische Zwecke haben wir stellenweise eine leicht andere Beschreibungsform gewählt. Diese Modifikationen beeinflussen allerdings nicht die Sicherheit. Teilweise sichern sie sogar gegen stärkere Angriffe als vorgesehen ab. Im Einzelnen betrifft unsere Darstellung folgende Punkte:

1. Die Schlüsselgenerierungsdienste bestehen jeweils aus drei Komponenten: einer HTTPS-Schnittstelle für die Kommunikation mit der Außenwelt; einer Request-verarbeitenden Einheit (RVE), die mit der HTTPS-Schnittstelle und der dritten Komponente, dem Hardware Security Modul (SGD-HSM), kommuniziert. Das SGD-HSM selbst kommuniziert nicht direkt mit der HTTPS-Schnittstelle. Die RVE übernimmt Aufgaben wie die Bereitstellung der Gültigkeitsprüfungsdaten von Zertifikaten per OCSP für das SHD-HSM; die Prüfung selbst erfolgt (auch) innerhalb des SGD-HSM. Das SGD-HSM muss hohe Sicherheitsanforderungen erfüllen, unter anderem eine FIPS-140-2-Zertifizierung auf Level 3 und das Firmware-Modul eine Sicherheitsbegutachtung durch eine anerkannte unabhängige Instanz. Für die kryptographische Analyse fassen wir die drei Komponenten zusammen. Die Kommunikation innerhalb eines SGD muss daher als sicher angenommen werden. Der Angreifer darf allerdings später einzelne SGD korrumpieren.

2. Die Zertifikatsprüfung auf Seiten des SGD (bzw. des RVE) kann zwischengespeichert vorliegen, beispielsweise wenn `GetPublicKey` noch nicht länger als 4 Stunden her ist. Wir nehmen hier zur Vereinfachung an, dass die Prüfung erst im Unterprotokoll `GetAuthenticationToken` gemacht wird.

3. Wir vereinheitlichen die flüchtigen Schlüssel $(\mathsf{esk}_C, \mathsf{epk}_C)$ von Clients, die nur in zwei

Ausführungen verwendet werden sollen, und die Schlüssel $(\mathsf{sk}_S, \mathsf{pk}_S)$ der SGDe, die spätestens alle 15 Minuten aktualisiert werden sollen, zu sogenannten temporären Schlüsseln. Wir überlassen es allerdings dem Angreifer, wann neue temporäre Schlüssel gewählt werden, eventuell werden die Client-Schlüssel dadurch sogar öfter verwendet. Dies verstärkt nur die Sicherheitsaussage.

4. Wir nehmen an, dass die Signatur s_C des Clients, die im `GetAuthenticationToken`-Protokoll für den eigenen flüchtigen Schlüssel und die Hashwerte der beiden SGD-Schlüssel (gemäß Kodierung nach *A_17900* in [gem20b]) erstellt wurde, in einer folgenden `KeyDerivation`-Ausführung wiederverwendet wird (sofern die Schlüssel gleich bleiben). Für die Sicherheit ist es nicht relevant, falls hier doch eine neue Signatur erstellt werden sollte.

5. Wir lassen den Angreifer am Ende der `KeyDerivation`-Operation die Daten $\mathtt{rx}^{\mathrm{ext}}$ wissen, die die (erweiterten) Regeldaten darstellen und als assoziierte Daten für die Verschlüsselung dienen. Diese Daten beinhalten unter anderem die `KVNR` des Anwenders, für die es keinen Grund zur Veröffentlichung außerhalb des Gesundheitsystems gibt. Dadurch machen wir in unserem Fall den Angreifer nur stärker und zeigen, dass selbst dann die abgeleiteten Schlüssel geheim bleiben, wenn diese Daten preisgegeben werden.

6. Der SGD muss bei einer gültigen `KeyDerivation`-Anfrage ohne Ableitungsschlüsselbezeichner `Bez` den aktuellen Wert eintragen. Wir spezifizieren hier nicht, welches dieser aktueller Bezeichner ist, da für unsere Analyse die langfristen Ableitungsschlüssel ltkd alle geheim bleiben müssen. Für die Sicherheit genügt es dann, einen beliebigen Bezeichner `Bez` zu wählen.

7. Die Kodierung der Daten wird in der Beschreibung spezifiziert. Wir haben hier zur Vereinfachung nur die essentiellen Einträge dargestellt. Für uns relevant ist dabei, dass die verschlüsselten Nachrichten sich durch einen Präfix unterscheiden (`CH` für `'Challenge'`, `RSP` für 'Response', KD für `'Key Derivation'`, und `OKKD` für `'OK-KeyDerivation'`.

Client		SGD
AUT-Zertifikat cert_C		SGD-HSM Zertifikat cert_S
für Signaturschlüssel $(\mathsf{sk}_C, \mathsf{pk}_C)$		
aktuelles Schlüsselpaar $(\mathsf{esk}_C, \mathsf{epk}_C)$		aktuelles Schlüsselpaar $(\mathsf{sk}_S, \mathsf{pk}_S)$
		mit Signatur s_S für pk_S unter cert_S
		aktueller Ableitungsschlüssel kd zu sk_S
		Langfristige Ableitungsschlüssel $\{(\mathsf{ltkd}_{\mathtt{Bez}}, \mathtt{Bez})\}$
$\mathsf{state} = (\mathsf{pk}_S, \mathsf{cert}_S)$		
$\mathsf{input} = (\mathsf{pk}_i, i)$ // $\mathsf{pk}_{3-i} \leftarrow \mathsf{pk}_S$		
.....	`GetAuthenticationToken`	.....
$\mathsf{epk}_C^{\mathrm{ext}} \leftarrow \mathsf{epk}_C \vert \mathsf{H}(\mathsf{pk}_1) \vert \mathsf{H}(\mathsf{pk}_2)$		
$H \leftarrow \mathsf{H}(\mathsf{epk}_C^{\mathrm{ext}} \vert \mathsf{cert}_C)$		
$s_C \leftarrow\!\!\$\ \mathsf{Sig}(\mathsf{sk}_C, H)$		
wähle $\mathsf{ch} \leftarrow\!\!\$\ \{0,1\}^{256}$		
$c_C^{\mathrm{at}} \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}_S, \mathtt{CH} \vert \mathsf{ch} \vert H)$	$\xrightarrow{\mathsf{epk}_C^{\mathrm{ext}}, s_C, \mathsf{cert}_C, c_C^{\mathrm{at}}}$	
		prüfe $\mathsf{H}(\mathsf{epk}_C^{\mathrm{ext}} \vert \mathsf{cert}_C), \mathsf{cert}_C, s_C$
		$m_S \leftarrow \mathsf{Dec}(\mathsf{sk}_S, c_C^{\mathrm{at}})$
		$A \leftarrow \mathsf{epk}_C^{\mathrm{ext}} \vert \mathsf{cert}_C$
		prüfe, dass $\mathtt{CH}$, $\mathsf{H}(A)$ in m_S korrekt
		$\mathsf{AT} \leftarrow \mathsf{KDF}(\mathsf{kd}, A)$
		$c_S^{\mathrm{at}} \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{epk}_C, \mathtt{RSP} \vert \mathsf{ch} \vert \mathsf{H}(A) \vert \mathsf{AT})$
	$\xleftarrow{c_S^{\mathrm{at}}}$	
$m_C \leftarrow \mathsf{Dec}(\mathsf{esk}_C, c_S^{\mathrm{at}})$		
prüfe, dass $\mathtt{RSP}$, ch und H in m_C korrekt		
.....		.....
$\mathsf{secret} \leftarrow \mathsf{AT}$		[$\mathsf{secret} \leftarrow \mathsf{AT}$]
$\mathsf{sid} \leftarrow (\mathtt{AT}, \mathsf{pk}_S, \mathsf{cert}_S, A)$		$\mathsf{sid} \leftarrow (\mathtt{AT}, \mathsf{pk}_S, \mathsf{cert}_S, A)$
$\mathsf{partner} \leftarrow \mathsf{cert}_S$		$\mathsf{partner} \leftarrow \mathsf{cert}_C$
$\mathsf{state} \leftarrow (\mathsf{pk}_S, \mathsf{cert}_S, \mathsf{AT})$		$\mathsf{state} \leftarrow \perp$

Abbildung 1.2: Operation `GetAuthenticationToken` zum Transfer eines Authentisierungstokens AT zum Client. Bei den Prüfungen der entschlüsselten Nachrichten wird verifiziert, dass die Werte an der richtigen Stelle in Klartext vorkommen. Der Client erhält als zusätzliche Eingabe input die beiden öffentlichen Schlüssel der SGDe, die er sich per `GetPublicKey` jeweils vorher hat geben lassen. Die Konstanten `CH` und `RSP` stehen für die Zeichenketten `'Challenge'` und `'Response'`. Im Protokoll speichert der SGD nicht das Authentisierungstoken; wir geben diesen Schritt hier nur zur Vereinfachung für die Analyse an.

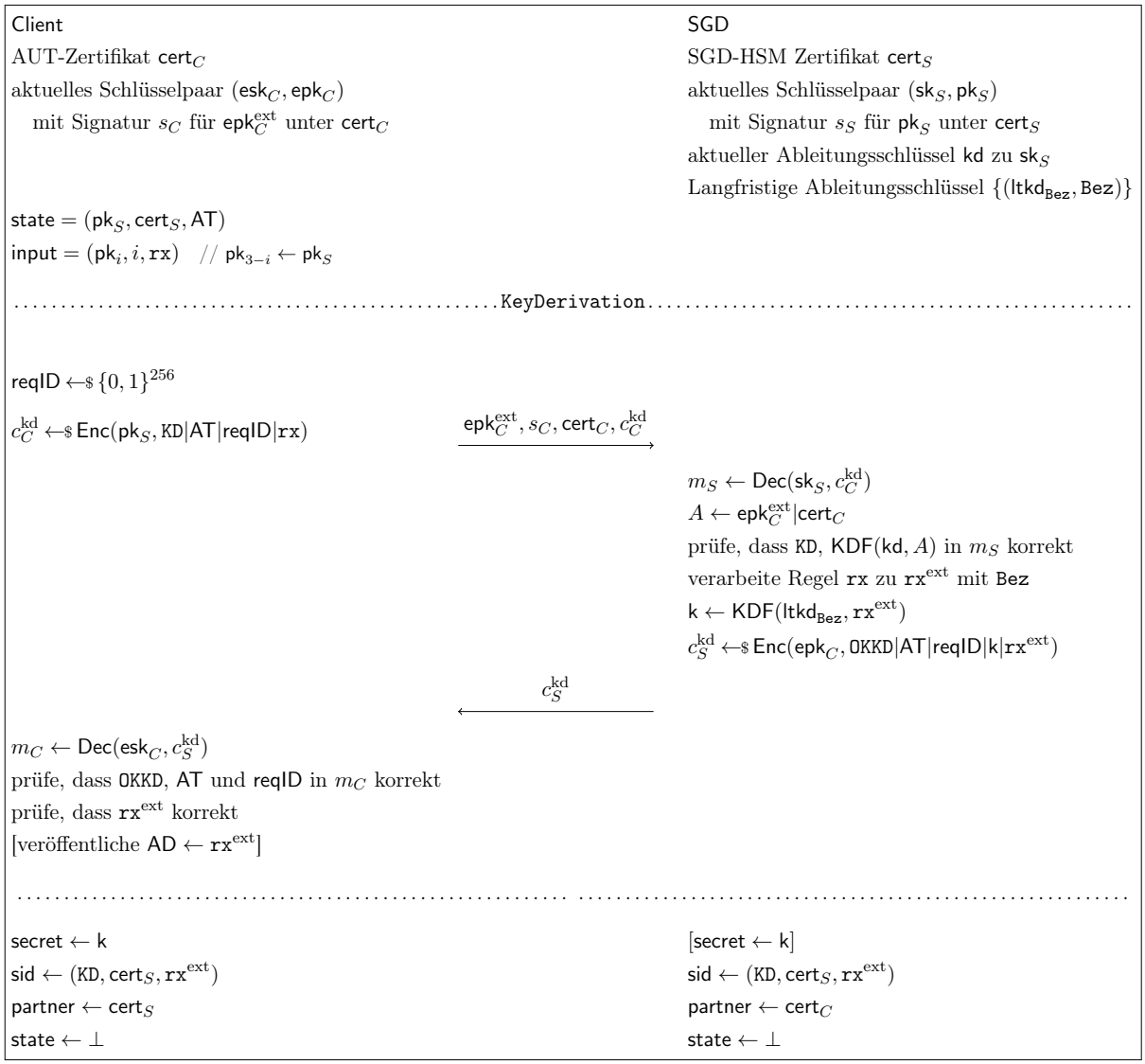

Abbildung 1.3: Operation `KeyDerivation`, die einen Authentisierungstoken in einen abgeleiteten Schlüssel k umwandelt. Bei den Prüfungen der entschlüsselten Nachrichten wird verifiziert, dass die Werte an der richtigen Stelle in Klartext vorkommen. Die Ableitungsregel `rx` des Clients wird durch den SGD in die Regel $\mathtt{rx}^{\text{ext}}$ überführt. Dabei werden auch die Indentitäten mit den AUT-Zertifikatseinträgen verglichen. Die Konstanten `KD` und `OKKD` stehen für die Zeichenketten `'KeyDerivation'` und `'OK-KeyDerivation'`. Laut Protokollbeschreibung veröffentlicht der Client nicht die assoziierten Daten AD, sondern legt sie zusammen mit dem verschlüsselten Daten ab, und auch der SGD speichert nicht den geheimen Schlüssel; wir haben es hier nur für die übliche kryptographische Protokollbeschreibung aufgeführt.

2 Sicherheitsmodell

Wir beschreiben in diesem Abschnitt das zugrundeliegende Sicherheitsmodell für die Schlüsselableitung. Da man die von den SGDen bereitgestellten Schlüssel auf jeweils beiden Seiten vorliegen, folgen wir der bewährten Darstellung aus Schlüsselaustauschverfahren, speziell dem sogenannten Bellare-Rogaway-Modell (BR-Modell) [BR93]. Dieses Modell wurde beispielsweise auch für die Analyse der Verfahren der elektronischen Ausweise [BFK09] oder des neuen TLS 1.3-Standards [DFGS21] herangezogen.

Das BR-Modell betrachtet gleichzeitig mehrer Ausführungen des Prokotolls zwischen verschiedenen Teilnehmern, aber auch mehrere gleichzeitige ablaufende Ausführung eines Teilnehmers mit anderen Kommunikationspartnern (beispielsweise ein SGD, der von mehreren Clients parallel angefragt wird, oder auch die laut Spezifikation vorgesehene parallele Anfrage eines Clients an beide SGDe). Zusätzlich erlaubt das Modell dem sogenannten Dolev-Yao-Angreifer die volle Kontrolle über die Netzwerkkommunikation, d.h. der Angreifer darf entscheiden, wann welche Protokollnachricht abgeliefert wird bzw. kann neue Nachrichten einfügen oder gesendete Nachrichten manipulieren. Insbesondere kann er einen sogenannten Man-in-the-Middle-Angriff starten. Zusätzlich zu den starken Angriffsmöglichkeiten und den damit verbundenen starken Sicherheitsgarantien deckt das BR-Modell Abhängigkeiten zwischen Ausführungen ab. Sicherheit der abgeleiteten Schlüssel (im BR-Modell üblicherweise Sitzungsschlüssel genannt) soll selbst dann garantiert sein, wenn andere abgeleitete Schlüssel bekannt werden. Auch Abhängigkeiten zwischen angegriffenen Schlüsseln werden betrachtet.

2.1 Identitäten, Schlüssel und Zertifikate

Das BR-Modell umfasst mehrere Teilnehmer, die durch eine administrative Identität id unterschieden werden. Wir gehen davon aus, dass der Angreifer die Menge $\mathcal{U}$ dieser Identitäten kennt. Jedem Teilnehmer id wird ein langfristiges, zertifiziertes Schlüsselpaar $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}, \mathsf{cert}_{\mathsf{id}})$ zugeordnet. Im Fall des SGD-Protokolls sind dies die Signaturschlüssel der Clients und SGDe, bei den SGDen beinhaltet der geheime Schlüssel zusätzlich die langfristigen Ableitungsschlüssel. Wir gehen davon aus, dass diese Schlüssel und Zertifikate zu Beginn des Angriffs erzeugt werden, und dass der Angreifer die öffentlichen Daten, also die öffentlichen Schlüssel $\mathsf{pk}_{\mathsf{id}}$ und die Zertifikate $\mathsf{pk}_{\mathsf{id}}$, erhält. In der Regel ist der öffentliche Schlüssel $\mathsf{pk}_{\mathsf{id}}$ Teil des Zertifikats; wir listen ihn allerdings oft explizit auf.

Neben den langfristigen Schlüsseln besitzen die Teilnehmer temporäre Schlüsseln, die nur kurzfristig verfügbar sind, aber eventuell in mehreren Ausführungen verwendet werden (wie die ECIES-Schlüssel der Clients und SGDe). Wie in der Kryptographie üblich verzichten wir auf die explizite Darstellung einer Zeitkomponente, um die Lebensdauer der temporären Schlüssel zu begrenzen. Stattdessen überlassen wir es dem Angreifer, wann diese Schlüssel erneuert werden. Im Unterschied zu den langfirstigen Schlüsseln können die temporären Schlüssel nicht korrumpiert werden.

Bei den Zertifikaten gehen wir davon aus, dass jeder Identität id genau ein Zertifikat cert zugeordnet werden kann. Wir schreiben auch $\mathsf{id}(\mathsf{cert})$ für diese eindeutig zugeordnete Identität zu einem Zertifikat cert. Ferner enthält das Zertifikat einen —im Unterschied zur adminstrativen Identität id— im Protokoll auslesbaren Kennzeichner $\mathtt{ID}$ wie beispielsweise die Versichertennummner $\mathtt{KVNR}$ oder die Telematik-ID $\mathtt{T\text{-}ID}$. Ein solcher Kennzeichner $\mathtt{ID}$ kann in mehreren Zertifikaten vorkommen, wenn beispielsweise ein neues Zertifikat für einen Versicherten ausgestellt wird. Auch hier schreiben wir bei Bedarf $\mathtt{ID}(\mathsf{cert})$ für diesen Kennzeichner eines Zertifikats cert.

Der Kennzeichner $\mathtt{ID}$ im Zertifikat wird im $\mathtt{KeyDerivation}$-Protokoll vom SGD gegen die Ableitungsregel geprüft. Für das SGD sind diesbezüglich verschiedene Zertifikate eines Kennzeichners $\mathtt{ID}$ nicht zu unterscheiden. Dadurch wird gewährleistet, dass ein Teilnehmer die Schlüssel immer noch abrufen kann, auch wenn das alte Zertifikat ungültig geworden ist. Daher gehen wir bei der Korruption eines Teilnehmers mit administrativer Identität id davon aus, dass alle "Klone" id' mit gleichem Kennzeichner $\mathtt{ID}$ ebenfalls korrumpiert werden. Sollte in der Realität ein Zertifikat mit $\mathtt{ID}$ unsicher werden, z.B. weil die Karte verloren gegangen ist, und ein neues Zertifikat für den Kennzeichner $\mathtt{ID}$ ausgestellt werden, so sorgt die Gültigkeitsprüfung in der Spezifikation für das Aussortieren des alten Zertifikats. Dies erfolgt aber außerhalb der kryptographischen Betrachtung des Protokolls.

2.2 Sitzungen

Zentral für das BR-Sicherheitsmodell ist der Begriff der Sitzung (*engl. session*). Eine Sitzung spiegelt die Sicht eines Teilnehmers einer Ausführung wieder. Jede Sitzung hat einen eindeutigen, administrativen Kennzeichner (*engl. label*) lbl, der es dem Angreifer erlaubt, spezifische Sitzung anzusprechen. Er wird bei der Initialisierung der Sitzung festgelegt und dem Angreifer mitgeteilt. Aus kryptographischer Sicht umfasst eine Sitzung neben den internen Daten des Teilnehmers zur Ausführung des Protokolls folgende Einträge:

owner und role: Beschreiben die Identität id und die Rolle (initiator oder responder) des Sitzungteilnehmers. Wird bei der Initialisierung der Sitzung vom Angreifer festgelegt.

status: Beschreibt den Status der Sitzung: initiated für neu initialisierte Sitzungen, running für laufende Sitzungen, accepted für erfolgreich beendete Sitzungen, und rejected für Sitzungen, die vorzeitig abgebrochen wurden (z.B. weil eine Entschlüsselung nicht funktionierte).

keys: Speichert die für die Sitzung aktuellen temporären Schlüsselteile der Teilnehmer, öffentlich und geheim.

type: Die Art der Sitzung beschreibt das entsprechende Unterprotokoll, das ausgeführt wird. Im Fall des SGD-Protokolls gibt es die Arten `GetPublicKey`, `GetAuthenticationToken`, und `KeyDerivation`, wobei eine Ausführung der Unterprotokolle `GetPublicKey` bzw. von `GetAuthenticationToken` anschließend von mehreren `GetAuthenticationToken`- bzw. `KeyDerivation`-Ausführungen gefolgt werden kann. Wir verwenden die Funktion next, um die Reihenfolge festzulegen:

$$\begin{aligned} \mathsf{next}(\texttt{GetPublicKey}) &= \texttt{GetAuthenticationToken}, \\ \mathsf{next}(\texttt{GetAuthenticationToken}) &= \texttt{KeyDerivation}, \\ \mathsf{next}(\texttt{KeyDerivation}) &= \bot. \end{aligned}$$

input: Beschreibt die eventuell übergebenen Eingabedaten bei der Intialisierung der Sitzung.

state: Beschreibt Daten, die beim Zusammenführen von verschiedenen Sitzungen übertragen werden, beispielsweise die öffentlichen Schlüssel aus verschiedenen Sitzungen mit SGDen. Wird am Ende der Sitzung gesetzt.

previous: Beschreibt die vorher ausgeführte Sitzung lbl' und erlaubt damit eine Referenz auf deren Daten, die beim Übergang zwischen den verschiedenen Sitzungen übertragen werden sollen, beispielsweise das Authentisierungstoken aus einer Sitzung der Art `GetAuthenticationToken`.

sid: Die Sitzungskennzeichner (*engl. session identifier*) sid ergibt sich sich aus den Protokolldaten einer Sitzung. Er erlaubt es, Sitzungen einzelner Teilnehmer zuzuordnen, so dass man zusammengehörige Aufführungen erkennt. Wird erst am Ende der Ausführung einer Sitzung gesetzt, sofern die Ausführung erfolgreich beendet wurde. Er besteht in der Regel aus öffentlichen Protokolldaten (vgl. [BFWW11]).

partner: Dieser Eintrag beschreibt den von der Sitzung vermeintlichen identifizierten Partner einer Sitzung, in der Regel durch dessen Zertifikat bestimmt.

security: Beschreibt den Sicherheitsstatus der Sitzung: revealed für Sitzungen, wo der Sitzungsschlüssel bekannt wurde, fresh für Sitzungen, bei denen der Sitzungsschlüssel geheim sein soll, und tested bei Sitzungen, bei denen der Angreifer den Schlüssel angreift.

secret: Beschreibt das Geheimnis einer Sitzung, also das Authentisierungstoken (im Fall des Unterprotokolls `GetAuthenticationToken`) bzw. den abgeleiteten Schlüssel (im Fall von `KeyDerivation`).

Für die einzelnen Einträge einer Sitzung mit Kennzeichner lbl schreiben wir oft lbl.owner, lbl.role, lbl.status usw.

2.3 Vertraulichkeit

Wir definieren zunächst die Vertraulichkeitseigenschaft (*engl. secrecy*) analog zu Schlüsselaustauschverfahren. Diese Eigenschaft besagt, dass Sitzungsschlüssel nicht von zufälligen Schlüsseln zu unterscheiden sind, selbst bei aktiven Angriffen, die andere Sitzungsschlüssel lernen. Dabei müssen wir einen trivialen Angriff ausschließen: Wenn der Client und der SGD einen Schlüssel ableiten, also die beiden Sitzungen zusammengehören und den gleichen Kennzeichner sid haben, dann darf der Angreifer nicht den einen Teilnehmer angreifen (also Test aufrufen), und den Kommunikationspartner nach dem Sitzungsschlüssel fragen (also Reveal auf dem Partner aufrufen). Sonst könnte er trivialerweise den angegriffen Schlüssel bestimmen. Um dies zu verhindern, erlauben wir keine solche Angriffskombination auf zusammengehörige Sitzungen mit dem gleichen Kennzeichner sid. Idealerweise werden auch wirklich nur die tatsächlichen zusammenghörigen Sitzungen so identifiziert. Das wird die Eigenschaft der Eindeutigkeit garantieren, die wir anschließend definieren.

Für das Sicherheitsspiel wird zu Beginn ein geheimes Bit b gezogen. Diese Bit b bestimmt, ob ein Angreifer später bei Test-Anfragen den richtigen Schlüssel oder einen zufälligen Wert bekommt. Ziel des Angreifers ist es, dieses Bit b signifikant besser zu bestimmen, als zu raten. Kann kein Angreifer dies, so folgt die Ununterscheidbarkeit der richtigen Schlüssel von zufälligen Werten. Zur Vereinfachung—da nur der Client dauerhaft den abgeleiteten Schlüssel verwendet— erlauben wir Test-Anfragen und Reveal-Anfragen nur für Client-Sitzungen der Art `KeyDerivation`. Initial wird die Menge $\mathcal{C} \subseteq \mathcal{U}$ der korrumpierten Teilnehmer $\mathcal{C} \leftarrow \emptyset$ auf die leere Menge gesetzt.

Der Angreifer kommuniziert während des Angriffs mit den Sitzungen und Teilnehmern über folgende Befehle:

INIT: Mit dem INIT-Befehl initialisiert der Angreifer eine neue Sitzung, die im folgenden mit dem SEND-Befehl angesprochen werden kann. Als Argumente übernimmt der INIT-Befehl eine Identität id des Teilnehmers, die Rolle r ($\mathsf{initiator}$ oder $\mathsf{responder}$), die Art t der Ausführung (hier die Arten `GetPublicKey`,`GetAuthenticationToken`, und `KeyDerivation`), sowie eine (eventuell leere) zusätzliche Eingabe I und eventuell einen Kennzeichner lbl'. Im Fall einer Eingabe lbl' wird zunächst geprüft, dass die vermeintlich vorige Sitzung mit Kennzeichner lbl' passt, also dass $\mathsf{lbl}'.\mathsf{owner} = \mathsf{id}$ und $\mathsf{lbl}'.\mathsf{status} = \mathsf{accepted}$ und $\mathsf{next}(\mathsf{lbl}'.\mathsf{type}) = t$. Es wird ferner geprüft, dass auch vorher eine enstprechende Ausführung wirklich stattgefunden hat, also dass ein lbl' übergeben wird, sofern es ein t' mit $\mathsf{next}(\mathsf{lbl}'.t') = t$ gibt. Sonst wird hier bereits abgebrochen.

Der Befehl generiert bei gültiger Anfrage eine neue Sitzung mit Kennzeichner lbl mit folgenden Einträgen:

- Setze $\mathsf{lbl}.\mathsf{owner} \leftarrow \mathsf{id}$, $\mathsf{lbl}.\mathsf{status} \leftarrow \mathsf{initiated}$, $\mathsf{lbl}.\mathsf{role} \leftarrow r$, $\mathsf{lbl}.\mathsf{type} \leftarrow t$, und $\mathsf{lbl}.\mathsf{keys} \leftarrow (\mathsf{sk}, \mathsf{pk})$ für das aktuelle temporäre Schlüsselpaar des Teilnehmers.
- Setze $\mathsf{lbl}.\mathsf{security} \leftarrow \mathsf{fresh}$, falls $\mathsf{id} \notin \mathcal{C}$ und $\mathsf{lbl}'.\mathsf{security} \neq \mathsf{revealed}$ (falls lbl' vorhanden) bzw. $\mathsf{lbl}.\mathsf{security} \leftarrow \mathsf{revealed}$, falls $\mathsf{id} \in \mathcal{C}$ oder $\mathsf{lbl}'.\mathsf{security} = \mathsf{revealed}$ (sofern lbl' vorhanden). Wenn bereits der Teilnehmer oder die vorherige Sitzung korrumpiert wurde, dann kann auch diese Sitzung nicht mehr getested werden.
- Setze $\mathsf{lbl}.\mathsf{input} \leftarrow I$ (sofern I vorhanden, sonst auf $\perp$). Setze $\mathsf{lbl}.\mathsf{previous} \leftarrow \mathsf{lbl}'$ (sofern lbl' vorhanden); sonst setze den Eintrag auf $\perp$. Erlaubt es, auf Daten aus der vorigen Sitzung oder auch aus anderen Sitzungen (z.B. den öffentlichen Schlüssel des anderen SGD) zuzugreifen.
- Alle anderen Einträge werden zu Beginn auf $\perp$ gesetzt.

SEND: Mit dem SEND-Befehl für Eingabe lbl und der (optionalen) Protokollnachricht m übergibt der Angreifer m an die Sitzung mit Kennzeichner lbl. Dazu wird zunächst geprüft, dass $\mathsf{lbl}.\mathsf{status} = \mathsf{running}$ oder $\mathsf{initiated}$. Wird dabei m ausgelassen, muss zusätzlich $\mathsf{lbl}.\mathsf{status} = \mathsf{initiated}$ und $\mathsf{lbl}.\mathsf{role} = \mathsf{initiator}$ gelten, also der Teilnehmer die Kommunikation beginnen soll. Setze zunächst $\mathsf{lbl}.\mathsf{status} \leftarrow \mathsf{running}$. Die Nachricht m wird dann an das Protokoll gegeben und erzeugt eine Protokollantwort bzw. der Initiator erzeugt die erste Protokollnachricht. Je nach Protokollausführung ändert sich der Status der Sitzung eventuell zu $\mathsf{accepted}$ oder $\mathsf{rejected}$. Im Fall von $\mathsf{accepted}$ muss ein Sitzungswert $\mathsf{lbl}.\mathsf{sid} \neq \perp$, ein Sitzungsschlüssel $\mathsf{lbl}.\mathsf{secret}$ und ein Partner $\mathsf{lbl}.\mathsf{partner}$ gesetzt werden. Ist $\mathsf{lbl}.\mathsf{status} = \mathsf{fresh}$ und gibt es dann eine weitere Sitzung lbl' mit $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl}.\mathsf{sid}$, bei der der Status $\mathsf{lbl}'.\mathsf{status} = \mathsf{tested}$ ist, dann setze auch $\mathsf{lbl}.\mathsf{status} \leftarrow \mathsf{tested}$. Wenn Partnersitzung bereits getested wurde, dann ist aus Kompatibilitätsgründen die Sitzung hier auch quasi schon getestet. Die Protokollantwort wird an den Angreifer zurückgegeben.

CORRUPT: Der CORRUPT-Befehl nimmt als Argument eine Teilnehmeridentität id und gibt den langfristigen Schlüssel $\mathsf{sk}_{\mathsf{id}}$ zurück. Füge id zu $\mathcal{C}$ hinzu. Verfahre entsprechend mit allen Identitäten id' (mit Zertifikat $\mathsf{cert}_{\mathsf{id}'}$) mit $\mathtt{ID}(\mathsf{cert}_{\mathsf{id}}) = \mathtt{ID}(\mathsf{cert}_{\mathsf{id}'})$ mit dem gleichen Kennzeicher im Zertifikat. Dieser Befehl CORRUPT darf nur vor den anderen Befehlen aufgerufen werden.

UPDTEMPKEY: Nimmt als Argument die Identität id eines Teilnehmers. Wählt ein neues temporäres Schlüsselpaar $(\mathsf{sk}, \mathsf{pk})$ für diesen Teilnehmer und speichert dieses under id ab. Von nun an wird dieses Schlüsselpaar in allen neu initiierten Ausführungen dieses Teilnehmers im Eintrag keys verwendet.

REVEAL: Nimmt einen Kennzeichner lbl als Argument. Gibt dann $\mathsf{lbl.secret}$ zurück, sofern $\mathsf{lbl.status} = \mathsf{accepted}$ und $\mathsf{lbl.security} = \mathsf{fresh}$ und $\mathsf{next}(\mathsf{lbl.type}) = \bot$, sonst $\bot$. Ist die Antwort nicht $\bot$, dann wird $\mathsf{lbl.security} \leftarrow \mathsf{revealed}$ gesetzt.

TEST: Nimmt einen Kennzeichner lbl als Argument. Gibt unmittelbar $\bot$ zurück, sofern $\mathsf{lbl.status} \neq \mathsf{accepted}$ oder $\mathsf{lbl.security} \neq \mathsf{fresh}$, also die Sitzung noch nicht akzeptiert hat, oder bereits getested oder korrumpiert wurde. Gibt auch $\bot$ zurück, wenn $\mathsf{next}(\mathsf{lbl.type}) \neq \bot$ oder $\mathsf{lbl.role} \neq \mathsf{initiator}$, also wenn nicht die Schlüsselableitung im finalen Unterprotokoll betroffen ist oder der Angriff nicht dem Client gilt. Sonst berechnet der Befehl in Abhängigkeit des geheimen Bits b die Antwort als $\mathsf{lbl.secret}$ (für $b = 0$) bzw. als zufälligen Schlüssel aus $\{0,1\}^{|\mathsf{lbl.secret}|}$ (für $b = 1$). Setze $\mathsf{lbl.status} \leftarrow \mathsf{tested}$. Setze auch alle gepartnerten Sitzungen lbl' mit $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl.sid}$ und $\mathsf{lbl}'.\mathsf{status} = \mathsf{fresh}$ auf getestet, $\mathsf{lbl}'.\mathsf{status} \leftarrow \mathsf{tested}$. (Bereits korrumpierte Sitzungen mit $\mathsf{lbl}'.\mathsf{status} = \mathsf{revealed}$ führen später dazu, dass der Angreifer verliert.)

Der Angreifer führt die oben angeführten Befehle in beliebiger Reihenfolge aus (bis auf CORRUPT). Er gibt am Ende einen Rateversuch a für das geheime Test-Bit b aus. Er gewinnt, wenn

- $a = b$; und
- es keine Sitzungen $\mathsf{lbl} \neq \mathsf{lbl}'$ mit $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$ und $\mathsf{lbl.status} = \mathsf{tested}$ und $\mathsf{lbl}'.\mathsf{status} = \mathsf{revealed}$ gibt—wenn der Angreifer also einen trivialen Angriff benutzt, bei dem eine von zwei gepartnerten Sitzungen getestet und die andere korrumpiert wird; und
- es keine Sitzung lbl mit $\mathsf{lbl.status} = \mathsf{tested}$ gibt, so dass $\mathsf{id}(\mathsf{lbl.partner}) \in \mathcal{C}$ gibt—also wenn die Sitzung mit einem Partner kommuniziert hat, der unter Kontrolle des Angreifers stand, und damit der Schlüssel trivialerweise nicht mehr geheim ist.

Ein Schlüsselgeneriergungsdienst ist sicher, wenn für alle (effizienten) Angreifer $\mathcal{A}$ die Erfolgswahrscheinlichkeit von $\mathcal{A}$ höchstens vernachläsigbar über der Ratewahrscheinlichkeit $\frac{1}{2}$ liegt. Sei $\mathbf{Adv}^{Secrecy}_{\mathsf{SGD},\mathcal{A}}$ die entsprechende Wahrscheinlichkeit über $\frac{1}{2}$ hinaus.

2.4 Eindeutigkeit

Die Sicherheitseigenschaft der Eindeutigkeit ist der "Gegenspieler" der Vertraulichkeit: Dort wurden triviale Angriffe auf Sitzungen und ihre Partnersitzung über die Kennzeichner sid ausgeschlossen. Üblicherweise müssen wir nun noch garantieren, dass die Einträge sid wirklich nur die zusammengehörigen Sitzungen kennzeichnen. Dazu definieren wir die Eindeutigkeit (*engl. match security*) der Sitzungskennzeichner.

Das erste Prinzip der Eindeutigkeit subsumiert die Korrektheit, nämlich dass für Sitzungen mit gleichem Kennzeicher sid auch die gleichen Geheimnisse secret abgeleitet werden. Wir lassen den Angreifer gegen die Eindeutigkeit also einerseits gewinnen, wenn es verschiedene Sitzungen $\mathsf{lbl}, \mathsf{lbl}'$ mit $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$, aber $\mathsf{lbl.secret} \neq \mathsf{lbl}'.\mathsf{secret}$ gibt. Mit anderen Worten: Der Angreifer gewinnt, wenn er dafür sorgt, dass zwei Sitzungen quasi zusammengehören, aber dennoch verschiedene Sitzungsgeheimnisse ableiten.

Das zweite Prinzip der Eindeutigkeit besagt, dass Kennzeichner sid quasi eindeutig sind, also dass es keine drei Sitzungen gibt, die den gleichen Kennzeichner erzeugen, obwohl nur die beiden zusammengehörigen Ausführungen zwischen Client und SGD jeweils den gleichen Wert sid erzielen sollten. Das SGD-Protokoll besitzt hinsichtlich der Eindeutigkeit aber eine Besonderheit gegenüber klassischen Schlüsselaustauschverfahren: Der Client kann einen einmal übertragenen abgeleiteten Schlüssel—den Sitzungsschlüssel einer Ausführung `KeyDerivation`— erneut anfordern. Damit er mit diesem erneut angeforderten Schlüssel den Akten- und Kontextschlüssel wieder erhalten kann, muss dabei wieder das gleiche Sitzungsgeheimnis wie zuvor erzeugt werden. Im Unterschied dazu nimmt man bei klassischen Schlüsselaustauschverfahren üblicherweise an, dass eine weitere Ausführung einen unabhängigen Schlüssel erzeugt.

Von dem Prinzip der paarweise eindeutigen Sitzungskennzeichner sid weichen wir hier wegen der Wiederherstellung der Schlüssel ab. Wir verwenden wie im Fall der sogenannten 0RTT-Variante von TLS 1.3 [FG17] das Prinzip der *wiederholbaren* (*engl. replayable*) Sitzungen. Dies besagt, dass es Unterprotokolle gibt, bei denen der gleiche Schlüssel und Sitzungskennzeicher mehrfach auftreten kann. Bei uns ist dies beispielsweise für den `KeyDerivation`-Schritt der Fall. Hier würden wir dementsprechend keine Eindeutigkeitsanforderung an sid stellen. Da wir aber in den anderen Unterprotokollen `GetPublicKey` und `GetAuthenticationToken` kein Test oder Reveal erlauben, benötigen wir dort keine weiteren Anforderungen an die Sitzungskennzeichern sid. Damit entfallen wegen der Wiederholbarkeit weitere Anforderungen an sid über die Korrektheit hinaus.

Wir sagen, dass $\mathcal{A}$ *Sitzungseindeutigkeit* erfolgreich bricht, wenn es am Ende des Angriffs verschiedene Sitzungen $\mathsf{lbl}, \mathsf{lbl}'$ mit $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$, aber $\mathsf{lbl.secret} \neq \mathsf{lbl}'.\mathsf{secret}$ gibt. Sei $\mathbf{Adv}^{Match}_{\mathsf{SGD},\mathcal{A}}$ die entsprechende Wahrscheinlichkeit. Ein Schlüsselgeneriergungsdienst ist (bis auf Wiederholungen) eindeutig, wenn für alle (effizienten) Angreifer diese Erfolgswahrscheinlichkeit von $\mathcal{A}$ höchstens vernachlässigbar ist.

3 Sicherheitsnachweis

Wir zeigen in diesem Abschnitt die Sicherheit des SGD-Protokolls. Zunächst spezifzieren wir noch einige Schritte des Protokolls gemäß des Sicherheitsmodells.

Wir gehen davon aus, dass es eine Menge von Clients mit Identitäten und langfristigen Signaturschlüsseln $(\mathsf{sk}_C, \mathsf{sk}_C)$ inklusive Zeritifkat cert_C gibt. Analog besitzt der SGD bzw. das SGD-HSM eine Identität und ein zertifiziertes Signatur-Schlüsselpaar $(\mathsf{sk}_S, \mathsf{pk}_S)$ für Zertifikat cert_S. Diese Schlüsselpaare werden gemäß der Signatur- und Zertifizierungsverfahren im SGD-Protokoll erzeugt. Wir nehmen zusätzlich an, dass die SGDe zu Beginn auch jeweils eine Liste $\{(\mathsf{ltkd}_{\mathtt{Bez}}, \mathtt{Bez})\}$ von zufälligen, langfristigen Ableitungsgeheimnisse bekommen, die gemäß einer Verteilung $\mathcal{V}_{\mathrm{lt}}$ mit mindestens 512 Bits Entropie erzeugt werden. Der Angreifer kennt alle einer Identität zugeordneteten Bezeichner `Bez`. Da wir im Modell nicht zwischen Clients und SGDen unterscheiden, geben wir eine solche Liste an alle Teilnehmer, auch wenn Clients diese Liste ignorieren. Die temporären Ableitungsschlüssel kd, die mindestens 256 Bits Entropie haben, werden gemäß einer Verteilung $\mathcal{V}$ gewählt. Formal bestehen damit die langfristigen Schlüssel eines Teilnehmers id aus dem geheimen Signaturschlüssel und den langfristigen Ableitungsschlüsseln $\{(\mathsf{ltkd}_{\mathtt{Bez}}, \mathtt{Bez})\}$ (für $\mathsf{sk}_{\mathsf{id}}$), und dem Verifizierschlüssel und den Bezeichnern $\{\mathtt{Bez}\}$ (für $\mathsf{pk}_{\mathsf{id}}$).

Um die beiden Sitzungen der SGDe für beide Schlüsselableitung zusammenzuführen, kann der Angreifer einen Schlüssel pk_i mit Position $i \in \{1, 2\}$ als zusätzliche Eingabe input bei Initialisierung einer Sitzung übergeben. Die Position gibt an, wo der öffentliche Schlüssel platziert werden soll. Der öffentliche Schlüssel pk_S des anderen SGD wird dann an Position $3 - i$ gesetzt.

Wenn der Angreifer einen Wert input mit einem Schlüssel pk_i als zusätzliche Eingabe für eine Client-Session der Art `GetAuthenticationToken` oder `KeyDerivation` gibt, dann muss einer dieser beiden Schlüssel genau der Schlüssel (an der richtigen Position) sein, den sich der Client vorher per `GetPublicKey` in $\mathsf{lbl.previous}$ hat geben lassen. Damit implementieren wir, dass sich der Client den Schlüssel merkt und in den folgenden Ausführungen korrekt verwendet.

3.1 Kryptographische Annahmen

In diesem Abschnitt diskutieren wir die kryptographischen Annahmen für die Sicherheit

des Protokolls. Weitere Details zu den Sicherheitseigenschaften der abstrakten Primitive findet man in [KL20] oder [MF21]. Details zu den im Protokoll verwendeten konkreten Verfahren findet man in [gem20b] und in [gem20a]. Die gewählten Verfahren und Parameter sind im Einklang mit den aktuellen Empfehlungen, z.B. denen des Bundesamts für Sicherheit in der Informationstechnik [BfSidI21].

Hashfunktion. Im Protokoll wird eine Hashfunktion H eingesetzt. Wir nehmen an, dass diese Funktion kollisionsresistent ist. Bezeichne für einen Algorithmus $\mathcal{C}$ der Vorteil $\mathbf{Adv}^{\mathrm{CR}}_{\mathsf{H},\mathcal{C}}$ die Wahrscheinlichkeit, dass $\mathcal{C}$ Werte $x \neq x'$ mit $\mathsf{H}(x) = \mathsf{H}(x')$ ausgibt. Um die bekannten Probleme mit kodierten Hash-Kollisionen zu umgehen, nehmen wir an, dass $\mathcal{C}$ konstruktiv bestimmt wird [MF21]. In der Spezifikation wird SHA-256 als Hashfunktion H angenommen. Da SHA-256 als sicher gilt und bis heute keine Kollisionen für SHA-256 bekannt sind, kann man davon ausgehen, dass $\mathbf{Adv}^{\mathrm{CR}}_{\mathsf{H},\mathcal{C}}$ vernachlässigbar für jeden konstruierten Algorithmus $\mathcal{C}$ ist.

Verschlüsselungssystem. Für das Verschlüsselungsverfahren $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ setzen wir die IND-CCA-Sicherheit gegen Chosen-Ciphertext-Angriffe voraus. Bei dieser Sicherheitseigenschaft darf sich der Angreifer $\mathcal{A}$ wiederholt eine von zwei Nachrichten m_0, m_1 (jeweils gleicher Länge) unter dem bekannten öffentlichen Schlüssel pk verschlüsseln lassen, wobei die Wahl m_b gemäß eines geheimen Bits b erfolgt. Der Angreifer erhält den jeweiligen Ciphertext und darf sich auch andere Ciphertext von $\mathsf{Dec}(\mathsf{sk}, \cdot)$ entschlüsseln lassen. Er gibt am Ende ein Bit a aus und gewinnt, wenn $a = b$ ist. Sei $\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{E},\mathcal{A}} := \left|\Pr[a = b] - \frac{1}{2}\right|$ der Vorteil des Angreifers $\mathcal{A}$ das Bit b über die Ratewahrscheinlichkeit von $\frac{1}{2}$ zu bestimmen.

Laut Spezifikation wird das Verfahren "Elliptic Curve Integrated Encryption Scheme" (ECIES) von Abdalla et al. [ABR99] zur Verschlüsselung eingesetzt. Das Verfahren gilt als IND-CCA-sicher (in der Spezifikation auch CCA2-Sicherheit genannt). Zwar beruht es auf dem Diffie-Hellman-Problem (über elliptischen Kurven) und gilt damit im Augenblick als verlässlich, aber langfristig könnte es mittels fortgeschrittener Quantencomputer gebrochen werden. In diesem Fall bleibt unsere Sicherheitsanalyse gültig, so dass dann hier einfach ein quanten-sicheres IND-CCA-Verfahren eingesetzt werden kann. Das ECIES-Verfahren wird als Hybridverfahren mit dem AEAD-Verfahren Galois/Counter Mode (GCM) für AES mit 256-Bit-Schlüsseln und zufälligen 96-Bit Initialisierungsvektoren verwendet [MV04]. Auch dieses Verfahren gilt als sicher und auch in Kombination mit ECIES als Hybridsystem.

Signaturverfahren. Ein geeignetes Signaturverfahren $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ erfüllt die EUF-CMA-Sicherheitseigenschaft. Bei einem solchen Angriff darf sich der Angreifer $\mathcal{A}$ zunächst beliebige Nachrichten m seiner Wahl per $\mathsf{Sig}(\mathsf{sk}, \cdot)$ unterschreiben lassen, nachdem er als Eingabe den zugehörigen öffentlichen Schlüssel pk erhalten hat. Dann soll er schließlich eine noch

nicht unterschriebene Nachricht m^* und eine Signatur s^* ausgeben, die vom Verifikationsalgorithmus $\mathsf{Vf}(\mathsf{pk}, m^*, s^*)$ akzeptiert wird. Sei $\mathbf{Adv}^{\text{EUF-CMA}}_{\mathcal{S},\mathcal{A}} := \Pr[\mathsf{Vf}(\mathsf{pk}, m^*, s^*) = 1 \wedge m^* \text{ neu}]$ die Wahrscheinlichkeit, dass $\mathcal{A}$ erfolgreich fälscht.

Laut Spezifikation wird das ECDSA-Verfahren zum Signieren mit Bit-Niveau 256 verwendet. Dieses Verfahren gilt nach heutigem Stand als sicher, auch wenn es wie das Verschlüsselungssystem langfristig nicht sicher gegen umfängliche Quantenrechner ist. Allerdings beschreibt die Spezifikaton selbst schon die Möglichkeit, auf stärkere Verfahren zu wechseln. Unsere Analyse bleibt davon ebenfalls unberührt, solange das Verfahren EUF-CMA ist.

Zertifizierung. (Nicht-hierarchische) Zertifizierung kann man vollständig analog zur Sicherheit von Signaturverfahren definieren. Auch in diesem Fall soll der Angreifer ein neues gültiges Zertifikat cert fälschen, nachdem er sich vorher beliebeige andere Zertifikate hat erstellen lassen. Für ein Zertifikatssystem $\mathcal{Z}$ definieren wir daher den Vorteil $\mathbf{Adv}^{\text{EUF-CERT}}_{\mathcal{Z},\mathcal{A}}$ entsprechend.

Zertifikate werden in der Veröffentlichung [gem20a] näher ausgeführt. Die dort angebenen Verfahren und Parameter gelten nach aktuellem Stand als sicher und garantieren die Sicherheit im obigen Sinne. Wir blenden hier zur Vereinfachung Aspekte wie das Revozieren und Angriffe gegen übergeordnete Zertifizierungsinstanzen aus.

Schlüsselableitung. Wir gehen davon aus, dass sich die Schlüsselableitungsfunktion KDF wie eine Pseudozufallsfunktion (PRF) verhält. Für einen (effizienten) Algorithmus $\mathcal{D}$ bezeichne

$$\mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{V},\mathcal{D}} := \left|\Pr\left[\mathcal{D}^{\mathsf{KDF}(\mathsf{KD},\cdot)} = 1\right] - \Pr\left[\mathcal{D}^{R}(\cdot) = 1\right]\right|$$

den Vorteil des Algorithmus $\mathcal{D}$, die Ableitungsfunktion $\mathsf{KDF}(\mathsf{KD}, \cdot)$ für einen zufälligen Schlüssel KD (gemäß Verteilung $\mathcal{V}$) von einer echt zufälligen Funktion R mit gleicher Ein- und Ausgabe zu unterscheiden, wobei Algorithmus $\mathcal{D}$ jeweils beliebige Eingaben für die Funktion adaptiv wählen darf. Für eine sichere Pseudozufallsfunktion muss der Vorteil vernachlässigbar klein sein.

In der Spezifikation wird HKDF (RFC 5869, [KE10]) mit SHA-256 als Ableitungsfunktion gewählt. Die Eingaben sind der Ableitungsschlüssel kd des i-ten SGD als "input key material" und ein Info-Wert (unterschiedlich in `GetAuthenticationToken` und `KeyDerivation`). Damit ergibt sich aus der Länge der abgeleiteten Schlüssel von 256 Bits, dass der Längenparameter L für HKDF als 32 Octets gewählt wird. Gemäß der HKDF-Spezifikation bedeutet dies, dass für SHA-256 die Ausgabe der Ableitungsfunktion als

$$\text{HMAC}(PRF, \texttt{info}|\texttt{0x01})$$

für $PRF \leftarrow \text{HMAC}(0 \ldots 0, \mathsf{kd})$ berechnet wird. Es ist davon auszugehen, dass sich die KDF-Funktion wie eine Pseudozufallsfunktion verhält [Kra10].

3.2 Sicherheitsbeweis: Eindeutigkeit

Wir beginnen mit der einfacheren Eindeutigkeitseigenschaft. Gemäß der Diskussion in Abschnitt 2.4 ist zu zeigen, dass es keine Sitzungen $\mathsf{lbl} \neq \mathsf{lbl}'$ gibt, so dass $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$, aber $\mathsf{lbl.secret} \neq \mathsf{lbl}'.\mathsf{secret}$. Dazu merken wir zunächst an, dass Gleichheit auf den Sitzungskennzeichnern bedeuten muss, dass beide Sitzungen das gleiche Unterprotokoll betreffen, da die Kennzeichner das entsprechende Unterprotokoll durch `PK`, `AT` und `KD` beinhalten. Im ersten Fall, wenn $\mathsf{lbl.type} = \mathsf{lbl}'.\mathsf{type} = \texttt{GetPublicKey}$ ist die Behauptung trivial, da beider Sitzungen den Wert $\mathsf{secret} = \bot$ und damit immer identisch setzen.

Für die beiden anderen Unterprotokolle argumentieren wir zunächst, dass bei jedem UpdTempKey-Aufruf mit hoher Wahrscheinlichkeit auf Seiten des SGD ein neuer öffentlicher Schlüssel pk_S erzeugt wird. Der Schlüssel wird zufällig in einer ellitpischen Kurve mit mindestens $n = 256$ Bits erzeugt, so dass bei maximal T Schlüsselerneuerung nach dem Geburtstagsparadoxon eine Wahrscheinlichkeit von höchstens $T^2 \cdot 2^{-256}$ besteht, eine Kollision auf den öffentlichen Schlüsseln zu erzeugen. Dabei nehmen wir an, dass die elliptsche Kurve 2^{256} Punkte hat. Selbst bei insgesamt einer Millionen SGDe, die jeweils alle minütlich ihre temporären Schlüssel aktualisieren, würden so in 100 Jahren maximal $T \leq 2^{47}$ öffentliche Schlüssel im Gesamtsystem erzeugt. Dann wäre die Wahrscheinlichkeit für eine Kollision unter allen Schlüsseln nur 2^{-162}—und damit sogar noch deutlich kleiner als die üblichen kryptographischen Schranken von 2^{-80} für vernachlässigbare Ereignisse.

Mit dem Aspekt der Eindeutigkeit der öffentlichen Schlüssel (der SGDe) können wir auch argumentieren, dass gleiche Sitzungskennzeichner sid den gleichen Schlüssel secret garantieren. Im Fall des `GetAuthenticationToken`-Protokolls besteht der Sitzungskennzeichner aus $\mathsf{sid} = (\texttt{AT}, \mathsf{pk}_S, \mathsf{cert}_S, A)$. Da wir annehmen können, dass der Schlüssel pk_S eindeutig ist, ist auch der temporäre Ableitungsschlüssel kd dadurch eindeutig bestimmt. Folglich ist auch das deterministisch berechnete Authentisierungstoken $\mathsf{AT} \leftarrow \mathsf{KDF}(\mathsf{kd}, A)$ bei gleichem Sitzungskennzeicher identisch.

Die Eindeutigkeit folgt analog für das `KeyDerivation`-Protokoll. Bei gleichem Kennzeichner $\mathsf{sid} = (\texttt{KD}, \mathsf{pk}_S, \mathsf{cert}_S, \texttt{rx}^{\text{ext}})$ zweier Sitzungen $\mathsf{lbl}, \mathsf{lbl}'$ muss für das eindeutige Zertifikat cert_S des SGDs auch die dem Zertifikat zugeordnete Menge $\{(\mathsf{ltkd}_{\texttt{Bez}}, \texttt{Bez})\}$ der langfristigen Ableitungsschlüssel übereinstimmen. Da ferner der Bezeichner `Bez` im Ableitungsvektor $\texttt{rx}^{\text{ext}}$ enthalten ist, folgt auch hier, dass $\mathsf{k} \leftarrow \mathsf{KDF}(\mathsf{ltkd}_{\texttt{Bez}}, A)$ in beiden Ausführungen identisch ist.

Insgesamt ergibt sich daher, dass das Protokoll die Eindeutigkeit mit überwältigender

Wahrscheinlichkeit garantiert:

Theorem 1 (Eindeutigkeit des SGD-Protokolls)**.** *Für jeden Angreifer $\mathcal{A}$, der während eines Angriffs maximal T temporäre Schlüssel per* UpdTempKey *erzeugen lässt, gilt*

$$\boldsymbol{Adv}^{Match}_{\mathsf{SGD},\mathcal{A}} \leq T^2 \cdot 2^{-256}.$$

Wir betonen, dass diese Schranke $T^2 \cdot 2^{-256}$ nicht besagt, dass eine gutartige Protokollausführung zwischen ehrlichen Teilnehmern mit dieser Wahrscheinlichkeit erfolglos bleibt, sondern dass kein Angreifer die Ausführung so stören kann, dass beide Teilnehmer mit verschiedenen Schlüsseln akzeptieren.

3.3 Sicherheitsbeweis: Vertraulichkeit

Es bleibt, die Vertraulichkeit der versichertenspezifischen Schlüssel zu zeigen. Dazu werden wir die kryptographischen Annahmen der Primitive verwenden. Der Beweis zeigt, dass die ehrlich abgeleiteten Schlüssel nicht von zufälligen Werten zu unterscheiden sind.

Theorem 2 (Vertraulichkeit des SGD-Protokolls)**.** *Sei S die maximale Anzahl von Sitzungen, die der Angreifer beginnt, T die maximale per* UpdTempKey *erzeugten temporären Schlüssel, U die maximale Anzahl aller Teilnehmer im System, und B die Gesamtzahl aller langfristigen Ableitungsschlüssel im System. Dann gibt es für jeden Angreifer $\mathcal{A}$ auf die Vertraulichkeit des SGD-Protokolls Angreifer $\mathcal{B}_{1a}, \mathcal{B}_{1b}, \mathcal{B}_{2a}, \mathcal{B}_{2b}, \mathcal{B}_{3-5}$ mit*

$$\begin{aligned}\boldsymbol{Adv}^{Secrecy}_{\mathsf{SGD},\mathcal{A}} \quad \leq \quad & U \cdot \boldsymbol{Adv}^{EUF\text{-}CMA}_{\mathcal{S},\mathcal{B}_{1a}} + U \cdot \boldsymbol{Adv}^{EUF\text{-}CERT}_{\mathcal{Z},\mathcal{B}_{1b}} + T \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{V},\mathcal{B}_{2a}} \\ & + B \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{V}_{lt},\mathcal{B}_{2b}} + 6T \cdot \boldsymbol{Adv}^{IND\text{-}CCA}_{\mathcal{E},\mathcal{B}_{3-5}} + 2S \cdot 2^{-256}.\end{aligned}$$

Dabei haben die Algorithmen $\mathcal{B}_{1a}, \mathcal{B}_{1b}, \mathcal{B}_{2a}, \mathcal{B}_{2b}, \mathcal{B}_{3-5}$ die gleiche Laufzeit wie $\mathcal{A}$, zuzüglich der Laufzeit zur Ausführung des Spiels.

Zur Interpretation: Unter der Annahme, dass die kryptographischen Bausteine das entsprechende Sicherheitsniveau erreichen, sind die Terme auf der rechten Seite (trotz der beschränkten Faktoren S, T, U und B) vernachlässigbar klein. Dies betrifft die Fälschungssicherheit des Signaturverfahrens und der Zertifizierung, die Pseudozufälligkeit der Ableitungsfunktion und die Sicherheit des Verschlüsselungssystems. Damit folgt aber auch, dass ein Angreifer echte, ehrlich abgeleitete Schlüssel nicht von unabhängigen, zufälligen Werten

unterscheiden kann. Nimmt man realistischerweise an, dass die kryptographischen Verfahren alle ein Sicherheitsniveau von 2^{-128} erreichen, und setzt eine überdimensionale Anzahl von einer Milliarde Clients an, die täglich eine Anfrage über 100 Jahre stellen, sowie eine Millionen SGDe, die die temporären Schlüssel minütlich und die langfristen Ableitungsschlüssel täglich über diesen Zeitraum von 100 Jahren wechseln, dann hätte man

$$S \leq 2^{47}, \quad T \leq 2^{48}, \quad U \leq 2^{31}, \quad B \leq 2^{36}.$$

In diesem Fall würde die Schranke des Theorems besagen, dass der Angreifer maximal mit Wahrscheinlichkeit 2^{-76} über der Ratewahrscheinlichkeit liegen kann.

Beweis. Der Beweis erfolgt per sogenannter Game-Hopping-Technik [BR06], bei der in einer Sequenz von leicht modifizierten Spielen das ursprüngliche Angriffsspiel in ein Spiel überführt wird, bei dem der Angreifer keinen Vorteil mehr über der Ratewahrscheinlichkeit hat. Jeder Übergang zum nächsten Spiel schließt eine Angriffsmöglichkeit aus. Diese "Wegnahme" wird in der Regel mittels der kryptographischen Annahmen über die Bausteine kompensiert.

Wir nennen im folgenden eine Sitzung lbl *ehrlich* (zu einem Zeitpunkt), wenn die Identität $\mathsf{lbl.id}$ des Teilnehmers nicht korrumpiert wurde, also $\mathsf{id} \notin \mathcal{C}$ ist, und wenn der Status $\mathsf{lbl.status} = \mathsf{fresh}$ oder tested ist. Diese ehrlichen Sitzungen sind die, bei denen das Geheimnis geschützt bleiben soll. Wir nennen einen Teilnehmer id ehrlich, wenn er nicht korrumpiert wurde, also $\mathsf{id} \notin \mathcal{C}$ gilt.

Sei $\mathcal{A}$ ein Angreifer gegen die Schlüsselgenerierung im Sinne des Modells in Abschnitt 2. Wir schreiben Game_i für den Erfolgsfall des Angreifers $\mathcal{A}$ in dem jeweiligen Spiel, wobei Game_0 das ursprüngliche Sicherheitsspiel sei, also $\Pr[\mathsf{Game}_0] = \mathbf{Adv}^{\mathrm{SGD}}_{\mathsf{SGD},\mathcal{A}} + \frac{1}{2}$.

Spiel Game_0. Entspricht dem ursprünglichen Angriff.

Spiel Game_1. Wie Spiel Game_0, allerdings stoppen wir sofort ohne Erfolg für den Angreifer, wenn

- der Angreifer in `GetPublicKey` einen Schlüssel pk_S an einen ehrlicen Client per SEND schickt, der vom Client akzeptiert wird, während der durch das Zertifikat cert_S definierte SGD mit Identität id nicht korrumpiert wurde, also $\mathsf{id} \notin \mathcal{C}$ ist, und der Schlüssel pk_S nicht vom SGD bereits signiert wurde; oder
- der Angreifer in `GetAuthenticationToken` einen Schlüssel epk_C an einen ehrlichen SGD per SEND schickt, der vom SGD akzeptiert wird, während der durch das Zertifikat cert_C definierte Client mit Identität id nicht korrumpiert wurde, also $\mathsf{id} \notin \mathcal{C}$ ist, und epk_C noch nicht vom Client signiert wurde.

Ein solcher Fall kann nur eintreten, wenn der Angreifer ein Zertifikat unter der Identität id fälscht, oder aber wenn der Angreifer eine Signatur unter dem richtigen Signaturschlüssel des Teilnehmers id fälscht (da der Schlüssel noch nicht signiert wurde). Dies ergibt unmittelbar per Reduktion Angreifer $\mathcal{B}_{1a}$ bzw. $\mathcal{B}_{1b}$ gegen die Sicherheit des Signaturverfahrens oder die des Zertifizierungsverfahrens. Wenn maximal U Teilnehmer im Angriff betrachtet werden, folgt:

$$\Pr[\mathsf{Game}_0] \leq \Pr[\mathsf{Game}_1] + U \cdot \mathbf{Adv}^{\text{EUF-CMA}}_{\mathcal{S},\mathcal{B}_{1a}} + U \cdot \mathbf{Adv}^{\text{EUF-CERT}}_{\mathcal{Z},\mathcal{B}_{1b}}.$$

Die Laufzeiten von $\mathcal{B}_{1a}$ und $\mathcal{B}_{1b}$ entsprechen der von $\mathcal{A}$, zuzüglich der Protokollschritte für den Angriff von $\mathcal{A}$.

Interpretation. Gemäß Spiel Game_1 kann der Angreifer keinen selbst gewählten, neuen temporären Schlüssel im Namen eines ehrliche Teilnehmers an den Partner schicken, so dass der Partner akzeptiert. Der Angreifer kann aber immer noch bereits signierte Schlüssel erneut senden, oder aber selbst gewählte Schlüssel unter korrumpierten Teilnehmern aus $\mathcal{C}$ schicken.

Spiel Game_2. Wie Spiel Game_1, aber ersetze alle Berechungen $\mathsf{KDF}(\cdot,\cdot)$ bei ehrlichen SGDen jeweils durch eine zufällige Funktion (bzw. effizient, per sogenanntem *Lazy Sampling* [BR06]), sowohl für temporäre als auch für langfristige Ableitungsschlüssel kd bzw. ltkd. Dies bedeutet, dass für jeden Schlüssel jeweils eine zufällige Funktion verwendet wird; für zufälligerweise identische Schlüssel aber auch die gleiche Funktion.

Dies ist wegen der Pseudozufälligkeit der KDF-Funktion möglich, und da bei ehrlichen SGDen der Ableitungsschlüssel kd zum vertraulichen temporären Schlüssel gehört bzw. die langfristigen Ableitungsschlüssel $\mathsf{ltkd}_{\mathsf{Bez}}$ stets geheim bleiben. Da es maximal T solcher temporärer Schlüssel für Verteilung $\mathcal{V}$ und maximal B solcher langfristigen Schlüssel für Verteilung $\mathcal{V}_{\text{lt}}$ gibt, folgt per einfachem Hybridargument und Reduktion, dass es Angreifer $\mathcal{B}_{2a}$ bzw. $\mathcal{B}_{2b}$ gegen KDF gibt, so dass:

$$\Pr[\mathsf{Game}_1] \leq \Pr[\mathsf{Game}_2] + T \cdot \mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{V},\mathcal{B}_{2a}} + B \cdot \mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{V}_{\text{lt}},\mathcal{B}_{2b}}.$$

Dabei haben $\mathcal{B}_{2a}, \mathcal{B}_{2b}$ jeweils die Laufzeit des gesamten Angriffs von $\mathcal{A}$.

Interpretation. Gemäß dieses Spiels erzeugen die ehrlichen SGDe nun alle Authentisierungstokens und abgeleitete Schlüssel echt zufällig statt pseudozufällig (aber immer noch konsistent, also identisch bei gleicher Eingabe). Dies bedeutet jedoch noch nicht, dass der Angreifer diese Werte nicht angreifen kann.

Spiel Game_3. Im nächsten Schritt ersetzen wir jede Verschlüsselung c^{at}_C und c^{kd}_C eines Clients an einen ehrlichen SGD mit Schlüssel pk_S durch eine Verschlüsselung einer gleichlangen

Nachricht, die nur aus 0-Bits besteht. Nach Erhalt eines solchen Ciphertexts auf SGD-Seite verwenden wir allerdings die ursprüngliche Nachricht $\texttt{CH}|\mathsf{ch}|H$ (für c_C^{at}) bzw. $\texttt{KD}|\mathsf{AT}|\mathsf{reqID}|\texttt{rx}$ (für c_C^{kd}).

Die Erfolgswahrscheinlichkeit des Angreifers kann sich wegen der IND-CCA-Sicherheit des ECIES-Verfahrens nicht wesentlich ändern. Dies folgt per Reduktion auf diesen Sicherheitsbegriff, indem man einen Angreifer $\mathcal{B}_3$ konstruiert, der gegen einen der (vertraulichen) Schlüssel pk_S spielt, und als Challenge-Paare die richtigen Nachrichten und 0-Bits übergibt, und der mit Hilfe des Entschlüsselungsorakels alle anderen Ciphertexte entschlüsseln kann. Schickt der Angreifer gegen das SGD-Protokoll dabei einen Challenge-Ciphertext erneut an ein ehrlichen SGD mit dem gleichen Schlüssel im gleichen Unterprotokoll, dann kennen wir den Inhalt der ursprünglichen Nachricht und können diesen verwenden. Sollte so ein Challenge-Ciphertext in dem jeweils anderen Unterprotokoll gesendet werden, wissen wir wegen des Präfixes `CH` oder `KD` in der Nachricht, dass er nicht gültig sein kann und daher unmittelbar verworfen werden kann. Die Reduktion nutzt dabei aus, das gemäß Spiel Game_1 die Schlüssel ehrlicher SGDe authentisch sind und vom SGD selbst erzeugt wurden.

Insgesamt können wir die maximal T vielen Schlüssel der SGD durch ein Hybridargument sukzessive ersetzen, und erhalten

$$\Pr[\mathsf{Game}_2] \leq \Pr[\mathsf{Game}_3] + 2T \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{E},\mathcal{B}_3}$$

für einen Angreifer $\mathcal{B}_3$. Der Faktor 2 in $2T$ kommt durch den Übergang des IND-CCA-Spiels für ein zufälliges Challenge-Bit zu einem festen Bit [KL20].

Interpretation. Die Ciphertexte an ehrliche SGDe geben selbst keine Informationen preis, da die Verschlüsselung unter sicheren Schlüsseln erfolgt. Beispielsweise ist der ch-Wert in `GetAuthenticationToken` in c_C^{at} geschützt. Allerdings ist damit noch nicht garantiert, dass andere Protokollschritte (z.B. die Antwort des SGD) diese Werte auch geheimhalten.

Spiel Game_4. Im nächsten Schritt ersetzen wir ebenfalls jede Verschlüsselung c_S^{at} eines ehrlichen SGDs an einen ehrlichen Client in einer Sitzung `GetAuthenticationToken` mit Schlüssel epk_C durch eine Verschlüsselung einer gleichlangen Nachricht, die nur aus 0-Bits besteht. Nach Erhalt eines solchen Ciphertexts auf Client-Seite verwenden wir allerdings die ursprüngliche Nachricht $\texttt{RSP}|\mathsf{ch}|\mathsf{H}(A)|\mathsf{AT}$.

Es folgt analog zum Fall der SGD-Verschlüsselung, dass man per Reduktion auf die IND-CCA-Sicherheit zeigen kann, dass sich die Erfolgswahrscheinlichkeit nicht wesentlich ändert. Auch hier nutzen wir, dass gemäß Spiel Game_1 die Schlüssel epk_C für ehrliche Clients authentisch sind.

$$\Pr[\mathsf{Game}_3] \leq \Pr[\mathsf{Game}_4] + 2T \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{E},\mathcal{B}_4}.$$

Interpretation. Es folgt wie im vorigen Spiel, dass die Ciphertexte c_S^{at} zur Übermittlung der Authentisierungstoken an Clients in `GetAuthenticationToken` selbst nichts preisgeben. Wir können hier allerdings noch nicht schließen, dass dies auch für die abgeleiteten Schlüssel in `KeyDerivation` gilt, da im `KeyDerivation`-Schritt die Zertifikate nicht mehr geprüft werden. Dazu werden wir über die Vertraulichkeit der Authentisierungstoken argumentieren: Nur ehrliche Clients können eine Anfrage mit gültigen Tokens stellen. Zunächst beschränken wir uns aber auf einen Sonderfall, nämlich, dass der Ciphertext c_S^{kd} für einen authentischen Schlüssel gebildet wird.

Spiel Game_5. Wie Spiel Game_4, allerdings ersetzen wir ebenfalls jede Verschlüsselung c_S^{kd} eines ehrlichen SGD an einen ehrlichen Client in einer Sitzung `GetAuthenticationToken` *mit dem korrekten Schlüssel* epk_C durch eine Verschlüsselung einer gleichlangen Nachricht, die nur aus 0-Bits besteht. Nach Erhalt eines solchen Ciphertexts auf Client-Seite verwenden wir allerdings die ursprüngliche Nachricht $\texttt{OKKD}|\mathsf{AT}|\mathsf{reqID}|\mathsf{k}|\mathtt{rx}^{\text{ext}}$.

Es folgt analog zum Fall der SGD-Verschlüsselung, dass man per Reduktion auf die IND-CCA-Sicherheit zeigen kann, dass sich die Erfolgswahrscheinlichkeit nicht wesentlich ändert. Hier nutzen wir allerdings aus, dass der Schlüssel epk_C als authentisch angenommen wird.

$$\Pr[\mathsf{Game}_4] \leq \Pr[\mathsf{Game}_5] + 2T \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{E},\mathcal{B}_5}.$$

Da die drei Angreifer $\mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5$ gegen das gleiche Verschlüsselungssystem agieren, fassen wir sie im Theorem zu einem Angreifer $\mathcal{B}_{3-5}$ zusammen.

Interpretation. Damit wissen wir nun, dass für einen ehrlichen Schlüssel epk_C in der Nachricht von einem ehrlichen SGD keine Information über AT, reqID und k an den Angreifer geht. Wir müssen uns nun noch überlegen, dass der Angreifer für einen gefälschten Schlüssel $\widetilde{\mathsf{epk}_C}$ in `KeyDerivation` keine Informationen über den Schlüssel k erhält. Dazu verwenden wir zunächst, dass der Angreifer das Authentisierungstoken nicht vorhersagen kann.

Spiel Game_6. Wie Spiel Game_5, aber der Angreifer verliert, wenn er unter dem Zertifikat cert_C eines ehrlichen Clients an ein ehrlichen SGD in `KeyDerivation` eine Protokollnachricht mit $\widetilde{\mathsf{epk}_C^{\text{ext}}}, \widetilde{c_C^{\text{kd}}}$ schickt, die SGD zusammen mit dem Zertifikat und der Signatur akzeptiert, aber für die der Client ein solches Paar $\widetilde{\mathsf{epk}_C^{\text{ext}}}, \widetilde{c_C^{\text{kd}}}$ in `KeyDerivation` in einer entsprechenden Nachricht noch nicht ausgegeben hat. (Wenn der Client $\widetilde{c_C^{\text{kd}}}$ in einer `GetAuthenticationToken`-Sitzung ausgegeben hat, kann wegen des Nachrichtenpräfix dieser Ciphertext in `KeyDerivation` nicht gültig sein.)

Wir unterscheiden zwei Fälle. Wenn der Angreifer ein neuen Wert $\widetilde{\mathsf{epk}_C^{\text{ext}}}$ wählt, den der Client noch nicht angefordert hat, dann ist gemäß Spiel Game_2 das zugehörige Authentisierungstoken zufällig und unabhängig. Dazu beachte man, dass die Eingabe der Ableitungsfunktion $\widetilde{A} = \widetilde{\mathsf{epk}_C^{\text{ext}}}|\mathsf{cert}_C$ ist. Dies bedeutet, dass alle anderen Abfragen für das Zertifikat cert_C zu diesem Zeitpunkt für andere Eingaben gewesen sein müssen, und für andere Zertifikate die Authentisierungstoken unabhängig gewählt werden. Wenn der Angreifer aber einen Wert $\mathsf{epk}_C^{\text{ext}}$ wählt, den der Client zuvor gewählt hatte, dann ist der Schlüssel des Clients vertraulich. Dann muss der Ciphertext $\widetilde{c_C^{\text{kd}}}$ des Angreifers aber neu sein. In diesem Fall ersetzen wir ihn aber nicht automatisch auf der SGD-Seite durch den richtigen Wert, sondern entschlüsseln ihn und der SGD vergleicht die Authentisierungstokens. Zu diesem Zeitpunkt aber hat der Angreifer keine Informationen über den richtigen Wert AT, da alle Ciphertexte, inklusive des eventuell schon erzeugten Ciphertexts c_S^{kd}, keine Informationen über AT enthalten, sondern nur 0-Bits.

In beiden Fällen kann der Angreifer den zufälligen Wert $\mathsf{KDF}(\mathsf{kd}, \widetilde{A})$ nur mit Wahrscheinlichkeit 2^{-256} raten. Damit können wir die Erfolgswahrscheinlichkeit im gesamten Spiel mit maximal S Sitzungen wie folgt beschränken:

$$\Pr[\mathsf{Game}_5] \leq \Pr[\mathsf{Game}_6] + S \cdot 2^{-256}.$$

Beachte, dass dies unabhängig von möglichen Reveal-Anfragen ist, da diese nur die abgeleiteten Schlüssel presigeben, aber nicht die Authentisierungstokens.

Intepretation. Zu diesem Zeitpunkt wissen wir also, dass der Angreifer sich keine Informationen über den abgeleiteten Schlüssel eines ehrlichen Clients vom SGD abgreifen kann. Wir wissen ferner, dass der Schlüssel sicher zwischen ehrlichen Teilnehmern übertragen wird. Damit ist die letzte Angriffsmöglichkeit erfasst, nämlich dass der Angreifer dem ehrlichen Client einen falschen Schlüssel unterschieben könnte—den er dann trivial von einem zufälligen Wert unterscheiden könnte. Allerdings ist der Angreifer daran gebunden, dass er sich als ehrlicher SGD mit Zertifikat cert_S ausgibt; sonst kann er diese Sitzung des Clients nicht erfolgreich testen, da der Partner korrumpiert wäre. Wir zeigen nun, dass der Angreifer dazu die zufällige reqID des Clients erraten müsste.

Spiel Game_7. Wie Spiel Game_6, aber stoppe und erkläre den Verlust des Angreifers, sofern er an einen ehrlichen Client in einer Ausüfhrung `KeyDerivation` dazu bringt, für einen Ciphertext $\widetilde{c_S^{\text{kd}}}$ und dem Zertifikat cert_S eines ehrlichen SGD zu akzpetieren, obwohl dieser SGD diesen Ciphertext zuvor nicht erzeugt hat.

Notwendig dazu ist, dass in dem Ciphertext $\widetilde{c_S^{\text{kd}}}$ der korrekte Wert reqID des Clients enthalten ist. Zu diesem Zeitpunkt hat der Angreifer allerdings keine Informationen über

reqID. Dies gilt selbst dann, wenn der Angreifer den Ciphertext c_C^{kd} an den ehrlichen SGD zunächst abliefert (per SEND) und der Angreifer die Antwort inklusive c_S^{kd} sieht. Gemäß der vorigen Spiele enthalten diese Ciphertext zwischen ehrlichen Teilnehmern und deren Schlüssel nur 0-Bits. Da der Ciphertext $\widetilde{c_S^{\text{kd}}}$ nach Voraussetzung verschieden ist von c_S^{kd} würden wir auch im Spiel nicht einfach die Antwort ersetzen, sondern entschlüsseln und vergleichen. Da reqID ein zufälliger, unbekannter Wert ist, wäre der Angreifer aber nur mit Wahrscheinlichkeit 2^{-256} in einer Sitzung erfolgreich, reqID richtig zu erraten. Damit können wir die Erfolgswahrscheinlichkeit im gesamten Spiel mit maximal S Sitzungen wie folgt beschränken:

$$\Pr[\mathsf{Game}_6] \leq \Pr[\mathsf{Game}_7] + S \cdot 2^{-256}.$$

Auch hier ist diese Betrachtung unabhängig von REVEAL-Anfragen gegen secret.

Fazit. Zu diesem Zeitpunkt haben wir gezeigt, dass der Angreifer (evtl. bis auf REVEAL-Anfragen) keine Informationen über den abgeleiteten Schlüssel eines ehrlichen Clients erhält. Für einen erfolgreichen Angriff auf eine Sitzung mit $\mathsf{lbl.sid} = (\texttt{KD}, \mathsf{cert}_S, \texttt{rx}^{\text{ext}})$ darf der Angreifer aber nur REVEAL-Anfragen für nicht gepartnerte Sitzungen stellen. Für eine solche Sitzung ist aber entweder das Zertifikat cert_S verschieden, und damit die langfristigen Ableitungsschlüssel $\mathsf{ltkd}_{\texttt{Bez}}$ unabhängig gewählt, oder aber die Eingabe $\texttt{rx}^{\text{ext}}$ neu. Dadurch ist der abgeleitete Schlüssel zufällig und unabhängig von solchen Sitzungen mit anderem Sitzungskennzeichner. Weitere TEST-Anfragen für gleiche Kennzeichner sid werden vom Sicherheitsmodell automatisch mit $\perp$ beantwortet.

Der letzte Schritt ist nun, festzuhalten, dass der Angreifer keinen korrupten Client $\widetilde{\mathsf{id}}$ benutzen kann, um die gleiche Eingabe $\texttt{rx}^{\text{ext}}$ wie ein ehrlicher Client id in der Ableitungsfunktion zu produzieren. Dazu beachte man, dass in $\texttt{rx}^{\text{ext}}$ der Kennzeichner $\texttt{ID}$ des Zertifikats enthalten ist und geprüft wird. Diese Kennzeichner müssten also übereinstimmen. Dann wäre aber bei der Anfrage CORRUPT($\widetilde{\mathsf{id}}$) auch der Client id wegen der Übereinstimmung auf $\texttt{ID}$ als korrupt markiert worden—und die Sitzung des Clients id wäre unmittelbar bei der Initialsierung auf REVEAL gesetzt worden und die TEST-Abfrage hätte $\perp$ zurückgegeben.

Insgesamt sind also die richtigen Schlüssel secret in testbaren Sitzungen in Spiel Game_7 alle uniform und unabhängig aus Sicht des Angreifers. Folglich sind alle getesteten Schlüssel für den Angreifer unabhängige Werte, in beiden Fällen $b = 0$ und $b = 1$, und in diesem Fall die Erfolgswahrscheinlichkeit genau $\frac{1}{2}$:

$$\Pr[\mathsf{Game}_7] \leq \frac{1}{2}.$$

Summiert man alle Wahrscheinlichkeiten auf, ergibt sich die Schranke aus dem Theorem. □

3.4 Bemerkungen

Die kaskadenhafte Verschlüsselung der Akten- und Kontextschlüssel wird hier nicht betrachtet. Dafür soll laut Spezifikation AES-GCM verwendet werden und die Ableitungsvektoren als assoziierte Daten dienen. Die Schlüssellänge beträgt jeweils 256 Bits und die Initialisierungsvektoren werden zufällig gewählt. Wir sehen damit keine grundsätzlichen Probleme.

Zur Vertraulichkeit der geheimen Schlüssel und Werte merken wir folgendes an:

1. Die langfristigen Ableitungsschlüssel der SGDe müssen geheim bleiben, sonst kann ein Angreifer bei Bekanntwerden der Ableitungsregeln die versichertenspezifischen Schlüssel rekonstruieren. Da aber wegen der doppelten Verschlüsselung des Akten- und Kontextschlüssels jeweils zwei Schlüssel benötigt werden, kann die Preisgabe eines Ableitungsschlüssels noch nicht unmittelbar genutzt werden, um den Akten- und Kontextschlüssels zu erhalten. In unserer Analyse wird dies dadurch abgebildet, dass der abgeleitete Schlüssel sicher ist, selbst wenn der andere SGD korrumpiert wurde.

2. Die Authentisierungstokens müssen geheim gehalten werden. Sonst könnte sich ein Angreifer bei Kenntniss eines solchen Tokens in `KeyDerivation` eventuell weitere abgeleitete Schlüssel für den Versicherten abholen.

3. Der Signaturschlüssel eines Clients muss sicher bleiben. Wenn der Angreifer den Schlüssel erlangt, kann er sich im Protokoll als der Client ausgeben und grundsätzlich versichertenspezifische Schlüssel vom SGD geben lassen. Insbesondere kann er sich auch bereits abgeleitete Schlüssel erneut geben lassen. Dies ist inhärent, da das System wiederholte Abfragen unterstützt. Das Protokoll unterstützt also keine vorwärts gerichtete Sicherheit (*engl. forward secrecy*) auf Client-Seite. Allerdings müsste sich der Angreifer zur Erlangung bereits abgeleiteter Schlüssel zuvor die assoziierten Daten vom Aktensystem, insbesondere den verwendeten zufälligen Wert RND in der Schlüsselableitung, geben lassen. Das Aktensystem selbst führt aber eine Autorisierungsprüfung durch, die einen solchen Angriff erschweren kann. Da das Aktensystem nicht Teil der Spezifikation des SGD-Verfahrens ist, geht dieser Aspekt hier allerdings nicht in die Analyse ein.

4. Der Signaturschlüssel eines SGDs muss vertraulich bleiben. Ansonsten könnte sich ein Angreifer als dieser SGD ausgeben und dem Client beispielsweise einen schwachen

Schlüssel unterschieben. Im Unterschied zum Client scheint der Signatur-Schlüssel allerdings vorwärts gerichtete Sicherheit zu gewährleisten: Solange der langfristige Ableitungsschlüssel des SGD geschützt bleibt, scheint es für den Angreifer dennoch keine Möglichkeit zu geben, bereits abgeleitete Schlüssel zu rekonstruieren.

5. Der Client verwendet eine zufällige Challenge `ch` bei der `GetAuthenticationToken`-Abfrage. Die Challenge sollte verhindern, dass der Angreifer ein falschen Authentisierungstoken beim Client setzen kann, genauso wie der zufällige Wert reqID in `KeyDerivation` verhindert, dass der Angreifer einen falschen abgeleiteten Schlüssel übertragen kann. Die zusätzliche Robustheit ist nützlich, aber für unsere Analyse nicht relevant: Der Angreifer müsste immer noch die Prüfung gegen das richtige Token auf der SGD-Seite überlisten.

6. Der Client-Schlüssel epk_C wird vor der Authentisierung noch an die temporären Schlüssel der SGDe gebunden, indem der Schlüssel zu $\mathsf{epk}_C^{\text{ext}} = \mathsf{epk}_C|\mathsf{H}(\mathsf{pk}_1)|\mathsf{H}(\mathsf{pk}_2)$ erweitert wird. Dieser Schritt sorgt für zusätzlichen Schutz, weil der Angreifer den temporären Client-Schlüssel während der Gültigkeit der SGD-Schlüssel brechen müsste, um aktive Angriffe zu starten. Da wir zeitliche Aspekte im Modell nicht abbilden, wird dies nicht in der Analyse verwendet.

Literaturverzeichnis

[ABR99] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHAES: an encryption scheme based on the Diffie-Hellman problem. *IACR Cryptol. ePrint Arch.*, page 7, 1999.

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2009.

[BfSidI21] Bundesamt für Sicherheit in der Informationstechnik. Technische Richtlinie BSI TR-02102-1: Kryptographische Verfahren: Empfehlungen und Schlüssellängen, version 2021-01. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf`, März 2021.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of bellare-rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 51–62. ACM, 2011.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[DFGS21] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *J. Cryptol.*, 34(4):37, 2021.

[FG17] Marc Fischlin and Felix Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 60–75. IEEE, 2017.

[gem20a] gematik. Übergreifende Spezifikation: Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur, version 2.18.0. `https://fachportal.gematik.de/fachportal-import/files/gemSpec_Krypt_V2.18.0.pdf`, November 2020.

[gem20b] gematik. Spezifikation Schlüsselgenerierungsdienst ePA, version 1.4.1. `https://fachportal.gematik.de/fachportal-import/files/gemSpec_SGD_ePA_V1.4.1.pdf`, November 2020.

[KE10] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Third Edition*. CRC Press, 2020.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, 2010.

[MF21] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography*. Information Security and Cryptography. Springer, 2021.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.