Effizientes Verifiable Encryption und Fairer Tausch von Geheimnissen

Christian Tobias

Diplomarbeit
am Fachbereich Mathematik
Johann-Wolfgang-Goethe-Universität
Frankfurt am Main

Betreuer: Prof. Dr. C.P. Schnorr

22. Juli 1999

Erklärung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen Hilfsmittel als die angegebenen Quellen verwendet habe.

Christian Tobias Frankfurt am Main, den 22. Juli 1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der vorliegenden Arbeit wird ein interaktives Beweisprotokoll für das Problem der *überprüfbaren Verschlüsselung (verifiable encryption)* vorgestellt. Mit Hilfe eines *Verifiable Encryption Protokolls (VEP)* beweist eine Person (der Prover) einer anderen Person (dem Verifier) effizient, daß ein vorher gesendeter Wert α die Verschlüsselung eines geheimen Wertes s ist. Den geheimen Wert s muß er dazu nicht offenlegen. Zur Verschlüsselung von s wird ein Public-Key-Verfahren und ein öffentlicher Schlüssel PK benutzt. PK gehört zum Schlüsselpaar einer dritten Partei, die nicht aktiv an der Protokollausführung beteiligt ist und die Rolle eines Notars einnimmt. Dem Verifier steht ein Wert d zur Verfügung, anhand dessen er entscheidet, ob er den Beweis akzeptiert oder verwirft. Akzeptiert der Verifier den Beweis des Provers, so kann er zwar mit an Sicherheit grenzender Wahrscheinlichkeit sagen, daß α eine Verschlüsselung von s unter dem öffentlichen Schlüssel PK ist. Er kann s jedoch nicht rekonstruieren, da er nicht im Besitz des zu PK gehörigen geheimen Schlüssels SK ist und der Beweis keine Informationen über s preisgibt.

Verifiable Encryption Protokolle werden unter anderem zum fairen Tausch digitaler Signaturen verwendet. Asokan, Shoup und Waidner stellen in [1] ein Protokoll zum fairen Signaturtausch vor, das im wesentlichen auf einem solchen VEP beruht. Dabei handelt es sich um ein optimistisches Protokoll, d.h. es wird zwar die Existenz einer vertrauenswürdigen und immer erreichbaren dritten Partei (Trusted Third Party, TTP) angenommen, diese greift aber nur bei Streitfällen in das Protokoll ein.
Das Problem des fairen Tauschs digitaler Signaturen tritt häufig bei Anwendungen im Bereich des Electronic Commerce auf. So ist es heute etwa schon möglich, Flugtickets über das Internet zu kaufen und diese mit elektronischem Geld online zu bezahlen. Sowohl die Flugtickets als auch die elektronischen Geldstücke werden dabei durch digitale Signaturen repräsentiert. Diese müssen so getauscht werden, daß eine Situation, in der einer der beteiligten Personen seine Signatur weggegeben hat, im Gegenzug aber selbst nichts erhalten hat, nicht auftreten kann.
Als weitere Anwendung von Protokollen zum fairen Tausch digitaler Signaturen

stelle man sich den Vertragsschluß im Internet vor. Hier einigen sich zwei Parteien auf einen Vertragstext, den jeder der beteiligten Personen digital signiert. Anschließend müssen die erstellten Signaturen noch (fair) ausgetauscht werden.

Auch bei Escrow-Verfahren können VEPs eingesetzt werden. Bei diesen Verfahren hinterlegt eine Person bestimmte Daten (z.B. seine Identität). Diese Daten sollen so lange vertraulich bleiben, bis eine Situation eintritt, die ihre Aufdeckung verlangt. Das VEP garantiert dabei, daß auch wirklich die gewünschten Daten hinterlegt wurden.
Beim sogenannten *Key-Escrow* müssen Personen, die Daten verschlüsselt versenden wollen, ihre Schlüssel bei einer dritten Partei hinterlegen. Auf richterlichen Beschluß können diese Schlüssel dann aufgedeckt und der Datenverkehr dieser Person (zur Verbrechensbekämpfung) mitgelesen werden.
Bei elektronischen Bezahlsystemen sollen die beteiligten Personen in der Lage sein, ihre Geschäftsvorgänge anonym abzuwickeln. Kommt es jedoch zu einem Betrugsversuch, so muß die Identität des Betrügers aufgedeckt werden können. Dies kann durch spezielle Escrow-Verfahren erreicht werden.

Diese Arbeit baut auf einer offenen Frage aus [1] auf. Dort wird das Problem des fairen Tauschs digitaler Signaturen zunächst auf das Problem des Tauschs zweier Urbilder s_1 und s_2 unter einem effizient zu berechnenden aber schwer zu invertierenden Gruppenhomomorphismus $\theta : G_1 \rightarrow G_2$ bei bekannten Bildern d_1 und d_2 reduziert. Die Werte s_1 und s_2 werden verschlüsselt und ausgetauscht. Das VEP garantiert dabei, daß es sich bei den gesendeten Werten tatsächlich um Verschlüsselungen von Urbildern von d_1 und d_2 handelt. Danach können s_1 und s_2 offen getauscht werden. Kommt es zu einem Disput, so kann jederzeit eine dritte Partei eingeschaltet werden, die das Protokoll fair zu Ende führt.
Das in [1] vorgestellte VEP arbeitet mit beliebigen (Einweg-) Homomorphismen und ist damit sehr allgemein. Es benutzt jedoch die Cut-and-Choose Technik, wodurch es ineffizient wird: um die Betrugswahrscheinlichkeit des Provers auf 2^{-n} zu drücken, muß das Protokoll n-mal durchlaufen werden.
In [1] können beliebige Verschlüsselungsfunktionen, die sicher gegen adaptive-chosen-ciphertext-Angriffe (stärkste Form des Angriffs auf Public-Key-Verfahren) sind, sowie beliebige (Einweg-) Homomorphismen θ benutzt werden. Die vorgestellten Reduktionsschemata erlauben einen Austausch von RSA- [36], DSS- [31], Schnorr- [38], Fiat-Shamir- [20], GQ- [26] und Ong-Schnorr-Signaturen [35] sowie elektronischem Geld nach Brands[8].

Das in dieser Arbeit vorgestellte VEP kommt mit nur einer Runde aus. Dadurch wird der benötigte Rechen- und Kommunikationsaufwand drastisch reduziert. Es ist jedoch nicht so allgemein wie das VEP aus [1].
Folgende Einschränkungen müssen gemacht werden:

- Je nachdem, ob die Urbildgruppe von θ additiv oder multiplikativ ist, muß das VEP leicht abgewandelt werden.
- Das verwendete Verschlüsselungsverfahren muß über homomorphe Verknüpfungseigenschaften verfügen. Für additive Urbildgruppen wird in die-

ser Arbeit die erweiterte ElGamal-Verschlüsselung aus [39] und für multiplikative Urbildgruppen eine abgewandelte Okamoto/Uchiyama-Verschlüsselung [34] verwendet.
Es ist jedoch auch möglich, andere Verschlüsselungsverfahren einzusetzen. Diese müssen über die erwähnten Verknüpfungseigenschaften verfügen und sicher gegen adaptive-chosen-ciphertext-Angriffe sein.

- Derzeit können nur sogenannte Diskreter Logarithmus-Unterschriften ausgetauscht werden. Hier müssen die Systemparameter (d.h. die Primzahlen p und q und der Generator g) für alle Teilnehmer fest vorgegeben werden.

  Um das Protokoll auch auf RSA-Signaturen [36] anwenden zu können, müßten neue Reduktionen (Teilprotokoll des Fair Exchange Protokolls, siehe Kapitel 5) gefunden werden.

Das neue VEP wird später in ein Protokoll zum fairen Tausch digitaler Signaturen (Fair Exchange Protokoll, FEP) integriert. Bei diesem FEP tauschen zwei Person A und B zunächst Verschlüsselungen ihrer Signaturen σ_A und σ_B aus. Mittels zweier Verifiable Encryption Protokolle beweisen sie sich gegenseitig, daß auch wirklich Verschlüsselungen von σ_A bzw. σ_B gesendet wurden. Anschließend werden die Signaturen im Klartext ausgetauscht. Versucht eine der Parteien zu betrügen oder kommt es zu einem technischen Defekt, kann durch Einschalten der TTP das Protokoll fair beendet werden.
Das vorgestellte FEP hat folgende Eigenschaften:

- Erhält Person A keine Antwort mehr (etwa weil Person B das Protokoll abgebrochen hat), so kann Person A jederzeit eine Beendigung des Protokolls erzwingen. Dazu muß sie sich an die TTP wenden. Der Signaturtausch wird entweder abgeschlossen oder es wird eine Situation erreicht, in der keine der Parteien verwertbare Daten erhalten hat.

- Das Protokoll erlaubt auch den Tausch von Unterschriften, die nicht von A oder B selbst erstellt wurden.
  Dies ist bei elektronischen Bezahlvorgängen wichtig. Hier wird die Gültigkeit von elektronischen Geldstücken durch die Unterschrift der Bank bestätigt.

- Die Parteien A und B können gegenüber der TTP anonym auftreten.

- Es wird eine zweite Versionen des FEP vorgestellt. In diesem FEP muß das VEP nur einmal durchlaufen werden. Es ist dann allerdings keine Anonymität gegenüber der TTP mehr gegeben.

Die Kapitel 2 und 3 geben eine kurze Einführung in die zum Verständnis des weiteren Textes nötigen kryptographischen Grundbegriffe. Zunächst wird in

Kapitel 2 das Konzept der digitalen Signatur erläutert. Neben den später verwendeten (auf dem Diskreten Logarithmus Problem basierenden) Signaturverfahren wird auch das RSA-Signaturverfahren eingeführt, da es weit verbreitet ist und das erste praktikable Signaturverfahren war. Kapitel 3 führt in interaktive Beweisprotokolle und Zero-Knowledge-Beweise ein. Die teilweise sehr abstrakten Begriffe werden am Beispiel des Fiat-Shamir Protokolls veranschaulicht.
In Kapitel 4 wird gezeigt, wie man das ElGamal-Verschlüsselungsverfahren [18] sicher gegen adaptive-chosen-ciphertext-Angriffe machen kann. Es wird versucht, das von Okamoto und Uchiyama auf der Eurocrypt '98 vorgestellte Verfahren [34] durch ähnliche Techniken ebenfalls sicher gegen adaptive-chosen-ciphertext-Angriffe zu machen. Formal wird allerdings nur ein etwas schwächeres Ergebnis bewiesen. Darauf aufbauend werden zwei Verifiable Encryption Protokolle entwickelt, die sich in der Struktur der Urbildgruppen des verwendeten Homomorphismus θ unterscheiden.

Im abschließenden fünften Kapitel wird zunächst ein Überblick über das Problem des fairen Tauschs digitaler Signaturen gegeben. Einer der Lösungsansätze dieses Problems [1] wird anschließend näher betrachtet. Es wird dann gezeigt, wie durch die Integration des hier vorgestellten VEPs einige Schwächen dieses Protokolls ausgebessert werden können.

Kapitel 2

Digitale Signaturen

Digitale Signaturen gehören zur Gruppe der asymmetrischen kryptographischen Verfahren. Zur Erstellung einer digitalen Signatur σ einer Nachricht m benötigt man einen geheimen Signaturschlüssel SK. Zur Verifikation der Signatur benötigt man die signierte Nachricht m, die Signatur σ und den zu SK gehörigen öffentlichen Schlüssel PK. Wie bei allen asymmetrischen Verfahren darf es (rechnerisch) nicht möglich sein, aus dem öffentlichen Schlüssel PK den geheimen Schlüssel SK abzuleiten.
Ferner darf es (rechnerisch) nicht möglich sein, ohne Kenntnis von SK digitale Signaturen zu erzeugen, die mit PK richtig verifiziert werden. Im Umkehrschluß bedeutet das, daß eine Signatur, die mit PK richtig verifiziert wird, unter Benutzung von SK erstellt wurde. Da SK aber (in der Regel) nur einer Person bekannt ist, beweist die Richtigkeit der Signatur auch gleichzeitig die Urheberschaft der Signatur.
Da die zu signierende Nachricht als Eingabe für die Signaturerzeugung verwendet wird, ergibt sich ein weiterer Vorteil digitaler Signaturen. Das Verändern der zu signierenden Nachricht wird zu einer anderen Signatur führen. Kann eine Signatur richtig verifiziert werden, so ist damit sichergestellt, daß die Nachricht im Nachhinein nicht verändert wurde.
Eine formale Definition digitaler Signaturen findet sich in [25].

Digitale Signaturen spielen in kryptographischen Anwendungen eine bedeutende Rolle. Sie können benutzt werden, um im email-Verkehr Nachrichtenintegrität (Unverfälschtheit der Nachricht) und Benutzerauthentizität (Urheberschaft) zu sichern. Cerification Authorities (CA) benutzen digitale Signaturen, um die Bindung eines öffentlichen Schlüssels an einen bestimmten Benutzer zu garantieren. Banken können durch ihre Signatur die Richtigkeit und Gültigkeit von elektronischen Geldstücken bestätigen. Nahezu alle Anwendungen, bei denen Wertgegenstände in Form von Bits und Bytes realisiert werden, benutzen digitale Signaturen.

Im folgenden Kapitel werden die am weitesten verbreiteten Signaturverfahren vorgestellt. Einen Schwerpunkt bilden hierbei die auf dem diskreten Logarithmus Problem beruhenden Signaturverfahren, da für diese Klasse von Signa-

turverfahren in den folgenden Kapiteln ein effizientes Protokoll für den fairen Tausch von Signaturen vorgestellt wird.
Zunächst wird allerdings das nach seinen Erfindern Rivest, Shamir und Adleman benannte RSA-Schema [36] vorgestellt, da es das erste praktikable Signaturverfahren war.

2.1 RSA-Signaturen

Rivest, Shamir und Adleman haben den RSA-Algorithmus entdeckt, als sie zeigen wollten, daß es die 1976 von Diffie und Hellman in [17] vorgestellten trapdoor Einwegfunktionen nicht geben kann. Sie haben damit den ersten Vertreter einer Klasse von Funktionen geliefert, deren Nicht-Existenz sie zeigen wollten.

Schlüsselerzeugung
Jeder Teilnehmer kann sich ein Schlüsselpaar erzeugen, indem er folgende Schritte durchläuft:

- Generiere zwei große Primzahlen p und q, die ungefähr gleich groß sind.
- Berechne $n = pq$ und $\phi(n) = (p-1)(q-1)$
- Wähle ein zufällige Zahl $e \in_R \mathbb{Z}^*_{\phi(n)}$
- Berechne (mit dem erweiterten euklidischen Algorithmus) die Zahl $d = e^{-1} \, mod \, \phi(n)$

Das Paar (e, n) bildet den öffentlichen und d den privaten Schlüssel.

Signaturerstellung
Um eine Nachricht $m \in \mathbb{Z}_n$ zu **signieren**, berechnet man

$$s = m^d \, mod \, n.$$

Der Wert s bildet dann die Signatur von m bzgl. des (öffentlichen) Schlüssels (e, n).

Signaturverifikation
Die Signatur ist gültig, wenn die Gleichung

$$s^e = m \, mod \, n$$

erfüllt ist.

Bemerkung 2.1 (Korrektheit des Verfahrens)
Wir haben gefordert, daß $ed = 1 \, mod \, \varphi(n)$, *d.h.* $\exists k \in \mathbb{Z}$ *mit* $ed = 1 + k \cdot \varphi(n)$.
Für Nachrichten $m \in \mathbb{Z}^*_n$ *folgt mit dem kleinen Satz von Fermat, daß* $(m^e)^d = m^{1+k\varphi(n)} = m \, mod \, n$.
Diese Beziehung gilt sogar für beliebige $m \in \mathbb{Z}_n$*:*
Angenommen, $ggt(m, n) \neq 1$ *(o.B.d.A. sei* $ggt(m, n) = p$*).*

In diesem Fall ist die Gleichung $m^{ed} = m \bmod p$ *trotzdem gültig (beide Seiten haben den Wert 0) und die Richtigkeit des Verfahrens folgt mit dem chinesischen Restsatz.*

Das RSA-Problem
Es seien eine ganze Zahl n, die das Produkt zweier Primzahlen p und q ist, eine ganze Zahl $e \in \mathbb{Z}^*_{\phi(n)}$ und eine Zahl $c \in \mathbb{Z}_n$ gegeben.
Finde eine ganze Zahl m mit $m^e = c \bmod n$.
Die *RSA-Annahme* besagt, daß das RSA-Problem schwer ist für (hinreichend große) zufällige RSA-Moduln n und zufällige $e \in_R \mathbb{Z}^*_{\phi(n)}$ und $c \in_R \mathbb{Z}_n$.

Das Faktorisierungsproblem
Es sei eine ganze Zahl n gegeben.
Finde die Primfaktorzerlegung von n, d.h. schreibe n als $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, wobei die p_i paarweise verschiedene Primzahlen und die $e_i \geq 1$ sind.
Die *Faktorisierungsannahme* besagt, daß das Faktorisierungsproblem schwer ist für (hinreichend große) RSA-Moduln n.

Ein Algorithmus, der das Faktorisierungsproblem effizient löst, kann benutzt werden, um das RSA-Problem zu lösen.
Die Umkehrung konnte bisher nicht bewiesen werden. Man vermutet aber, daß auch sie richtig ist.

Sicherheit von RSA

1. **Multiplikativität von RSA**
   Seien $s_1 = (m_1)^d \bmod n$ und $s_2 = (m_2)^d \bmod n$ gültige Signaturen von m_1 bzw. m_2.
   Dann ist $s_1 s_2 = (m_1 m_2)^d \bmod n$ eine gültige Signatur von $(m_1 m_2)$. Diese kann ohne Kenntnis des geheimen Schlüssels d aus s_1 und s_2 berechnet werden.

2. **Common Modulus Attack**
   Das RSA-Schema ist unsicher, wenn mehrere Personen denselben Modul n verwenden, da aus der Kenntnis von e und d bereits die Primfaktoren p und q des Moduls n berechnet werden können.
   Mittels der Faktoren p und q kann man dann den geheimen Schlüssel eines jeden anderen Teilnehmers, der auch diesen Modul benutzt, aus dessen öffentlichem Schlüssel berechnen.

3. **Kleine geheime Exponenten**
   In [44] hat M. Wiener gezeigt, daß man den geheimen Exponenten d effizient aus e und N berechnen kann, wenn p und q so gewählt wurden, daß $q < p < 2q$ gilt, und d so gewählt wurde, daß $d < \frac{1}{3} n^{0.25}$ gilt.
   Boneh und Durfee haben in [7] den Angriff von Wiener modifiziert und so die Grenze auf $d < n^{0.292}$ verbessert.

Einen guten Überblick über die Sicherheit von RSA bietet [6].

2.2 DSA-Signaturen

Der Digital Signature Algorithm (DSA) [31] ist ein Teil des Digital Signature Standards (DSS). Der DSA wurde im Mai 1994 vom National Institute of Standards and Technology (NIST) zum Standard erklärt. Er ist eine Variante der Verfahren von ElGamal und Schnorr.

Systemparameter:
Primzahlen p und q. Für p wird (laut Standard) eine Länge von l Bits gefordert, wobei l ein Vielfaches von 64 ist und mindestens 512 beträgt. Die Bitlänge von q ist auf 160 festgelegt. Für q wird ferner gefordert, daß $q \mid (p-1)$.
Ferner sei g ein Generator der eindeutigen Untergruppe von $\mathbb{Z}_p^*$ der Ordnung q. Dieser Generator kann effizient berechnet werden:

1. Wähle ein zufälliges $x \in_R \mathbb{Z}_p^*$.
   Berechne $g = x^{(p-1)/q} \bmod p$.
2. Ist $g = 1$, so beginne erneut bei Schritt 1.

Die Werte p, q und g können von jedem Teilnehmer selbst gewählt und als Teil ihres öffentlichen Schlüssels veröffentlicht werden. Es ist jedoch auch möglich, daß eine Gruppe von Teilnehmern die gleichen Parameter p, q und g benutzt, ohne daß die Sicherheit des Verfahrens beeinträchtigt wird.

Jeder Teilnehmer wählt sich zufällig eine Zahl $a \in_R \mathbb{Z}_q^*$ als geheimen Schlüssel und berechnet daraus seinen öffentlichen Schlüssel $y = g^a \bmod p$.
Ferner ist eine Hashfunktion $h : \{0,1\}^* \to \mathbb{Z}_q$ gegeben. Der Standard [31] sieht den Secure Signature Algorithm (SHA-1) als Hashfunktion vor.

Signaturerstellung
Um eine Nachricht m beliebiger Länge zu signieren, macht man folgendes:

- Wähle eine zufällige Zahl k mit $0 < k < q$.
- Berechne $r = (g^k \bmod p) \bmod q$
- Berechne $k^{-1} \bmod q$
- Berechne $s = k^{-1}(h(m) + ar) \bmod q$

Das Paar (r, s) ist die Signatur von m bzgl. des öffentlichen Schlüssels $y = g^a \bmod p$.

Signaturverifikation
Um eine Signatur (r, s) unter dem öffentlichen Schlüssel $y = g^a \bmod p$ zu prüfen, mache folgendes:

- Prüfe, ob $0 < r, s < q$
  Ist dies nicht der Fall, so brich ab.
- Berechne $w = s^{-1} \bmod q$ und $h(m)$.

- Berechne $u_1 = wh(m) \, mod \, q$ und $u_2 = rw \, mod \, q$.
- Berechne $v = (g^{u_1} y^{u_2} \, mod \, p) \, mod \, q$.
- Akzeptiere genau dann, wenn $v = r$.

Der Vorgang der Signaturprüfung schlägt fehl, wenn $s = 0$ (in diesem Falle existiert s^{-1} nicht). Der Signaturersteller sollte also prüfen, ob tatsächlich $s \neq 0$ gilt und gegebenenfalls einen neuen Wert k wählen und die Signaturerzeugung wiederholen. Dieser Fall tritt aber nur ungefähr mit Wahrscheinlichkeit 2^{-160} auf (s ist gleichverteilt in $\mathbb{Z}_q$).

Sicherheit von DSA

- Kennt ein Angreifer den Zufallswert $k \in \mathbb{Z}_q^*$, der für die Erstellung einer DSA-Signatur (r, s) einer Nachricht m verwendet wurde, so kann er den geheimen Schlüssel des Signaturerstellers berechnen:

  Aus Kenntnis von k lassen sich die Werte $r = (g^k \, mod \, p) \, mod \, q$ und $k^{-1} \, mod \, q$ berechnen.
  Von der Gleichung $s = k^{-1}(h(m) + ar) \, mod \, q$ sind alle Werte bis auf a bekannt. Der geheime Schlüssel ergibt sich also durch Berechnung von $a = (sk - h(m))r^{-1} \, mod \, q$.

  Ein guter Zufallszahlengenerator ist daher bei jeder Implementierung von DSA entscheidend für die Sicherheit des Verfahrens.

- Ein ähnlicher Angriff ergibt sich, wenn zwei Nachrichten m_1 und m_2 mit Hilfe derselben Zufallszahl k verschlüsselt wurden. In diesem Fall kann mit hoher Wahrscheinlichkeit der Wert k (und daraus der geheime Schlüssel a) rekonstruiert werden:

  Seien $s_1 = k^{-1}(h(m_1) + ar) \, mod \, q$
  und $s_2 = k^{-1}(h(m_2) + ar) \, mod \, q$
  $\Rightarrow (s_1 - s_2)k = (h(m_1) - h(m_2) \, mod \, q$
  Für $s_1 \neq s_2$ folgt also $k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \, mod \, q$

2.3 Schnorr-Signaturen

Genau wie bei DSA werden auch im Schnorr-Signaturverfahren die Berechnungen in einer Untergruppe von $\mathbb{Z}_p$ der Ordnung q ausgeführt. Das Schnorr-Signaturverfahren wurde auf der Crypto 1989 vorgestellt. Es beruht auf einem Identifizierungsprotokoll, das (wie das Signaturverfahren auch) in [37] veröffentlicht wurde.

Systemparameter
Zwei große Primzahlen p und q mit $q \mid (p-1)$.
Anders als bei DSA sind im Schnorr-Verfahren keine Bedingungen an die Größe von p und q geknüpft.
Sei $g \in_R \mathbb{Z}_p^*$ mit $ord(g) = q$, d.h. g ist wieder ein Generator der (eindeutigen) Untergruppe von $\mathbb{Z}_p^*$ der Ordnung q.
Eine Hashfunktion $h : \{0,1\}^* \to \mathbb{Z}_q$

Die Schlüsselerzeugung läuft wie bei DSA, d.h. jeder Teilnehmer wählt sich ein zufälliges $a \in_R \mathbb{Z}_q^*$ als geheimen Schlüssel und berechnet daraus den öffentlichen Schlüssel $y = g^a \, mod \, p$.

Signaturerstellung
Sei $(PK, SK) = ((p, q, g, y), r)$ das Schlüsselpaar eines Benutzers. Dieser Benutzer signiert eine Nachricht m beliebiger Länge, indem er folgendes macht:

- Wähle ein zufälliges $k \in_R \mathbb{Z}_q^*$
- Berechne $r = g^k \, mod \, p$
- Berechne $e = h(m, r)$
- Berechne $s = ae + k \, mod \, q$

Das Paar $(s, e) \in \mathbb{Z}_q^2$ ist die Signatur der Nachricht m unter dem öffentlichen Schlüssel PK.

Signaturverifikation
Das Paar (s, e) ist genau dann eine gültige Schnorr-Signatur von m unter dem (öffentlichen) Schlüssel (p, q, g, y), wenn die Gleichung

$$e = h(m, g^s y^{-e})$$

erfüllt ist.

Sicherheit
In [39] zeigen Schnorr und Jakobsson, daß die Schnorr-Signatur sicher gegen One-more-Forgery Angriffe ist (Im Random Oracle und generischen Modell).
Wir nehmen an, daß A ein probabilistischer polynomialzeit Angreifer ist, der Zugriff auf ein Signaturorakel hat. Diesem Orakel darf er beliebige Fragen stellen, d.h. er legt dem Orakel beliebige Nachrichten vor, für die ihm das Orakel gültige Signaturen schickt.
Ein Signaturverfahren heißt sicher gegen One-more-Forgery Angriffe, wenn es einem solchen Angreifer nicht gelingt nach n Interaktionen mit dem Signa- turorakel $n + 1$ gültige Signaturen auszugeben.

Performance
Um eine Schnorr-Signatur zu erstellen, müssen lediglich zwei modulare Exponentiationen durchgeführt werden. Davon muß lediglich eine berechnet werden. Die Exponentiation modulo der größeren Primzahl p kann schon im Vorfeld ausgeführt werden.
Ein weiterer Vorteil des Schnorr-Signaturverfahrens ist die kurze Unterschriftenlänge (von ca. $2 \cdot 160 = 320$ Bits).

2.4 Okamoto-Schnorr-Signaturen

Eine Variante der Schnorr-Signatur ist das Okamoto-Schnorr-Signaturschema [33]. Genau wie bei der Schnorr-Signatur beruht das in [33] vorgestellte Signaturschema auch auf einem Identifizierungsprotokoll.

Systemparameter

- Zwei große Primzahlen p und q mit $q \mid (p-1)$.
  In der Originalarbeit schlägt der Autor für q eine Länge von 140 Bits und für p eine Länge von 512 Bits vor.
- $g_1, g_2 \in \mathbb{Z}_p^*$ mit Ordnung q.
- Zufallszahlen $s_1, s_2 \in_R \mathbb{Z}_q$
- $v = g_1^{-s_1} g_2^{-s_2}$

Das Tupel (p, q, g_1, g_2, v) bildet den öffentlichen und das Paar (s_1, s_2) den privaten Schlüssel. Das Tupel (p, q, g_1, g_2) kann wieder systemweit vorgegeben sein oder von jedem Benutzer selbst gewählt werden.

Signaturerstellung
Um eine Nachricht $m \in \{0,1\}^*$ zu signieren, macht man folgendes:

- Wähle zwei Zufallszahlen $r_1, r_2 \in_R \mathbb{Z}_q$.
- Berechne $x = g_1^{r_1} g_2^{r_2} \, mod \, p$
- Berechne $e = h(x, m) \in \mathbb{Z}_q$
- Berechne $y_1 = r_1 + e s_1$ und $y_2 = r_2 + e s_2$

Das Tupel (e, y_1, y_2) ist die digitale Signatur von m bzgl. des (öffentlichen) Schlüssels (p, q, g_1, g_2, v).

Signaturverifikation
Das Tupel (e, y_1, y_2) ist genau dann eine gültige Signatur von m unter dem (öffentlichen) Schlüssel (p, q, g_1, g_2), wenn folgende Gleichung erfüllt ist:

$$e = h(g_1^{y_1} g_2^{y_2} v^e \, mod \, p, m)$$

Sicherheit
In [33] zeigt Okamoto, daß sein Signaturschema existentially unforgeable gegen adaptive-chosen-message-Angriffe ist, sofern das diskrete Logarithmus Problem schwer ist und es sich bei h um eine Hashfunktion handelt, die kollisionsfrei ist.

2.5 ElGamal-Signaturen

Das ElGamal-Signaturschema [19] wurde 1984 von Taher ElGamal vorgestellt. Es handelt sich um ein probabilistisches Verfahren, das als Vorlage für zahlreiche weitere Signaturverfahren, wie z.B. für DSA- und Schnorr-Signaturen, diente.

Systemparameter
Sei p eine große Primzahl (ca. 1024 Stellen) und g ein Generator der multiplikativen Gruppe $\mathbb{Z}_p^*$.
Ferner sei eine kryptographische Hashfunktion $h : \{0,1\}^* \rightarrow \mathbb{Z}_p$ gegeben.
Jeder Teilnehmer wählt sich eine ganze Zahl a mit $1 \leq a \leq p-2$ als geheimen Schlüssel. Der dazugehörige öffentliche Schlüssel ergibt sich dann als $y = g^a \, mod \, p$ (bzw. als (p, q, g, y)).

Signaturerstellung
Um eine Nachricht m beliebiger Länge zu signieren, macht man folgendes:

- Wähle eine geheime Zahl k mit $1 \leq k \leq p-2$ und $ggt(k, p-1) = 1$.
- Berechne $r = g^k \, mod \, p$
- Berechne $s = k^{-1}(h(m) - ar) \, mod \, (p-1)$

Das Paar (r, s) ist die digitale Signatur von m unter dem (öffentlichen) Schlüssel (p, q, g, y).

Signaturverifikation
Sei (p, q, g, y) der öffentliche Schlüssel des Signaturerstellers.
Die Signatur (r, s) von m ist genau dann gültig, wenn die beiden folgenden Bedingungen erfüllt sind:

- Überprüfe, ob $1 \leq r \leq p-1$
  Ist dies nicht der Fall, so breche ab.
- Überprüfe, ob $y^r r^s = g^{h(m)} \, mod \, p$

Sicherheit

- Es sollte für jede Nachricht, die signiert wird, ein anderes k gewählt werden. Ansonsten kann der Wert k und daraus der geheime Schlüssel a mit hoher Wahrscheinlichkeit rekonstruiert werden:

Seien $s_1 = k^{-1}(h(m_1) - ar) \, mod \, (p-1)$
und $s_2 = k^{-1}(h(m_2) - ar) \, mod \, (p-1)$
$\Rightarrow (s_1 - s_2)k = (h(m_1) - h(m_2)) \, mod \, (p-1)$
Gilt $s_1 - s_2 \neq 0$, so folgt $k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \, mod \, (p-1)$.

- Wenn keine Hashfunktion benutzt wird, d.h. $s = k^{-1}(m - ar) \, mod \, (p-1)$, können gültige Signaturen ohne Kenntnis des geheimen Schlüssels erstellt werden:

  Wähle $u, v \in_R \mathbb{Z}_p$ mit $ggt(v, p-1) = 1$
  Berechne $r = g^u y^v = g^{u+av} \, mod \, p$ und $s = -rv^{-1} \, mod \, (p-1)$.
  Das Paar (r, s) ist eine gültige Signatur von m=su unter dem (öffentlichen) Schlüssel (p, q, g, y).

- Wenn bei der Signaturverifikation die Bedingung $1 \leq r \leq p-1$ nicht überprüft wird, so können aus einer gültigen Signatur (ohne Kenntnis des geheimen Schlüssels) mit hoher Wahrscheinlichkeit weitere gültige Signaturen erstellt werden:

  Sei (r, s) eine gültige Signatur der Nachricht m und sie $\hat{m}$ eine weitere (beliebige) Nachricht.
  Berechne $u = h(\hat{m})(h(m))^{-1} \, mod \, (p-1)$ (vorausgesetzt, $h(m)^{-1} \, mod \, p$ existiert).
  Berechne $\hat{s} = su \, mod \, (p-1)$ und ein $\hat{r}$, für das $\hat{r} = ru \, mod \, (p-1)$ und $\hat{r} = r \, mod \, p$ gilt. Ein solches $\hat{r}$ existiert immer (chinesischer Restsatz).
  Dann ist das Paar $(\hat{r}, \hat{s})$ eine gültige Unterschrift von $\hat{m}$ (sofern $1 \leq r \leq p-1$ nicht überprüft wird).

Weitere Hinweise bzgl. der Sicherheit des ElGamal-Signaturschemas finden sich in [30], [29] und [42].

Kapitel 3

Kryptographische Grundbegriffe

3.1 Interaktive Beweisprotokolle

In einem interaktiven Beweisprotokoll tauschen zwei Personen (ein Prover und ein Verifier) Nachrichten aus. Ziel des Protokolles ist es, daß der Prover dem Verifier eine bestimmte Behauptung beweist (etwa, daß er ein Geheimnis kennt). Prover und Verifier sind probabilistische interaktive Turing-Maschinen, die jeweils ein Band mit internen Münzwürfen zur Verfügung haben, von dessen Inhalt sie ihre Berechnungen abhängig machen können.
Am Ende des Protokolls wird der Verifier entweder akzeptieren oder verwerfen.

Definition 3.1 (Completeness-Eigenschaft)
Ein interaktives Beweisprotokoll heißt complete, wenn das Protokoll bei ehrlichem Prover und Verifier mit überwältigender Wahrscheinlichkeit zum Erfolg führt (d.h. der Verifier akzeptiert).

Die Completeness-Eigenschaft sagt etwas über die Durchführbarkeit des Protokolls aus. Sie ist die Grundvoraussetzung für ein praktikables interaktives Beweisprotokoll.

Definition 3.2 (Soundness-Eigenschaft)
Ein interaktives Beweisprotokoll heißt sound, wenn ein Algorithmus M existiert mit im Mittel polynomieller Laufzeit und den folgenden Eigenschaften:
Wenn ein unehrlicher Prover mit nicht-vernachlässigbarer Erfolgswahrscheinlichkeit das Protokoll mit dem Verifier ausführen kann, dann kann M benutzt werden, um aus den Eingaben des Provers Wissen zu extrahieren, mit dem das Protokoll in folgenden Ausführungen mit überwältigend großer Wahrscheinlichkeit erfolgreich bestanden werden kann.

Aus der Definition folgt, daß jeder, der in die Rolle des Provers schlüpfen und das Protokoll mit nicht-vernachlässigbarer Wahrscheinlichkeit bestehen kann,

bereits über Wissen verfügen muß, das äquivalent zur Kenntnis des Geheimnisses des Provers ist. Damit stellt ein interaktives Beweisprotokoll, das sound ist, einen Beweis der Kenntnis des Geheimnisses (proof of knowledge) dar.

Definition 3.3
Die Zufallsvariable view(x,h) beschreibt die Folge der während der Protokollausführung (auf Eingabe x) ausgetauschten Daten. Der Wert h steht für weitere (geheime) Eingaben, die der Verifier zur Verfügung hat (Die Länge von h muß polynomiell beschränkt in der Eingabelänge $\mid x \mid$ sein, d.h. $\mid h \mid < poly(\mid x \mid)$). Die Zufallsvariable view(x,h) hängt nur von x, h und den internen Zufallsbits von Prover und Verifier ab.

Ein Simulator S ist ein Algorithmus, der nur die zu beweisende Aussage als Eingabe hat und bei der Protokollausführung Prover und (eventuell unehrliche) Verifier simuliert.
Die Zufallsvariable $S(x,h)$ sei für simulierte Protokolle analog zu view(x,h) definiert. $S(x,h)$ hängt nur von den Zufallsbits des Simulators S ab.

Definition 3.4 (Zero-Knowledge-Eigenschaft)
Ein interaktives Beweisprotokoll heißt

- ***perfect zero-knowledge*** *, wenn ein Simulator S existiert, so daß die Zufallsvariablen view(x,h) und S(x,h) identische Verteilungen haben.*
- ***statistical zero-knowledge*** *, wenn ein Simulator S existiert, so daß*

$$\sum_{\alpha} \mid \Pr[M(x,h)=\alpha] - \Pr[view(x,h)=\alpha] \mid < \frac{1}{|x|^c}$$

  für alle $c > 0$ und hinreichend große $\mid x \mid$ gilt.

- ***computational zero-knowledge*** *, wenn ein Simulator S existiert, so daß für alle poly-zeit Algorithmen A, für alle Konstanten $a, d > 0$, für alle hinreichend großen $\mid x \mid$ und alle h mit $\mid h \mid < \mid x \mid^a$ gilt:*

$$\Pr[C_{|x|}(\alpha) = 1 \mid \alpha \text{ zufällig aus } S(x,h)] -$$
$$\Pr[C_{|x|}(\alpha) = 1 \mid \alpha \text{ zufällig aus } view(x,h)] < \frac{1}{|x|^d}$$

Aus „perfect zero-knowledge“ folgt „statistical zero-knowledge“ und daraus wiederum „computational zero-knowledge“.

Anschaulich bedeutet statistical zero-knowledge, daß ein (in seiner Rechenleistung nicht beschränkter) Beobachter die Ausgaben $S(x,h)$ des Simulators mit hoher Wahrscheinlichkeit nicht von den Werten $view(x,h)$, die während einer Protokollausführung mit dem echten Prover anfallen, unterscheiden kann.

Computational zero-knowledge bedeutet anschaulich, daß ein probabilistischer polynomialzeit Beobachter diese Werte nicht unterscheiden kann.

Eine besondere Form von zero-knowledge-Protokollen sind die sogenannten **honest-verifier** zero-knowledge-Protokolle, bei denen angenommen wird, daß der Verifier sich an das Protokoll hält. Dies ist eine Einschränkung gegenüber obiger Definition der zero-knowledge-Eigenschaft, bei der auch Verifier zugelassen waren, die sich nicht an das Protokoll halten. Ein solcher „unehrlicher" Verifier könnte zum Beispiel versuchen, Werte, die laut Protokoll zufällig zu wählen sind, so zu berechnen, daß sie ihm doch weitere Informationen liefern.
In [15] und [24] werden Konstruktionen vorgestellt, um aus einem honest-verifier statistical zero-knowledge Protokoll ein statistical zero-knowledge-Protokoll (bzgl. beliebiger Verifier) zu konstruieren.

Zero-Knowledge im allgemeinen bedeutet, daß ein Beobachter keine zusätzlichen Informationen bekommt, außer der, daß die zu beweisende Aussage wahr ist. Keine zusätzlichen Informationen bedeutet, daß er nichts erfährt, was er nicht schon aus den öffentlichen Daten effizient berechnen könnte.

Eine Vielzahl von Zero-Knowledge-Protokollen hat folgende Form:

A→B: commitment
A←B: challenge
A→B: response

Im ersten Schritt legt sich A auf einen Zufallswert aus einer Menge fest und sendet ein commitment dieses Wertes an B. Dadurch wird eine Zufallsgröße eingebracht, die diese Protokollausführung von bereits durchlaufenen Protokollausführungen unterscheidet. Im zweiten Schritt wählt B aus dieser Menge möglicher Fragen eine aus (challenge) und stellt sie A. B wird im dritten Schritt genau dann akzeptieren, wenn A die richtige Antwort sendet.
Die entsprechenden Mengen (Menge von A's Zufallswerten, Menge von B's challenges und die Menge von A's responses) sind vorweg festgelegt.

Oftmals kann A betrügen, indem er im voraus errät, welche challenge ihm B senden wird. Die Menge der challenges sollte also hinreichend groß sein oder das Protokoll sollte mehrfach durchlaufen werden.

Zero-Knowledge-Protokolle werden oft für Identifikationsverfahren benutzt. Der einzelne Teilnehmer identifiziert sich, indem er zeigt, daß er ein seiner Person zugeordnetes Geheimnis kennt. Sein Gegenüber überprüft die Behauptung anhand einer (öffentlichen) Größe.
So kann sich ein Teilnehmer zum Beispiel dadurch identifizieren, daß er zero-knowledge beweist, den diskreten Logarithmus einer (öffentlichen) Größe zu kennen.
Genau wie bei Public-Key-Verfahren ist auch hier die Authentizität der Zuordnung einer Person zu ihrem öffentlichen Schlüssel wesentlich. Kann diese

Zuordnung nicht eindeutig sichergestellt werden, so sind diese Verfahren nicht sicher (Impersonalisierungsangriffe sind dann möglich). Die Zuordnung einer Person zu einer festen Größe kann (wie bei Public-Key-Verfahren auch) durch Zertifizierung einer vertrauenswürdigen Instanz erfolgen.
Der Vorteil gegenüber den normalen Paßwortverfahren, bei denen ein Benutzer sein Geheimnis einfach nennt, ist offensichtlich. Da der Benutzer sein Geheimnis selbst nicht preisgibt, kann es auch von keinem anderen mißbraucht werden.

3.1.1 Beispiel: Fiat-Shamir Identifikation

Das Fiat-Shamir Protokoll [20] wurde 1986 von A. Fiat und A. Shamir vorgestellt. Es beruht auf der Schwierigkeit Quadratwurzeln modulo einer zusammengesetzten Zahl zu ziehen, deren Faktorisierung nicht bekannt ist. Dieses Problem ist „äquivalent“ zum Faktorisieren von n.

Schlüsselgenerierung

- Eine vertrauenswürdige Instanz generiert ein RSA-Modul $n = pq$ und veröffenlicht n. Die Primfaktoren p und q werden nicht veröffentlicht.
- Jeder Teilnehmer wählt sich nun ein zufälliges $s \in_R \mathbb{Z}_n^*$ als seinen geheimen Schlüssel, berechnet daraus $v = s^2 \, mod \, n$ und veröffentlicht v als seinen öffentlichen Schlüssel.

Das Fiat-Shamir Protokoll
Die folgenden Schritte werden t-mal wiederholt. B akzeptiert genau dann, wenn er in allen t Runden akzeptiert.

1. A wählt ein zufälliges commitment $r \in_R \mathbb{Z}_n^*$ und sendet $x = r^2 \, mod \, n$ als witness an B.
2. B wählt ein zufälliges challenge $b \in_R \{0, 1\}$ und sendet es an A.
3. A sendet $y = rs^b$ als response an B.
4. B verwirft, falls $y = 0$.
   Andernfalls akzeptiert B genau dann, wenn $y^2 = xv^b \, mod \, n$.

Ein betrügerischer A, der den geheimen Schlüssel von A nicht kennt, kann das Protokoll mit einer Wahrscheinlichkeit von (mindestens) $\frac{1}{2}$ bestehen, indem er das Challenge-Bit b von B im voraus rät.
Für $b = 0$ sendet A die Werte $x = r^2$ und $y = r$ und für $b = 1$ die Werte $x = \frac{r^2}{v}$ und $y = r$ an B (mit zufälligem $r \in_R \mathbb{Z}_n^*$).

Andererseits kann er das Protokoll dann auch nur mit einer Wahrscheinlichkeit von höchstens $\frac{1}{2}$ bestehen. Angenommen, er kann das Protokoll mit y_0 für $b = 0$ und mit y_1 für $b = 1$ bestehen, d.h. $y_0^2 = x$ und $y_1^2 = xv$. Daraus folgt, daß $\frac{y_1^2}{y_0^2} = v$ und damit, daß $\frac{y_1}{y_0}$ eine Quadratwurzel von v modulo n ist.

Zero-Knowledge-Eigenschaft
Wir beweisen die Zero-Knowledge-Eigenschaft durch Angabe des Simulators S:

- S wählt zufällig ein Bit $c \in_R \{0,1\}$ und ein zufälliges $r \in_R \mathbb{Z}_n^*$.
  Er berechnet $x = r^2 v^{-c}$ und sendet x an B.
- B antwortet mit einem Bit b.
- Ist $b = c$, so sendet S den Wert $y = r$ an B.
  Ist $b \neq c$, so wird die Simulation dieser Runde erneut gestartet.

Der Simulator muß im Erwartungswert zweimal ausgeführt werden, um eine Runde des Fiat Shamir Protokolls zu simulieren. Um das gesamte Protokoll zu simulieren, muß er bei sequentieller Wiederholung des Protokolls im Erwartungswert also 2t-mal ausgeführt werden.

3.2 Das Random Oracle Modell

Vielfach stellen Hashfunktionen in Protokollen ein Problem bei Sicherheitsbeweisen dar. Um dieses Problem zu umgehen, benutzt man ein Modell, in dem Hashfunktionen wie zufällige Funktionen behandelt werden.
Zufällige Funktionen stellen insofern idealisierte kryptographische Hashfunktionen dar, da es (rechnerisch) unmöglich ist, Kollisionen unter den Funktionswerten zu finden.
Alle an der Protokollausführung beteiligten Personen sowie eventuelle Angreifer haben in diesem Modell Zugriff auf ein Hashorakel, dem sie beliebige Bitstrings als Fragen stellen dürfen und den dazugehörigen Hashwert als Antwort bekommen.
Das Orakel wählt aus der Menge $F_k = \{f : \{0,1\}^* \rightarrow \{0,1\}^k\}$ eine Funktion zufällig, d.h. uniform, aus und berechnet mit dieser Funktion seine Orakelantworten. Damit können die gegebenen Antworten als zufällig angesehen werden. Ferner sind Antworten auf (unterschiedliche) Fragen stochastisch unabhängig, d.h.

$$(\forall m \in \mathbb{N})(\forall x_1, \ldots, x_m \in \{0,1\}^*)(\forall y_1, \ldots, y_m \in \{0,1\}^k)$$

$$(\forall x \neq x_1, \ldots, x_m)(\forall y \in \{0,1\}^k)$$

$$Pr[f(x) = y \mid f(x_1) = y_1, \ldots, f(x_m) = y_m] = Pr[f(x) = y] = \tfrac{1}{2^k}$$

Die Wahrscheinlichkeit läuft in diesem Fall über alle $f \in F_k$.
Werden Beweise der Zero-Knowledge-Eigenschaft im Random Oracle Modell geführt, so wird angenommen, daß der Simulator außer dem Prover auch das Hashorakel unter Kontrolle hat.
Für eine detailiertere Darstelung des Random Oracle Modells sei auf [4] und [10] verwiesen.

3.3 Das generische Modell

Im generischen Modell geht man von Gruppen G aus, die insofern als ideal betrachtet werden können, als die binäre Darstellung von Gruppenelementen wertlos für kryptographische Attacken ist (ideal group assumption).
Man glaubt, daß diese Annahme für die meisten elliptischen Kurven und Untergruppen $G \subset \mathbb{Z}_p^*$ (p prim, Ordnung $G = q$ wobei q prim und $q \mid (p-1)$) gilt.
Im generischen Modell bekommt ein Angreifer deshalb nicht die binäre Darstellung von Gruppenelementen. Elemente der Gruppe G können nur für Gleichheitstests und Gruppenoperationen verwendet werden.
Das generische Modell geht auf Nechaev [32] zurück und wurde von Shoup [40] wieder aufgegriffen. Das hier benutzte Modell ist eine leichte Modifikation des Modells von Shoup. Es orientiert sich stark am generischen Modell, wie es von Schnorr und Jakobsson [39] benutzt wird. Ein Angreifer hat zusätzlich noch Zugang zu einem Entschlüsselungs-Orakel. Je nach Anforderung können andere Erweiterungen des Modells von Shoup vorgenommen werden (z.B. Interaktion mit einem Signaturorakel, siehe [39]).

Folgende Operatioen werden in dieser Arbeit als *generische Schritte* gezählt:

- Gruppenoperationen, d.h. multivariable Exponentationen
  $mex_a : G^d \to G,\ (g_1, \ldots, g_d) \mapsto \prod_i g_i^{a_i}$ mit $a = (a_1, \ldots, a_d) \in \mathbb{Z}_q^d$
- Befragen des Hashorakels H
  (nur wenn gleichzeitig noch das Random Oracle Modell benutzt wird)
- Interaktionen mit einem Entschlüsselungs-Orakel

Ein generischer Angreifer A ist ein deterministischer interaktiver Algorithmus, der Elemente der Gruppe G (*group-data*) nur für generische Gruppenoperationen und Gleichheitstests benutzen darf. Mit Daten, die nicht zur Gruppe G gehören (*non-group-data*), darf A beliebige Rechnungen durchführen. Diese Rechnungen sind kostenlos.
Der Determinismus von A ist keine Einschränkung, da zusätzliche Münzwürfe wertlos wären. A kann sich eine Folge von t optimalen generischen Schritten und eine Folge von optimalen internen Münzwürfen wählen, die seine Erfolgswahrscheinlichkeit maximieren. Der Wahrscheinlichkeitsraum besteht dabei aus den zufälligen Gruppenelementen der Eingabe (z.B. eines Generator g) und der zufälligen Wahl der Hashfunktion H.
Bestehen die Berechnungen von A aus t generischen Schritten, so erhält er $t' \leq t$ Gruppenelemente $f_1, \ldots, f_{t'}$ und non-group-data (enthält keine Gruppenelemente) als Ergebnis. Die Folge $f_1, \ldots, f_{t'}$ beginnt mit den Gruppenelementen, die A als Eingabe bekommt. Gruppenelemente in der Eingabe zählen also auch zu den generischen Schritten. Existieren $i \neq j$ mit $f_i = f_j$, so nennen wir das eine Kollision von Gruppenelementen.

Kapitel 4

Verifiable Encryption

Ein Verifiable Encryption Protokoll (VEP) ist ein interaktives Zwei-Parteien Protokoll zwischen einem Prover P und einem Verifier V.
Als gemeinsame Eingabe haben sie den Schlüssel PK eines asymmetrischen Verschlüsselungsverfahrens E, einen Wert d und eine Relation R. Nach Ausführung des Protokolls wird V entweder abbrechen oder akzeptieren und die Verschlüsselung $E_{PK}(s)$ eines Wert s unter PK erhalten, der $(s, d) \in R$ erfüllt.
Das Protokoll muß sicherstellen, daß V Verschlüsselungen von „falschen“ S nur mit vernachlässigbar kleiner Wahrscheinlichkeit akzeptiert. Ferner sollte der Verifier durch die Protokollausführung nichts lernen außer der Verschlüsselung von s und deren Korrektheit.
Typischerweise gehört der Schlüssel PK einer vertrauenswürdigen dritten Partei (Trusted Third Party, TTP), die am Verifiable Encryption Protokoll jedoch nicht teilnimmt. Ist das VEP selbst Teil einer größeren kryptographischen Anwendung, so ist die TTP in Konfliktfällen (eine Partei versucht zu betrügen oder es kommt zu einem technischen Problem) in der Lage, den Wert $E_{PK}(s)$ zu entschlüsseln und den Wert s zu rekonstruieren.

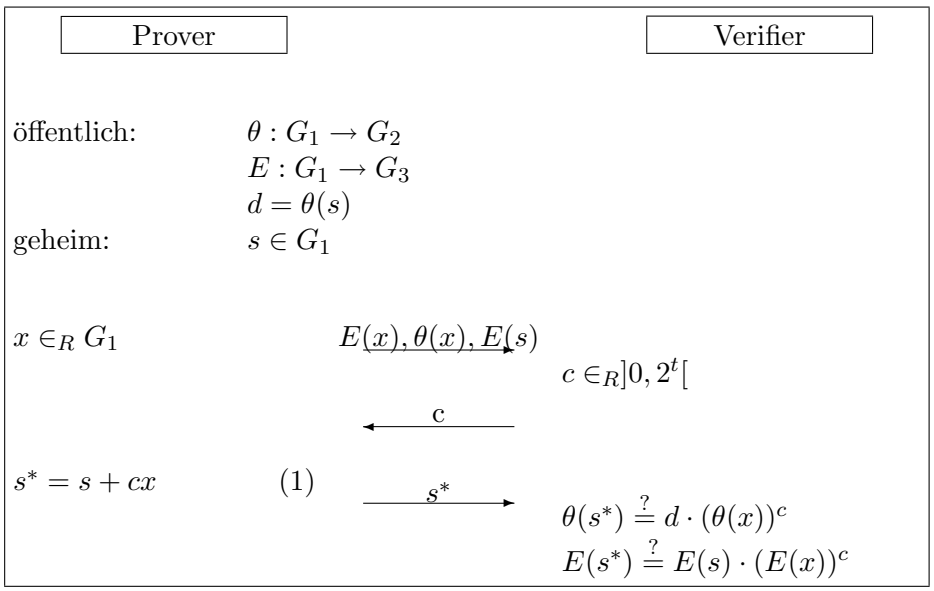
Ein solches VEP kann zum Beispiel für den fairen Tausch digitaler Signaturen benutzt werden. Es gelte genau dann $(s, d) \in R$, wenn s eine gültige Signatur von d ist. Anstatt ihre Signaturen direkt auszutauschen, tauschen sie zunächst deren Verifiable Encryptions aus. Danach können sie die Signaturen im Klartext tauschen, da jeder Teilnehmer durch Einschalten der TTP die gewünschte Signatur rekonstruieren kann. Damit das Protokoll auch wirklich fair ist, d.h. A bekommt die Signatur von B genau dann, wenn B auch die Signatur von A bekommt, darf die TTP nicht jede Anfrage auf Entschlüsselung beantworten (siehe auch Kapitel 5).
Asokan, Shoup und Waidner stellen in [1] ein ähnliches Protokoll für den fairen Tausch digitaler Signaturen vor. Allerdings wird dort in einem ersten Schritt das Problem des fairen Tauschs digitaler Signaturen auf das Problem des Tauschs von Urbildern eines Homomorphismus θ bei bekannten Bildern reduziert. Das VEP arbeitet dann auf der Relation $(s \in \theta^{-1}(d), d)$ bei öffenlich bekanntem θ und d (siehe auch Kapitel 5).

VEPs können auch zur Realisierung von „Escrow-Techniken“ genutzt werden. Bei Escrow-Techniken möchte man, daß die Teilnehmer bestimmte Werte

(z.B. ihre Identität) hinterlegen. Diese Werte sollen so hinterlegt werden, daß sie für andere Benutzer zunächst nicht sichtbar sind. In Bedarfsfällen (etwa bei Betrugsversuchen dieses Teilnehmers) soll eine Aufdeckung dieses Wertes unter Mitwirkung einer vertrauenswürdigen Partei jedoch möglich sein. Natürlich muß auch hier überprüfbar sein, ob tatsächlich die gewünschten Werte hinterlegt wurden. Kilian und Patrank stellen in [28] eine Lösung für „Identity Escrow“ vor, die auf einem VEP basiert.

Diese Diplomarbeit baut auf [1] auf und stellt ein neues VEP vor, das eine effiziente Alternative zu dem in [1] vorgestellten VEP darstellt.
In unserem Fall sind s und d (wie in [1]) Urbild und Bild eines effizient zu berechnenden aber (rechnerisch) nicht invertierbaren Gruppenhomomorphismus $\theta : G_1 \rightarrow G_2$. Ein Beispiel für eine solche Abbildung θ, das uns im folgenden noch begegnen wird, ist die diskrete Exponentiation $x \mapsto g^x \, mod \, p$.
Für die Verschlüsselungsfunktion E fordern wir, daß sie über homomorphe Verknüpfungseigenschaften verfügt. Gleichzeitig muß sie aber auch sicher gegen adaptive-chosen-ciphertext-Angriffe sein, da die TTP als ein Entschlüsselungsorakel angesehen werden kann. Für θ gibt es kein solches Orakel (insbesondere erfüllt die TTP diese Eigenschaft nicht).

Die Grundidee des neuen Verifiable Encryption Protokolls stellt sich (schematisch) folgendermaßen dar:



Bei Gleichung (1) wird vorausgesetzt, daß es sich bei der Urbildgruppe G_1 um eine additive Gruppe handelt. Für multiplikative Urbildgruppen ist diese Gleichung in $s^* = s \cdot x^c$ abzuändern.
Bei G_2 handelt es sich bei den hier betrachteten Fällen immer um eine multiplikative Gruppe.

Der Verifier akzeptiert genau dann, wenn die beiden Gleichungen, die er gegen Ende des Protokolls prüft, erfüllt sind.

Durch die erste Gleichung überzeugt sich der Verifier, daß der Prover das Geheimnis s wirklich kennt und daß der Wert s^* richtig berechnet wurde.
Durch Überprüfung der zweiten Gleichung überzeugt sich der Verifier, daß in $E(s)$ auch wirklich ein Urbild von d verschlüsselt wurde.

Beide Gleichungen überprüfen die Gültigkeit von $s^* = s + cx$, allerdings in zwei verschiedenen Gruppen. Im ersten Fall wird die Gültigkeit der Gleichung in der Bildgruppe G_2 überprüft. Im zweiten Fall im Raum aller möglichen Ciphertexte der Verschlüsselungsfunktion E.
Damit diese Überprüfung in beiden Gruppen vorgenommen werden kann, müssen die Parameter der Verschlüsselungsfunktion E sorgfältig auf die Gruppe G_2 abgestimmt werden. Die Parameter von E gehören aber zum Schlüsselpaar der TTP und müssen deshalb über längere Zeit als fest angesehen werden. Insbesondere kann unser Protokoll also nicht mit wechselnden Gruppen G_2 arbeiten.
Benutzt man dieses VEP im Fair Exchange Protokoll von [1], so wird sich zeigen, daß der Homomorphismus θ und dessen Bildgruppe G_2 beim Tausch von RSA-Signaturen vom benutzten Modul n und somit vom öffentlichen Schlüssel eines speziellen Benutzers abhängen. Das hier vorgestellte VEP eignet sich also nicht zum Tausch von RSA-Signaturen nach [1].
Im Fall von diskreten Logarithmus Unterschriftenverfahren können mehrere Benutzer dieselben Systemparameter benutzen (siehe Kapitel 2 über digitale Signaturen). Folgt man [1], so können alle Benutzer, die mit den gleichen Systemparametern arbeiten, auch mit denselben Homomorphismen θ und Gruppen G_2 arbeiten. Das vorliegende VEP eignet sich damit zum Tausch aller diskreten Logarithmus Unterschriften

Frühere Arbeiten über VEP

Asokan, Shoup und Waidner stellen in [1] ein VEP vor, das auf der Cut-and-Choose Technik basiert. Das dort vorgestellte Protokoll arbeitet mit surjektiven Gruppenhomomorphismen $\theta : G_1 \to G_2$, an die man nur die Einschränkung macht, daß sie effizient zu berechnen aber (rechnerisch) nicht invertierbar sind. Es ist damit generisch, jedoch nicht allzu effizient.
Beiden Teilnehmern ist ein Wert $d \in G_2$ und dem Prover ein Wert $s \in G_1$ mit $\theta(s) = d$ als geheime Eingabe bekannt. In jeder Runde des Protokolls verschlüsselt der Prover einen Zufallswert $\hat{s} \in G_1$ und sendet diese Verschlüsselung und den Wert $\theta(\hat{s})$ an den Verifier. Abhängig vom Challenge-Bit des Verifiers muß der Prover im nächsten Schritt entweder den Zufallswert $\hat{s}$ offenlegen oder ein s^* senden, das die Gleichung $s^* = \hat{s} + s$ erfüllt. Im ersten Fall überprüft der Verifier, ob der Prover für die Berechnung der von ihm gesendeten Werte dasselbe $\hat{s}$ benutzt hat. Im zweiten Fall überzeugt sich der Verifier, ob $\theta(s^*) = d + \theta(\hat{s}) = \theta(s + \hat{s})$ erfüllt ist und damit, ob der Prover den Wert s kennt.
Ein betrügerischer Prover kann das Protokoll genau dann bestehen, wenn er das Challenge-Bit des Verifiers im voraus raten kann. Um die Betrugswahrscheinlichkeit des Provers auf 2^{-k} zu drücken, muß das Protokoll also k mal (parallel) ausgeführt werden.

Es wird bewiesen, daß das vorgestellte VEP sicher im Random Oracle Modell ist.

In [9] stellen Camenisch und Damgard eine Verbesserung des Protokolls aus [1] vor. Es wird vorausgesetzt, daß der geheime Wert s und der öffentliche Wert d in einer (öffentlichen) Relation R enthalten sind, d.h. $(s, d) \in R$. Von dieser Relation wird gefordert, daß ein 3-Schritt (Commitment, Challenge, Response) proof of knowledge für sie existiert.
Unter diesen Voraussetzungen kann ein VEP für diese Relation konstruiert werden, das nur mit $O(\log k)$ (k ist der Sicherheitsparameter) Verschlüsselungen auskommt (im Vergleich zu $O(k)$ bei [1]). Ferner kann die Sicherheit dieses VEP ohne Benutzung des Random Oracle Modells bewiesen werden.
Durch das Bilden von Paaren der Form $(d,\ s \in \theta^{-1}(d))$ (für spezielle Gruppenhomomorphismen θ) kann man eine Relation für die Homomorphismen aus [1] bilden.
Die Relation aus [1] enthält Paare der Form $(x,\ \theta^{-1}(x))$ für spezielle Gruppenhomomorphismen θ.

Ebenfalls mit einer Off-line TTP arbeiten Bao, Deng und Mao in [3]. Sie nennen ihr VEP *Certificate of Encrypted Message Being a Signature (CEMBS)*. Als Verschlüsselungsfunktion benutzen sie ElGamal. Die Autoren stellen zwei Versionen ihres CEMBS vor.
Eines arbeitet mit DSA-ähnlichen Signaturschemata. Es benutzt einen von Stadler in [41] vorgestellten nicht-interaktiven Beweis für die Gleichheit eines diskreten Logarithmus (von $y = x^a$) und des diskreten Logarithmus eines diskreten Logarithmus (von $Y = X^{z^a}$) (Proof of Equivalence of Discrete Logarithm to Discrete LogLogarithm, PEDLDLL).
Das andere CEMBS arbeitet mit Guillou-Quisquater Signaturen [26] und basiert auf einem Beweis für die Gleichheit zweier diskreter Logarithmen (Proof of Equivalence of Discrete Logarithms, PEDL).

Feng Bao stellt in [2] ein nicht-interaktives Verifiable Encryption Protokoll für diskrete Logarithmen vor.
Er benutzt das asymmetrische Verschlüsselungsverfahren von Okamoto und Uchiyama [34] in der nicht-randomisierten Version (siehe Kapitel 4.3) als trapdoor one-way Funktion. Die Verschlüsselung besteht also einfach aus einer diskreten Exponentialfunktion $g \mapsto g^x \, mod \, n$ für einen Modul $n = p^2 q$ (p und q prim). Die Sicherheit der Verschlüsselungsfunktion beruht auf der Schwierigkeit, den Modul n zu faktorisieren.
Beiden Teilnehmern ist ein öffentlicher Schlüssel PK für das Okamoto/Uchiyama-Verfahren und ein Wert d bekannt. Der Prover kennt ferner s, einen diskreten Logarithmus von d.
Der Prover verschlüsselt s anschließend mit dem öffentlichen Schlüssel PK. Dieses Chiffrat ($E_{PK}(s) = g^s \, mod \, n$) und der Wert d sind beides Bilder von s unter einer diskreten Exponentialfunktion. Der Prover sendet nun das Chiffrat von s zusammmen mit einem nicht-interaktiven Zero-Knowledge Beweis für die Gleichheit der diskreten Logarithmen von d und $E_{PK}(s)$ an den Verifier. Ak-

zeptiert dieser, so ist er davon überzeugt, daß in $E_{PK}(s)$ der gewünschte Wert s verschlüsselt wurde.
Bao zeigt, daß sein Protokoll im Random Oracle Modell sicher ist, sofern der Modul $n = p^2q$ nicht faktorisiert werden kann.

Markus Stadler benutzt in [41] VEPs zur Konstruktion von *Publicly Verifiable Secret Sharing Schemes*. Er stellt ein VEP für diskrete Logarithmen und ein VEP für e-te Wurzeln (modulo n) vor. In beiden Fällen wird die ElGamal-Verschlüsselung benutzt.
VEP für diskrete Logarithmen: Gegeben sei ein öffentliches Element $V = g^v$ einer Gruppe G sowie ein öffentlicher ElGamal-Schlüssel $y = g^z \, mod \, p$.
Der Prover verschlüsselt v in $(A, B) = (h^\alpha \, mod \, p, \, v^{-1}y^\alpha \, mod \, p)$. Anschließend beweist er dem Verifier, daß $\alpha = \log_h A = \log_y(\log_g V^B)$.
VEP für e-te Wurzeln: Gegeben sei ein öffentlicher Wert $V = v^e \, mod \, n$. Die Parameter des ElGamal-Schemas müssen nun so angepaßt werden, daß sie auch auf $\mathbb{Z}_n$ anstatt wie üblich auf $\mathbb{Z}_p$ arbeiten. Sei $y = g^z \, mod \, n$ der öffentliche ElGamal-Schlüssel.
Der Prover sendet eine ElGamal-Verschlüsselung von v, d.h.
$(A, B) = (g^\alpha \, mod \, n, \, my^\alpha \, mod \, n)$, an den Verifier und beweist, daß $B^e y^{-\alpha e} = V$.
Die Schwäche des VEP für e-te Wurzeln besteht darin, daß für jeden Modul n ein anderes ElGamal-Schlüsselpaar benutzt werden muß. Es ist daher nicht möglich, **ein** Schlüsselpaar (der TTP) über längere Zeit beizubehalten.

In [28] stellen Kilian und Petrank eine Lösung für „Identity Escrow“ vor, die auf einem VEP basiert. Beim Identity Escrow soll der Benutzer seine Identität in verschleierter Form hinterlassen, die unter Zutun einer dritten Partei aufgedeckt werden kann. Ferner soll jeder Teilnehmer überprüfen können, ob tatsächlich die richtige Identität hinterlassen wurde.
Anwendungen für solche Techniken finden sich zum Beispiel bei elektronischen Bezahlsystemen, in denen Benutzer so lange anonym agieren können, bis eine Unregelmäßigkeit auftaucht. Die dritte Partei (meist die Bank) deckt dann die Identität des Betrügers auf und zieht ihn zur Verantwortung.
Das VEP hat in diesem Fall die Aufgabe, die Hinterlegung der verschlüsselten Identität eines Benutzers überprüfbar zu machen.

4.1 ElGamal-Verschlüsselung

4.1.1 Das Verschlüsselungsschema von ElGamal

Das folgende Verschlüsselungsverfahren geht auf Taher ElGamal zurück und wurde in [18] vorgestellt.
Die (öffentlich bekannten) Systemparameter bestehen aus einer großen Primzahl p und einem Generator g von $\mathbb{Z}_p^*$. Die Primzahl p sollte nach heutigem Stand der Technik mindestens 1024 (Binär-)Stellen haben.
Eine Möglichkeit ist es, p und g als Systemparameter vorzugeben, d.h. alle Benutzer rechnen mit diesen Parametern. Es wäre aber auch möglich, daß sich jeder Benutzer seine eigene Primzahl p und seinen eigenen Generator g wählt und diese als Teil seines öffentlichen Schlüssels veröffentlicht. Ersteres hat den Vorteil, daß die Exponentiationen durch Vorberechnungen (precomputations) schneller berechnet werden können. Andererseits existiert mit dem Index-Calculus-Algorithmus zur Berechnung des diskreten Logarithmus ein Verfahren, bei dem ein großer Teil der Berechnungen im Vorfeld ausgeführt und in einer Datenbank abgelegt werden können. Da für den Fall, daß p als Systemparameter genutzt wird, auch die Sicherheitsanforderungen an p größer sind, sollte ein größerer Modul p gewählt werden.

Schlüsselerzeugung
Jeder Teilnehmer wählt sich zufällig eine Zahl $x \in_R \mathbb{Z}_p^*$ als privaten Schlüssel. Der zugehörige öffentliche Schlüssel berechnet sich als $y = g^x \, mod \, p$. Wegen der Schwierigkeit des diskreten Logarithmus Problems ist es (rechnerisch) unmöglich, vom öffentlichen auf den privaten Schlüssel zu schließen.

Verschlüsselung
Nachrichten $m \in \mathbb{Z}_p^*$ können dann wie folgt verschlüsselt werden:

- Wähle $r \in_R \mathbb{Z}_p^*$
- Berechne $\gamma = g^r \, mod \, p$
- Berechne $\delta = my^r \, mod \, p$

Das Paar $c = (\gamma, \delta)$ bildet nun den Ciphertext der Nachricht m.

Zur **Entschlüsselung** eines solchen Ciphertextes macht man folgendes:

- Berechne $\gamma^{-x} \, mod \, p$
- Die Nachricht ergibt sich dann als $m = (\gamma^{-x})\delta \, mod \, p$

Verknüpfungseigenschaften

Für $m_1, m_2 \in \mathbb{Z}_p^*$ erhält man eine nützliche Verknüpfungseigenschaft:

$$\begin{aligned} E(r_1, m_1) \cdot E(r_2, m_2) &= (g^{r_1}, m_1 y^{r_1}) \cdot (g^{r_2}, m_2 y^{r_2}) \\ &= (g^{r_1+r_2}, m_1 m_2 y^{r_1+r_2}) \\ &= E(r_1 + r_2, m_1 m_2) \end{aligned}$$

Aus dieser Eigenschaft läßt sich direkt eine zweite Verknüpfungseigenschaft ableiten:

$$E(r, m)^c = (g^r, my^r)^c = (g^{rc}, m^c y^{rc}) = E(rc, m^c)$$

Bemerkung 4.1 (Effizienz)
Zur Verschlüsselung einer Nachricht sind im wesentlichen zwei modulare Exponentiationen durchzuführen. In bestimmten Fällen können diese aber schon im Vorfeld berechnet werden und brauchen nicht online bei der eigentlichen Verschlüsselung berechnet zu werden.
Die ElGamal-Verschlüsselung hat eine Nachrichtenexpansion von 2, d.h. der ciphertext ist doppelt so lang wie die Nachricht, die verschlüsselt wurde.

Die Sicherheit der ElGamal-Verschlüsselung hängt eng mit der Schwierigkeit des Diffie-Hellman Problems zusammmen.

Definition 4.2 (Diffie-Hellman Problem, DHP)
Seien eine Primzahl p, ein Generator g von $\mathbb{Z}_p^*$ sowie zwei Werte $g^a \bmod p$ und $g^b \bmod p$ gegeben. Finde $g^{ab} \bmod p$.

Bemerkung 4.3 (Sicherheit)

1. *Das Brechen der ElGamal-Verschlüsselung, d.h. das Finden von m, wenn p, g, $y = g^x$, γ und δ gegeben sind, ist äquivalent zum Lösen des Diffie-Hellman-Problems:*

   Sei $E(r, m) = (\delta, \gamma) = (g^r, my^r)$ eine ElGamal-Verschlüsselung von m bzgl. des öffentlichen Schlüssels $y = g^x$ und des Zufallswertes r.
   Angenommen, wir können DHP lösen, dann liefern uns δ und y einen Wert $g^{xr} = y^r$ mit dessen Hilfe wir die Nachricht $m = \gamma(g^{-xr})$ rekonstruieren können.
   Sei andererseits (g, g^a, g^b) eine Instanz von DHP.

Wir konstruieren uns mit $k \in_R \mathbb{Z}_p^*$ eine ElGamal-Verschlüsselung (g^b, k) einer (uns unbekannten) Nachricht $m \in \mathbb{Z}_p^*$ bzgl. des öffentlichen Schlüssels g^a und des (uns unbekannten) Zufallswertes b. Durch die verschlüsselte Nachricht m können wir $g^{ab} = km^{-1}$ rekonstruieren.

2. *Genau wie bei der ElGamal-Signatur (siehe Kapitel 2) ist es kritisch, mehrere ElGamal-Verschlüsselungen mit demselben Zufallswert r zu berechnen:*

   Seien $E(m_1, r) = (\delta_1, \gamma_1)$ und $E(m_2, r) = (\delta_2, \gamma_2)$ ElGamal-Verschlüsselungen von m_1 bzw. m_2 bzgl. des gleichen Zufallswertes r.
   Dann kann ohne Kenntnis des geheimen Schlüssels $\delta_1/\delta_2 = m_1/m_2$ berechnet werden. Aus der Kenntnis eines Klartextes würde automatisch auch die Kenntnis des anderen Klartextes folgen.

3. *Die ElGamal-Verschlüsselung ist nicht sicher gegen adaptive-chosen-ciphertext-Angriffe (CCA):*

   Seien $E(m_1) = (\gamma_1, \delta_1)$ eine ElGamal-Verschlüsselung von m_1 und $E(m_2) = (\gamma_2, \delta_2)$ eine ElGamal-Verschlüsselung von m_2.
   Dann ist $(\gamma_1\gamma_2, \delta_1\delta_2)$ eine gültige Verschlüsselung von m_1m_2.

Bemerkung 4.4
Für die ElGamal-Verschlüsselung wird typischerweise die Gruppe $\mathbb{Z}_p^*$ verwendet. Man kann jedoch auch beliebige andere endliche, zyklische Gruppen G benutzen. Für die Sicherheit der ElGamal-Verschlüsselung auf G ist es wichtig, daß das Diskreter Logarithmus Problem in G schwer ist. Man nimmt an, daß dies für Untergruppen der Ordnung q von $\mathbb{Z}_p^*$ (mit $q \mid p-1$) zutrifft.
In G stellt sich die ElGamal-Verschlüsselung wie folgt dar:
Sei n die Ordnung der Gruppe G und g ein Generator von G. Jeder Teilnehmer wählt ein zufälliges x mit $1 \leq x \leq n-1$ als geheimen Schlüssel. Der zugehörige öffentliche Schlüssel ergibt sich als g^x.
Die Verschlüsselung einer Nachricht $m \in G$ bzgl. des Schlüsselpaares (x, g^x) und dem Zufallswert r ($1 \leq r \leq n-1$) ist das Paar $c = (\delta, \gamma) = (g^r, m \cdot (g^x)^r)$.
Der Empfänger erhält die Nachricht m, indem er $m = (\delta^{-x}) \cdot \gamma$ berechnet.

Für unsere Zwecke wäre ein Verschlüsselungsverfahren wünschenswert, das sowohl die in der letzten Bemerkung erwähnten multiplikativen Eigenschaften hat als auch sicher gegen adaptive-chosen-ciphertext-Angriffe ist. Das eben beschriebene ElGamal-Verfahren kann mit einem kleinen Trick so verbessert werden, daß es diesen Ansprüchen genügt.

4.1.2 Die signierte ElGamal-Verschlüsselung

Um das ElGamal-Schema sicher gegen adaptive-chosen-ciphertext-Angriffe zu machen, werden die Ciphertexte um eine Signatur erweitert. Da in unserem Fall die am Signaturtausch beteiligten Personen (zumindest gegenüber der TTP) anonym bleiben sollen, kann man für diese Signatur nicht das Schlüsselpaar

einer dieser Personen verwenden. Stattdessen wird das Paar (r, g^r) aus dem Verschlüsselungsschema als Schlüsselpaar verwendet.
Dieses Schema wurde unabhängig voneinander von Tsiounis und Yung [43] und Jakobsson [27] vorgeschlagen. In [39] zeigen Schnorr und Jakobsson, daß das neue Schema semantisch sicher gegen adaptive-chosen-ciphertext-Angriffe ist (im generischen und Random Oracle Modell).
Die neue Verschlüsselungsfunktion E sieht wie folgt aus.

Verschlüsselung einer Nachricht $m \in \mathbb{Z}_p^*$

Schlüsselpaar: $x \in_R \mathbb{Z}_p^*,\ y = g^x \bmod p$
Hashfunktion: $H : \{0,1\}^* \to \mathbb{Z}_p^*$

- wähle $r, s \in_R \mathbb{Z}_p^*$
- Berechne $g^r \bmod p$, $g^s \bmod p$ und $f = my^r \bmod p$
- $c = H(g^s, g^r, my^r)$
- $z = s + cr \bmod p$

Das Tupel $C = (g^r, my^r, c, z)$ ist die Verschlüsselung der Nachricht m bzgl. der Zufallswerte r und s unter dem (öffentlichen) Schlüssel y.

Die ersten beiden Komponenten des Ciphertextes stellen eine ElGamal-Verschlüsselung der Nachricht m unter dem geheimen Schlüssel x dar. Die letzten beiden Komponenten sind eine Schnorr-Signatur der ersten beiden Komponenten unter dem geheimen Schlüssel r.
Der Ciphertext ist nur dann gültig, wenn die enthaltene Schnorr-Signatur gültig ist. Der Empfänger einer Nachricht prüft die Signatur, bevor er die eigentliche Nachricht entschlüsselt.

Bemerkung 4.5
Analog zur „normalen“ kann die signierte ElGamal-Verschlüsselung auch auf beliebigen endlichen, zyklischen Gruppen G anstatt $\mathbb{Z}_p^*$ gebildet werden. In G muß das Diskreter Logarithmus Problem wieder schwer sein.

Bezeichnungen

Sei eine beliebige Nachricht $m \in \mathbb{Z}_p^*$ gegeben. Im weiteren bezeichne

$$E(r, m) = (g^r \bmod p, my^r \bmod p)$$

eine („normale“) ElGamal-Verschlüsselung von m bzgl. des Zufallswertes r und

$$E^*(r, m, s) = (g^r, my^r, c, z)$$

eine signierte ElGamal-Verschlüsselung von m bzgl. der Zufallswerte r und s.

4.2 Verifiable Encryption für multiplikative Gruppen

Die signierte ElGamal-Verschlüsselung, die zum einen über homomorphe Verknüpfungseigenschaften verfügt, gleichzeitig aber sicher gegen adaptive-chosen-ciphertext-Angriffe ist, nutzen wir jetzt, um ein Verifiable Encryption Protokoll für multiplikative Urbildgruppen zu konstruieren.

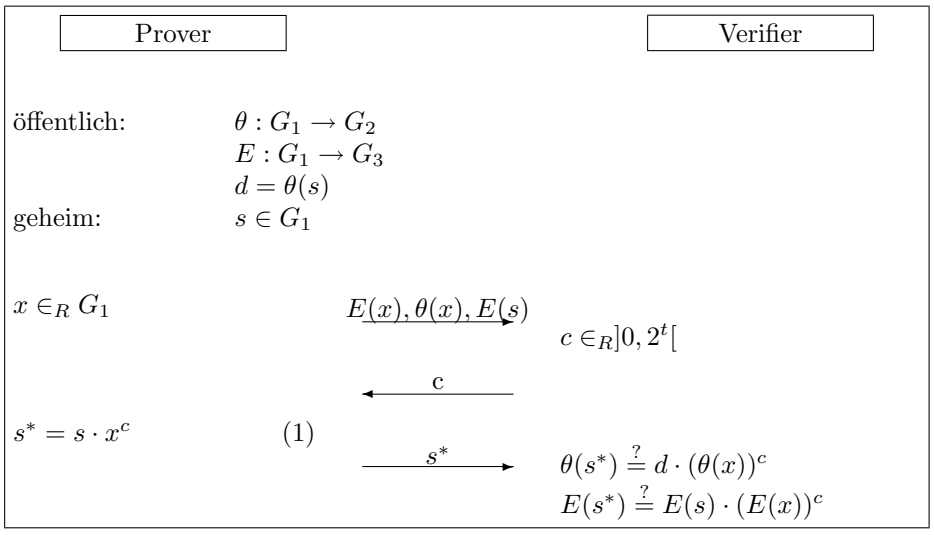
Sei $\theta : G_1 \to G_2$ ein surjektiver Gruppenhomomorphismus, dessen Urbildgruppe G_1 multiplikativ ist, d.h. θ habe die Eigenschaft

$$\theta(x_1 \cdot x_2) = \theta(x_1) \cdot \theta(x_2) \text{ für alle } x_1, x_2 \in G_1$$

Wir nehmen ferner an, daß G_1 eine (multiplikative) zyklische Gruppe der Ordnung $p-1$ (mit p prim) ist, d.h. G_1 ist isomorph zu $\mathbb{Z}_p^*$. Der Einfachheit halber nehmen wir an, daß $G_1 = \mathbb{Z}_p^*$.

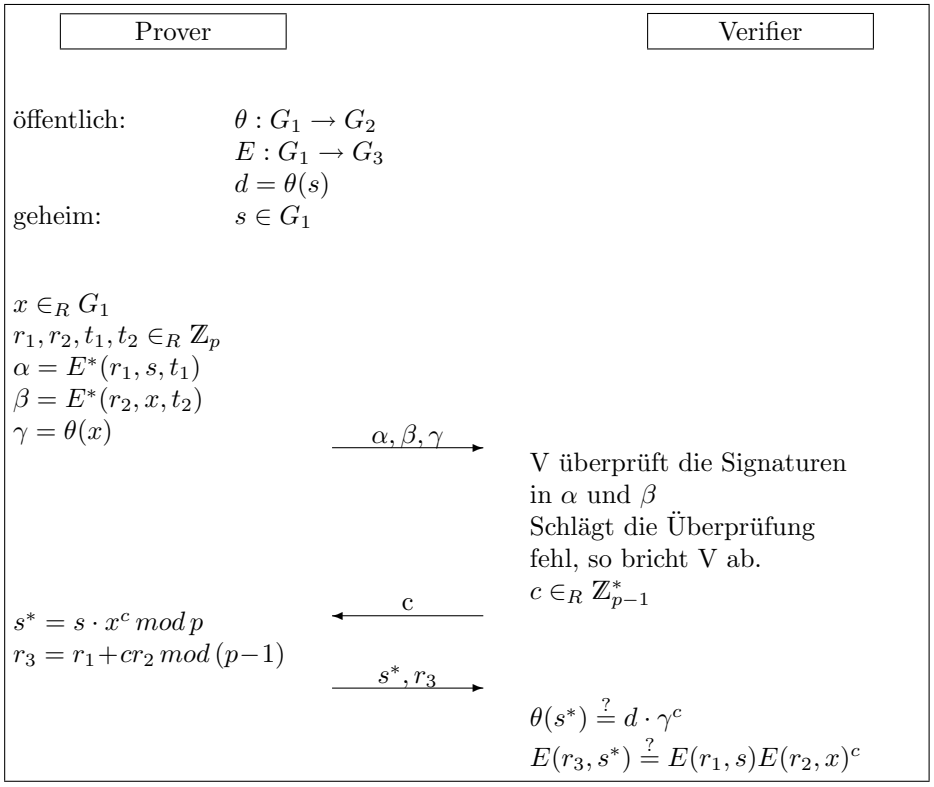
Es muß darauf geachtet werden, daß die Parameter des ElGamal-Schemas so gewählt wurden, daß es auf der Nachrichtenmenge $\mathbb{Z}_p^*$ (bzgl. derselben Primzahl p) operiert. Grund dafür ist die Tatsache, daß Gleichung (1), die in unserem Fall modulo p berechnet wird, in zwei Gruppen erfüllt sein muß. Sie gilt in der Bildgruppe G_2, da G_1 isomorph zu $\mathbb{Z}_p^*$ und θ ein surjektiver Gruppenhomomorphismus ist. Ferner darf die Gültigkeit dieser Gleichung auch durch die ElGamal-Verschlüsselung nicht verlorengehen.

Es sei noch einmal an unser schematisches Protokoll erinnert:



Verwendet man nun als Verschlüsselungsfunktion E die (signierte) ElGamal-Verschlüsselung, so ergibt sich mit den Notationen aus dem letzten Abschnitt folgendes Protokoll:

VEP für multiplikative Gruppen



Bemerkung 4.6

1. *Die Werte $E(r_1, s)$ und $E(r_2, x)$ erhält der Verifier, indem er von α und β die letzten beiden Komponenten abschneidet.*

2. *Die Wahl der Challenge $c \in_R \mathbb{Z}^*_{p-1}$ stellt sicher, daß die Abbildung $G_1 \to G_1, \quad x \mapsto x^c \bmod p$ bijektiv ist.*
   Der Prover sollte bei Erhalt von c prüfen, ob wirklich $c \in \mathbb{Z}^*_{p-1}$ gilt.

Satz 4.7 (Completeness)
Wenn P und V ehrlich sind, dann wird V für alle s, d mit $\theta(s) = d$ akzeptieren.

Beweis:
Sei y der öffentliche Schlüssel der TTP.
Der Verifier überprüft zwei Dinge:

- $\theta(s^*) = \theta(s * x^c) = \theta(s) * \theta(x)^c = d * \gamma^c$

- $E(r_3, s^*) = E(r_1 + cr_2, sx^c) = (g^{r_1+cr_2}, sx^c y^{r_1+cr_2})$

$$= (g^{r_1}, sy^{r_1})(g^{r_2}, xy^{r_2})^c = E(r_1, s)E(r_2, x)^c$$

□

Satz 4.8 (Soundness)
Sei θ ein Einweg-Homomorphismus und $mu(\theta) = \frac{|G_1|}{|\theta(G_1)|}$ dessen Multiplizität. Ferner sei E sicher gegen adaptive-chosen-ciphertext-Angriffe.
Angenommen der Prover besteht mit Wahrscheinlichkeit $> \frac{mu(\theta)}{p-1}$. Dann ist α eine Verschlüsselung eines Urbildes von d (unter dem Schlüssel der TTP).

Beweis:
Sei $\gamma = \theta(x)$ und $d = \theta(s)$.
Angenommen, in α und β wurden andere Werte als s und x (nämlich $\hat{s}$ und $\hat{x}$) verschlüsselt.
Der Prover besteht das Protokoll, d.h. es gilt
$E(r_3, s^*) = E(r_1, \hat{s}) \cdot E(r_2, \hat{x})^c$
$\Leftrightarrow g^{r_3} = g^{r_1+cr_2}$ *(1)*
und $s^* y^{r_3} = \hat{s} y^{r_1} \hat{x}^c y^{cr_2}$ *(2)*
(Dabei bezeichnet y den öffentlichen Schlüssel der TTP.)

(1) $\Leftrightarrow r_3 = r_1 + cr_2 \,(mod\,(ord\, g))$
$\Rightarrow y^{r_3} = y^{r_1+cr_2} \,mod\, p$, *da* $y = g^{SK_{TTP}} \,mod\, p$
Damit folgt aus Gleichung (2):
$s^* = \hat{s} \cdot \hat{x}^c$
Ferner gilt (wird vom Verifier überprüft):
$\theta(s^*) = d\gamma^c = \theta(s)\theta(x)^c = \theta(sx^c)$
$\Rightarrow \theta(\hat{s}\hat{x}^c) = \theta(sx^c)$

Um zu betrügen, muß der Prover also Werte $\hat{s}$ und $\hat{x}$ finden, die diese Gleichung erfüllen, bevor er den Wert c kennt.
Hat sich der Prover auf Werte $\hat{s}$ und $\hat{x}$ festgelegt, so gibt es genau ein $c \in \mathbb{Z}_p^*$, so daß die Gleichung $\hat{s}\hat{x}^c = k$ (für ein festes $k \in G_1$) erfüllt ist (die Abbildung $G_1 \to G_1, x \mapsto x^c$ ist bijektiv).
Die Wahrscheinlichkeit, daß für fest gewählte $\hat{s}$ und $\hat{x}$ die Gleichung $\theta(\hat{s}\hat{x}^c) = \theta(sx^c)$ erfüllt ist, beträgt also $\frac{mu(\theta)}{p-1}$.
Im Regelfall ist θ ein Isomorphismus (das muß sogar gefordert werden, wenn Geheimnisse anstatt Signaturen ausgetauscht werden sollen). In diesem Fall beträgt die Betrugswahrscheinlichekit also $\frac{1}{p-1}$.
□

Bemerkung 4.9 *Für isomorphe θ sagt dieser Satz, daß in α wirklich das gewünschte Geheimnis s verschlüsselt wurde.*

Satz 4.10 (Zero-Knowledge)
Sei θ ein Einweg-Homomorphismus und E sicher gegen adaptive-chosen-ciphertext-Angriffe. Dann ist das vorliegende Protokoll honest-verifier computational zero-knowledge (im Random Oracle Modell).

Beweis:
Zu zeigen ist, daß ein Simulator S existiert, der Ausgaben erzeugt, die (rechnerisch) nicht von denen des richtigen Protokolls unterschieden werden können. Angabe eines Simulators S:

- *Wähle zufällige $\hat{x}, s^* \in_R G_1$, $r_1, r_2, t_1, t_2 \in_R \mathbb{Z}_p$ und $c \in_R \mathbb{Z}^*_{p-1}$*
- $\gamma = (\theta(s^*) \cdot \theta(s)^{-1})^{\hat{c}}$, *wobei* $\hat{c} = c^{-1} \bmod (p-1)$
- $\hat{s} = s^* \hat{x}^{-c}$
- $\alpha = E^*(r_1, \hat{s}, t_1)$
- $\beta = E^*(r_2, \hat{x}, t_2)$
- $r_3 = r_1 + cr_2$

Schnorr-Signaturen sind für alle Nachrichten m und alle Schlüsselpaare (x, g^x) gleichverteilt (im Random Oracle Modell). Die an die Ciphertexte angehängten Signaturen tragen also nichts zur Unterscheidung von echten und simulierten Protokollausführungen bei.

Angenommen, es existiert ein Algorithmus A, der simulierte von echten Protokollausführungen unterscheiden kann. Es wird nun gezeigt, daß dann das signierte ElGamal-Schema nicht ununterscheidbar ist. Ununterscheidbarkeit ist äquivalent zu semantischer Sicherheit. Da das signierte ElGamal-Schema semantisch sicher ist, kann es einen solchen Algorithmus A also nicht geben.

Seien zwei Werte $s_1, s_2 \in G_1$ in beliebiger Reihenfolge und ein Ciphertext $\alpha = E^*(r_1, s, t_1)$ gegeben, wobei $s = s_i$ für $i = 1$ oder 2.
Mache für $i = 1, 2$ folgendes:

- *Wähle zufällig $c_i \in_R \mathbb{Z}^*_{(p-1)}$ und $r_{3i} \in_R \mathbb{Z}_{p-1}$.*
- *Berechne $\hat{c}_i = c_i^{-1} \bmod p - 1$.*
- *Berechne $g^{r_{3i}} \bmod p$. Der Ciphertext α liefert uns zusätzlich $g^{r_1} \bmod p$. Da $r_{31} = r_1 + c_i r_{2i} \bmod (p-1)$ gelten soll, berechnet sich $g^{r_{2i}}$ als $g^{r_{2i}} = (g^{r_{3i}})^{\hat{c}_i} (g^{r_1})^{-\hat{c}_i} \bmod p$.*
- *Berechne analog den Wert $s_i y^{r_{2i}}$.*
- *Sei $x_i = y_i \cdot s_i$ für $y_i \in_R G_1$. Durch die in den letzten beiden Punkten berechneten Werte kann $\beta_i = E(r_{2i}, x_i)$ berechnet werden.*
- *Berechne $s_i^* = s_i \cdot x_i^{c_i}$ und $\theta(s_i^*)$.*

- *Berechne* $\gamma_i = (\theta(s_1^*) \cdot \theta(s_i)^{-1})^{\hat{c}_i}$.
- *Gib das Tupel* $(\alpha, \beta_i, \gamma_i, c_i, s_i^*, r_{3i})$ *aus.*

Für $s = s_i$ *gilt* $\gamma_i = \theta(x_i)$. *Die erzeugten Tupel sind also von der Form, wie sie bei einer richtigen Protokollausführung anfallen. Die erzeugten Tupel für* $s \neq s_i$ *entsprechen den Tupeln, die der Simulator erzeugt. Solche Tupel können laut Annahme vom Algorithmus A unterschieden werden.*

□

Bemerkung 4.11
Wählt man als Challenge einen Hashwert der zuvor gesendeten Nachrichten, so kann man das Protokoll in ein nicht-interaktives computational zero-knowledge Protokoll (im Random Oracle Modell) transformieren:

Nicht-Interaktives VEP für multiplikative Gruppen

Prover		Verifier
öffentlich:	$\theta : G_1 \to G_2$ $E : G_1 \to G_3$ $d = \theta(s)$ $H : \{0,1\}^* \to \mathbb{Z}_p^*$	
geheim:	$s \in G_1$	
$x \in_R G_1$ $r_1, r_2, t_1, t_2 \in_R \mathbb{Z}_p$ $\alpha = E^*(r_1, s, t_1)$ $\beta = E^*(r_2, x, t_2)$ $\gamma = \theta(x)$ $c = H(\alpha, \beta, \gamma)$ $s^* = s \cdot x^c$ $r_3 = r_1 + cr_2$	$\alpha, \beta, \gamma, s^*, r_3 \rightarrow$	V überprüft die Signaturen in α und β Schlägt die Überprüfung fehl, so bricht V ab. $c = H(\alpha, \beta, \gamma)$ $\theta(s^*) \stackrel{?}{=} d \cdot \gamma^c$ $E(r_3, s^*) \stackrel{?}{=} E(r_1, s)E(r_2, x)^c$

4.3 Das Verschlüsselungsschema von Okamoto/Uchiyama

Das Verschlüsselungsverfahren von Okamoto/Uchiyama [34] beruht auf der Tatsache, daß man bei Kenntnis der Faktorisierung des benutzten Moduls $n = p^2q$ auf einer p-Sylow-Untergruppe Γ von $\mathbb{Z}_n^*$ den diskreten Logarithmus effizient berechnen kann.

Zunächst eine kurze Wiederholung aus der Algebra:

Definition 4.12 ***p-Sylow-Untergruppe***
H heißt p-Sylow-Untergruppe von G, wenn ord H=p^n die höchste p-Potenz ist, die die Ordnung von G teilt.

Satz 4.13 *Sei G eine endliche Gruppe und p ein Primteiler von ord G. Dann existiert eine p-Sylow-Untergruppe von G.*

Der Satz besagt, daß für $n = p^2q$ die Gruppe $\mathbb{Z}_n^*$ eine p-Sylow-Untergruppe Γ besitzt, deren Ordnung p ist ($\mathbb{Z}_n^*$ hat die Ordnung p(p-1)(q-1)).
Diese p-Sylow-Untergruppe sieht in unserem Fall folgendermaßen aus:

$$\Gamma = \{x \in \mathbb{Z}_{p^2}^* \mid x = 1 \, mod \, p\}$$

Um den diskreten Logarithmus für Werte aus Γ berechnen zu können, betrachten wir folgende Funktion:

$$\begin{array}{c} L : \Gamma \to \mathbb{F}_p \\ x \mapsto \frac{x-1}{p} \end{array}$$

Satz 4.14
Die Funktion L hat folgende Eigenschaften:

1. *L ist wohldefiniert auf* Γ
2. *Für alle* $a, b \in \Gamma$ *gilt:*

$$L(ab) = L(a) + L(b) \, mod \, p$$

   Die Multiplikation ab wird in Γ *ausgeführt, d.h.* $ab \, mod \, p^2$.

3. *L ist ein Isomorphismus.*

Satz 4.15
Sei $x \in \Gamma$ mit $L(x) \neq 0 \, mod \, p$ und $y = x^m \, mod \, p^2$ für $m \in \mathbb{Z}_p$.
Dann gilt:

$$m = \frac{mL(x)}{L(x)} = \frac{L(x^m)}{L(x)} = \frac{L(y)}{L(x)} = \frac{y-1}{x-1} \, mod \, p$$

Die Bedingung $L(x) \neq 0 \, mod \, p$ bedeutet, daß $x \neq 1 \, mod \, p^2$ und damit, daß x ein erzeugendes Element von Γ ist.
Der obige Satz sagt also aus, daß man unter Kenntnis der Funktion L in der Gruppe Γ effizient den diskreten Logarithmus berechnen kann. Die Kenntnis von L ist gleichbedeutend mit der Kenntnis der Faktorisierung von n.

Aus den obigen Beobachtungen läßt sich nun wie folgt ein asymmetrisches Verschlüsselungsverfahren bauen:

4.3.1 Das Verschlüsselungsschema

Systemparameter

p und q seien große Primzahlen mit Bitlänge k, d.h. $2^{k-1} < p, q \leq 2^k$.
Setze $n = p^2 q$ und wähle zufällig ein $g \in_R \mathbb{Z}_n^*$, so daß die Ordnung von $g_p = g^{p-1}$ in $\mathbb{Z}_{p^2}^*$ gleich p ist.
Setze ferner $h = g^n \, mod \, n$.
Das Tupel (n, g, h, k) bildet den öffentlichen Schlüssel und das Paar (p, q), d.h. die Faktorisierung von n, den geheimen Schlüssel.

Bemerkung 4.16 ***(zur Wahl von g_p)***
Für alle $g \in \mathbb{Z}_n^*$ gilt, daß $g \in \mathbb{Z}_p^*$ und damit $g^{p-1} = 1 \, mod \, p$ (nach dem kleinen Satz von Fermat) und damit wiederum, daß $g_p \in \Gamma$.
Γ hat die Ordnung p. Alle Elemente aus Γ haben damit entweder die Ordnung 1 oder die Ordnung p. Das einzige Element aus Γ, das die Ordnung 1 hat, ist die 1 selbst. Alle anderen (p-1) Elemente haben die Ordnung p.
Bei zufälliger Wahl von g aus $\mathbb{Z}_n^*$ ist g_p gleichverteilt in Γ. Die Wahrscheinlichkeit, daß für ein zufälliges g aus $\mathbb{Z}_n^*$ der Wert $g_p = g^{p-1}$ in $\mathbb{Z}_{p^2}^*$ die Ordnung p hat, beträgt also $\frac{p-1}{p}$.

Bemerkung 4.17 ***(zur Ordnung von g)***
Das Schema von Okamoto/Uchiyama ist gebrochen, wenn es gelingt, diskrete Logarithmen zur Basis g zu berechnen. Die von g erzeugte Untergruppe sollte also hinreichend groß sein.
Aus der Wahl der Parameter folgt bereits, daß $ord \, g \geq p$ in $\mathbb{Z}_{p^2}^*$ ist, denn:
Angenommen: $ord \, g = l < p$ in $\mathbb{Z}_{p^2}^*$
$\Rightarrow g^l = 1 \, mod \, p^2$
$\Rightarrow g^{l(p-1)} = 1 \, mod \, p^2$
$\Rightarrow ord \, g^{p-1} \mid l < p$ *in* $\mathbb{Z}_{p^2}^*$
Das ist ein Widerspruch zur Wahl von g, wo gefordert wurde, daß die Ordnung von $g_p = g^{p-1}$ in $\mathbb{Z}_{p^2}^*$ den Wert p hat.

Da ferner gilt, daß $ord_n\, g = kgV(ord_q\, g, ord_{p^2}\, g)$, folgt, daß die Ordnung von g in $\mathbb{Z}_n^*$ mindestens gleich p ist.

Verschlüsselung

Die zu verschlüsselnde Nachricht m muß im Bereich $0 < m < 2^{k-1}$ liegen.
Zunächst muß ein zufälliger Wert $r \in_R \mathbb{Z}_n$ gewählt werden.
Der Ciphertext zur Nachricht m errechnet sich dann als:

$$C = E(m, r) = g^m h^r \, mod \, n$$

Entschlüsselung

Bei gegebenem Ciphertext C berechnet man zunächst $C_p = C^{p-1} \, mod \, p^2$.
Der Klartext ergibt sich dann durch

$$\frac{L(C_p)}{L(g_p)} = \frac{L(g_p^m g_p^{nr})}{L(g_p)} = \frac{L(g_p^m)}{L(g_p)} + \frac{L(g_p^{nr}}{L(g_p)} = \frac{mL(g_p)}{L(g_p)} + \frac{nrL(g_p)}{L(g_p)}$$

$$= m + nr = m + rp^2q = m \, mod \, p$$

Verknüpfungseigenschaften

Für $m_1 + m_2 < 2^{k-1}$ erhält man eine nützliche Verknüpfungseigenschaft:

$$E(m_1, r_1)E(m_2, r_2) = g^{m_1} h^{r_1} g^{m_2} h^{r_2} = g^{m_1+m_2} h^{r_1+r_2} = E(m_1 + m_2, r_1 + r_2)$$

Aus dieser Eigenschaft läßt sich direkt eine zweite Verknüpfungseigenschaft ableiten:

$$(E(m, r))^c = E(cm, cr) \text{ für } c \in \mathbb{N} \text{ mit } cm < 2^{k-1}$$

Blendungseigenschaft

Eine Person kann einen Ciphertext $C = E(m, r)$ so in einen Ciphertext $\hat{C} = Ch^r$ verändern, daß $\hat{C}$ nach wie vor eine gültige Verschlüsselung von m ist.
Dazu muß er weder die verschlüsselte Nachricht noch den geheimen Schlüssel kennen.

4.3.2 Die Sicherheit des Verfahrens

In der Originalarbeit, in der das eben beschriebene Verfahren vorgestellt wird [34], sind einige Sätze bezüglich der Sicherheit des Systems aufgeführt und bewiesen, die hier der Vollständigkeit halber zitiert werden:

- Unter der Faktorisierungsannahme ist die Verschlüsselungsfunktion eine Trapdoor-Funktion.
- Das Brechen des Kryptosystems ist äquivalent zum Faktorisierungsproblem. Mit Brechen ist hier die Existenz eines probabilistischen polynomialzeit Algorithmus gemeint, der mit nicht-vernachlässigbarer Wahrscheinlichkeit Ciphertexte entschlüsseln kann.
- Es ist genauso schwierig, die Verschlüsselungen zweier beliebiger Nachrichten m_1 und m_2 zu unterscheiden, wie die Verschlüsselungen der Nachrichten 0 und 1.
  Das vorgestellte Schema ist also semantisch sicher unter der p-subgroup-Annahme.
  Die p-subgroup-Annahme besagt, daß $E(0,r) = h^r \, mod \, n$ und $E(1,\hat{r}) = gh^{\hat{r}} \, mod \, n$ (rechnerisch) ununterscheidbar sind, wobei r und $\hat{r}$ uniform und unabhängig aus $\mathbb{Z}_n$ gezogen werden.

Ein Angriff auf das Verfahren
Der Angriff basiert auf der Idee, Nachrichten $m > 2^k$ zu verschlüsseln (Das System schreibt vor, daß Nachrichten im Bereich $0 < m < 2^{k-1}$ liegen müssen). Wir nehmen an, daß ein Angreifer Zugriff auf ein Entschlüsselungsorakel hat (in unserem Fall kann die TTP als ein solches Orakel angesehen werden). Der Angreifer verschlüsselt eine Nachricht $m > 2^k$ und bekommt vom Orakel einen Wert $\hat{m}$ mit $\hat{m} = m \, mod \, p$ zurückgeliefert. Er weiß nun, daß $p \mid (\hat{m} - m)$ und $p \mid n$, also daß $p \mid ggT((\hat{m} - m), n)$, woraus er den geheimen Schlüssel p schnell berechnen kann.

4.3.3 Die signierte Okamoto/Uchiyama-Verschlüsselung

Das Verschlüsselungsverfahren von Okamoto und Uchiyama ist nicht sicher gegen chosen-ciphertext-Angriffe. Diese Eigenschaft wird aber später für das Verifiable Encryption Protokoll benötigt.
Um Sicherheit gegen adaptive-chosen-ciphertext-Angriffe zu erreichen, benutzen wir den Trick, der schon in Kapitel 2 bei der Absicherung der ElGamal-Verschlüsselung gegen adaptive-chosen-ciphertext-Angriffe verwendet wurde.
Um diesen Trick anwenden zu können, müssen wir aber zunächst noch kleine Veränderungen am originalen Okamoto/Uchiyama-Schema vornehmen.

Ein abgewandeltes Okamoto/Uchiyama-Schema

Die signierte Okamoto/Uchiyama-Verschlüsselung soll (genau wie die signierte ElGamal-Verschlüsselung) aus dem eigentlichen Ciphertext und einer angehängten digitalen Signatur des Ciphertextes bestehen. Der verwendete Signaturschlüssel soll wieder aus dem Verschlüsselungsvorgang hervorgehen. Da das Okamoto/Uchiyama-Schema auch auf dem diskreten Logarithmus basiert, können wir wieder die Schnorr-Signatur verwenden.
Die Ordnung des öffentlichen Elementes g muß im Okamoto/Uchiyama-Schema

geheim bleiben, da sie Rückschlüsse auf $\varphi(n)$ und damit auch auf den Modul n selbst zulassen würde. Andererseits muß die Gleichung $z = r + cx$ im Signaturvorgang modulo einem Vielfachen der Ordnung von g reduziert werden, um so wenig wie möglich über den geheimen Schlüssel x preiszugeben. Wir fordern deshalb, daß die Ordnung von g den Wert p hat (und damit n teilt) und reduzieren modulo n.
Im einfachen Okamoto/Uchiyama-Schema besteht der Verschlüsselungsvorgang nur aus einer (diskreten) Exponentiation ($m \mapsto g^{m+nr}$). Als Schlüsselpaar für die angehängte Signatur kommt also eigentlich nur das Paar $(m+nr, g^{m+nr} \bmod n)$ in Frage. Bei Benutzung von $m+nr$ als geheimem Schlüssel und n als Modul vereinfacht sich jedoch die zweite Komponente der angehängten Schnorr-Signatur zu $z = r + c(m + nr) = r + cm \bmod n$. Der Wert c ist bekannt, da er ebenfalls Teil der Signatur ist. Bei gültigen Signaturen läßt sich außerdem g^r als $g^r = g^z E(m,s)^{-c} = g^{r+cm}(g^m g^{ns})^{-c} \bmod n$ berechnen, da $(ord\, g) \mid n$ gilt. Aus den bekannten Werten kann man also $g^m \bmod n$ berechnen. Damit wurde der zufällige Term aus der Okamoto/Uchiyama-Verschlüsselung entfernt. Damit ist das System insbesondere nicht mehr semantisch sicher.
Es wird jetzt zunächst eine nicht-randomisierte Version der Okamoto/Uchiyama-Verschlüsselung vorgestellt. Diese werden wir dann durch Anhängen einer Schnorr-Signatur des Ciphertextes erweitern, um sie unseren Bedürfnissen anzupassen.

Systemparameter
Es seien wieder p und q zwei Primzahlen mit Bitlänge k, d.h. $2^{k-1} < p, q \leq 2^k$, $n = p^2 q$ und $g \in \mathbb{Z}_n^*$, so daß die Ordnung von $g_p = g^{p-1}$ in $\mathbb{Z}_{p^2}^*$ gleich p ist. Ferner sei $h = g^n \bmod n$.
Zusätzlich fordern wir jetzt jedoch, daß die Ordnung von g in $\mathbb{Z}_n^*$ den Wert p hat. Eine solche Wahl von g ist mit den Anforderungen, die an g im originalen Schema gestellt werden, vereinbar und läßt sich effektiv berechnen:

- Generiere einen Generator g von $\mathbb{Z}_{p^2}^*$
- Exponentiation von g mit $d = k(p-1)$ für ein $k \in \mathbb{Z}_p^*$ liefert genau eines der $(p-1)$-vielen $\hat{g}$, die in $\mathbb{Z}_{p^2}^*$ die Ordnung p haben.
- Der chinesische Restsatz liefert durch Lösen der beiden Gleichungen $x = \hat{g} \bmod p^2$ und $x = 1 \bmod q$ genau ein $x \in \mathbb{Z}_n^*$ mit den gewünschten Eigenschaften.

Das abgewandelte Schema

Verschlüsselung:
$C = E(m) = g^m \bmod n$

Entschlüsselung:
Sei $C_p = C^{p-1} \bmod p^2$.
Der gesuchte Klartext ergibt sich durch Berechnung von $\frac{L(C_p)}{L(g_p)} \bmod p$

Die Verschlüsselung ist in diesem Fall einfach eine diskrete Exponentiation. Zur Entschlüsselung benötigt man die Faktorisierung von n als Trapdoorinformation.

Dadurch bleibt insbesondere die Verknüpfungseigenschaft

$$(E(m,r))^c = E(cm,cr) \text{ für } c \in \mathbb{N} \text{ mit } cm < 2^{k-1}$$

erhalten.

Dadurch, daß die Randomisierung weggefallen ist, lassen sich von den ursprünglichen Sicherheitsbehauptungen jedoch nur noch die beiden folgenden aufrechterhalten:

- Unter der Faktorisierungsannahme ist die Verschlüsselungsfunktion eine Trapdoor-Funktion.
- Das Brechen des Kryptosystems ist äquivalent zum Faktorisierungsproblem.

Das Verschlüsselungsschema von Okamoto und Uchiyama wird in seiner nichtrandomisierten Version ebenfalls von Feng Bao in [2] benutzt, um ein Verifiable Encryption Protokoll für diskrete Logarithmen zu bauen.

Die signierte Okamoto/Uchiyama-Verschlüsselung

Um eine Nachricht m mit $0 < m < 2^{k-1}$ zu verschlüsseln, muß folgendes gemacht werden:

- $C = E(m) = g^m \, mod \, n$
- $r \in_R \mathbb{Z}_n^*$
- berechne $k = g^r \, mod \, n$
- $c = H(C,k)$
- $z = r + cm \, mod \, n$

Das Paar (c,z) bildet eine Schnorr-Signatur des Ciphertextes C unter dem Schlüsselpaar $(m, g^m \, mod \, n)$.
Die Signatur ist genau dann gültig, wenn die Gleichung $e = H(C, g^z (g^m)^{-c} \, mod \, n)$ erfüllt ist.

Um die signierte Okamoto/Uchiyama-Verschlüsselung in einem Verifiable Encryption Protokoll einsetzen zu können, brauchen wir, daß es sicher gegen adaptive-chosen-ciphertext-Angriffe ist, d.h. ein Angreifer A kann folgendes Spiel nicht gewinnen:

- Dem Angreifer A wird ein gültiger Ciphertext C vorgelegt.

- A darf einem Entschlüsselungsorakel Ciphertexte $C_1, \ldots, C_n$ schicken und bekommt vom Orakel die zugehörigen Klartexte.
  Dabei muß $C_i \neq C$ für $1 \leq i \leq n$ gelten.

A hat gewonnen, wenn es ihm gelingt, den Klartext von C zu berechnen.

Anschaulich kann man argumentieren, daß ein interaktiver Angreifer, der Zugriff auf ein Entschlüsselungsorakel hat, nicht stärker als ein nicht-interaktiver Angreifer sein kann. Um dem Orakel nämlich einen Ciphertext vorlegen zu können, dessen Klartext dem Angreifer nicht bekannt ist, müßte eine korrekte Schnorr-Signatur gebildet werden. Dem Angreifer stünde dazu nur der öffentliche Schlüssel zur Verfügung (der geheime Schlüssel entspricht in unserem Fall dem Klartext, von dem wir angenommen haben, daß ihn der Angreifer nicht kennt). Bisher sind jedoch noch keine ernstzunehmenden Schwächen der Schnorr-Signatur bekannt (siehe Kapitel 2). Der Angreifer könnte dem Orakel also höchstens Ciphertexte vorlegen, deren Klartexte er schon kennt.

Einen formalen Beweis dafür, daß das signierte Okamoto/Uchiyama-Schema sicher gegen adaptive-chosen-ciphertext-Angriffe ist, kann ich an dieser Stelle leider nicht geben. Der folgende Satz bekräftigt jedoch obige informelle Argumentation. Er sagt aus, daß ein interaktiver Angreifer seine Erfolgswahrscheinlichkeit, gültige Ciphertext ohne Kenntnis des zugehörigen Klartextes zu erzeugen, nur um einen Faktor p^{-1} pro Interaktion verbessern kann (im generischen und Random Oracle Modell). Allerdings ist die Gruppe $\mathbb{Z}_n^*$, in der wir rechnen, nachweislich keine Gruppe, für die die *ideal group assumption* (siehe Abschnitt 3.3) zutrifft. Die Sicherheit des Okamoto/Uchiyama-Verfahrens beruht auf der Tatsache, daß der Modul $n = p^2q$ nicht faktorisiert werden kann. In dem von uns betrachteten generischen Modell sind Operationen auf non-group-data (und damit auch Faktorisierung) aber kostenlos.
Aus Satz 4.16 folgt jedoch, daß ein Angreifer, der aus Interaktionen mit dem Entschlüsselungsorakel wesentliche Vorteile ziehen möchte, Operationen benutzen muß, die über die von uns vorgestellten generischen Operationen hinausgehen.

Satz 4.18
Ein generischer Angreifer, der l Interaktionen mit dem Entschlüsselungsorakel eingeht, hat eine um l/p höhere Erfolgswahrscheinlichkeit als ein nicht-interaktiver Angreifer (im generischen und Random Oracle Modell).

Beweis:
Wir beschränken uns auf den Fall, daß keine Kollisionen unter den Gruppenelementen auftreten. Werden t' viele Gruppenelemente $f_1, \ldots, f_{t'}$ von A im Laufe seines Angriffes berechnet, so treten Kollisionen unter den Gruppenelementen nur mit Wahrscheinlichkeit $1-\binom{t'}{2}$ auf (siehe [39]).
Der Angreifer A wird durch Elimination der jeweils ersten Interaktion mit dem Entschlüsselungs-Orakel in einen nicht-interaktiven Angreifer transformiert. Sei (C, c, z) die erste Anfrage von A. Wir nehmen an, daß es sich dabei um einen gültigen Ciphertext handelt, d.h. (c, z) ist eine gültige Schnorr-Signatur von C

unter dem Schlüsselpaar $(\log_g C, C$*. Der Angreifer muß* $c = H(C, k)$ *(*k *gehört zu den berechneten Gruppenelementen) vor* z *berechnet haben, da ansonsten die Signatur nur mit Wahrscheinlichkeit* p^{-1} *gültig ist (da der Hashwert zufällig ist). A muß dann den Wert* z *so berechnen, daß* $z = \log_g k + c \log_g C$ *gilt.*
Der Angreifer A hat nur den Generator $g \in \mathbb{Z}_n^*$ *und den challenge-Ciphertext* $E(m, r) = (g^m, c, z)$ *als Gruppendaten-Eingabe zur Verfügung. Alle Gruppenelemente, die A bekannt sind, lassen sich also schreiben als* $f_i = \sum_{j=1}^{2} f_j^{\alpha_{i,j}}$ *für* $i = 1, \ldots, t'$*, wobei der Exponentenvektor* $\alpha_i = (\alpha_{i,1}, \alpha_{i,2})$ *von den zuvor berechneten non-group-Daten abhängt.*
Der Wert k *gehört zu den berechneten Gruppenelementen, d.h.* $k = f_i = g^{\alpha_{i,1}}(g^m)^{\alpha_{i,1}}$ *und damit* $\log_g k = \alpha_{i,1} + m\alpha_{i,2}$*. Da ferner* $\log_g C = m$ *gilt, folgt* $z = \alpha_{i,1} + m(\alpha_{i,2} + c)$*.*
Die Werte $\alpha_{i,1}$ *und* $\alpha_{i,2}$ *müssen gewählt werden, bevor das Hashorakel nach* $c = H(C, k)$ *befragt wird, da sie zur Berechnung von* k *nötig sind. Ein Angreifer, der* m *nicht kennt, kann die Gleichung nur mit Wahrscheinlichkeit* p^{-1} *erfüllen.*
Sei l *die Anzahl der Interaktionen zwischen dem generischen Angreifer A und dem Entschlüsselungsorakel. Durch obiges Vorgehen eliminiere man die jeweils erste Interaktion. Die Erfolgswahrscheinlichkeiten eines interaktiven und eines nicht-interaktiven Angreifers unterscheiden sich dann gerade um* l/p*.* □

Bezeichnungen

Sei eine Nachricht m mit $0 < m < 2^{k-1}$ gegeben. Im weiteren bezeichne

$$E(m) = g^m \bmod n$$

eine („normale“) Okamoto/Uchiyama-Verschlüsselung von m und

$$E^*(m, r) = (g^m \bmod n,\ c,\ z)$$

eine signierte Okamoto/Uchiyama-Verschlüsselung von m bzgl. des Zufallswertes r.

4.4 Verifiable Encryption für additive Gruppen

Im Unterschied zum VEP für multiplikative Gruppen, betrachten wir nun einen Gruppenisomorphismus θ, dessen Urbildgruppe die (additve) Gruppe $\mathbb{Z}_p$ (p prim) ist, d.h. θ habe die Eigenschaft

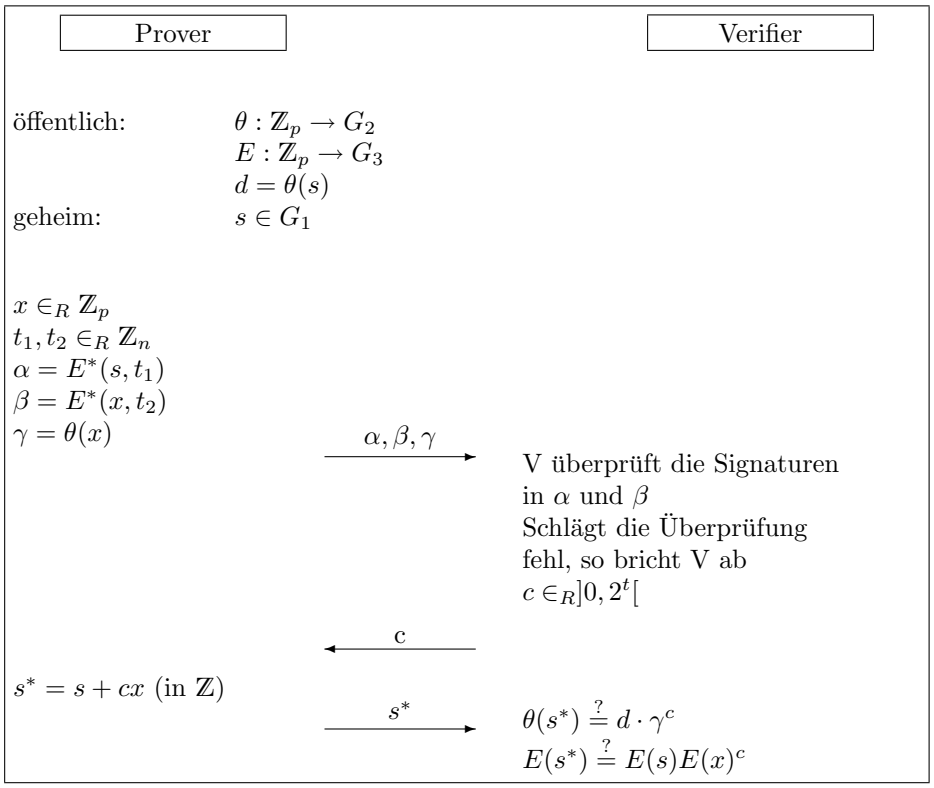
$$\theta(x_1 + x_2) = \theta(x_1) \cdot \theta(x_2) \text{ für alle } x_1, x_2 \in \mathbb{Z}_p$$

Ferner seien $n = q^2 r$ (q und r prim) und $g \in \mathbb{Z}_n^*$ mit $ord\, g = q$ die Parameter für das (signierte) Okamoto/Uchiyama-Schema.

Wie bisher ist beiden Teilnehmern ein Wert $d \in G_2$ bekannt. Als geheime Eingabe ist dem Prover ferner ein Wert s, mit $\theta(s) = d$, bekannt.
Bei t handelt es sich um einen Sicherheitsparameter, der sorgfältig gewählt werden muß, wie im folgenden noch gezeigt wird.

Setzt man im schematischen Protokoll nun das Okamoto/Uchiyama-Schema ein und paßt Gleichung (1) an, so erhält man ein

VEP für additive Gruppen



Bemerkung 4.19

- *Die Werte $E(s)$ und $E(x)$ erhält man wieder, indem man die letzten beiden Komponenten von α bzw. β abschneidet.*

- *Der Wert s^* wird vom Prover in $\mathbb{Z}$ berechnet.*
  Da im Okamoto/Uchiyama-Schema nur Nachrichten eindeutig verschlüsselt werden können, die kleiner als 2^{k-1} sind (k ist die Bitlänge der Primzahlen q und r), müssen die Parameter des VEP für additive Gruppen so gewählt werden, daß stets $0 < s^* < 2^{k-1}$ gilt.
  In Kapitel 5 wird ein Protokoll (Unterprotokoll Check-Size vorgestellt, mit dessen Hilfe die dritte Partei vor Entschlüsselung prüfen kann, ob wirklich ein Wert $< 2^{k-1}$ verschlüsselt wurde.

- *Wir nehmen im folgenden an, daß $2^t < p$. Das bedeutet, daß die Abbildung $\mathbb{Z}_p \to \mathbb{Z}_p$, $x \mapsto cx$ bijektiv ist.*

Satz 4.20 (Completeness)
Wenn P und V ehrlich sind, dann wird V für alle s, d mit $\theta(s) = d$ akzeptieren.

Beweis:
Der Verifier überprüft zwei Dinge:

1. $\theta(s^*) = \theta(s + cx) = \theta(s) \cdot \theta(x)^c = d \cdot \gamma^c$
2. $E(s^*, r_3) = g^{s^*} h^{r_3} = g^{s+cx} h^{r_1+cr_2}$

$$= g^s h^{r_1} (g^x h^{r_2})^c = E(s, r_1) E(x, r_2)^c$$

□

Lemma 4.21
Seien θ ein Einweg-Homomorphismus, $\gamma = \theta(x)$, $d = \theta(s)$ und $s^* = \hat{s} + c\hat{x}$.
Wenn der Verifier akzeptiert, dann gilt mit überwältigender Wahrscheinlichkeit $x = \hat{x} \bmod p$ und $s = \hat{s} \bmod p$ und damit $s^* = s + cx \bmod p$.

Beweis:
$\theta(s^*) = \theta(\hat{s} + c\hat{x}) = \theta(\hat{s}) \cdot \theta(\hat{x})^c$
Nach Annahme akzeptiert V, d.h. es gilt ferner:
$\theta(s^*) = d \cdot \gamma^c = \theta(s) \cdot \theta(x)^c$
$\Rightarrow \theta(\hat{s}) = \theta(s + c(x - \hat{x}))$
Da θ bijektiv ist, folgt:
$\hat{s} = s + c(x - \hat{x}) \bmod p$
Die Abbildung $\mathbb{Z}_p \to \mathbb{Z}_p$, $x \mapsto cx$ ist bijektiv (siehe Bemerkung 4.17).
Für feste $\hat{s}$ und $\hat{x}$ gibt es also genau ein c, das obige Gleichung erfüllt. Da aber $\hat{s}$ und $\hat{x}$ vom Prover gewählt werden müssen, bevor er den Wert c vom Verifier bekommt, ist die Wahrscheinlichkeit, daß der Verifier akzeptiert, wenn der Prover $\hat{x} \neq x$ oder $\hat{s} \neq s$ gewählt hat, gerade $\frac{1}{p}$. □

Satz 4.22 (Soundness)
Seien θ ein Einweg-Homomorphismus und E sicher gegen adaptive-chosen-ciphertext-Angriffe.
Besteht der Prover das Protokoll mit nicht-vernachlässigbarer Wahrscheinlichkeit, so ist α eine Verschlüsselung eines Urbildes von d (unter dem Schlüssel der TTP).

Beweis:
Sei $\gamma = \theta(x)$ und $d = \theta(s)$.
Angenommen, in α und β wurden andere Werte als s und x (nämlich $\hat{s}$ und $\hat{x}$) verschlüsselt.
Nach Voraussetzung akzeptiert der Verifier, d.h. es gilt
$E(s^*, r_3) = E(s, r_1)E(x, r_2)^c$
$\Leftrightarrow g^{s^*} h^{r_3} = g^{\hat{s}} h^{r_1} g^{c\hat{x}} h^{cr_2}$
$\Leftrightarrow g^{s^*+nr_3} = g^{\hat{s}+c\hat{x}+n(r_2+cr_1}$
$\Leftrightarrow g^{r_3} = g^{\frac{\hat{s}-s^*+c\hat{x}}{n}+(r_2+cr_1)}$
$\Leftrightarrow r_3 = \frac{\hat{s}-s^*+c\hat{x}}{n} + (r_2 + cr_1) \, mod \, q$
Es gilt aber $q \mid n$, da $n = q^2 r$, d.h. n ist nicht invertierbar mod q.
$\Rightarrow \hat{s} - s^* + c\hat{x} = 0 \, mod \, q$
$\Leftrightarrow s^* = \hat{s} + c\hat{x} \, mod \, q$
$\Leftrightarrow q \mid (s^* - \hat{s} - c\hat{x})$
1. Fall: $s^* - \hat{s} - c\hat{x} = 0$
$\Rightarrow s^* = \hat{s} + c\hat{x} \, mod \, p$
Mit Lemma 4.19 folgt $s = \hat{s} \, mod \, p$
2. Fall: $s^* - \hat{s} - c\hat{x} \neq 0$
$\Rightarrow s^* - \hat{s} - c\hat{x} = lq$ *für ein* $l \neq 0$ *aus* $\mathbb{Z}$
In diesem Fall würde die Berechnung von $ggt(s^* - \hat{s} - c\hat{x}, n)$ einen Teiler des Moduls n liefern. □

Das vorliegende Protokoll hat nicht die Zero-Knowledge-Eigenschaft, da aus Kenntnis von s^* und c der Wert $s \, mod \, c$ errechnet werden kann.
Der nächste Satz zeigt, daß der Verifier nur $s \, mod \, c$ (und nicht mehr) lernt. Der Verifier kann damit zwar wertvolle Informationen aus dem VEP gewinnen, in der Praxis ist das Protokoll aber immer noch sicher.
Benutzt man das VEP (für additive Gruppen) zum fairen Tausch von Diskreter Logarithmus Unterschriften, so hat die Urbildgruppe G_1 in der Regel eine Ordnung von ca. 2^{160}. Die Challenge c hat eine Länge von t Bits. Der Prover erfährt den Wert $s \, mod \, c$ und weiß damit, daß s in der Untergruppe $\{g \in G_1 \mid g = s \, mod \, c\}$ von G_1 liegt. Diese hat aber eine Ordnung von ca. 2^{160-t}.
Hier zeigt sich insbesondere auch, daß die richtige Wahl der Größe des Sicherheitsparameters t von immenser Bedeutung für die Sicherheit des Protokolls ist.

Wir nehmen im folgenden an, daß ein Angreifer, der Werte $\alpha_1 = E(s_1)$, $\alpha_2 = E(s_2)$ und $d = \theta(s)$ mit $s_i = s$ für $i = 1$ oder 2 vorgelegt bekommt, nicht

entscheiden kann, für welches i die Gleichung $s_i = s$ gilt.
Alternativ läßt sich dieses Problem auch so formulieren:
Es seien zwei Paare $(\alpha_1 = E(s_1), d = \theta(s))$ und $(\alpha_2 = E(2_2), d = \theta(s))$ mit $s_i = s$ für $i = 1$ oder 2 gegeben. Entscheide, für welches i die Urbilder von α_i und d gleich sind.
Dieses Problem werden wir im folgenden als *Urbild-Problem* bezeichnen. Das Urbild-Problem ist leicht, wenn die Bildgruppen von E und θ speziell gewählt sind. Daß dieses Problem in unserem Fall schwer ist, sollen folgende Überlegungen begründen:

Die Funktionen E und θ bilden in multiplikative zyklische Gruppen der Ordnung q bzw. p ab $(p \neq q)$. Diese sind isomorph zu Untergruppen von $\mathbb{Z}_m^*$ mit $m = q^2p^2$. Seien g und $\hat{g} \in \mathbb{Z}_m^*$ Erzeugende dieser Untergruppen.
Die Generatoren g und $\hat{g}$ erzeugen in $\mathbb{Z}_m^*$ eine Untergruppe $\{g^k\hat{g}^l \mid k \in \mathbb{Z}_q, l \in \mathbb{Z}_p\} = \{(g\hat{g})^k \mid k \in \mathbb{Z}_{pq}\}$ der Ordnung $p \cdot q$.
Den Paaren (α_1, d) und (α_2, d) entsprechen in dieser Untergruppe die Werte $g^{s_1}\hat{g}^s$ und $g^{s_2}\hat{g}^s$. Für $s_i = s$ schreibt sich dieses Element der Untergruppe als $(g\hat{g})^s$ (mit $s < p$) und für $s_i \neq s$ als $(g\hat{g})^k$ für ein $k \in \mathbb{Z}_{pq}$ mit $k > p$.
Ein Angreifer A, der das Urbild-Problem in $\mathbb{Z}_n^*$ lösen kann, kann also entscheiden, ob der diskrete Logarithmus von $g^{s_i}\hat{g}^s$ zur Basis $(g\hat{g})$ größer oder kleiner als p ist. Ein solcher Angreifer kann s bereits berechnen:

- Bestimme das i, für das $\log_{g\hat{g}}(g^{s_i}\hat{g}^s) < p$ gilt.
  Setze $G = g^{s_i}\hat{g}^s = (g\hat{g})^s$ (für dieses i).
- Der Angreifer kann durch seinen Unterscheider eine Folge δ_i konstruieren, so daß die Folge $G_i = (g\hat{g})^{s+\delta_i}$ gegen $(g\hat{g})^p$ konvergiert.
- Hat der Angreifer ein δ_k gefunden mit $G_k = (g\hat{g})^p$, so gibt er $s = p - \delta_k$ aus.

Die δ_i können so gewählt werden, daß in jedem Schritt ein Bit von $p - s$ errechnet wird. Im Extremfall müssen also $\log_2 p$ Runden dieses Algorithmus zur Bestimmung von s durchlaufen werden.

Der nächste Satz zeigt, daß unter Annahme der Schwierigkeit des Urbild-Problems ein ehrlicher Prover nichts dazu lernt außer $s \bmod c$. Um dies zu zeigen, wird ein Simulator angegeben, der außer den öffentlichen Eingaben noch über ein Element der Menge $\{g \in G_1 \mid g = s \bmod c\}$ verfügt (hat der Simulator $s \bmod c$ als weitere Eingabe, kann er sich einen solchen Wert selbst konstruieren).
Anhand dieser zusätzlichen Eingabe kann der Simulator Protokollausführungen simulieren, die rechnerisch nicht von echten Protokollausführungen unterschieden werden können. Die simulierten Protokollausführungen können keine Informationen über s (außer dem Werte $s \bmod c$) preisgeben, da sie ohne Kenntnis des geheimen Wertes s erstellt wurden. Da simulierte von echten Durchläufen (rechnerisch) nicht unterschieden werden können, kann der Prover auch aus

echten Ausführungen keine Informationen über s (außer $s \bmod c$) extrahieren.

Satz 4.23
Sei θ ein Einweg-Homomorphismus und E sicher gegen adaptive-chosen-ciphertext-Angriffe.
Dann existiert ein probabilistischer polynomialzeit Simulator S, der Protokollausführungen simulieren kann, die (rechnerisch) nicht von echten Protokollausführungen zu unterscheiden sind (im Random Oracle Modell und für honest verifier), sofern S einen Wert $\hat{s} \in_R \{g \in G_1 \mid g = s \bmod c\}$ als zusätzliche Eingabe bekommt und das Urbild-Problem für die Funktionen E und θ schwer ist.

Beweis:
Der Simulator S macht folgendes:

- *Wähle $\hat{s} \in_R \{g \in G_1 \mid g = s \bmod c\}$*
- *Wähle $t_1, t_2 \in_R \mathbb{Z}_n$*
- *Berechne $\alpha = E^*(\hat{s}, t_1)$*
- *Wähle $\hat{x} \in_R G_1$ und berechne $\beta = E^*(\hat{x}, t_2)$.*
- *Wähle $c \in_R]0, 2^t[$*
- *Berechne $s^* = \hat{s} + c\hat{x}$ (in $\mathbb{Z}$)*
- *Berechne $\gamma = (\theta(s^*) \cdot d^{-1})^{\hat{c}}$, wobei $\hat{c} = c^{-1} \bmod p$*
- *Gib das Tupel $(\alpha, \beta, \gamma, c, s^*)$ aus.*

Schnorr-Signaturen sind für alle Nachrichten m und alle Schlüsselpaare (x, g^x) gleichverteilt (im Random Oracle Modell). Die an die Ciphertexte angehängten Signaturen tragen also nichts zur Unterscheidung von echten und simulierten Protokollausführungen bei.

Angenommen, es existiert ein Algorithmus, der echte von simulierten Protokollausführungen unterscheiden kann. Dann kann dieser Algorithmus benutzt werden, um das Urbild-Problem für die Funktionen E und θ zu lösen.

Seien $\alpha_1 = E(s_1)$, $\alpha_2 = E(s_2)$ und $\theta(s)$ mit $s = s_i$ für $i = 1$ oder 2 gegeben.
Mache für $i = 1, 2$ folgendes:

- *Wähle zufällig $c_i \in_R]0, 2^t[$.*
- *Berechne $\hat{c}_i = c_i^{-1} \bmod p$ und $\check{c}_i = c_i^{-1} \bmod n$.*
- *Wähle $s_i^* \in_R]0, 2^{k-1}[$*
  (k ist die Bitlänge der Primzahlen q und r des Okamoto/Uchiyama-Schemas)
- *Berechne $E(s_i^*)$ und $\theta(s_i^*)$.*

- *Berechne* $\gamma_i = (\theta(s_i^*) \cdot \theta(s)^{-1})^{\hat{c}_i}$
- *Berechne* $\beta = (E(s_i^*) \cdot \alpha_i^{-1})^{\check{c}_i}$
- *Gib das Tupel* $(\alpha_i, \beta_i, \gamma_i, c_i, s_i^*)$ *aus.*

Für $s_i = s$ *entstehen dabei Tupel, wie sie bei echten Protokollausführungen vorkommen, für* $s_i \neq s$ *Tupel, wie sie der Simulator produziert.* □

Auch für das VEP für additive Gruppen läßt sich wieder eine nicht-interaktive Version angeben.

Nicht-Interaktives VEP für additive Gruppen

Prover		Verifier
öffentlich:	$\theta : \mathbb{Z}_p \to G_2$ $d = \theta(s)$ $H : \{0,1\}^* \to]0, 2^t[$	
geheim:	$s \in G_1$	
$x \in_R G_1$ $t_1, t_2 \in_R \mathbb{Z}_n$ $\alpha = E^*(s, t_1)$ $\beta = E^*(x, t_2)$ $\gamma = \theta(x)$ $c = H(\alpha, \beta, \gamma)$ $s^* = s + cx$ (in $\mathbb{Z}$)	$\xrightarrow{\alpha, \beta, \gamma, s^*}$	V überprüft die Signaturen in α und β Schlägt die Überprüfung fehl, so bricht V ab $c = H(\alpha, \beta, \gamma)$ $\theta(s^*) \stackrel{?}{=} d + c \cdot \gamma$ $E(s^*) \stackrel{?}{=} E(s)E(x)^c$

Kapitel 5

Fair Exchange of Secrets

Bei dem Fair Exchange Problem sind A und B im Besitz von Geheimnissen S_A und S_B, die sie fair austauschen wollen. Fair bedeutet dabei, daß A den Wert S_B genau dann bekommen soll, wenn B im Gegenzug den Wert S_A erhält.

Der naive Ansatz ist, daß eine Partei, z.B. A, ihr Geheimnis S_A zuerst sendet und anschließend S_B von B geschickt bekommt. Bricht B jedoch das Protokoll ab, nachdem er S_A erhalten hat, so hat A zwar sein Geheimnis preisgegeben, im Gegenzug S_B jedoch nicht bekommen. Das Protokoll ist also nicht fair.

Ein Ausweg ist es, die Geheimnisse in kleinere Einheiten aufzuspalten und blockweise auszutauschen. Hierbei muß sichergestellt sein, daß ein öffentliches Prädikat existiert, anhand dessen A und B nach jedem Schritt überprüfen können, ob der erhaltene Block korrekt ist. Andererseits dürfen die bereits erhaltenen Blöcke (fast) nichts über die übrigen Blöcke aussagen. Wählt man die Blockgröße hinreichend klein, so ist das Protokoll fair, da eine Person immer nur einen minimalen rechnerischen Vorteil gegenüber der anderen Person hat. Problematisch wird es allerdings, wenn die am Tausch beteiligten Personen über eine unterschiedliche Rechenleistung verfügen.
Den Ansatz, die Geheimnisse blockweise auszutauschen, wählte unter anderem Ivan Damgard [14], dessen Resultat von Fujisaki und Okamoto [22] bezüglich Kommunikations- und Rechenaufwand verbessert wurde.

Ein anderer Ansatz ist es, eine vertrauenswürdige dritte Partei (Trusted Third Party, TTP) in das Protokoll aufzunehmen, die als ein Art Notar fungiert. Hier wäre der triviale Ansatz, daß beide Parteien ihre Geheimnisse zunächst an die TTP senden, die diese dann weiterleitet, sobald sie beide Geheimnisse erhalten hat und deren Richtigkeit (anhand eines öffentlichen Prädikats) überprüft hat. Außer der Vertrauenswürdigkeit müßte man von dieser dritten Partei ferner fordern, daß sie immer erreichbar ist. Die Anwesenheit einer dritten Partei erhöht die Kosten und verringert die Effizienz des Protokolls und ist deshalb nicht wünschenswert. Protokolle, die sich einer dritten Partei bedienen, sind unter anderem in [13], [16] und [21] beschrieben.

In letzter Zeit ist man deshalb vermehrt zu optimistischen Protokollen übergegangen. Optimistische Protokolle bedienen sich einer offline-TTP, die auch als vertrauenswürdig und immer erreichbar angesehen wird, in der Regel jedoch nicht in das Protokoll eingreift. Nur wenn das Protokoll nicht ordnungsgemäß

abläuft, etwa weil eine der beteiligten Personen zu betrügen versucht oder weil ein technisches Problem aufgetreten ist, wenden sich die beteiligten Parteien an die TTP, um das Protokoll fair zu beenden.
Bei diesem Ansatz tauschen A und B zunächst Werte aus, deren Richtigkeit vom jeweils anderen Teilnehmer überprüft werden kann, aus denen gleichzeitig aber keinerlei Informationen über die Geheimnisse S_A bzw. S_B selbst gewonnen werden können. Unter Beteiligung der TTP können aus diesen Werten (im Falle eines Disputs) die gewünschten Geheimnisse rekonstruiert werden. Im Anschluß werden dann die Geheimnisse selbst ausgetauscht. Sind beide Parteien ehrlich, so muß die TTP nicht eingreifen, da am Ende des Protokolls jeder Teilnehmer das Geheimnis des jeweils anderen erhalten hat. Ein weiterer Vorteil besteht darin, daß die Geheimnisse als ganzes ausgetauscht werden und der Kommunikationsaufwand gegenüber dem blockweisen Tausch erheblich geringer ist. Asokan, Shoup und Waidner stellen in [1] ein optimistisches Protokoll vor, das auf einem Verifiable Encryption Protokoll beruht. Liqun Chen [12] stellt ein Protokoll vor, das sich Chaums *designated confirmer signatures* (siehe [11]) bedient.

Im folgenden Kapitel wird ein Protokoll von Asokan, Shoup und Waidner [1] genauer vorgestellt. Es wird sich herausstellen, daß das darin benutzte Verifiable Encryption Protokoll ineffizient ist. Es ist jedoch möglich, die in Kapitel 4 vorgestellten Verifiable Encryption Protokolle in das Fair Exchange Protokoll von [1] zu integrieren und damit dieses Manko zu beseitigen.

5.1 Ein Ansatz mittels Offline-TTP

Asokan, Shoup und Waidner stellen in [1] ein Protokoll zum fairen Tausch digitaler Unterschriften vor. Dieses Protokoll ist optimistisch, d.h. es bedient sich einer offline-TTP, die nur dann eingreift, wenn das Protokoll aus irgendeinem Grund nicht ordnungsgemäß zu Ende geführt wurde.

Zunächst wird mittels eines Secure Reduction Schemes das Problem des fairen Tausches digitaler Signaturen auf das Problem des Tauschs zweier Urbilder von öffentlich bekannten Homomorphismen θ_1 und θ_2 reduziert. Aus diesen Urbildern kann der Empfänger dann die gewünschte Signatur rekonstruieren.

Das Herzstück des Protokolls aus [1] ist ein Verifiable Encryption Protokoll. Das dort vorgestellte Protokoll weist jedoch noch Schwächen auf. Es benutzt eine Cut-and-Choose-Technik. Deshalb muß das Protokoll n mal (parallel) ausgeführt werden, um die Betrugswahrscheinlichkeit des Provers auf 2^{-n} zu drücken. Wünschenswert wäre ein Protokoll, das ohne Cut-and-Choose auskommt. Diesen Schwachpunkt werden wir gegen Ende des Kapitels beheben, indem wir das in [1] vorgestellte VEP durch die in Kapitel 4 beschriebenen ersetzen.

Das eigentliche Fair Exchange Protokoll bedient sich dieser beiden Bausteine (Reduktionsschema und Verifiable Encryption Protokoll). Es gewährt Anonymität der am Tausch beteiligten Personen gegenüber der TTP. Diese Anonymität ist bei Bezahlvorgängen wichtig, bei denen keine eigenen Signaturen sondern von einer Bank signierte Geldstücke ausgetauscht werden. Ist

Anonymität nicht erwünscht, so kann das Fair Exchange Protokoll wesentlich effizienter gestaltet werden (siehe Abschnitt 5.1.5).

Asokan, Shoup und Waidner stellen ebenfalls eine Version ihres Protokolls vor, das mit Off-Line Coupons arbeitet (siehe Abschnitt 5.1.4). Die am Tausch beteiligten Personen müssen sich vor dem Tausch an die TTP wenden und sich einen solchen Off-Line Coupon ausstellen lassen. Das VEP benötigt dann nur noch eine Runde. Ein Coupon darf nur einmal verwendet werden. Nachteil ist, daß die TTP nun wieder (indirekt) am Protokoll teilnimmt.

5.1.1 Das Secure Reduction Scheme

Das Secure Reduction Scheme dient dazu, das Problem des fairen Tausches digitaler Signaturen auf das Problem des Tausches von Urbildern öffentlich bekannter Homomorphismen zu reduzieren.

Teilnehmer A berechnet (in einem Algorithmus *reduce*) zunächst aus seinen öffentlichen Signaturparametern, der signierten Nachricht und der Signatur zwei Werte s und d, für die $\theta(s) = d$ unter einem öffentlich bekannten Einweg-Homomorphismus θ gilt. Der Homomorphismus θ hängt vom verwendeten Signaturverfahren und den Signaturparametern von A ab. Den Wert d sendet er an B.

B überprüft (in einem Algorithmus *verify*), ob er einen korrekten Wert bekommen hat, d.h. ob er aus dem Urbild s von d bzgl. θ die gewünschte Signatur (mittels des Algorithmus *recover*) rekonstruieren kann.

Definition 5.1 (Secure Reduction Scheme)
Sei Σ ein Signaturverfahren.
Ein reduction scheme ordnet jedem öffentlichen Schlüssel PK einen effizient berechenbaren Gruppenhomomorphismus $\theta : G_1 \rightarrow G_2$ zu und besteht aus folgenden Algorithmen:

1. ***reduce***
   Eingabe: öff. Schl. PK, Nachricht m und Signatur σ
   Ausgabe: $d \in G_2, c \in \{0,1\}^*$ und $s \in \theta^{-1}(d)$
   Der Wert c enthält zusätzliche Informationen für die Algorithmen verify und recover.
   reduce darf für wenige Ausnahmen fehlschlagen (wird bei DSA-Signaturen benötigt).

2. ***verify***
   Eingabe: PK, m, d und c
   Ausgabe: accept oder reject

3. ***recover***
   Eingabe: PK, m, c und $s \in G_1$
   Ausgabe: $\hat{\sigma}$

Ein reduction scheme heißt sicher (secure), wenn es folgende Eigenschaften erfüllt:

Completeness
Wenn reduce(PK,m,σ)=(d,c,s), dann gilt verify(PK,m,d,c)=accept.

Soundness
Ein Angreifer kann nur mit vernachlässigbar kleiner Wahrscheinlichkeit Werte PK, m, d und c finden, für die verify(PK,m,d,c)=accept aber recover(PK,m,c,s) keine gültige Signatur von m ist, und zwar für alle $s \in \theta^{-1}(d)$.

Secrecy
Ein Angreifer kann folgendes Spiel nicht gewinnen:

B1 *Der Schlüsselerzeugungsalgorithmus wird ausgeführt.*
Der geheime Schlüssel wird an das Signaturorakel S und der öffentliche Schlüssel PK an den Angreifer gegeben.

B2 *Der Angreifer darf S beliebige Fragen stellen.*

B3 *Der Angreifer erzeugt eine Nachricht m, die verschieden von denen in B2 ist.*
S erzeugt eine Signatur σ von m unter PK und gibt dem Angreifer die Werte d und c, wobei (d,c,s)=reduce(PK,m,σ).

B4 *Der Angreifer darf S beliebige Fragen (ungleich m) stellen.*

Der Angreifer gewinnt das Spiel, wenn er eine gültige Signatur von m erzeugen kann.

Beispiel 5.2 (Reduktion für das Schnorr-Schema)
Sei (c, s) eine Schnorr-Signatur der Nachricht m unter dem Schlüsselpaar $(x, y = g^x)$. Zur Verifikation der Signatur überprüft ein Empfänger, ob die Gleichung $c = h(m, g^s y^{-c})$ erfüllt ist.
Der Homomorphismus θ und die Algorithmen reduce, verify und recover sehen im Fall der Schnorr-Signatur folgendermaßen aus:

- θ : $\mathbb{Z}_q \to \mathbb{Z}_p^*, \qquad x \mapsto g^x \bmod p$
- *recover:* $(c, s) \mapsto (c, \theta(s)) = (c, g^s \bmod p)$
- *verify:* $c = h(m, \theta(s) y^{-c})$
- *recover:* $c \mapsto (c, z)$

5.1.2 Das Verifiable Encryption Scheme

Definition 5.3 ***Verifiable Encryption Scheme***
Ein Verifiable Encryption Scheme besteht aus einem Schlüsselerzeugungs-Algorithmus, einen Prover P, einem Verifier V, einem Entschlüsselungs-Algorithmus D und einem Recovery-Algorithmus R.
P und V haben $d \in G_2$, $x \in \{0,1\}^*$ und PK als gemeinsame Eingabe.
P hat ferner $s \in \theta^{-1}(d)$ als geheime Eingabe.
Am Ende des Protokolls wird V entweder akzeptieren und einen Wert α ausgeben, oder verwerfen.
Der Wert α kann an D und anschließend an R gegeben werden, um s (mit $\theta(s) = d$) zu erhalten.

Ein Verifiable Encryption Scheme heißt sicher (secure), wenn es folgende Bedingungen erfüllt:

Completeness
Wenn P und V ehrlich sind, so wird V für alle s,d,x mit $\theta(s) = d$ akzeptieren.

Soundness
Für alle d und x und alle Prover P^* gilt:
Wenn V akzeptiert und einen Wert α ausgibt, dann gilt $\theta(R(D(\alpha, x))) = d$ mit überwältigender Wahrscheinlichkeit.

Zero-Knowledge
Folgendes Spiel wird gegen einen Angreifer gespielt:

C1 *Der Schlüsselerzeugungs-Algorithmus wird gestartet. Der private Schlüssel wird an D und der öffentliche an den Angreifer gegeben.*

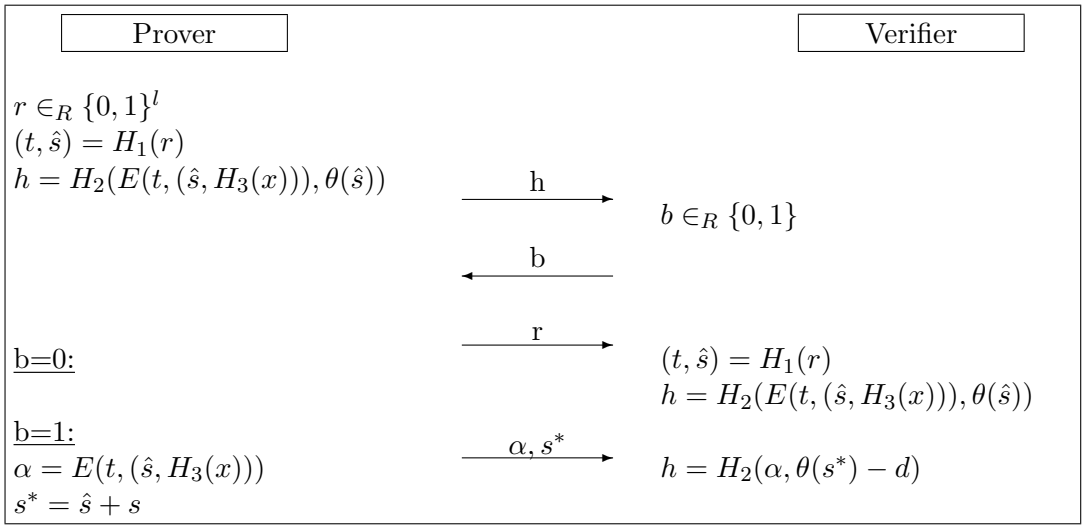
C2 *Der Angreifer darf beliebige Anfragen $D(\hat{\alpha}, \hat{x})$ stellen.*

C3 *Der Angreifer generiert s,d und x mit $\theta(s) = d$ und gibt s,d und x an P zusammen mit dem öffentlichen Schlüssel.*

C4 *Der Angreifer macht beliebige Anfragen an D und P, aber nach seiner ersten Anfrage an P darf er D nicht mit Label x befragen.*

Ein Simulator ist eine Maschine, die die Rollen des Schlüsselerzeugungs-Algorithmus, des Entschlüsselungs-Orakels D und des Provers P übernimmt. Er bekommt als Eingabe jedoch nur d und x (und nicht s).
Im Random Oracle Modell generiert der Simulator außerdem die Ausgaben der Hashfunktion.
Zero-Knowledge bedeutet, daß ein Simulator $\hat{P}$ existiert, so daß der Angreifer nicht zwischen echten und simulierten Spielen unterscheiden kann.

Das Verifiable Encryption Protokoll aus [1]



Das Protokoll wird n-mal parallel ausgeführt. Der Verifier akzeptiert genau dann, wenn er in jeder der n Runden akzeptiert.
Es verwendet die Cut-and-Choose Technik, d.h. in jeder Runde überprüft der Verifier genau eine von zwei Bedingungen. In unserem Fall entweder, ob h mit dem korrekten $\hat{s}$ gebildet wurde (im Fall $b = 0$) oder ob das s^* richtig gebildet wurde (im Fall $b = 1$).
Der Prover kann betrügen, wenn er die Bits raten kann, die der Verifier ihm senden wird.
Er hat also eine Betrugswahrscheinlichkeit von 2^{-n}. Da online geantwortet werden muß, erachten die Autoren eine Rundenanzahl von $n = 40$ als ausreichend.

5.1.3 Das Fair Exchange Protokoll

E1 A berechnet $reduce(PK_A, m_A, \sigma_A) = (d_A, c_A, s_A)$ und sendet d_A, c_A an B.

E2 B überprüft, ob $verify(PK_A, m_A, d_A, c_A) = accept$. Wenn nicht, so bricht B ab.
B berechnet $reduce(PK_B, m_B, \sigma_B) = (d_B, c_B, s_B)$ und sendet d_B, c_B an A.

E3 A überprüft, ob $verify(PK_B, m_B, d_B, c_B) = accept$. Wenn nicht, so bricht A ab.
A wählt $r \in_R DefBereich(f)$ und berechnet $v = f(r)$.
A und B durchlaufen nun das Verifiable Encryption Protokoll mit A als Prover und B als Verifier, d.h. A sendet ein verified encrypt α von s_A mit Label $(v, d_B, desc(\theta(B)))$ an B.

E4 Akzeptiert B den Beweis in E3 nicht, so bricht B ab.
Ansonsten durchlaufen A und B wieder das Verifiable Encryption Protokoll, diesmal aber mit B als Prover und A als Verifier, d.h. B sendet ein verified encrypt β von s_B mit Label v an A.

E5 Akzeptiert A den Beweis aus E4 nicht, so wendet er sich im Unterprotokoll **abort** an die TTP.
Ansonsten sendet er s_A an B.

E6 B überprüft, ob $\theta_A(s_A) = d_A$.
Ist dies nicht der Fall, so wendet sich B im Unterprotokoll **B-resolve** an die TTP.
Andernfalls sendet B den Wert s_B an A, gibt $recover(PK_A, m_A, c_A, s_A)$ aus und hält.

E7 A überprüft, ob $\theta_B(s_B) = d_B$.
Ist dies nicht der Fall, so wendet sich A im Unterprotokoll **A-resolve** an die TTP.
Andernfalls gibt A den Wert $recover(PK_B, m_B, c_B, s_B)$ aus und hält.

Unterprotokoll 1 abort

A sendet r, d_B und $desc(\theta_B)$ an die TTP, die folgendes macht:

if $(deposit, f(r), d_B, s_B, desc(\theta_B)) \in S$ **then**
  sende s_B an A
else if $(no - abort, f(r)) \in S$ **then**
  sende „Abbruch nicht erlaubt“ an A
else
  füge $(abort, f(r))$ der Liste S hinzu
  sende „Tausch abgebrochen“ an A
end if

Unterprotokoll 2 B-resolve

B sendet α, v, s_B und $desc(\theta_B)$ an die TTP, die folgendes macht:

if $(abort, v) \in S$ **then**
  sende „Tausch abgebrochen“ an B
else
  füge $(deposit, v, d_B, s_B, desc(\theta_B))$ der Liste S hinzu
  Entschlüssel α bzgl. des Labels $(v, \theta_B(s_B), desc(\theta_B))$
  und sende das Ergebnis an B
end if

Unterprotokoll 3 A-resolve

A sendet β und r an die TTP, die folgendes macht:

if $(abort, f(r)) \in S$ **then**
  sende „Tausch abgebrochen“ an A
else
  füge $(no - abort, f(r))$ der Liste S hinzu
  Entschlüssel β bzgl. des Labels $f(r)$
  und sende das Ergebnis an A
end if

Anschaulich kann man sich diese Unterprotokolle so vorstellen, daß die TTP mehrere Listen führt. In einer Liste sind die Anfragen aufgelistet, die entschlüsselt und beantwortet wurden (das sind die Einträge aus S, deren erste Komponente „deposit“ lautet). Die zweite Liste enthält die Protokollausführungen (durch das zugehörige Label identifiziert), die abgebrochen wurden und für die keine Antworten mehr gegeben werden dürfen („abort“ als ersten Eintrag). Die dritte Liste enthält die Protokollausführungen, die auf keinen Fall auf die Abbruchliste gesetzt werden dürfen („no-abort“ als ersten Eintrag).

Generell gibt es zwei Betrugsmöglichkeiten. Eine der beteiligten Personen bricht das Fair Exchange Protokoll vorzeitig ab oder eine der Personen wendet sich mit einem der Unterprotokolle an die TTP, obwohl dies das Fair Exchange Protokoll zu diesem Zeitpunkt nicht vorsieht.
B hat grundsätzlich nur die Möglichkeit, sich mit dem Unterprotokoll B-resolve an die TTP zu wenden. Das kann B sofort nach Ausführung von Schritt E3 machen. B selbst hat zu diesem Zeitpunkt zwar noch nichts brauchbares an A gesendet, muß jedoch sein Geheimnis s_B bei der TTP hinterlegen, um Antwort von der TTP zu bekommen. Dieses Geheimnis kann sich A später (etwa wenn er keine Antwort mehr von B bekommt oder B ihm falsche Werte schickt) im Unterprotokoll abort von der TTP holen.
Bekommt A nach dem Schritt E3 keine Antwort (oder falsche Werte) von B, so hat sich B entweder schon an die TTP gewendet (siehe letzter Absatz) oder noch nicht. Im zweiten Fall läßt A diese Protokollausführung sperren (das Label wird auf die Sperrliste gesetzt). Wendet sich B später an die TTP, so stellt die TTP fest, daß das Label auf der Sperrliste steht und wird B's Anfrage nicht beantworten. Keine der Parteien hat also etwas erhalten.
A kann sich mit dem Protokoll A-resolve frühestens nach Schritt E4 an die TTP wenden. Bekommt er eine Antwort von der TTP, so wird das Label auf die Liste der Protokollausführungen gesetzt, die keinesfalls gesperrt werden dürfen, um B ein Einlösen des (schon erhaltenen) Wertes α bei der TTP zu ermöglichen. Hat A bereits das Label auf die Sperrliste setzen lassen, um B zu blockieren, und wendet sich nachher (in A-resolve) an die TTP, so erhält er keine Antwort.

Die Sicherheit des Fair Exchange Protokolls wird in [1] formal mit folgendem Satz bewiesen.

Satz 5.4 *Unter der Annahme, daß f eine Einweg-Funktion ist und daß die zugrundeliegenden reduction und verifiable encryption Schemata sicher sind (im Random Oracle Modell), ist das vorgestellte Fair Exchange Protokoll sicher (im Random Oracle Modell).*

5.1.4 Verifiable Encryption with Off-Line Coupons

Asokan, Shoup und Waidner stellen in [1] außerdem eine Variante dieses Protokolls mit einem effizienteren Verifiable Encryption Protokoll vor. Allerdings muß bei dieser Variante im Vorfeld jeder Protokollausführung die TTP kontaktiert werden, die sogenannte Off-Line Coupons ausstellt. Diese Coupons können auch gebündelt ausgegeben werden, d.h. bei einem Kontakt mit der TTP läßt sich ein Teilnehmer gleich mehrere Coupons ausstellen, so daß er vor den folgenden Protokollausführungen die TTP nicht mehr kontaktieren muß.
Außer dem Schlüsselpaar für das asymmetrische Verschlüsselungsschema E habe die TTP zusätzlich ein Schlüsselpaar für ein Signaturschema. Der öffentliche Signaturschlüssel sei beiden Teilnehmern bekannt.
Wendet sich einer der Teilnehmer mit der Bitte um Ausstellung eines Off-Line Coupons an die TTP, so tut diese folgendes:

- Die TTP wählt zufällig einen Wert $\hat{s} \in_R G_1$.
  Dann berechnet sie $\hat{d} = \theta(\hat{s})$ und $\alpha = E(\hat{s})$.
- Die TTP generiert ein Schlüsselpaar (PK, SK) für das Signaturschema.
- Die TTP unterschreibt mit ihrem geheimen Schlüssel den Coupon, d.h. das Tupel $(\hat{d}, \alpha, PK)$.
- Die TTP sendet den unterschriebenen Coupon mitsamt $\hat{s}$ und SK an den Teilnehmer.

Mit ihrer Unterschrift garantiert die TTP, daß in α wirklich ein Urbild von $\hat{d}$ verschlüsselt wurde.

Bei Benutzung dieser Coupons vereinfacht sich das Verifiable Encryption Protokoll dann zu:

- Der Prover berechnet $s^* = \hat{s} + s$ und unterschreibt mit SK das Tupel (α, x).
  Anschließend sendet der Prover diese beiden Werte sowie den von der TTP unterschriebenen Coupon $(\hat{d}, \alpha, PK)$ an den Verifier.
- Der Verifier überprüft die Signatur der TTP unter dem Coupon und anhand des im Coupon enthaltenen öffentlichen Schlüssels die Signatur des Verifiers des Paares (α, x).
  Sind beide Signaturen korrekt, so prüft er ferner, ob $\theta(s^*) = d + \hat{d}$ gilt.
  Ist diese Gleichung erfüllt, so wird der Verifier akzeptieren.

Wendet sich eine Person mit der Bitte um Entschlüsselung an die TTP, so muß zusätzlich der Coupon $(\hat{d}, \alpha, PK)$ samt Unterschrift der TTP und das Paar (α, x) samt Unterschrift des Provers vorgelegt werden. Die TTP wird nur antworten, wenn beide Signaturen korrekt sind.

Diese Konstruktion ist sicher, solange Coupons nur ein einziges Mal benutzt werden.
Die Unterschrift des Paares (α, x) dient dem Zweck, das Label x eindeutig an die Protokollausführung zu binden.
Im allgemeinen kann ein ausgestellter Coupon nur für ein Signaturverfahren und meist auch nur von einer bestimmten Person benutzt werden. Dies liegt daran, daß der Homomorphismus θ sowie dessen Urbildgruppe G_1 vom verwendeten Signaturverfahren und oft sogar vom öffentlichen Schlüssel des Teilnehmers (z.B. bei RSA-Signaturen) abhängt.

5.1.5 Ergänzungen

Ein effizienteres Fair Exchange Protokoll ohne Anonymität gegenüber der TTP

Ein Merkmal der Arbeit von Asokan, Shoup und Waidner ist, daß Anonymität der beteiligten Personen gegenüber der TTP besteht. Ist diese Anonymität nicht unbedingt erforderlich, so kann das Fair Exchange Protokoll so gestaltet werden, daß das Verifiable Encryption Protokoll nur einmal durchlaufen werden muß.
Bei dieser Version kann auf Labels und die Unterprotokolle abort, A-resolve und B-resolve verzichtet werden. Seien σ_A die Signatur von A der Nachricht m_A und σ_B die Signatur von B der Nachricht m_B, so muß jedoch eine Bindung zwischen $\alpha = E(s_A)$ und m_B gewährleistet sein. Dies ist nötig für den Fall, daß B sich an die TTP wendet. Um den Wert α entschlüsselt zu bekommen, muß B nämlich die Signatur σ_B hinterlegen, die von der TTP an A weitergeleitet wird. Diese Bindung wird durch A's Signatur des Paares (α, m_B) realisiert.

E1 A berechnet $reduce(PK_A, m_A, \sigma_A) = (d_A, c_A, s_A)$ und sendet d_A, c_A an B.

E2 B überprüft, ob $verify(PK_A, m_A, d_A, c_A) = accept$. Wenn nicht, so bricht B ab.
A und B durchlaufen nun das Verifiable Encryption Protokoll mit A als Prover und B als Verifier, d.h. A sendet ein verified encrypt α von s_A an B.
Außerdem sendet A noch eine Unterschrift des Paares (α, m_B).

E3 B überprüft die Signatur des Paares (α, m_B). Ist die Signatur nicht korrekt oder akzeptiert B im Verifiable Encryption Protokoll nicht, so bricht er ab.
Ansonsten sendet er s_B an A.

E4 A überprüft, ob $\theta(s_B) = d_B$.
Ist dies nicht der Fall, so bricht A das Protokoll ab.
Ansonsten sendet er s_A an B und berechnet mittels des Algorithmus recover die gewünschte Signatur σ_B.

E5 B überprüft, ob $\theta(s_A) = d_A$.
Ist dies nicht der Fall, so wendet sich B an die TTP. Ansonsten berechnet er mittels des Algorithmus recover die gewünschte Signatur σ_A und hält.

Wendet sich A an die TTP, so muß er das Paar (α, m_B) samt Signatur von A und σ_B an die TTP senden. Die TTP überprüft die Signatur. Ferner überprüft sie, ob σ_B eine korrekte Signatur von B des Wertes m_B darstellt. Sind beide Bedingungen erfüllt, so entschlüsselt sie den Wert α und sendet das Ergebnis an B. Den Wert σ_B leitet sie weiter an A.

B hat zwei Möglichkeiten zu betrügen. Entweder sendet er in Schritt E3 keine oder falsche Werte an A, oder er wendet sich direkt nach Ablauf des VEP in Schritt E2 an die TTP. In beiden Fällen muß er sich an die TTP wenden, um aus den ihm bekannten Werten die gewünschte Signatur σ_A zu erhalten. Die TTP verlangt allerdings seine Signatur σ_B, die sie dann an A weiterleitet. Beide Teilnehmer haben also die gewünschte Signatur erhalten.
A kann nur betrügen, indem er im Laufe des Protokolls falsche Werte sendet. Macht er das in den ersten beiden Schritten, so wird B nicht akzeptieren und das Protokoll abbrechen. Macht er das in Schritt E4, so kann B sich mit dem Wert α an die TTP wenden und erhält die gewünschte Signatur.
Diese Version des Fair Exchange Protokolls wird in etwas abgewandelter Form von Liqun Chen in [12] benutzt.

5.2 Integration des neuen VEP

Das in [1] benutzte VEP kann nicht ohne weiteres durch die in Kapitel 4 beschriebenen VEPs ersetzt werden. Zum einen muß ein Label an das verifiable encrypt gebunden werden. Zum anderen muß beim VEP für additive Urbildgruppen verhindert werden, daß ein Teilnehmer einen Wert x größer p (mit dem Okamoto/Uchiyama-Schema) verschlüsselt, diese Verschlüsselungen an die TTP sendet und sich von ihr den Wert $x \, mod \, p$ zurückliefern läßt (siehe Kapitel 4).
Die folgenden Überlegungen beziehen sich auf das VEP für additive Urbildgruppen. Das VEP für multiplikative Urbildgruppen könnte vollkommen analog auch in das Fair Exchange Protokoll integriert werden. Bei den existierenden Reduktions-Schemata, die einen Homomorphismus θ mit multiplikativer Urbildgruppe liefern, hängen θ und die Urbildgruppe G_1 jedoch nicht nur vom verwendeten Signaturverfahren sondern auch vom Schlüsselpaar des Benutzer ab (bei RSA z.B. vom Modul n). Da in unserem Fall aber die Parameter des Schlüsselpaares der TTP auf die Parameter der Gruppe G_1 abgestimmt werden müssen, bräuchte die TTP für jeden Benutzer ein eigenes Schlüsselpaar, was nicht praktikabel ist.

5.2.1 Das Unterprotokoll „Check-Size“

Das Okamoto/Uchiyama-Schema (und damit auch das Verifiable Encryption und das Fair Exchange Protokoll) ist unsicher, wenn ein Teilnehmer einen Wert $x > p$ verschlüsselt, ihn der TTP vorlegt und von ihr $x \bmod p$ zurückgeliefert bekommt (siehe Abschnitt 4.3.2). Da die Primzahl p Teil des geheimen Schlüssels ist, die Bitlänge k von p (d.h. $2^{k-1} < p \leq 2^k$) aber allen bekannt ist, fordern wir, daß für verschlüsselte Nachrichten $m < 2^{k-1}$ gilt.

Da die TTP nach Entschlüsselung eines Ciphertextes nicht mehr überprüfen kann, ob die verschlüsselte Nachricht kleiner als 2^{k-1} war, fordern wir, daß ein Teilnehmer, der sich mit der Bitte um Entschlüsselung an die TTP wendet, die Werte $\alpha = E^*(s, t_1)$, $\beta = E^*(x, t_2)$, c und s^* vorlegen muß. Bei Erhalt entschlüsselt die TTP die Werte α und β und erhält $s \bmod p$ und $x \bmod p$. Anschließend überprüft sie, ob $s^* = (s \bmod p) + c \cdot (x \bmod p)$ (in $\mathbb{Z}$) gilt. Gelingt es einem Teilnehmer, einen Wert $x > p$ zu finden, der diese Gleichung erfüllt, so kann er daraus bereits die Primzahl p berechnen. Die TTP kann sich in diesem Fall also sicher sein, daß ein Wert $x < p$ verschlüsselt wurde.
Wünschenswert ist es, auch ein „Herantasten“ 0.05cm an p zu verhindern. Unter Herantasten verstehen wir hier, daß Werte $x > 2^{k-1}$ verschlüsselt und der TTP vorgelegt werden. Je nachdem, ob die TTP antwortet oder nicht, kann man folgern, ob $x > p$ gilt oder nicht. Mit dieser Vorgehensweise ließe sich mit jedem neuen Durchgang ein Bit der Primzahl p ermitteln. Aus diesem Grund wird, nachdem sichergestellt ist ,daß $x < p$ gilt, auch noch geprüft, ob $x < 2^{k-1}$ gilt.
Zusammenfassend stellt sich dieser Test also folgendermaßen dar:

Unterprotokoll 4 Check-Size

Eingaben: $\alpha = E^*(s, t_1)$
$\beta = E^*(x, t_2)$
c, s^*

- Entschlüssele die Werte α und β.
- Teste für die resultierenden Werte $s \bmod p$ und $x \bmod p$, ob die Gleichung $s^* = (s \bmod p) + c(x \bmod p)$ (in $\mathbb{Z}$) erfüllt ist.
  Ist die Gleichung nicht erfüllt, so gib „unkorrekt“ aus.
- Überprüfe, ob $(x \bmod p) = x < 2^{k-1}$ gilt.
  Gilt dies nicht, so gib „unkorrekt“ aus.
  Ansonsten gib „korrekt“ aus.

5.2.2 VEP mit Labels

Für die Sicherheit des Fair Exhange Protokolls nach [1] ist es notwendig, daß an jede Protokollausführung ein eindeutiger Wert gebunden wird. Diesen Wert werden wir im folgenden als Label bezeichnen.
Prover und Verifier müssen sich vor Protokollausführung auf dieses Label einigen und keiner der beiden darf in der Lage sein, das Label im Nachhinein zu ändern. Ferner sollte das Label neben den öffentlichen Werten d_A und d_B Beschreibungen $desc(\theta_A)$ und $desc(\theta_B)$ der benutzten Homomorphismen sowie zwei Zufallswerte r_A und r_B enthalten, die verhindern sollen, daß zwei Protokollausführungen dasselbe Label tragen.
Anhand dieses Labels identifiziert die TTP die Protokollausführung und entscheidet, ob sie vorgelegte Werte entschlüsseln darf oder nicht.
Wir werden die Schnorr-Signatur im signierten Okamoto/Uchiyama-Verfahren nutzen, um das Label L an den Wert $\alpha = E^*(s, t_2)$ zu binden. Der Verschlüsselungsvorgang von s bzgl. des Zufallswertes t_2 sei hier noch einmal dargestellt:

- Berechne $E(s) = g^s \, mod \, n$
- Wähle $r \in_R \mathbb{Z}_n^*$
- Berechne $k = g^r \, mod \, n$
- $c = H((C, L), k)$
- $z = r + cm \, mod \, n$

Das Tripel (C, c, z) ist Verschlüsselung von s im signierten Okamoto/Uchiyama-Verfahren bzgl. des Zufallswertes t_1.
Anstatt den Ciphertext C vom s im („normalen “ nicht-randomisierten) Okamoto/Uchiyama-Verfahren wird in diesem Fall also das Paar (C, L), bestehend aus dem Ciphertext C und dem Label L, signiert. Die Sicherheit der Schnorr-Signatur garantiert, daß das Label im Nachhinein nicht mehr geändert werden kann. Da das Label L ein öffentlicher Wert ist, kann trotzdem jeder die Gültigkeit der angehängten Schnorr-Signatur (c, z) überprüfen, indem er testet, ob $c = H((C, L), \, g^z (g^m)^{-c} \, mod \, n)$ erfüllt ist.

5.2.3 Zusammenfassung

Mit obigen Überlegungen haben das Fair Exchange Protokoll und seine Unterprotokolle folgende Form:

E1 A berechnet $reduce(PK_A, m_A, \sigma_A) = (d_A, c_A, s_A)$ und sendet d_A, c_A an B.

E2 B überprüft, ob $verify(PK_A, m_A, d_A, c_A) = accept$. Wenn nicht, so bricht B ab.
B berechnet $reduce(PK_B, m_B, \sigma_B) = (d_B, c_B, s_B)$ und sendet d_B, c_B an A.

E3 A überprüft, ob $verify(PK_B, m_B, d_B, c_B) = accept$. Wenn nicht, so bricht A ab.
A und B durchlaufen nun das Verifiable Encryption Protokoll mit A als Prover und B als Verifier für die Eingabe s_A und das Label $L = (r_A, r_B, d_A, d_B, desc(\theta_A), desc(\theta_B))$.

E4 Akzeptiert B den Beweis in E3 nicht, so hält er an.
Ansonsten durchlaufen A und B wieder das Verifiable Encryption Protokoll, diesmal aber mit B als Prover und A als Verifier, für die Eingabe s_B und das Label L.

E5 Akzeptiert A den Beweis aus E4 nicht, so wendet er sich an das Unterprotokoll **abort**.
Ansonsten sendet er s_A an B.

E6 B überprüft, ob $\theta_A(s_A) = d_A$.
Ist dies nicht der Fall, so wendet sich B an das Unterprotokoll **B-resolve**.
Andernfalls sendet B den Wert s_B an A, gibt $recover(PK_A, m_A, c_A, s_A)$ aus und hält.

E7 A überprüft, ob $\theta_B(s_B) = d_B$.
Ist dies nicht der Fall, so wendet sich A an das Unterprotokoll **A-resolve**.
Andernfalls gibt A den Wert $recover(PK_B, m_B, c_B, s_B)$ aus und hält.

Unterprotokoll 5 abort

A sendet L und d_B an die TTP, die folgendes macht:

if $(deposit, L, d_B, s_B) \in S$ **then**
  sende s_B an A
else if $(no - abort, L) \in S$ **then**
  sende „Abbruch nicht erlaubt“ an A
else
  füge $(abort, L)$ der Liste S hinzu
  sende „Tausch abgebrochen“ an A
end if

Unterprotokoll 6 B-resolve

B sendet L, s_B und die Werte (α, β, c, s^*) aus dem VEP an die TTP, die folgendes macht:

if $(abort, L) \in S$ **then**
  sende „Tausch abgebrochen“ an B
else
  if Check-Size(α, β, c, s^*)=unkorrekt **then**
    Abbruch des Protokolls
  else if Check-Label(α, L)=unkorrekt **then**
    Abbruch des Protokolls
  else
    füge $(deposit, L, d_B, s_B)$ der Liste S hinzu
    Entschlüssel α bzgl. des Labels L und sende das Ergebnis an B
  end if
end if

Unterprotokoll 7 A-resolve

A sendet L und die Werte (α, β, c, s^*) aus dem VEP an die TTP, die folgendes macht:

if $(abort, L) \in S$ **then**
  sende „Tausch abgebrochen“ an A
else
  if Check-Size(α, β, c, s^*)=unkorrekt **then**
    Abbruch des Protokolls
  else if Check-Label(α, L)=unkorrekt **then**
    Abbruch des Protokolls
  else
    füge $(no - abort, L)$ der Liste S hinzu
    Entschlüssel α bzgl. des Labels L
    und sende das Ergebnis an A
  end if
end if

Unterprotokoll 8 Check-Size

Eingaben: $\alpha = E^*(s, t_1)$
$\beta = E^*(x, t_2)$
c, s^*

- Entschlüssele die Werte α und β.
- Teste für die resultierenden Werte $s \, mod \, p$ und $x \, mod \, p$, ob die Gleichung $s^* = (s \, mod \, p) + c(x \, mod \, p)$ (in $\mathbb{Z}$) erfüllt ist.
  Ist die Gleichung nicht erfüllt, so gib „unkorrekt“ aus.
- Überprüfe, ob $(x \, mod \, p) = x < 2^{k-1}$ gilt.
  Gilt dies nicht, so gib „unkorrekt“ aus.
  Ansonsten gib „korrekt“ aus.

Unterprotokoll 9 Check-Label

Eingaben: $\alpha = E^*(s, t_1)$
L

Der Wert $\alpha = E^*(s, t_1) = (C, (c, s))$ ist eine signierte Okamoto/Uchiyama-Verschlüsselung des Wertes s, an die zusätzlich das Label L gebunden wurde. Die TTP überprüft nun, ob das richtige Label an die Verschlüsselung gebunden wurde, indem sie überprüft, ob die Gleichung

$$c = H((C, L), g^z (g^m)^{-c} \, mod \, n)$$

erfüllt ist.

Gilt diese Gleichung, so gib „korrekt“ aus. Ansonsten gib „unkorrekt“ aus.

Literaturverzeichnis

[1] N. Asokan, V. Shoup, M. Waidner
Optimistic Fair Exchange of Digital Signatures
Advances in Cryptology - Eurocrypt '98, LNCS 1403, S. 591-606

[2] F. Bao
An Efficient Verifiable Encryption Scheme for Encryption of Discrete Logarithms
Cardis '98

[3] F. Bao, R. Deng, W, Mao
Efficient and Practical Fair Exchange Protocols with Off-line TTP
Proceedings of 1998 IEEE Symposium on Security and Privacy, Oakland, California, IEEE Computer Press (1998), S. 77-85

[4] M. Bellare, P. Rogaway
Random Oracles are Practical: a paradigm for designing efficient protocols
Proceedings of the 1st ACM Conference on Computer and Communications Security, 62-73, Fairfax (Vorgrinia, USA), 1993, ACM press

[5] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway
Relations Among Notions of Security for Public-Key Encryption Schemes
Advances in Cryptology - Crypto '98, LNCS 1462

[6] D. Boneh
Twenty Years of Attacks on the RSA Cryptosystem
Notices of the American Mathematical Society (AMS), 46(2), S. 203-213, 1999

[7] D. Boneh, G. Durfee
Cryptanalysis of RSA with private key d less than $n^{0.292}$
Advances in Cryptology - Eurocrypt '99, LNCS 1592, S. 1-11

[8] S. Brands
Untraceable off-line cash in wallets with observers
Advances in Cryptology - Crypto '93, LNCS 773, 302-318

[9] J. Camenisch, I. Damgard
Verifiable Encryption and Applications to Group Signatures and Signature Sharing
BRICS Technical Report RS-98-32

[10] R Canetti, O. Goldreich, S. Halevi
The Random Oracle Methodology, Revisited
Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC - 98), S. 209-218, ACM Press, 1998

[11] D. Chaum
Designated confirmer signatures
Advances in Cryptology - Eurocrypt '94, LNCS 950, S.86-91

[12] Liqun Chen
Efficient Fair Exchange with Verifiable Confirmation of Signatures
Advences in Cryptology - Asiacrypt '98, LNCS 1514, S. 286-299

[13] B. Cox, D. Tygar, M. Sirbu
NetBill security and transaction protocol
First USENIX Workshop on Electronic Commerce, S. 77-88, 1995

[14] Ivan Damgard
Practical and Provably Secure Release of a Secret and Exchange of Signature
Advances in Cryptology - Eurocrypt '93, LNCS 765, 200-214

[15] I. Damgard, O. Goldreich, T. Okamoto, A. Widgerson
Honest Verifier vs Dishonest Verifier in Public Coin Zero-Knowledge Proofs
Advances in Cryptology - Crypto '95, LNCS 963, S. 325-338

[16] R.H. Deng, L. Gong, A. Lazar, W. Wang
Practical protocols for certified electronic mail
Journal of Network and Systems Management, 4(3), 1996

[17] W. Diffie, M.E. Hellman
New directions in cryptography
IEEE Transactions on Information Theory, 22 (1976), S. 644-654

[18] T. ElGamal
A public key cryptosystem and a signature scheme based on discrete logarithms
Advances in Cryptology - Crypto '84, LNCS 196, S. 10-18

[19] T. ElGamal
A public key cryptosystem and a signature scheme based on discrete logarithms.
IEEE Transactions on Information Theory, 31 (1985), 469-472
Eine frühere Version ist [18].

[20] A. Fiat, A. Shamir
How to prove yourself: practical solutions to identification and signature problems
Advances in Cryptology - Crypto '86, LNCS 263, 186-194

[21] M.K. Franklin, M.K. Reiter
Fair exchange with a semi-trusted third party
4th ACM Conference on Computer and Communications Security, S. 1-5, 1997

[22] E. Fujisaki, O. Okamoto
Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations
Advances in Cryptology - Crypto '97, LNCS 1294, 16-30

[23] Oded Goldreich
Foundations of Cryptography - Fragments of a Book, Version 2.03

[24] O. Goldreich, A. Sahai, S. Vadhan
Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge
Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC - 98), S. 399-408, ACM Press, 1998

[25] S. Goldwasser, S. Micali, R. Rivest
A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks
SIAM J. Computing 17, 2 (April 1998), S. 281-308

[26] L. Guillou, J. Quisquater
A „paradoxical" identity-based signature scheme resulting from zero-knowledge
Advances in Cryptology - Crypto '88, LNCS 403, 216-231

[27] M. Jakobsson
A Practical Mix
Advances in Cryptology - Eurocrypt '98, LNCS 1403, S. 448-461

[28] J. Kilian, E. Petrank
Identity Escrow
Advances in Cryptology - Crypto '98, LNCS 1642, S. 169-185

[29] C. Mitchell, F. Piper, P. Wild
Digital signatures
G.J. Simmons (editor):„Contemporary Cryptology: The Science of Information Integrity", 325-378, IEEE Press, 1992

[30] Menezes, Oorschot, Vanstone
Handbook of Applied Cryptography
CRC Press, 1997

[31] National Institute of Standards and Technology
NIST FIPS PUB 186
Digital Signature Standard
U.S. Department of Commerce, may 1994

[32] V.I. Nechaev
Complexity of a Determinate Algorithm for the Discrete Logarithm
Mathematical Notes 55 (1994), S. 165-172

[33] T. Okamoto
Provably secure and practical identification schemes and corresponding signature schemes
Advances in Cryptology - Crypto '92, LNCS 740, 31-53

[34] T. Okamoto, S. Uchiyama
A New Public-Key Cryptosystem as Secure as Factoring
Advances in Cryptology - Eurocrypt '98, LNCS 1403, S. 306 ff.

[35] H. Ong, C.P. Schnorr
Fast signature generation with a Fiat Shamir-like scheme
Advances in Cryptology - Eurocrypt '90, LNCS 473, 432-440

[36] R.L. Rivest, A. Shamir, L.M. Adleman
A method for obtaining digital signatures and public-key cryptosystems
Communications of the ACM, 21 (1978), 120-126

[37] C.P. Schnorr
Efficient identification and signatures for smart cards
Advances in Cryptology - Crypto '89, LNCS 435, S. 239-252

[38] C.P. Schnorr
Efficient signature generation by smart cards
Journal of Cryptology, 4 (1991), 161-174

[39] C.P. Schnorr, M. Jakobsson
Security of Discrete Log Cryptosystems in the Random Oracle + Generic Model
Manuskript, Bell Laboratories, Murray Hill (New Jersey, USA), 1998

[40] V. Shoup
Lower Bounds for Discrete Logarithms and Related Problems
Advances in Cryptology - Eurocrypt '97, LNCS 1233, S. 256-266

[41] M. Stadler
Publicly Verifiable Secret Sharing
Advances in Cryptology - Eurocrypt '96, LNCS 1070, S.190-199

[42] D.R. Stinson
Cryptography: Theory and Practice
CRC Press, Boca Raton, Florida, 1995

[43] Y. Tsiounis, M. Yung
On the Security of ElGamal Based Encryption
Public Key Cryptography '98, LNCS 1431, Springer-Verlag 1998, S. 117-134

[44] M. Wiener
Cryptanalysis of short RSA secrets exponents
IEEE Transactions on Information Theory, 36 (1990), S. 553-558

Index