TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Lehrstuhl für Datenschutz und Datensicherheit

Dudle: Mehrseitig sichere Web 2.0-Terminabstimmung

Dissertation

zur Erlangung des akademischen Grades eines Doktoringenieurs (Dr.-Ing.)

Benjamin Kellermann

geboren am 29. März 1980 in Dresden

Betreuende Hochschullehrer:
Prof. Dr. rer. nat. Andreas Pfitzmann († 23. September 2010)
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Dresden, im September 2011

1. Gutachter: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
2. Gutachter: Prof. Dr. rer. nat. Michael Waidner

Disputation am 16. Dezember 2011

Danksagung

Besonderer Dank gilt Andreas Pfitzmann, welcher mir die Möglichkeit gab diese Promotion zu schreiben. Seine Tür war jederzeit offen und war stets ein angenehmer und unkonventioneller weitblickender Doktorvater.

Alexander Schill danke ich, dass er die Betreuung meines Themas nach Andreas' Tod so selbstverständlich und unkompliziert übernommen hat.

Genauso danke ich Michael Waidner, für die Hinweise und Diskussionen.

Ich danke allen Kollegen der DuD Gruppe, die nicht müde wurden, sich neue Protokollverbesserungen anzuhören. Besonderer Dank gilt hierbei Rainer Böhme, ohne den die erste Veröffentlichung wahrscheinlich nie geschrieben worden wäre.

Stefan Köpsell und Sebastian Clauß haben durch ihr unermüdliches Feedback einen sehr großen Anteil an der Qualität dieses Textes.

Vielen Dank auch an Sandra Steinbrecher, Dagmar Schönfeld und Stephan Groß, welche sich viel Zeit nahmen, die Arbeit Korrektur zu lesen.

Manuela Berg danke ich für die Hilfe bei den formalen Details.

Ich danke Josef Schmid für die anregende Diskussion im Metalab Wien, welche mir half das anonyme Protokoll entscheidend zu verbessern.

Manuel Odendahl danke ich für die Pair-Programming-Session nach dem Datenspuren Symposium, in der wir den JavaScript-Watchdog überlisteten.

Vielen Dank an Christina Hochleitner und Peter Wolkerstorfer vom Center of Usability Research, Wien. Beide kritisierten wieder und wieder das Interface der Implementierung und haben damit zur Verbesserung der Applikation beigetragen.

Auch den Freiwilligen, die die Applikation in viele Sprachen übersetzt haben soll an dieser Stelle gedankt werden. Sie haben mir durch ihre unaufgeforderten Zuschriften sehr viel Mut zum weiterschreiben gegeben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Definitionen I

Definitionen II (Anforderungen)

Symbole

Symbol	Beschreibung
$\boldsymbol{k}_u$	Schlüsselmatrix für Teilnehmer u
$\vec{k}_{u_a,u_b}$	Schlüsselvektor von Teilnehmer u_a mit Teilnehmer u_b
$k^t_{u_a,u_b}$	Schlüssel für die DC-Netz-Runde zwischen Teilnehmer u_a und u_b für den Zeitpunkt t
n	DC-Netz-Modulus
Δ	Menge aller DC-Netz-Runden
δ	Tupel, welches eine DC-Netz-Runde eindeutig identifiziert
g	Diffie–Hellman Generator
dh_{u_a,u_b}	Symmetrischer Schlüssel, welcher mittels der Diffie–Hellman Schlüssel zwischen den Teilnehmern u_a und u_b berechnet wurde
q	Diffie–Hellman Modulus
pub_u	Öffentlicher Diffie–Hellman Schlüssel von Teilnehmer u
sec_u	Geheimer Diffie–Hellman Schlüssel von Teilnehmer u
U	Geordnete Menge von Teilnehmern
UUID	Menge aller für Umfragen eindeutige einmalige IDs
T	Geordnete Menge von Zeitpunkten
$\mathcal{B}^{u_m}_{\text{all}}(\cdot)$	Funktion, die aus Sicht aller ehrlichen Teilnehmer (nur u_m greift an) überprüft, ob die Abstimmung ohne Angriffe durchgeführt wurde
$\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$	Funktion, die aus Sicht von Teilnehmer u überprüft, ob die Abstimmung ohne Angriffe durchgeführt wurde (engl. *blame*)

Symbol	Beschreibung
$\lvert X \rvert$	Kardinalität der Menge X
$\mathcal{D}_{\mathcal{B}}\left(\boldsymbol{d}, \boldsymbol{k}_u\right)$	Funktion, die die Menge von DC-Netz-Runden ausgibt, für die Teilnehmer u beweisen kann, dass ein Angriff stattgefunden hat (engl. *disclose*)
$\mathcal{D}_{\mathcal{C}}\left(\boldsymbol{d}\right)$	Funktion, die die Menge von DC-Netz-Runden ausgibt, an denen die Konsistenzprüfung zwischen normaler und invertierter Abstimmung fehlgeschlagen ist (engl. *disclose*)
$\mathcal{D}_{\mathcal{V}}\left(\boldsymbol{d}\right)$	Funktion, die die Menge von DC-Netz-Runden ausgibt, an denen die DC-Netz-Summe am niedrigsten ist (engl. *disclose*)
$\vec{\psi}_{u_a,u_b}$	Vektor, welcher die Commitments für die Schlüssel $\vec{k}_{u_a,u_b}$ enthält
$\psi^{\delta}_{u_a,u_b}$	Commitment für den Schlüssel $k^{\delta}_{u_a,u_b}$
$\mathrm{commit}\left(\cdot\right)$	Funktion, die ein Commitment zu einem Wert berechnet
$\mathcal{C}\left(\boldsymbol{d}\right)$	Funktion, die die Konsistenz zwischen normaler und invertierter Abstimmung überprüft (engl. *consistency*)
$\mathrm{decr}^{\mathrm{sym}}_{\mathsf{key}}\left(\cdot\right)$	Symmetrische Entschlüsselungsfunktion
$\mathrm{encAuth}^{\mathrm{sym}}_{\mathsf{key}}\left(\cdot\right)$	Symmetrische Verschlüsselungs- und Authentikationsfunktion
$\mathrm{enc}^{\mathrm{sym}}_{\mathsf{key}}\left(\cdot\right)$	Symmetrische Verschlüsselungsfunktion
$\boldsymbol{d}$	Matrix, welche alle DC-Netz-Nachrichten enthält, die von allen Teilnehmern gesendet wurden
$d_u^{(t,i)}$	Verschlüsselte Teilstimme von Teilnehmer u für Zeitpunkt t und DC-Netz-Index i
$\vec{d}_u$	Verschlüsselter Verfügbarkeitsvektor von Teilnehmer u
d_u^t	Verschlüsselte Verfügbarkeit von Teilnehmer u zum Zeitpunkt t
$\mathrm{enc}_u\left(\cdot\right)$	Funktion, die eine Nachricht asymmetrisch an Teilnehmer u verschlüsselt
$\mathrm{h}(\cdot)$	Kollisionsresistente Hashfunktion
$\phi_u^{(t,\lambda)}$	Für $\lambda = 0$: die Verfügbarkeit des Teilnehmers u zum Zeitpunkt t. Für $\lambda = 1$: die Prüfverfügbarkeit

Symbol	Beschreibung
ℓ	Anzahl gleichzeitig stattfindender DC-Netz-Runden
$\boldsymbol{k}_u$	Schlüsselmatrix für Teilnehmer u mit allen Schlüsseln für alle simultanen DC-Netze
$k_{u_a,u_b}^{(t,i)}$	Schlüssel für die DC-Netz-Runde (t, i) zwischen Teilnehmer u_a und u_b
$\varphi_u^{(t,i)}$	Teilstimme von Teilnehmer u für Zeitpunkt t im DC-Netz mit dem DC-Netz-Index i
u	Teilnehmer einer Terminplanung (engl. *user*)
uuid	Für jede Umfrage eindeutige einmalige ID ($\mathsf{uuid} \in \mathsf{UUID}$)
$P(e)$	Eintrittswahrscheinlichkeit für Ereignis e (engl. *probability*)
r	Zufallszahl (engl. *random*)
$\vec{\sigma}$	Ergebnisvektor, welcher die Anzahl der verfügbaren Teilnehmer zu allen Zeitpunkten enthält
σ^t	Anzahl der zum Zeitpunkt t verfügbaren Teilnehmer
$\mathcal{S}_{\max}$	Funktion, die den ersten Zeitpunkt aus der Menge der zur Wahl stehenden Zeitpunkte auswählt, an dem die meisten Teilnehmer verfügbar sind
$\mathcal{S}$	Funktion, die einen Zeitpunkt aus der Menge der zur Wahl stehenden Zeitpunkte auswählt (engl. *selection rule*)
$\mathrm{sig}_u(\cdot)$	Digitale Signatur einer Nachricht des Teilnehmers u
t	Zur Wahl stehender Zeitpunkt (engl. *time slot*)
$\mathcal{V}(\boldsymbol{d})$	Funktion, die ohne das Wissen über das Abstimmverhalten der Teilnehmer überprüft, ob die Abstimmung ohne Angriffe durchgeführt wurde (engl. *verify*)
$\vec{\phi}_u$	Verfügbarkeitsvektor von Teilnehmer u
ϕ_u^t	Verfügbarkeit des Teilnehmers u zum Zeitpunkt t

1. Einleitung

> Freitag, 13:30 Uhr, Besprechung. Nachdem die wichtigsten Neuigkeiten der Gruppe geklärt wurden, berichtet jeder Mitarbeiter über seine Tätigkeiten der letzten Woche. Ein Kollege bringt die bevorstehende Weihnachtsfeier zur Sprache und gibt zu bedenken, dass noch kein Termin gefunden wurde, der allen Mitarbeitern angenehm ist. Hektisch kramen einige Mitarbeiter in ihren Taschen und versuchen ihren Terminkalender ausfindig zu machen. Andere öffnen ihre Laptops mit diversen Kalenderprogrammen und nennen Termine, an denen sie auf Dienstreise sind. Wieder andere Kollegen rufen Tage dazwischen, an welchen eine Planung aus verschiedenen anderen Gründen ungünstig wäre. Der Chef wendet ein, dass er gerne – so wie die letzten Jahre – die ehemaligen Mitarbeiter einladen möchte. Da man auch auf deren Termine Rücksicht nehmen möchte, wird die Terminfindung gänzlich verschoben. Der Kollege, der die Problematik „Weihnachtsfeier“ aufgebracht hat, wird beauftragt, sich um einen geeigneten Termin zu kümmern und alle Externen nach ihren Terminwünschen zu fragen. Auf dem Rückweg ins Büro gibt ein anderer Mitarbeiter noch zu bedenken, dass erst kürzlich in einer Schulung darauf hingewiesen wurde, dass das Verwenden externer Terminplanungsprogramme aus Datenschutzgründen zu unterlassen ist.

Dieses oder ähnliche Szenarien hat sicher jeder schon in seinem Berufsleben erlebt. Schon länger verwalten Firmen die Kalender ihrer Mitarbeiter zentral, um dieses Problem zu lösen. Existieren solche Lösungen nicht oder werden Termine zwischen unterschiedlichen Organisationen gesucht, wird der Termin nicht selten nur durch lange E-Mail-Debatten gefunden. Oft werden hierbei Excel-Tabellen verteilt, in denen jeder seine Terminwünsche einträgt. Seit spätestens 2001 versucht auch das Web 2.0 eine Lösung für dieses Problem anzubieten. Mit Meet-O-Matic [Mee] wurde ein Dienst erschaffen, der eine einfache Tabelle darstellt, in der jeder Teilnehmer seine Terminwünsche eintragen kann. Später wurde dieses Prinzip durch Michael Näf aufgegriffen und mit Doodle [Näf] zu der heute wahrscheinlich bekanntesten Applikation dieser Art ausgebaut. Heute finden sich neben Meet-O-Matic und Doodle aber auch noch unzählige andere Terminabstimmungsapplikationen [FS; Pro; Sol; SDV; Kle+; Tun; Tim; Mad; Sof; 12g].

Mehrere Sicherheitsprobleme treten bei den genannten Applikationen auf. Zum einen ist es nicht immer gewünscht, dass alle Teilnehmer, die an Terminabstimmungen teilnehmen, die Terminwünsche aller anderen Teilnehmer sehen können. Ein weiteres Problem ist, dass man sich bei manchen Umfragen sicher sein will, dass niemand die Umfrage gefälscht hat, beispielsweise in dem er die Terminwünsche eines anderen bearbeitet. Die folgenden Beispiele sollen einige Datenschutz- und Datensicherheitsprobleme etwas verdeutlichen.

Beispiel 1 Die vier Vorstandsvorsitzenden von vier über Deutschland verteilten Firmen möchten zu einem geheimen Gipfeltreffen zusammenkommen. Keiner der Vorsitzenden hätte ein Problem damit, den anderen seine Terminpräferenzen mitzuteilen, bzw. die Befürchtung, ein anderer würde seine Stimme fälschen. Allerdings befürchten alle, dass Außenstehende Informationen über den Termin oder das Treffen erlangen. □

Beispiel 2 Ein Professor möchte ermitteln, an welchen Tagen der größte Bedarf an Prüfungsterminen seiner Studenten besteht. Er konfiguriert seine Terminabstimmungsapplikation so, dass sich die Studenten für die jeweiligen Tage eintragen können, an denen es ihnen möglich ist die Prüfung abzulegen. Um die Privatsphäre der Studenten zu schützen, wer an welchen Tagen seine Prüfung haben wird, sollten die Studenten nicht sehen, wofür ihre Kommilitonen jeweils gestimmt haben.

Aus Sicht des Professors ist der Datenschutz der Studenten allerdings weniger kritisch. Im Gegenteil, es ist für ihn evtl. sogar notwendig zu wissen, welcher Student für welchen Tag seine Wünsche geäußert hat, damit er die genauen Zeiten mit den jeweiligen Studenten vereinbaren kann. □

Beispiel 3 Eine Gruppe von 20 Computer-affinen, über die Welt verteilten Personen, die gemeinsam an einem freien Softwareprojekt arbeiten, möchten sich innerhalb des nächsten halben Jahres zu einem gemeinsamen Wochenende treffen. Es möchte jeder überprüfen können, dass das Wochenende fair aus den 26 zur Verfügung stehenden Wochenenden gewählt wurde. Gleichzeitig möchte allerdings nicht jeder jedem anderen Auskunft über seine Wochenendgestaltung geben. □

In keiner der zu Beginn erwähnten Applikationen lassen sich die eben erwähnten Szenarien befriedigend umsetzen. Hierbei fehlt es den Applikationen insbesondere an geeigneten Maßnahmen bzgl. des Datenschutzes und der Datensicherheit. Diese Arbeit widmet sich deshalb der Erstellung einer Web 2.0-Applikation, welche das Problem der Terminabstimmung mehrseitig sicher löst. Dabei wird auf die unterschiedlichen Schutzbedürfnisse der in den einzelnen Szenarien beteiligten Entitäten Rücksicht genommen.

Die Gliederung dieser Arbeit ist wie folgt. Erst werden Primär- und Sekundäranforderungen definiert (Kapitel 2). Kapitel 3 stellt Verfahren vor, bei denen mindestens eine Entität vertrauenswürdig ist, in Kapitel 4 wird gezeigt, wie eine Terminabstimmung mit minimalen Annahmen über andere möglich ist. Existierende Arbeiten werden jeweils am Anfang der Kapitel betrachtet (Abschnitte 3.1 und 4.1). Viele vorgestellte Verfahren wurden in Form einer Web 2.0-Applikation umgesetzt. Eine Diskussion der Implementierungen findet sich in Kapitel 5. Kapitel 6 fasst die gesamte Arbeit zusammen und gibt einen Ausblick auf mögliche Verbesserungen im Protokoll sowie der Implementierung.

2. Anforderungsanalyse

Im Folgenden werden die Anforderungen diskutiert, welche an eine Web 2.0-Terminabstimmungsapplikation gestellt werden können. Benötigte Begriffe werden zuerst definiert.

2.1. Begriffsdefinitionen/Primärfunktionalität

Die Planung von Terminen kann teilweise von sehr komplexen Bedingungen abhängen. Einige Bedingungen könnten beispielsweise sein:

- „Alice ist montags bis 13 Uhr in Hamburg."
- „Die Reisezeit von Hamburg nach Berlin beträgt zwei Stunden."
- „Die Reisezeit von Hamburg nach München beträgt vier Stunden."
- „Der Meetingraum in Berlin ist montags von 9–12 Uhr blockiert."
- „Der Meetingraum in München ist montags von 10–14 Uhr blockiert."
- „Der Meetingraum in Berlin fasst fünf Personen."
- „Der Meetingraum in München fasst sieben Personen."
- „Von den Gruppenleitern Alice und Bob wird mindestens eine Person benötigt."
- „Aus jeder Gruppe sollten mindestens eine, maximal zwei Personen teilnehmen."

Mithilfe solcher Bedingungen können Regeln für Applikationen aufgestellt werden, die zum Planen von Ereignissen verwendet werden. In der Praxis werden solche Applikationen benutzt, um Stundenpläne von Universitäten oder Tagungsprogramme zu planen oder auch, um Abflugs- und Landezeiten von Flugzeugen in eine Reihenfolge zu bringen (unter Berücksichtigung von Flugzeug- und Rollbahngrößen, sowie den Randbedingungen anderer Flughäfen). Applikationen, welche solche Terminplanungen vornehmen, lesen eine Menge von Bedingungen bzw. Regeln, aggregieren sie und wenden anschließend eine *Terminauswahlfunktion* auf sie an, die den Zeitpunkt ausgibt, an dem das Ereignis stattfinden sollte.

Definition 1 (Terminauswahlfunktion) Eine Terminauswahlfunktion $\mathcal{S}$ ist eine Funktion, welche den den Termin, an dem ein Ereignis stattfinden soll bestimmt. □

Bei vielen Terminplanungsproblemen ist es nicht nötig, die Randbedingungen in komplexen Regeln zu spezifizieren. Eine einfache Form, Randbedingungen für eine Terminplanung anzugeben, ist die Spezifikation von *Verfügbarkeitsvektoren*.

Definition 2 (Verfügbarkeit) Eine Verfügbarkeit $\phi_u^t \in \{0, 1\}$ gibt an, ob ein Teilnehmer u zu einem Zeitpunkt t an einem Ereignis teilnehmen kann oder nicht.

$$\phi_u^t := \begin{cases} 1 & \text{Teilnehmer } u \text{ kann zum Zeitpunkt } t \text{ am Ereignis teilnehmen,} \\ 0 & \text{sonst.} \end{cases} \tag{2.1}$$

□

Definition 3 (Verfügbarkeitsvektor) Ein Verfügbarkeitsvektor einer Person u ist ein Vektor $\vec{\phi}_u$, welcher für eine geordnete Menge T von Zeitpunkten angibt, zu welchen Zeitpunkten die Person verfügbar ist.

$$\vec{\phi}_u := \left(\phi_u^{t_1}, \ldots, \phi_u^{t_{|T|}}\right), \left\{t_1, \ldots, t_{|T|}\right\} = T. \tag{2.2}$$

□

Im Web 2.0 haben sich einfache Terminabstimmungsapplikationen durchgesetzt, bei denen keine komplexen Bedingungen und Terminauswahlfunktionen spezifiziert werden, sondern nur Verfügbarkeitsvektoren.

Definition 4 (Terminabstimmungsapplikation) Eine Terminabstimmungsapplikation (auch Termin*umfrage*applikation) ist ein Programm, welches die Aufgabe hat, eine Menge von Teilnehmern U bei der Ermittlung eines Termins für ein Ereignis zu unterstützen. Die Eingabe einer Terminabstimmungsapplikation ist eine endliche geordnete Menge möglicher Zeitpunkte T, sowie die $|U|$ Verfügbarkeitsvektoren der Teilnehmer über die Zeitpunkte T. Die Ausgabe der Applikation ist eine aggregierte Form der Verfügbarkeitsvektoren, auf die eine Terminauswahlfunktion angewandt werden kann. □

Die Menge möglicher Zeitpunkte T, sowie die Teilnehmermenge U werden zu Beginn einer Terminabstimmung von einem *Initiator* festgelegt.

Die Terminauswahlfunktion ist hierbei nicht Teil der Applikation. Nach Durchführung der Abstimmung besteht die Terminauswahlfunktion darin, dass der Zeitpunkt mithilfe der aggregierten Werte entweder von einem Teilnehmer (z. B. dem Initiator) festgelegt oder von den Teilnehmern ausgehandelt wird. Die Teilnehmer einer Terminabstimmungsapplikation müssen sich vor Durchführung der Umfrage nicht auf eine konkrete Auswahlfunktion festlegen.

Eine wahrscheinlich häufig benutzte Terminauswahlfunktion ist die Funktion, welche den ersten Zeitpunkt wählt, an dem die größte Anzahl an Teilnehmern verfügbar ist. Diese Funktion benötigt als Eingabe einen Vektor, welcher die Summe aller Verfügbarkeitsvektoren enthält $\vec{\sigma} = \sum_{u \in U} \vec{\phi}_u$.

Definition 5 (Terminauswahlfunktion maximaler Verfügbarkeit) Die Terminauswahlfunktion $\mathcal{S}_{\max} : \{0, \ldots, |U|\}^{|T|} \to T$, die den ersten Zeitpunkt wählt, an dem die meisten Teilnehmer verfügbar sind, wird definiert als:

$$\mathcal{S}_{\max}(\vec{\sigma}) := \min\left\{t : \sigma^t = \max(\vec{\sigma})\right\}, \sigma^t \in \vec{\sigma}. \tag{2.3}$$

□

Nachdem der Begriff der Terminabstimmung eingeführt wurde, wird nun noch definiert, was unter einer Web 2.0-Applikation zu verstehen ist:

Definition 6 (Web-Applikation) Eine Web-Applikation ist eine Applikation, welche clientseitig nur mit einem Web-Browser interagiert. □

Definition 7 (Web 2.0-Applikation) Als Web 2.0-Applikation wird eine Web-Applikation genau dann bezeichnet, wenn deren Inhalt durch ihre Nutzer über das Web erstellt, verwaltet und gelesen werden kann. □

Eine Web 2.0-Terminabstimmungsapplikation unterstützt daher eine Menge von Teilnehmern bei der Ermittlung eines Termins für ein Ereignis über das Web. *Web* bedeutet hierbei, dass die Bedienung vollständig mittels eines Browsers durchgeführt werden kann.

2.2. Anforderungen

In diesem Abschnitt werden die Anforderungen, welche an eine Terminabstimmungsapplikation gestellt werden, vorgestellt und diskutiert. Die Anforderungen werden hierbei in zwei Kategorien eingeteilt: Benutzbarkeits- und Sicherheitsanforderungen.

2.2.1. Benutzbarkeit

Bei jeder Applikation spielt die Benutzbarkeit (engl. *usability*) eine wichtige Rolle. Hierbei kann die Benutzbarkeit von einem zu Grunde liegenden Schema eingeschränkt werden, was in den folgenden Anforderungen zum Ausdruck gebracht werden soll.

Die Auswahlfunktion einer Terminplanung mittels einer Terminabstimmungsapplikation kann als Eingabe nur das enthalten, was die Applikation an aggregierten Werten ausgibt. In der einfachsten Form besteht die Aggregation aus der identischen Abbildung.[1] Diese Form der Aggregation würde alle Informationen der Eingabe vollständig erhalten. Im Laufe dieser Arbeit wird gezeigt, dass es unter manchen Umständen sinnvoll ist, die Menge der Information, welche in der Eingabe enthalten ist, in der Ausgabe zu verringern. Eine Möglichkeit, die enthaltene Information zu verringern, ist die Entropie der Ausgabe zu verringern.[2] Mit jeder Einschränkung leidet allerdings die *Flexibilität* der Auswahlfunktion. Wird als Aggregation beispielsweise ein Vektor gebildet, der die Summe aller Verfügbarkeitsvektoren enthält, so kann als Auswahlfunktion nur noch eine Funktion verwendet werden, die diesen einen Vektor einliest (z. B. $\mathcal{S}_{\max}$). Ist diese Einschränkung für manche Terminplanungen kein Problem, kann es für andere jedoch wichtig sein, dass die Anwesenheit einer Teilmenge der Teilnehmer notwendig ist, oder dass aus verschiedenen Personengruppen je eine Person anwesend sein muss.

Anforderung 1 (Flexibilität) Die Ausgabe einer Terminabstimmungsapplikation sollte so viel Flexibilität wie möglich für die Wahl der Terminauswahlfunktion lassen. Die höchste Flexibilität ist gegeben, wenn bei der Aggregation der Verfügbarkeitsvektoren keine Information verloren geht. □

[1] In konkreten Applikationen werden hier die einzelnen Verfügbarkeitsvektoren (nur) graphisch dargestellt.
[2] Eine Beispielrechnung über die Entropie eines Verfügbarkeitsvektors findet sich in Anhang A.

Weitere Benutzbarkeitsanforderungen an eine Terminabstimmungsapplikation sind:

Anforderung 2 (Installationsaufwand) Der Benutzer einer Web 2.0-Applikation sollte wenig bzw. keinen zusätzlichen[3] Installationsaufwand haben. □

Anforderung 3 (Reaktionszeit) Die Zeit, die ein Nutzer in einer Web 2.0-Applikation zwischen zwei Eingaben warten muss, sollte den Nutzer in seinem Arbeitsablauf nicht beeinträchtigen. □

Inwieweit ein Benutzer beeinträchtigt wird, ist natürlich eine sehr subjektive Aussage. In der Literatur finden sich viele Studien darüber, wie lange Nutzer bereit sind, auf Webseiten zu warten. Eine Angabe über die Dauer, nach wie vielen Sekunden Nutzer von einer Aufgabe abgelenkt werden, schwankt hierbei zwischen 8 und 30 Sekunden [Mil68; CNM83; New90]. Klar erscheint dabei, dass Nutzer nicht von ihrer Aufgabe abgelenkt werden, wenn eine Web 2.0-Applikation nach einer Eingabe weniger als 8 Sekunden Wartezeit benötigt. Beansprucht ein Schema mehr als 30 Sekunden, so ist dies in diesem Kontext auf jeden Fall zu lang.

Die Reaktionszeit einer Applikation wird von zwei Faktoren beeinflusst, dem Berechnungsaufwand und dem Kommunikationsaufwand.

Definition 8 (Berechnungsaufwand) Der Berechnungsaufwand gibt an, wie viel Zeit ein Algorithmus benötigt, um auf einer Zielplattform sein Ergebnis zu erzielen. □

Der Berechnungsaufwand hängt zwar von der Berechnungskomplexität eines Algorithmus ab, allerdings kann ein Algorithmus mit einer schlechten Berechnungskomplexität auch einen Berechnungsaufwand haben, welcher die Anforderung der Reaktionszeit erfüllt.

Wird eine Berechnung im Hintergrund ausgeführt während der Benutzer nach Eingaben gefragt wird, so darf sie mehr Zeit in Anspruch nehmen als eine Berechnung, auf die der Nutzer zwischen zwei Eingaben warten muss. Des Weiteren hängt die Zeit, die eine Berechnung benötigt, sehr stark von der Leistung des Rechners, sowie der jeweiligen Implementierung ab.

Das Optimum für Anforderung 2 ist, dass keinerlei zusätzliche Installation beim Benutzer nötig ist. Unter dieser Berücksichtigung steht dem Benutzer allerdings für eine Web 2.0-Applikation nur ein Web-Browser ohne jegliche Erweiterungen wie etwa Flash oder Java zur Verfügung. Heutige Web-Browser können in der Regel JavaScript interpretieren, welches für clientseitige Berechnungen verwendet werden kann. Wie schnell heutige JavaScript-Interpreter sind, ist in Tabelle 2.1 auf der nächsten Seite dargestellt.[4]

Hier wurde die Performance einer diskreten Exponentation in einer Gruppe mit 2^{1024} Elementen mit mehreren JavaScript-BigInteger Bibliotheken gemessen. Diese Exponentationen sind bei asymmetrischen kryptographischen Operationen oft zu finden.

[3] *Zusätzlich* bedeutet hier, dass ein Betriebssystem sowie gängiger Web-Browser vorausgesetzt werden kann.

[4] Um die Implementierungen der einzelnen Browser vergleichen zu können, wurden alle Browser im gleichen Betriebssystem (Windows XP, SP3) und auf dem gleichen Rechner (Intel Pentium 4 Duo mit 2,8 GHz CPU und 2 GB RAM) gemessen.

Tabelle 2.1.: Ausführungszeit für eine diskrete Exponentation in einer Gruppe mit 2^{1024} Elementen mit mehreren JavaScript-BigInteger Bibliotheken. Die schnellsten Zeiten sind jeweils fett markiert. Gemessen wurde auf einem Intel Pentium 4 Duo mit 2,8 GHz CPU und 2 GB RAM unter Windows XP SP3.

	IE8 8.0	IE7 7.0	IE6 6.0	FF 5.0	Chrome 12.0	Safari 5.1	Opera 11.50
[Wu]	**4,89 s**	**5,39 s**	**5,49 s**	0,94 s	0,10 s	1,38 s	0,70 s
[Bai]	8,46 s	13,43 s	13,56 s	**0,26 s**	**0,08 s**	**0,29 s**	**0,31 s**
[Sha]	33,25 s	50,73 s	49,70 s	1,71 s	0,57 s	1,61 s	1,85 s
[Cru]	297,03 s	616,88 s	624,61 s	24,78 s	9,57 s	28,22 s	(stürzt ab)
1000 × AES256	1,02 s	2,51 s	2,52 s	0,05 s	0,06 s	0,08 s	0,07 s

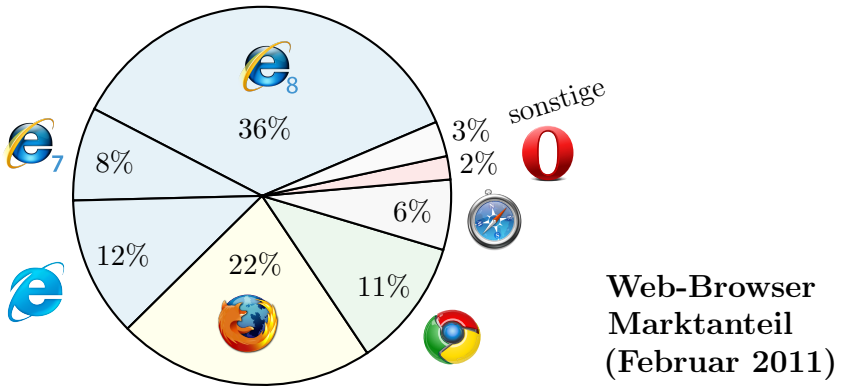


Abbildung 2.1.: Marktanteil verschiedener Web-Browser. Quelle: Net Applications [Net]

Es ist ersichtlich, dass solche diskreten Exponentationen durchaus eine relevante Größe im Berechnungsaufwand haben. Dies wird besonders deutlich, wenn man Statistiken betrachtet, welche den Marktanteil der Web-Browser erfassen. In Abbildung 2.1 ist eine Statistik der Firma Net Applications dargestellt, welche auch Informationen über die einzelnen Browserversionen bietet. Hier ist zu sehen, dass der Browsermarkt immer noch vom Internet Explorer – dem Browser mit der langsamsten JavaScript Engine – sowie von Firefox dominiert wird. Darüber hinaus ist zu sehen, dass immer noch sehr viele Nutzer den sehr alten Internet Explorer 6 benutzen. Die einzelnen Versionen der anderen Browser wurden nicht extra dargestellt, da hier von der überwiegenden Mehrheit die aktuellen Versionen benutzt werden.

Würde ein Schema beispielsweise eine asymmetrische Operation pro Nutzer und Zeitpunkt benötigen, so müsste ein Nutzer des Internet Explorer 8 bei einer Umfrage mit fünf

Benutzern und zehn Zeitpunkten allein $5 \cdot 10 \cdot 4{,}89\,\mathrm{s} = 4\,\mathrm{min}\ 4\,\mathrm{s}$ auf die Berechnung der asymmetrischen kryptographischen Operationen warten.

Legt man etwas weniger Wert auf Anforderung 2, so könnte man Flash oder Java im Browser des Nutzers voraussetzen. Insbesondere Java hätte hierbei einen positiven Einfluss auf den Berechnungsaufwand, da die gleichen Operationen in einer Java Virtuellen Maschine deutlich schneller ausgeführt werden können. Auf konkrete Messungen zu Ausführungszeiten mit Java und Flash, sowie deren Implementierung wird in Abschnitt 5.3.3 eingegangen.

Zusätzlich ist in Tabelle 2.1 in einer Zeile dargestellt, wie schnell symmetrische Operationen im Vergleich zu den Asymmetrischen sind. Gemessen wurde die Ausführungszeit von 1000 AES256 Berechnungen. Es ist leicht zu sehen, dass symmetrische Operationen wesentlich weniger kritisch für den Berechnungsaufwand einer Applikation sind.

Definition 9 (Kommunikationsaufwand) Der Kommunikationsaufwand gibt an, wie viel Zeit ein Algorithmus benötigt, um auf einer Zielplattform mit bestimmter Kanalkapazität die benötigten Daten zu übertragen. □

Wie auch beim Berechnungsaufwand spielt es für die Reaktionszeit keine Rolle, ob wenige Bits oder mehrere Kilobytes übertragen werden, so lange der Arbeitsablauf des Nutzers nicht beeinträchtigt wird.

Eine weitere Anforderung betrifft die Anzahl der Interaktionen mit einer Applikation. Hierbei sei angemerkt, dass man bei einer Web 2.0-Applikation nicht voraussetzen kann, dass Teilnehmer bzw. ihre Rechner dauerhaft verfügbar sind.

Anforderung 4 (Blockierende Protokollrunden) Existiert für eine gegebene Terminabstimmungsapplikation eine Eingabe $T, U, \left\{\vec{\phi}_{u_1}, \ldots, \vec{\phi}_{u_{|U|}}\right\}$, bei der eine Protokollrunde erst durchgeführt werden kann, nachdem die Eingabe *aller* Teilnehmer erfolgt ist, so ist dies eine blockierende Protokollrunde.

Die Anzahl der blockierenden Protokollrunden einer Terminabstimmungsapplikation ist durch die Eingabe bestimmt, welche die meisten blockierenden Protokollrunden aufweist (worst case). Eine Terminabstimmungsapplikation sollte möglichst wenige blockierende Protokollrunden haben. □

Beispiel 4 Angenommen, eine Terminabstimmungsapplikation würde für eine Menge T an Zeitpunkten versuchen herauszufinden, wann der erste Zeitpunkt ist, an dem die meisten Teilnehmer verfügbar sind. Damit die Teilnehmer nicht alle ihre Verfügbarkeiten preisgeben müssen, wird jeder Teilnehmer der Menge U einzeln nach allen Zeitpunkten gefragt. Es wird daher erst nach Zeitpunkt t_1 gefragt, und wenn einer der Teilnehmer seine Verfügbarkeit für diesen Zeitpunkt verneint, wird nach dem nächsten Zeitpunkt $t_2, t_1 < t_2$ gefragt. Gibt es einen Zeitpunkt, an dem alle Teilnehmer verfügbar sind, so wird abgebrochen und dieser Zeitpunkt gewählt.

Im schlimmsten Fall verneint bei jedem Zeitpunkt der letzte gefragte Teilnehmer seine Verfügbarkeit für diesen Zeitpunkt, wodurch $|T|$ Protokollrunden nötig sind. Daher hat dieses Protokoll $|T|$ blockierende Protokollrunden. □

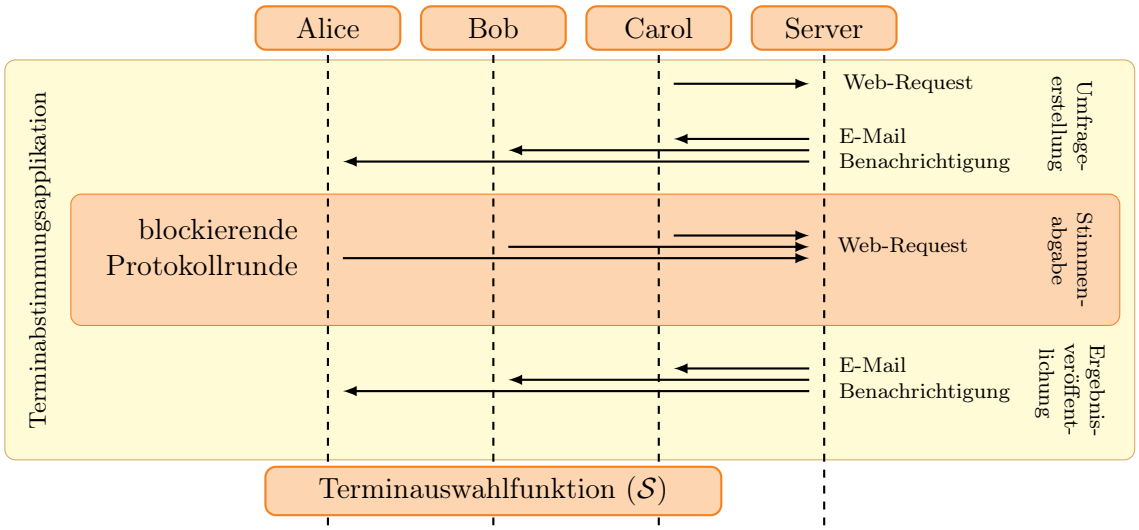


Abbildung 2.2.: Phasen einer typischen Terminabstimmungsapplikation

Diese Anforderung wird um so bedeutender, je mehr Teilnehmer an einer Terminabstimmung teilnehmen sollen, da es dann um so langwieriger ist, darauf zu warten, dass alle Teilnehmer die Protokollrunde abgearbeitet haben.

Jede Terminabstimmungsapplikation benötigt mindestens eine blockierende Protokollrunde, da mindestens einmal auf die Eingabe aller Personen gewartet werden muss.[5] Benötigt eine Umfrage nur eine blockierende Protokollrunde, so ist die Anzahl der Runden minimal. Die Runden finden üblicherweise in drei Phasen statt:

1. Das Anlegen einer Abstimmung (Umfrageerstellung),
2. das Senden der persönlichen Präferenzen zum Abstimmungsserver (Stimmenabgabe) und
3. die Benachrichtigung über das Ergebnis, welche oft via E-Mail geschieht (Ergebnisveröffentlichung).

Da das Ergebnis nur aus aggregierten Werten besteht, muss hierauf nun die Terminauswahlfunktion angewandt werden. Abbildung 2.2 illustriert diesen Ablauf. Für eine weitere Beschreibung kann an dieser Stelle ein Blick in Abschnitt 5.1.1 hilfreich sein. Dort wird vorgestellt, wie die in dieser Arbeit entstandene Implementierung die einzelnen Phasen behandelt.

Zusätzlich ist es in manchen Applikationen üblich, dass sich Teilnehmer vorher einmalig registrieren. Will ein Teilnehmer nur an einer einzigen Umfrage teilnehmen, so bedeutet

[5] Natürlich wird hier unterstellt, dass die Verfügbarkeiten bzw. die Bedingungen aller Teilnehmer dem Umfrageninitiator oder dem Umfragenserver vorher nicht bekannt sind.

dies eine zusätzliche Protokollrunde. Mit steigender Anzahl der Umfragen, an denen sich ein Teilnehmer beteiligt, wird diese zusätzliche Runde allerdings unbedeutender. Im Folgenden soll dieser Fall getrennt zu den Protokollrunden einer Umfrage betrachtet werden. Benötigt ein Protokoll zusätzlich zur Registrierung nur eine blockierende Protokollrunde, so wird von *minimaler Anzahl blockierender Protokollrunden mit Registrierung* gesprochen.

2.2.2. Mehrseitige Sicherheit

Für die vorliegende Arbeit wird von der Definition für *mehrseitige Sicherheit* ausgegangen, wie sie von Federrath und Pfitzmann [FP97] eingeführt wurde:

Definition 10 (Mehrseitige Sicherheit) „Mehrseitige Sicherheit bedeutet die Einbeziehung der Schutzinteressen aller Beteiligten sowie das Austragen daraus resultierender Schutzkonflikte [...]." [FP97] □

Prinzipiell können Angreifer in *protokolltreue* und *protokollverletzende* Angreifer unterschieden werden. Während protokolltreue Angriffe nicht erkannt werden können, da sie sich an die Regeln halten, die ihnen vorgegeben wurden und im Falle einer Beschuldigung immer argumentieren können, sie hätten nur nach Spezifikation gehandelt, sind protokollverletzende Angriffe in der Regel erkennbar.

Darüber hinaus kann ein Angreifer *passiv* oder *aktiv* agieren. Eine genauere Betrachtung hierzu findet sich u. a. in [MOV97].

Im Folgenden werden wir den Begriff *mehrseitige Sicherheit* getrennt nach *Mehrseitigkeit* und *Sicherheit* untersuchen.

Sicherheit

Unter dem Begriff Sicherheit werden in dieser Arbeit die drei Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit verstanden. Jedes dieser Schutzziele sollte anschließend gegenüber jeder Entität betrachtet werden, um den Aspekt der Mehrseitigkeit zu berücksichtigen.

Anforderung 5 (Vertraulichkeit) Es sollen so wenig wie möglich Daten über einen Teilnehmer einer Terminabstimmung bekannt werden. □

Daten, die in einer Terminabstimmung auftreten können, sind zum einen die Primärdaten, welche aus dem Verfügbarkeitsvektor einer Person abgelesen werden können. Des weiteren existieren aber auch Sekundärdaten, wie beispielsweise der Zeitpunkt, an dem ein Teilnehmer an einer Abstimmung teilnimmt, seine IP-Adresse, seine Bandbreite oder die Geschwindigkeit, mit der der Rechner des Teilnehmers Berechnungen durchführt. Während die Primärdaten durch das Terminabstimmungsschema geschützt werden müssen, ist der Schutz der Sekundärdaten Aufgabe anderer Werkzeuge (z. B. AN.ON oder TOR zum Verbergen der IP-Adresse).

Anforderung 6 (Integrität) Jeder Teilnehmer sollte jede Manipulation an einer Terminabstimmung erkennen können. □

Die Integrität kann dahingehend zerlegt werden, in welchem Maße eine Umfrage durch einen Teilnehmer überprüft werden kann.

Definition 11 (Individuelle Überprüfbarkeit) Eine Umfrage ist individuell überprüfbar, wenn ein Teilnehmer überprüfen kann, dass *seine eigene Stimme* richtig in die Berechnung des Ergebnisses eingegangen ist. □

Definition 12 (Universelle Überprüfbarkeit) Eine Umfrage ist universell überprüfbar, wenn ein Teilnehmer überprüfen kann, dass *die Stimmen aller Teilnehmer* richtig in die Berechnung des Ergebnisses eingegangen sind. □

Es ist leicht ersichtlich, dass individuelle Überprüfbarkeit von der universellen vorausgesetzt wird. Um Integrität zu liefern, sollte eine Umfrage universell überprüfbar sein.

Kann man die Integrität einer Umfrage dritten gegenüber beweisen, so wird von Zurechenbarkeit gesprochen:

Definition 13 (individuelle Zurechenbarkeit) Eine Umfrage ist individuell zurechenbar, wenn man jedem Teilnehmer nachweisen kann, welchen Verfügbarkeitsvektor er gesendet hat. □

Definition 14 (universelle Zurechenbarkeit) Eine Umfrage ist global zurechenbar, wenn man dritten gegenüber beweisen kann, dass das Ergebnis einer Umfrage richtig berechnet wurde. □

Universelle Zurechenbarkeit ist eine Anforderung, die besonders für E-Voting Verfahren von Bedeutung ist. In Terminabstimmungsapplikationen geht es in der Regel um Termine in geschlossenen Gruppen ohne die Anforderung, dass die Korrektheit des Ergebnisses dritten gegenüber beweisbar sein muss. Es kann allerdings vorkommen, dass die Teilnahme einzelner Personen wichtig ist und eine individuelle Zurechenbarkeit erwünscht ist.

Es ist leicht ersichtlich, dass individuelle Zurechenbarkeit gegenüber bestimmten Entitäten die Vertraulichkeit gegenüber diesen Entitäten ausschließt.

Das Gegenteil von Zurechenbarkeit, wenn die Integrität eines Verfügbarkeitsvektors dritten gegenüber nicht bewiesen werden kann, ist Abstreitbarkeit.

Definition 15 (Abstreitbarkeit) Eine Umfrage ist abstreitbar, wenn man einem Teilnehmer weder nachweisen kann, welchen Verfügbarkeitsvektor er gesendet hat, noch Teile eines Verfügbarkeitsvektors beweisbar sicher zugeordnet werden können. □

Im weiteren Verlauf dieser Arbeit werden sowohl abstreitbare Abstimmungsschemata als auch ein individuell zurechenbares Schema vorgestellt.

Anforderung 7 (Verfügbarkeit) Das System sollte nach einer angemessenen Zeit ein integeres Ergebnis liefern. □

Wie aus der Definition für Verfügbarkeit ersichtlich ist, beeinflusst die Integrität direkt die Verfügbarkeit. Kann eine Entität also die Integrität stören, so kann sie gleichzeitig auch die Verfügbarkeit stören. Um die Motivation zu senken, eine Integritätsstörung zur Störung der Verfügbarkeit zu nutzen, sollte es möglich sein, die angreifende Entität zu identifizieren. Dieses soll im Folgenden mit Störeridentifikation bezeichnet werden.

Definition 16 (Störeridentifikation) Jede protokollkonforme Entität kann beweisen, dass sie sich protokollkonform verhalten hat. Eine protokollverletzende Entität kann identifiziert werden. □

Beteiligte Entitäten (Mehrseitigkeit)

Verschiedene Personen, die in einer Terminabstimmungsapplikation als Angreifer auftreten können, sind

- der Administrator des Abstimmungsservers (*Serveradministrator*),[6]
- der Initiator einer Terminabstimmung (*Umfrageninitiator*),
- ein *Teilnehmer* einer Abstimmung,[7]
- ein Angreifer, der zwischen den Teilnehmern und dem Abstimmungsserver Zugriff auf die Leitungen hat (*Netzwerkbetreiber*) oder
- ein *Außenstehender*, der nicht an der Terminabstimmung teilnimmt.

Manche dieser Personen können in der Rolle einer anderen Person auftreten. So kann der Administrator des Servers auch in der Rolle des Netzwerkbetreibers auftreten und als außenstehende Person agieren. Der Initiator einer Umfrage ist in der Regel auch Teilnehmer der Abstimmung und kann immer als Außenstehender auftreten. In Abbildung 2.3 wird diese Hierarchie durch ein Diagramm illustriert. Alle Zugeständnisse, die man bzgl. des Vertrauens gegenüber einem Teilnehmer machen muss, müssen auch gegenüber dem Umfrageninitiator getroffen werden. Genauso kann der Administrator des Umfragenservers mindestens so viele Angriffe auf eine Umfrage durchführen, wie der Netzwerkbetreiber oder ein Außenstehender.

Das Vertrauen, welches man dem Initiator einer Umfrage, und das, was man dem Serveradministrator entgegen bringen muss, lässt sich jedoch nicht vergleichen. So könnte der Serveradministrator in einer konkreten Umsetzung beispielsweise relativ leicht die Verfügbarkeit einer Umfrage stören, was für einen Umfrageninitiator nicht so einfach möglich ist. Dafür kann durch eine symmetrische Verschlüsselung aller Inhalte Vertraulichkeit gegenüber dem Serveradministrator erreicht werden, während dem Umfrageninitiator

[6] Theoretisch könnte eine Web-Applikation auch verteilt mit mehreren Servern realisiert sein. Da der Fokus dieser Arbeit allerdings darin liegt, das Vertrauen unter den Teilnehmern zu verteilen und nicht unter mehreren Servern und darüber hinaus der Administrations- und Betriebsaufwand erheblich gesteigert werden würde, wird in dieser Arbeit nur von Applikationen und Schemata ausgegangen, in denen nur *ein* Abstimmungsserver verwendet wird.

[7] In dieser Arbeit wird davon ausgegangen, dass alle Teilnehmer gleich behandelt werden.

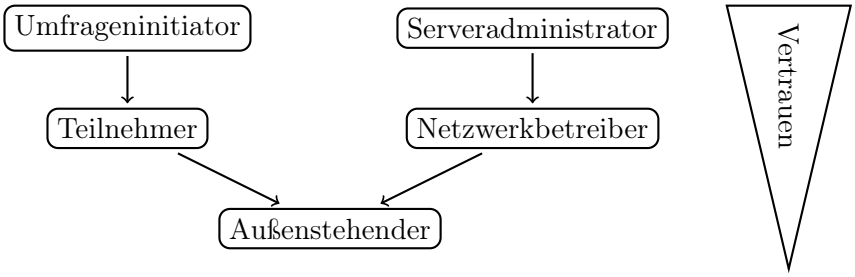


Abbildung 2.3.: Diagramm über das Vertrauen, welches man den Entitäten einer Terminabstimmung entgegen bringen muss. Die Pfeile zeigen welche Rolle eine Entität annehmen kann. Das Vertrauen, welches in Entitäten entgegengebracht werden muss, nimmt in Pfeilrichtung niemals zu.

in so einem Fall bzgl. Vertraulichkeit vertraut werden muss. Ein Vergleich, welches der Schutzziele (Vertraulichkeit vs. Verfügbarkeit) in diesem Beispiel wichtiger ist, kann grundsätzlich nicht geführt werden – das Vertrauen, welches in beide Entitäten erbracht werden muss, ist nicht vergleichbar.

2.2.3. Beziehungen der Anforderungen

Zusammenfassend lassen sich folgende Anforderungen aufstellen:

1. hohe Flexibilität in der Wahl der Terminauswahlfunktion,
2. wenig Installationsaufwand,
3. niedrige Reaktionszeit,
4. wenige blockierende Protokollrunden,
5. hohe Vertraulichkeit,
6. hohe Integrität und
7. hohe Verfügbarkeit.

Dabei sind die Sicherheitsanforderungen jeweils zwischen allen Entitäten zu untersuchen.

Zwischen manchen Anforderungen bestehen Beziehungen. Wolf und Pfitzmann stellten bereits Beziehungen zwischen den Sicherheitsschutzzielen dar [WP00]. In Abbildung 2.4 auf der nächsten Seite sind ähnlich zu Wolf/Pfitzmann die Beziehungen zwischen allen hier vorgestellten Anforderungen zusammengefasst. Im Folgenden werden die Beziehungen nochmal erläutert:

Installationsaufwand vs. Reaktionszeit Alle gängigen Browser unterstützen JavaScript, während Java erst in Form einer virtuellen Maschine nachinstalliert werden muss. Die Java Virtuelle Maschine kann allerdings Operationen schneller ausführen als

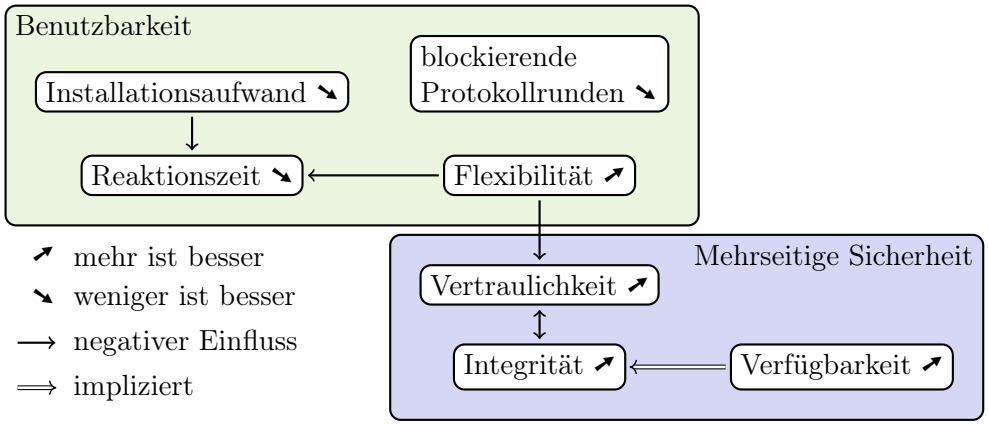


Abbildung 2.4.: Beziehungen zwischen den Anforderungen

gängige JavaScript-Interpreter. Daher hat es einen positiven Einfluss auf den Berechnungsaufwand und damit auf die Reaktionszeit, wenn weniger Ansprüche auf den Installationsaufwand gestellt werden.

Flexibilität vs. Reaktionszeit Um flexibel in der Auswahl der Terminauswahlfunktion zu sein, müssen möglichst genaue Daten vorliegen, weshalb die Verfügbarkeiten der Teilnehmer möglichst genau und umfassend abgefragt werden müssen. Um so mehr Daten erhoben und verarbeitet werden, um so höher ist der Berechnungs- sowie Kommunikationsaufwand, den eine Applikation hat. Daher hat eine höhere Flexibilität einen negativen Einfluss auf die Reaktionszeit.

Flexibilität vs. Vertraulichkeit Wie für den Berechnungs- und Kommunikationsaufwand ist es schlecht für die Vertraulichkeit, wenn mehr persönliche Informationen über die Teilnehmer abgefragt werden.

Vertraulichkeit vs. Integrität In einem individuell zurechenbaren Schema ohne Vertraulichkeit kann die korrekte Ausführung der Auswahlfunktion problemlos von jedem Teilnehmer für seine eigene Stimme überprüft werden. Je weniger Informationen über ihn bekannt werden, desto schwerer wird es auch für ihn die Korrektheit der Verarbeitung zu überprüfen.

Verfügbarkeit vs. Integrität Da die Definition für die Verfügbarkeit voraussetzt, dass das Ergebnis integer ist, impliziert die Verfügbarkeit die Integrität.

Auf das Schutzziel Verfügbarkeit wird im weiteren Verlauf der Arbeit nicht näher eingegangen. Es ist klar, dass ein aktiv angreifender Serveradministrator oder Netzwerkbetreiber die Verfügbarkeit immer stören kann. Des Weiteren können Außenstehende versuchen das Netz bzw. den Umfrageserver zu überlasten. Betrachtet man nur das jeweilige Protokoll, so wird Verfügbarkeit im gleichen Maß erreicht, wie Integrität erreicht wird.

3. Abstimmungsverfahren mit mindestens einer vertrauenswürdigen Entität

Dieses Kapitel gliedert sich in zwei Teile. Zuerst werden bestehende Terminabstimmungsapplikationen und deren Verfahren vorgestellt (Abschnitt 3.1). Diese benötigen alle das Vertrauen in den Administrator des Abstimmungsservers. Anschließend (Abschnitt 3.2) werden neue Verfahren vorgestellt, welche ähnliche Vertrauensmodelle haben, wie die in Abschnitt 3.1, jedoch reduziert um das notwendige Vertrauen in den Serveradministrator.

Tabellen 3.1 und 3.2, welche sich jeweils am Ende der Abschnitte befinden, fassen die Verfahren der Abschnitte jeweils zusammen.

Auf Verfahren ohne einzelne vertrauenswürdige Entität, welche mehr mit E-Voting-Protokollen vergleichbar sind, wird in Kapitel 4 eingegangen.

3.1. Existierende Verfahren mit vertrauenswürdigem Serveradministrator

Es gibt bereits eine Fülle an Applikationen, welche Web 2.0-Terminabstimmungen implementieren. Die wahrscheinlich älteste Applikation dieser Art ist Meet-O-Matic [Mee], welche schon seit 2001 existiert. 2004 wurde Doodle [Näf] ins Leben gerufen, die heute wahrscheinlich meistgenutzte Web 2.0-Terminabstimmungsapplikation. Es finden sich daneben aber auch noch viele andere, z. B. Moreganize [FS], agreeAdate [Pro] etc.

Im Folgenden wird diskutiert, welche Anforderungen von den einzelnen Applikationen erfüllt werden. Da der Hauptunterschied in der Erfüllung der mehrseitigen Sicherheit liegt, sind die folgenden Betrachtungen nach dieser Anforderung gegliedert.

3.1.1. Ohne Zugriffskontrolle

Im einfachsten Fall einer Umfrage, wie sie in Abbildung 2.2 auf Seite 9 vorgestellt wurde, muss man keinerlei Berechtigungen vorzeigen, um an der Umfrage teilzunehmen bzw. sich Ergebnisse anzuschauen. Das Protokoll besteht aus den folgenden drei Schritten:

Umfrageerstellung Der Umfrageninitiator legt die Menge der möglichen Zeitpunkte T fest und schickt allen Teilnehmern einen Link zur Umfrage (URL).

Stimmenabgabe Alle Teilnehmer senden ihre unverschlüsselten Verfügbarkeitsvektoren zum Server.

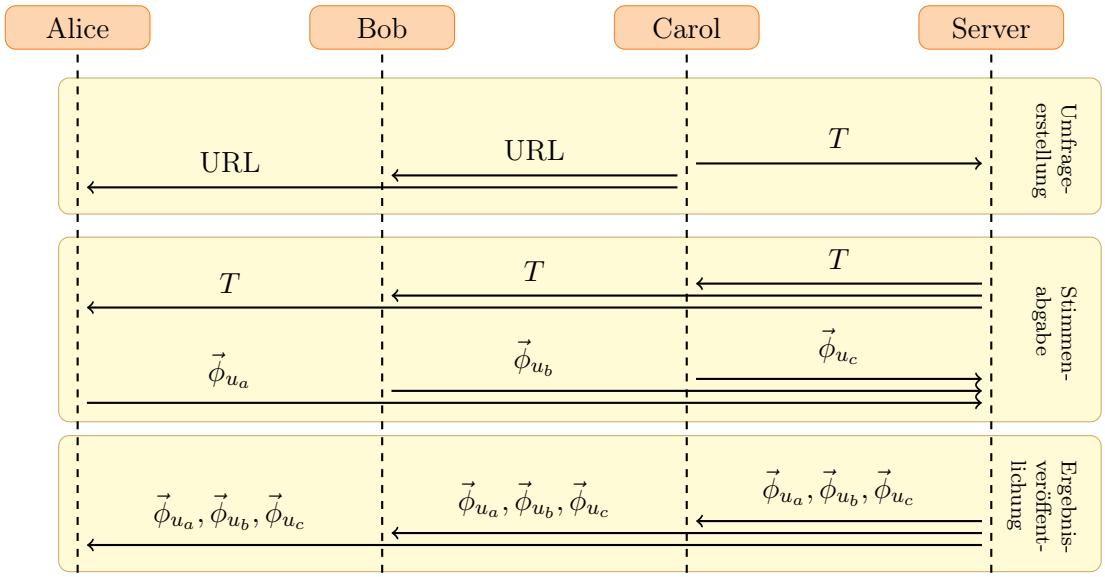


Abbildung 3.1.: Beispiel für den Protokollablauf ohne Zugriffskontrolle (Schema 0)

Ergebnisveröffentlichung Alle Teilnehmer rufen die unverschlüsselten Verfügbarkeitsvektoren vom Server ab.

Abbildung 3.1 zeigt ein Beispiel für einen Protokollablauf mit drei Teilnehmern.

In diesem Protokoll hat man keinerlei Einschränkung für die Terminauswahlfunktion. Installations-, Berechnungs- und Kommunikationsaufwand, sowie die Anzahl der blockierenden Protokollrunden sind minimal. Kurz gesagt erfüllt diese Lösung alle Benutzbarkeitsanforderungen sehr gut. Bezüglich der Sicherheitsanforderungen muss in so einem Fall jedoch allen in Abbildung 2.3 auf Seite 13 gezeigten beteiligten Entitäten vertraut werden.

Dieses Verfahren steht in Tabelle 3.1 auf Seite 21 in der ersten Zeile und ist als Schema 0 gekennzeichnet. Die Tabelle ist nach der Zugriffskontrolle geordnet. Schreibende Zugriffskontrolle bedeutet im Kontext einer Web 2.0-Terminabstimmungsapplikation die Zugriffskontrolle über das Abstimmen. Lesender Zugriff steht dafür, dass man sich das Ergebnis anschauen will. In der Zeile für Schema 0 ist zu erkennen: Es ist keine Zugriffskontrolle erforderlich, wodurch zwar alle Benutzbarkeitsanforderungen erfüllt sind, jedoch werden keinerlei Sicherheitsanforderungen erfüllt.

3.1.2. Schutz gegen außenstehende Angreifer

Eine einfache Lösung, eine Umfrage gegen Außenstehende abzusichern, ist, sie mittels eines Passworts zu schützen. Ist die Passworteingabe nur im Falle einer Stimmenabgabe notwendig und kann jeder die Umfrage ohne Passwort einsehen, so wird ein Integritätsschutz gegen Außenstehende erreicht (Schema 1 in Tabelle 3.1). Wird das Passwort

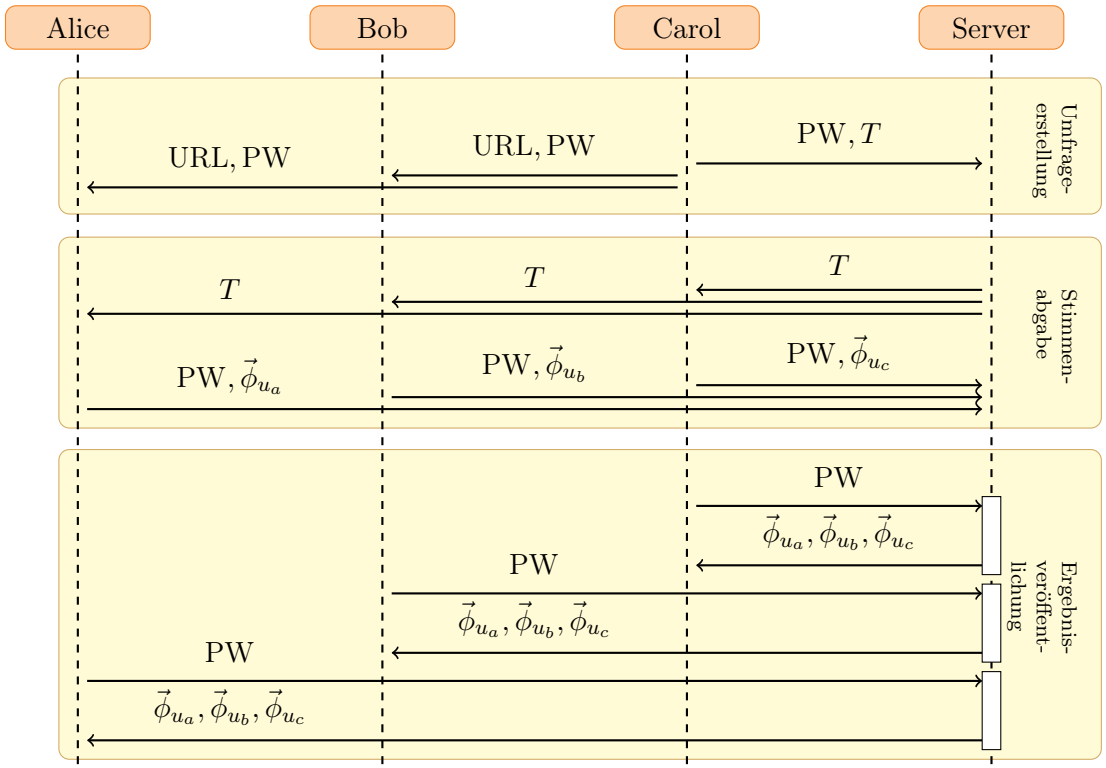


Abbildung 3.2.: Beispiel für den Protokollablauf mit Schutz gegen außenstehende Angreifer (Schema 2)

zusätzlich benötigt, um die Umfrageergebnisse einzusehen, so schützt es gleichzeitig gegen unbefugtes Lesen und erhöht somit die Vertraulichkeit der Umfrage (Schema 2 in dieser Tabelle).

Der Ablauf von Schema 2 besteht aus den folgenden Schritten, welche auch in Abbildung 3.2 dargestellt sind:

Umfrageerstellung Der Initiator der Umfrage legt die Menge der möglichen Zeitpunkte T fest und setzt ein Abstimmungspasswort PW. Das Passwort wird den Teilnehmern zusammen mit dem Link zur Umfrage geschickt.

Stimmenabgabe Jeder Teilnehmer u authentifiziert sich mit dem Abstimmungspasswort PW und sendet seinen unverschlüsselten Verfügbarkeitsvektor $\vec{\phi}_{u_u}$ zum Server.

Ergebnisveröffentlichung Jeder Teilnehmer bekommt nach Passwortauthentifikation alle unverschlüsselten Verfügbarkeitsvektoren.

Das Passwort kann in der Initialisierungsphase entweder vom Umfrageninitiator gewählt oder vom Server festgelegt werden. Letzteres ist in gängigen Implementierungen der

Fall (Doodle, Moreganize, agreeAdate, Meet-O-Matic). Hier wird beim Erstellen einer Umfrage eine zufällige URL erzeugt[1]. Geht man davon aus, dass diese genügend Entropie enthält, so kann die URL als Passwort zum Abstimmen bzw. zum Anschauen der Umfrage angesehen werden.

3.1.3. Schutz gegen angreifende Teilnehmer

Soll Integrität gegenüber anderen Teilnehmern sowie dem Umfrageninitiator erreicht werden, so müssen die einzelnen Stimmen durch unterschiedliche Passwörter (Teilnehmerpasswörter) geschützt werden. Diese Passwörter können entweder von den Teilnehmern selbst in einem Registrierungsschritt gewählt werden (Schema 3 in Tabelle 3.1) oder vom Server für die jeweilige Umfrage festgelegt werden (Schema 4). Der Nachteil, wenn Passwörter selbst gewählt werden, ist die zusätzliche blockierende Protokollrunde in Form einer einmaligen Registrierung am Umfragenserver (vgl. minimale Anzahl blockierender Protokollrunden mit Registrierung auf Seite 10). In Tabelle 3.1 ist diese Einschränkung in Schema 3 mit einem geklammerten Haken dargestellt. Die Schritte für Schema 3 sind:

Registrierung Jeder Teilnehmer u vereinbart ein Passwort PW_u mit dem Server.

Umfrageerstellung Der Initiator der Umfrage legt die Menge der möglichen Zeitpunkte T und die Teilnehmermenge U fest. Der Link zur Umfrage wird den Teilnehmern geschickt.

Stimmenabgabe Jeder Teilnehmer u authentifiziert sich mit seinem Passwort PW_u und sendet seinen unverschlüsselten Verfügbarkeitsvektor $\vec{\phi}_{u_u}$ zum Server.

Ergebnisveröffentlichung Jeder Teilnehmer bekommt nach Eingabe seines Passworts alle unverschlüsselten Verfügbarkeitsvektoren.

Abbildung 3.3 zeigt einen beispielhaften Ablauf mit drei Teilnehmern.

In den gängigen Applikationen sind beide Varianten implementiert. Bei Doodle, Moreganize und agreeAdate ist eine Nutzerregistrierung möglich. Moreganize und agreeAdate setzen darüber hinaus noch servergenerierte Passwörter ein, welche in Form einer URL an die Nutzer geschickt werden. Geht man davon aus, dass die E-Mails, in denen die Passwörter (bzw. URLs) verschickt werden, vertraulich[2] sind und deren Empfänger erreichen[3], so wird universelle Überprüfbarkeit erreicht. Jeder Teilnehmer kann davon ausgehen, dass die Person, die sich hinter der jeweiligen E-Mail-Adresse verbirgt, auch die Stimme abgegeben hat, die angegeben ist.

Die Realisierung von Vertraulichkeit gegenüber anderen Teilnehmern ist auf zwei Arten möglich. Die erste Möglichkeit ist, dass man dem Initiator der Umfrage vertraut. Ist das der Fall, so können die Ergebnisse der Umfrage nur ihm sichtbar gemacht werden (Schema 5, implementiert bei Doodle, Moreganize, agreeAdate und Meet-O-Matic).

[1] Z. B. `http://www.moreganize.ch/bzcGgZx4vpG`

[2] Unter der Annahme, es existiert kein passiver Angreifer, der die E-Mail liest.

[3] Unter der Annahme, es existiert kein aktiver Angreifer, der die E-Mail zurückhält oder eine Reaktion darauf verhindert.

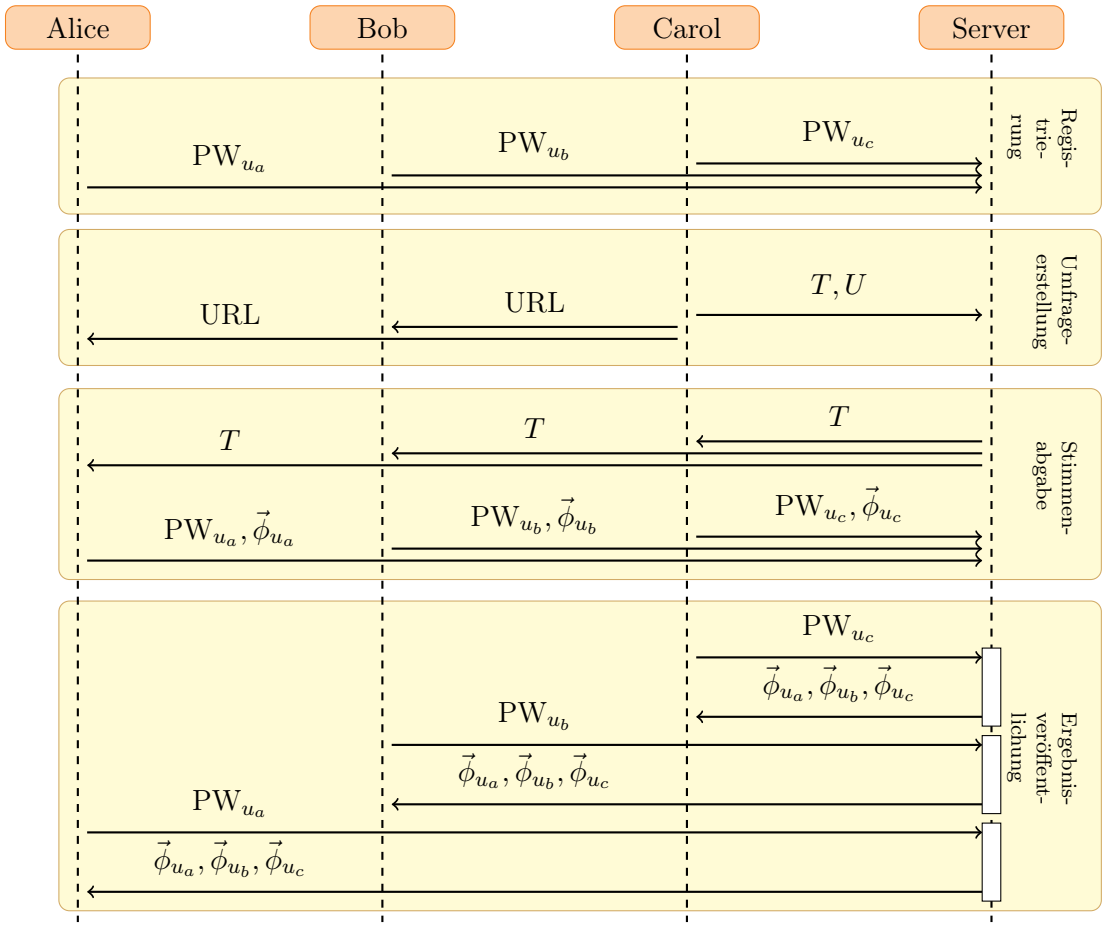


Abbildung 3.3.: Beispiel für den Protokollablauf mit Integritätsschutz gegen Teilnehmerangriffe (Schema 3)

Vertraulichkeit gegenüber den anderen Teilnehmern und dem Initiator kann durch eine geeignete Aggregation der Daten erreicht werden. Eine hier häufig verwendete Methode ist, die Verfügbarkeitsvektoren zu addieren, nur deren Summe zu veröffentlichen und eine Terminauswahlfunktion zu verwenden, die mit dieser Aggregation auskommt (z. B. $\mathcal{S}_{\max}$). Geht man davon aus, dass die Entropie der so aggregierten Werte gering genug ist, so wird Vertraulichkeit gegenüber den anderen Teilnehmern, dem Initiator sowie Außenstehenden erreicht. Um die Integrität der Umfrage in diesem Fall zu gewährleisten, kann auf Teilnehmerpasswörter zurückgegriffen werden (Schema 7). Moreganize ist die einzige der bisher vorgestellten Applikationen, die diese Funktionalität umsetzt, mit der Ausnahme, dass nur ein Passwort (die URL) verwendet wird, welches für alle gleich ist (Schema 6 der Tabelle 3.1). Außer den bisher vorgestellten Applikationen gibt es noch

zahlreiche Umfragedienste, die diese Funktionalität anbieten, z. B. Limeservice [Sch] und SurveyMonkey [Fin].

Auf eine Darstellung von Schema 5,6 und 7 wird hier verzichtet.

Eine weitere Möglichkeit, Vertraulichkeit gegenüber anderen Teilnehmern zu erreichen, ist, dem Server die Terminauswahlfunktion mitzuteilen, worauf dieser nur den geplanten Termin bekannt gibt. Eine solche Lösung hätte zwar einen sehr negativen Einfluss auf die Flexibilität (vgl. Seite 5), würde dafür allerdings mit einer hohen Vertraulichkeit einhergehen. Die Veröffentlichung der Summe der verfügbaren Teilnehmer ist daher ein Kompromiss zwischen Vertraulichkeit und Flexibilität. Diese Lösung findet sich in keiner dem Autor bekannten Applikationen. Interessant wäre, ob sich eine solche Umsetzung beim Benutzer durchsetzen würde.

3.1.4. Schutz gegen den Netzwerkbetreiber

Vertraulichkeit und Integrität gegenüber dem Netzwerkbetreiber zu erreichen ist vergleichsweise einfach. Hier genügt es, eine Ende-zu-Ende-Verschlüsselung/Authentikation zwischen dem Server und dem Benutzer des Dienstes herzustellen. Während nahezu alle Dienste SSL unterstützen, um die Web-Kommunikation zu verschlüsseln und zu authentisieren (Moreganize, Meet-O-Matic, Doodle[4]), bietet kein Dienst eine vergleichbare Maßnahme für die bereits erwähnten Passwort-E-Mails an. Das Vertrauen, welches man dem Netzwerkbetreiber entgegen bringen muss, hängt also hier von der Verschlüsselung der E-Mail-Kommunikation ab.

3.1.5. Zusammenfassung

Eine Übersicht über die vorgestellten Realisierungen findet sich in Tabelle 3.1 auf der nächsten Seite. Die Sicherheitsbetrachtung gegenüber dem Netzwerkbetreiber wurde der Übersichtlichkeit halber nicht in der Tabelle dargestellt. Geht man davon aus, dass die Kommunikation Ende-zu-Ende-verschlüsselt und -authentisiert ist, so wird die gleiche Sicherheit erreicht, wie gegenüber Außenstehenden. Ist dies nicht der Fall und alle Daten werden unverschlüsselt übertragen, so erreicht man die gleiche Sicherheit wie gegen den Serveradministrator.

Aus der Tabelle ist ersichtlich, dass keine der Applikationen versucht, die Notwendigkeit des Vertrauens in den Server in irgendeiner Form zu reduzieren. Im weiteren Verlauf wird sich diese Arbeit also speziell der Frage widmen, wie das notwendige Vertrauen in den Serveradministrator reduziert werden kann.

3.2. Neue Verfahren mit vertrauenswürdigen Teilnehmern

In diesem Abschnitt wird vorgestellt, wie das nötige Vertrauen zum Serveradministrator reduziert werden kann. Es wird dabei ähnlich wie in Abschnitt 3.1 vorgegangen, indem in jedem Unterabschnitt das Vertrauen, welches allen Entitäten entgegengebracht werden

[4]SSL-Verschlüsselung wird nur gegen Bezahlung angeboten.

Tabelle 3.1.: Überblick über Abstimmungsverfahren mit vertrauenswürdigem Serveradministrator

Schema		Zugriffskontrolle schreiben: Abstimmen	Zugriffskontrolle lesen: Ergebnis anschauen	Benutzbarkeit: Flexibilität	Installationsaufwand	Reaktionszeit	block. Protokollrunden	Mehrseitige Sicherheit: Umfrageninitiator	Teilnehmer	Außenstehender	Serveradministrator
0	d, Moreganize, agreeAdate, Meet-O-Matic	—	—	✓	✓	✓	✓				
1		Abstimmungs-PW	—	✓	✓	✓	✓				
2	d, Moreganize, agreeAdate, Meet-O-Matic	Abstimmungs-PW	Abstimmungs-PW	✓	✓	✓	✓				
3	d, Moreganize, agreeAdate	Teilnehmer-PW†	PW	✓	✓	✓	(✓)				
4	Moreganize, agreeAdate	Teilnehmer-PW‡	PW	✓	✓	✓	✓				
5	d, Moreganize, agreeAdate, Meet-O-Matic	Teilnehmer-PW‡	Initiator-PW	✓	✓	✓	✓				
6	Moreganize, LimeService, SurveyMonkey	Abstimmungs-PW	nur Summe		✓	✓	✓				
7		Teilnehmer-PW‡	nur Summe		✓	✓	✓				

d ... Doodle [Näf]
Moreganize ... Moreganize [FS]
agreeAdate ... agreeAdate [Pro]
Meet-O-Matic ... Meet-O-Matic [Mee]
LimeService ... LimeService [Sch]
SurveyMonkey ... SurveyMonkey [Fin]
PW ... Passwort:
Abstimmungs-PW ... alle Teilnehmer haben das gleiche Passwort
Teilnehmer-PW ... jeder Teilnehmer hat ein anderes Passwort
† ... vom Teilnehmer selbst festgelegt
‡ ... vom Server festgelegt und per E-Mail an die Teilnehmer geschickt
✓ ... erfüllt die Anforderung vollständig
(✓) ... erfüllt die Anforderung nahezu (hier: minimale Anzahl blockierender Protokollrunden mit Registrierung)
Vertraulichkeit ... Vertraulichkeit
Integrität ... Integrität

muss, etwas verringert wird. Die Verfahren in Abschnitt 3.1 benötigten alle (mit Ausnahme der Verbindungsverschlüsselung) keine kryptographischen Sicherungsmethoden. In diesem Abschnitt wird zu jedem Verfahren ohne kryptographische Methoden ein Verfahren aufgezeigt, welches ähnliche Sicherheitseigenschaften mittels Kryptographie realisiert. Der Vorteil, kryptographische Methoden zu benutzen, ist hierbei, dass dem Serveradministrator weniger Vertrauen entgegen gebracht werden muss.

3.2.1. Allen Teilnehmern vertrauen

Es wurde in Abschnitt 3.1.2 bereits gezeigt, dass das notwendige Vertrauen, welches im einfachsten Abstimmungsverfahren einem Außenstehenden entgegengebracht werden muss, mit einer Passwortabfrage reduziert werden kann (Schema 0 vs. Schema 1–4 in Tabelle 3.1). Das Passwort konnte entweder nur zum Abstimmen verlangt werden (Schema 1) oder zusätzlich zum Anschauen des Ergebnisses (Schema 2–4), um die Vertraulichkeit zu schützen.

Benutzt man dieses Passwort als Schlüssel zum Authentisieren der Stimme mittels eines symmetrischen Authentikationssystems, so ist der gleiche Integritätsschutz gegenüber Außenstehenden erreicht wie im Schema ohne Kryptographie. Zusätzlich schützt die Authentikation allerdings noch gegen unbefugte Änderungen durch den Serveradministrator. Um das zu erreichen, muss das Passwort natürlich clientseitig generiert werden. Wird davon ausgegangen, dass die Passwörter vertraulich zu den Teilnehmern geschickt werden, so kann darüber hinaus noch Integrität gegenüber dem Netzwerkbetreiber erreicht werden.

Dieses Schema ist u. a. in Tabelle 3.2 auf Seite 27 dargestellt. Es ist dort mit $1'$ markiert, da es sich um eine Erweiterung des Schemas 1 handelt. Es ist zu sehen, dass zusätzlich zur Integrität gegen Außenstehende ein Integritätsschutz gegenüber dem Serveradministrator erfolgt.

Äquivalent zur Benutzung des Passworts für die Authentikation der einzelnen Stimmen kann das Passwort dazu benutzt werden, die Verfügbarkeitsvektoren zu verschlüsseln (Schema $2'$ als Erweiterung zu Schema 2 in Tabelle 3.2). Auf diese Weise wird Vertraulichkeit gegenüber dem Serveradministrator bzw. dem Netzwerkbetreiber erreicht. Das Schema benötigt folgende Schritte:

Umfrageerstellung Der Umfrageninitiator legt die Menge der möglichen Zeitpunkte T sowie ein Abstimmungspasswort PW fest. Das Abstimmungspasswort dient als Schlüssel für eine symmetrische Verschlüsselungs- und Authentikationsfunktion[5] $\text{encAuth}_{\text{PW}}^{\text{sym}}(\cdot)$, mit der alle Nachrichten auf Clientseite verschlüsselt und authentifiziert werden. Das Passwort und der Link zur Umfrage (URL) werden vertraulich zu allen Teilnehmern verschickt.

Stimmenabgabe Jeder Teilnehmer u verschlüsselt und authentifiziert seinen Verfügbarkeitsvektor und überträgt diesen zum Server ($\text{encAuth}_{\text{PW}}^{\text{sym}}\left(\vec{\phi}_u\right)$).

[5] Hier kann eine Chiffre in einem Betriebsmodus benutzt werden, der Vertraulichkeit und Authentizität garantiert.

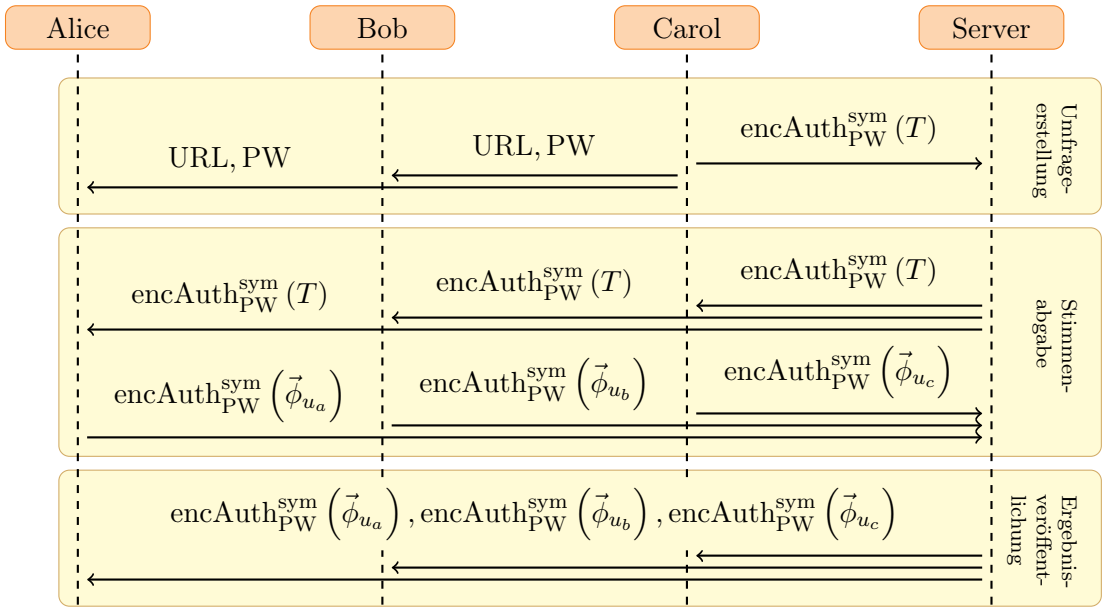


Abbildung 3.4.: Beispiel für den Protokollablauf mit Schutz gegen außenstehende Angreifer sowie den Serveradministrator (Schema 2′)

Ergebnisveröffentlichung Alle Teilnehmer rufen die verschlüsselten Verfügbarkeitsvektoren vom Server ab, entschlüsseln diese und überprüfen die Authentizität der Nachrichten.

Abbildung 5.9 verdeutlicht den Kommunikationsablauf an einem Beispiel.

Ein so arbeitendes Schema schränkt die Benutzbarkeit nicht zwangsläufig ein. Symmetrische Verfahren sind schnell genug[6], um keinen signifikanten Einfluss auf den Berechnungsaufwand auszuüben. An der Flexibilität in der Wahl der Terminauswahlfunktion, der Anzahl der blockierenden Protokollrunden sowie am Kommunikationsaufwand ändert sich nichts.

3.2.2. Nur dem Umfrageinitiator vertrauen

In den Verfahren, die ohne Kryptographie auskommen, wurde Integrität gegenüber anderen Teilnehmern erreicht, indem jeder Teilnehmer ein anderes Passwort benutzt (Schema 3 und 4). Wählt man statt des Passworts ein digitales Signaturverfahren, so kann das Vertrauen in den Serveradministrator weiter eingeschränkt werden. Die Schlüssel, welche für die digitalen Signaturen verwendet werden, können entweder vom Initiator der Umfrage festgelegt oder aber von jedem Teilnehmer selbst generiert werden. Legt der Initiator die

[6] Symmetrische Verfahren sind auch in einer JavaScript Implementierung schnell genug und können damit ohne Einfluss auf den Installationsaufwand benutzt werden.

Schlüssel fest (Schema 4′), so benötigt man keinen initialen Registrierungsschritt, muss dafür allerdings dem Initiator der Umfrage bzgl. der Integrität vertrauen. Generieren alle Teilnehmer ein eigenes Schlüsselpaar (Schema 3′), so entfällt dieses Vertrauen und das Schema bietet eine individuell zurechenbare Abstimmung.

Egal wer die Schlüssel generiert, es muss in beiden Fällen unterstellt werden, dass der Schlüsselaustausch authentisch von statten geht. Im Falle, der Initiator wählt die Schlüssel, muss die Kommunikation zwischen ihm und den Teilnehmern darüber hinaus noch vertraulich statt finden. Ist dies nicht der Fall, so hat der Netzbetreiber die Möglichkeit die Schlüssel mitzuhören bzw. zu verändern.

Beide Verfahren können mit einer symmetrischen Verschlüsselung kombiniert werden, um Vertraulichkeit gegenüber dem Serveradministrator und dem Netzwerkbetreiber zu erreichen.

Der Ablauf von Schema 3′ ist wie folgt:

Registrierung Jeder Teilnehmer u hinterlegt einen digitalen Signaturschlüssel auf einem Schlüsselserver.

Umfrageerstellung Der Umfrageninitiator legt die Menge der möglichen Zeitpunkte T sowie ein Abstimmungspasswort PW fest. Das Abstimmungspasswort dient als Schlüssel für eine symmetrische Verschlüsselungsfunktion $\mathrm{enc}^{\mathrm{sym}}_{\mathrm{PW}}(\cdot)$, mit der alle Nachrichten auf clientseitig verschlüsselt werden. Das Passwort sowie die URL werden vertraulich zu den Teilnehmern geschickt.

Stimmenabgabe Jeder Teilnehmer u signiert seinen Verfügbarkeitsvektor digital, verschlüsselt ihn anschließend symmetrisch und überträgt diesen zum Abstimmungsserver $(\mathrm{enc}^{\mathrm{sym}}_{\mathrm{PW}}\left(\vec{\phi}_u, \mathrm{sig}_u\left(\vec{\phi}_u\right)\right))$.

Ergebnisveröffentlichung Alle Teilnehmer rufen die verschlüsselten und signierten Verfügbarkeitsvektoren vom Server ab, entschlüsseln diese und überprüfen die Authentizität der Nachrichten.

Abbildung 3.5 zeigt einen beispielhaften Kommunikationsablauf des Protokolls.

Die Reaktionszeit wird bei diesem Protokoll nur wenig beeinflusst. Beim Abstimmen muss jeder Teilnehmer nur eine Signatur für seinen Verfügbarkeitsvektor berechnen. Um das Ergebnis zu überprüfen muss jeder Teilnehmer die Signaturen der anderen Teilnehmer überprüfen.

Wird dem Initiator der Umfrage vollständig vertraut und soll Vertraulichkeit gegenüber den anderen Teilnehmern der Umfrage erreicht werden, konnten die Ergebnisse nur dem Initiator sichtbar gemacht werden (Schema 5). Um dieses Verfahren kryptographisch abzusichern und Vertraulichkeit gegenüber dem Serveradministrator zu erreichen, sollten die Verfügbarkeitsvektoren der Teilnehmer asymmetrisch an den Initiator der Umfrage verschlüsselt werden (Schema 5′). In Kombination mit digitalen Signaturen wird hier wiederum sichergestellt, dass die Stimmen nur von berechtigten Teilnehmern abgeschickt wurden. Die Schlüssel für die Signaturen können vom Initiator gewählt werden, da diesem ohnehin bzgl. Integrität vertraut werden muss.

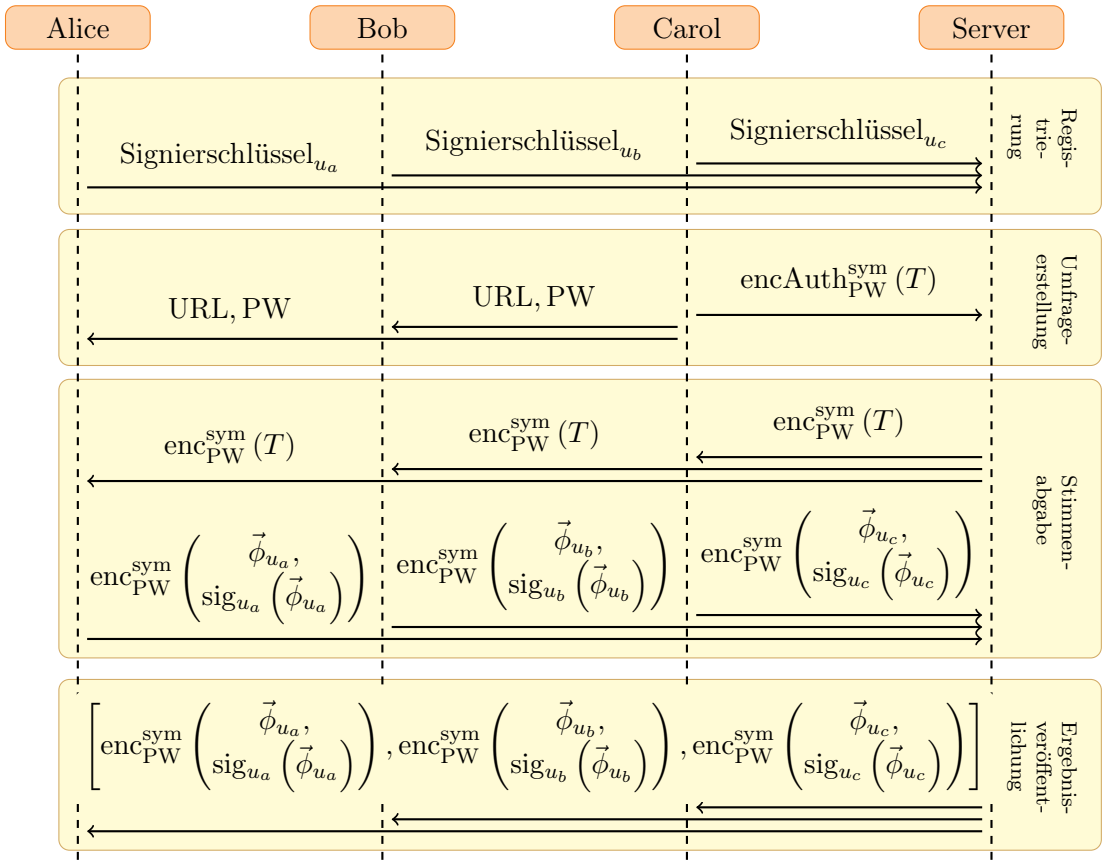


Abbildung 3.5.: Beispiel für den Protokollablauf mit Integritätsschutz gegen Teilnehmerangriffe sowie Integritäts- und Vertrauluchkeitsschutz gegen den Serveradministrator (Schema 3′)

3.2.3. Ausblick auf Schemata ohne vertrauenswürdige Entitäten

Die letzte Möglichkeit, die in Abschnitt 3.1.3 vorgestellt wurde, mit der sowohl Vertraulichkeit als auch Integrität gegenüber allen Teilnehmern sowie dem Umfrageninitiator erreicht wurden, war nur aggregierte Werte der Verfügbarkeitsvektoren zu veröffentlichen (Schema 6 und 7). Während die bisher vorgestellten Methoden mit 1-2 kryptographischen Primitiven pro Schema auskamen, um die Sicherheitsschutzziele gegenüber dem Serveradministrator zu erreichen, muss hier deutlich mehr Aufwand betrieben werden. Das Problem besteht darin, dass nach der Veröffentlichung des aggregierten Wertes (z. B. der Summe aller Verfügbarkeitsvektoren) niemand auf einfache Weise nachvollziehen kann, welche Verfügbarkeitsvektoren von welchen Teilnehmern in die Berechnung eingegangen sind. In der einfachen Version wurde hierbei darauf vertraut, dass der Umfragenserver die Berechnung korrekt durchführt und auch nur berechtigte Teilnehmer an der Umfrage

teilnehmen lässt. Ohne diese überprüfende Instanz entsteht der typische Konflikt zwischen Überprüfbarkeit (Integrität) und Vertraulichkeit.

Auf eine Lösung für dieses Problem wird gesondert in Kapitel 4 eingegangen. In Tabelle 3.2 ist diese Lösung schon als Schema 6′ und 7′ dargestellt.

Schema 7′ macht minimale Vertrauensannahmen, kommt dafür aber nicht ohne Registrierungsschritt aus. Wird der Umfrageninitiator als vertrauenswürdig angesehen, so kann er die Schlüssel für dieses Verfahren generieren (Schema 6′). Dieses kommt ohne Registrierungsschritt aus (wie Schema 6), allerdings muss dem Initiator bzgl. Vertraulichkeit sowie Integrität vertraut werden. Des Weiteren leidet die Flexibilität in der Wahl der Auswahlfunktion bei Schema 6′ und 7′ natürlich auf gleiche Weise, wie sie es bei der Aggregation ohne kryptographische Methoden getan hat.

3.2.4. Zusammenfassung

Eine tabellarische Zusammenfassung der einzelnen Verfahren wird in Tabelle 3.2 gezeigt. Hier sind die bestehenden Verfahren aus Tabelle 3.1 jeweils ausgegraut über ihrem Pendant dargestellt, welches Kryptographie verwendet, um das nötige Vertrauen in den Serveradministrator zu reduzieren. In der letzten Spalte jeder Zeile ist zu sehen, dass das Vertrauen in den Serveradministrator jeweils im Vergleich zur vorherigen Zeile etwas eingeschränkt wurde.

Abbildung 3.6 zeigt den Vertrauensgewinn der einzelnen Schemata der Tabelle mit Ausnahme der Schemata 4, 4′, 6 und 6′ nochmals graphisch. Diese Darstellung veranschaulicht, dass mit Hilfe der kryptographischen Methoden zu jedem Modus ohne Kryptographie ein Pendant gefunden wurde, in welchem das benötigte Vertrauen geringer ist.

Die Schemata 4 und 6 stellen hierbei eine Ausnahme dar. Bei diesen wurden die Passwörter vom Server gewählt, bei den Schemata 4′ und 6′ vom Initiator. Daher muss bei Schema 4′ und 6′ dem Initiator der Umfrage vertraut werden, was bei 4 und 6 nicht nötig war.

Auf die Beispiele zurückblickend, die in Kapitel 1 gebracht wurden, kann man feststellen, dass sich die vier Vorstandsvorsitzenden aus Beispiel 1 wohl am ehesten auf Schema 2′ einlassen würden. In diesem vertrauen die Teilnehmer einander, sind aber gegen Angriffe von außerhalb sicher. Der Professor, der den Prüfungsbedarf ermitteln wollte (Beispiel 2) greift am besten auf Schema 5′ zurück, in welchem nur ihm als Umfrageninitiator vertraut werden muss. Schließlich bleiben noch die 20 Programmierer aus Beispiel 3. Diesen Personen sei Schema 7′ empfohlen, da sie in diesem keiner anderen Entität vertrauen schenken müssen.

Hier ist ersichtlich, dass keines der vorgestellten Schemata das optimale Schema für Jedermann ist. Vielmehr gilt es abzuwägen, gegen welche Art Angreifer man sich schützen will und welche Benutzbarkeitseinbußen man dafür in Kauf nimmt.

Tabelle 3.2.: Überblick über Abstimmungsverfahren ohne vertrauenswürdigem Serveradministrator

Schema	Zugriffskontrolle: schreiben (Abstimmen)	Zugriffskontrolle: lesen (Ergebnis anschauen)	Benutzbarkeit: Flexibilität	Benutzbarkeit: Installationsaufwand	Benutzbarkeit: Reaktionszeit	Benutzbarkeit: block. Protokollrunden	Mehrseitige Sicherheit: Umfrageninitiator	Mehrseitige Sicherheit: Teilnehmer	Mehrseitige Sicherheit: Außenstehender	Mehrseitige Sicherheit: Serveradministrator
1	Abstimmungs-PW	—	✓	✓	✓	✓			🔒	
1′	symmetrische Auth.	—	✓	✓	✓	✓			🔒	🔒
2	Abstimmungs-PW	Abstimmungs-PW	✓	✓	✓	✓			🔒	
2′	symmetrische Auth.	symmetrische Verschl.	✓	✓	✓	✓			🔒	🔒
3	Teilnehmer-PW†	PW	✓	✓	✓	(✓)		🔒	🔒	
3′	digitale Signatur*	symmetrische Verschl.	✓	✓	✓	(✓)		🔒	🔒	🔒
4	Teilnehmer-PW‡	PW	✓	✓	✓	✓		🔒	🔒	
4′	digitale Signatur**	symmetrische Verschl.	✓	✓	✓	✓		🔒	🔒	🔒
5	Teilnehmer-PW‡	Initiator-PW	✓	✓	✓	✓		🔒	🔒	
5′	digitale Signatur**	asym. Verschl. an Initiator	✓	✓	✓	✓		🔒	🔒	🔒
6	Abstimmungs-PW	nur Summe		✓	✓	✓	🔒	🔒	🔒	
6′	digitale Signatur**	nur Summe berechenbar		✓	✓	✓		🔒	🔒	🔒
7	Teilnehmer-PW‡	nur Summe		✓	✓	✓	🔒	🔒	🔒	
7′	digitale Signatur*	nur Summe berechenbar		✓	✓	(✓)	🔒	🔒	🔒	🔒

PW . . . Passwort:
Abstimmungs-PW . . . alle Teilnehmer haben das gleiche Passwort
Teilnehmer-PW . . . jeder Teilnehmer hat ein anderes Passwort

† . . . vom Teilnehmer selbst festgelegt

* . . . Schlüssel vom Teilnehmer selbst festgelegt

‡ . . . vom Server festgelegt und per E-Mail an die Teilnehmer geschickt

** . . . Schlüssel vom Initiator festgelegt und per E-Mail an die Teilnehmer geschickt

✓ . . . erfüllt die Anforderung vollständig

(✓) . . . erfüllt die Anforderung nahezu (hier: minimale Anzahl blockierender Protokollrunden mit Registrierung)

🔒 . . . erfüllt die Anforderung Vertraulichkeit gegenüber der jeweiligen Entität

🔴 . . . erfüllt die Anforderung Integrität gegenüber der jeweiligen Entität

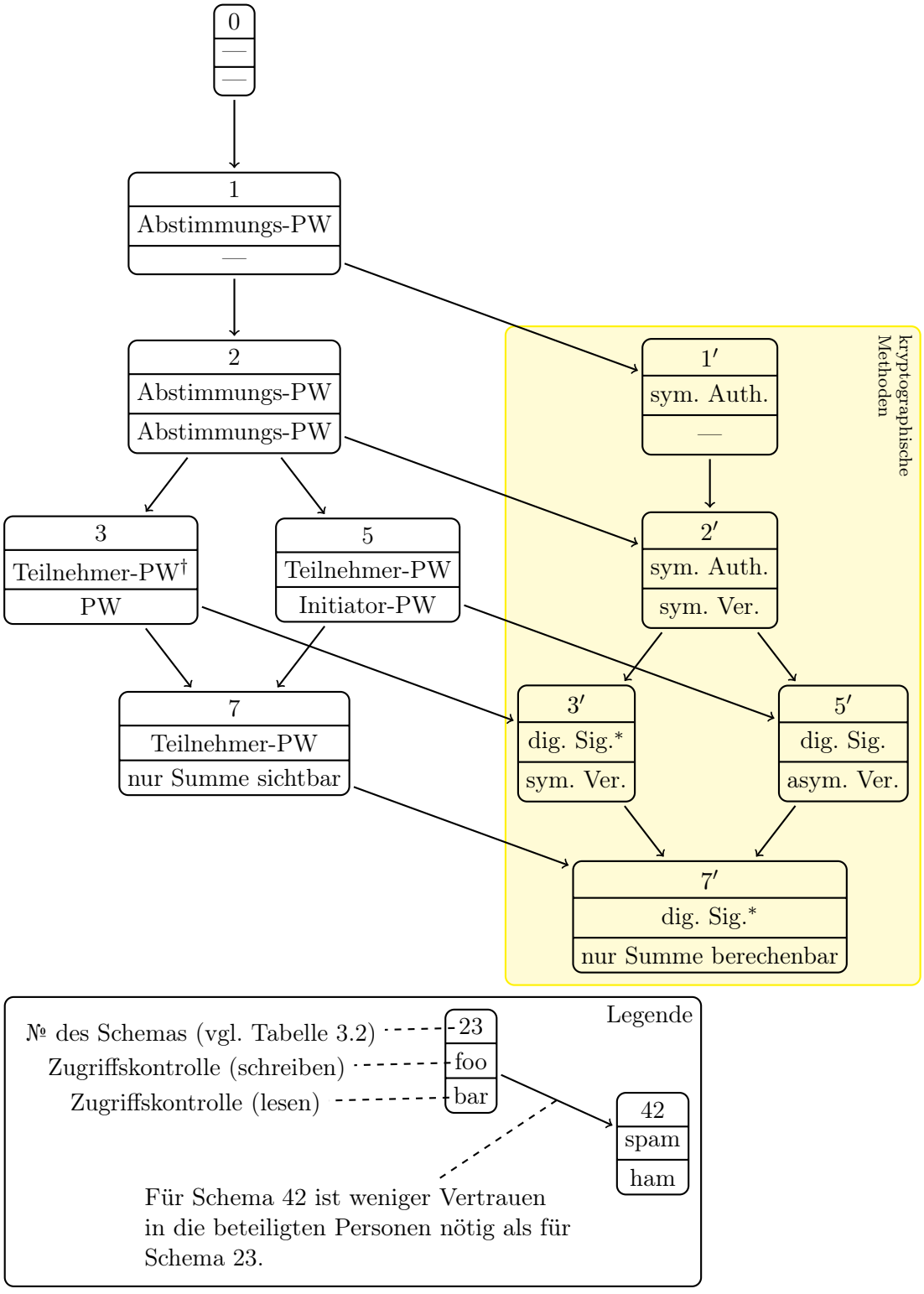


Abbildung 3.6.: Diagramm über das notwendige Vertrauen, welches den verschiedenen Entitäten entgegengebracht werden muss.

4. Abstimmungsverfahren ohne vertrauenswürdige Entitäten

Im Folgenden wird ein Terminabstimmungsprotokoll vorgestellt, welches die Abstimmung über einen Termin unter minimalen Annahmen über die anderen Entitäten leistet (vgl. Abschnitt 2.2.2). Es aggregiert dabei die einzelnen Verfügbarkeitsvektoren der Teilnehmer, indem die Summe der verfügbaren Teilnehmer berechnet wird. Darüber hinaus werden keine Informationen über die Verfügbarkeitsvektoren der einzelnen Teilnehmer veröffentlicht. Gleichzeitig erlaubt es die Integrität der Abstimmung zu überprüfen.

Im ersten Abschnitt dieses Kapitels wird die aktuelle Literatur zu diesem Thema (Abschnitt 4.1) betrachtet. Ein vereinfachtes Protokoll, welches von protokolltreuen Teilnehmern ausgeht wird in Abschnitt 4.2 vorgestellt. Diese Grundidee wurde gemeinsam mit Rainer Böhme entwickelt [KB09]. Das Protokoll wird anschließend auf protokollverletzende Angreifer innerhalb der Teilnehmer erweitert (Abschnitt 4.3, [Kel09; Kel11]). Die zwei wesentlichen Protokollschritte (Stimmenabgabe und Ergebnisveröffentlichung) sind in Anhang B und C zusammengefasst.

4.1. Existierende Verfahren

Es findet sich nur wenig Literatur, welche sich mit speziellen Protokollen zur Terminabstimmung beschäftigen. Speziell über die mehrseitig sichere Web 2.0-Terminabstimmung gibt es noch keine Literatur. Mehrseitig sichere Terminabstimmung lässt sich jedoch mit verschiedenen anderen Methoden durchführen. Diese werde im Folgenden vorgestellt.

4.1.1. E-Voting

In E-Voting Protokollen wird ein ähnliches Ziel verfolgt wie in einer Terminabstimmung. Viele Wählern stimmen über eine oder mehrere Optionen ab. Ziel ist es pro Option die Anzahl der Wähler herauszufinden, ohne Kenntnis über die einzelnen Stimmen zu erlangen.

Es existiert bereits viel Literatur über E-Voting, z. B. [Cha81; FOO92; CBT94; SK95; CGS97; HS00; JCJ05]. Manche Protokolle wurden auch bereits in Implementierungen umgesetzt [Her97; Adi08; Cla+08].

Wird ein E-Voting Protokoll benutzt um eine Terminabstimmung durchzführen, können drei Beobachtungen getroffen werden:

Keine „short ballot assumption“ Einzelne Wähler können aufgrund der im Wahlzettel enthaltenen Entropie identifiziert werden. Je mehr unterschiedliche Optionen ein

Wähler hat (je länger der Wahlzettel ist), desto mehr Entropie beinhaltet der Wahlzettel und desto einfacher ist es einen Wähler zu identifizieren. Hat ein Angreifer Kenntnis über einen Teil der Optionen, die ein Wähler wählen wird, so kann er darüber den Wahlzettel identifizieren und auf den Rest des Stimmzettels schließen. Dieser Angriff gelingt umso besser, je kleiner die Anonymitätsmenge der Wähler ist, da mit der Größe der Anonymitätsmenge auch die Wahrscheinlichkeit bestimmt wird, dass zwei Stimmzettel auftauchen, die das gleiche Muster haben. In E-Voting Systemen muss daher die *short ballot assumption* [RS07] gelten.

Im Gegensatz zu einem (oft) sehr kurzen Wahlzettel enthält ein Verfügbarkeitsvektor viel Entropie. Zusätzlich ist die Anzahl der Teilnehmer einer Terminumfrage deutlich geringer als die Anzahl der Wähler in einer von E-Voting adressierten Wahl. Deshalb gilt die short ballot assumption bei Terminabstimmungen nicht.

Um die short ballot assumption zu umgehen muss ein E-Voting Schema für jeden zur Wahl stehenden Zeitpunkt einzeln durchgeführt werden. So wird die durch den gesamten Verfügbarkeitsvektor entstehende Entropie auf einzelne Zeitpunkte entkoppelt.

Teilnehmer = Wahlobmann Bei E-Voting Systemen wird das Vertrauen typischerweise auf einige Wahlobmänner (engl. *election officials*) verteilt. Wie schon in Abschnitt 2.2.2 diskutiert wurde, wird in dieser Arbeit versucht, das Vertrauen auf die Teilnehmer der Web-Applikation zu verteilen. Um das zu erreichen, muss jeder Teilnehmer ein Wahlobmann sein.

keine Erpressungsfreiheit Schutz vor Beeinflussung der Stimmenabgabe durch Erpressung o. ä. (engl. *coercion resistance*) und die damit verbundene Quittungsfreiheit sind bei Terminabstimmungssystemen unkritisch.

Im Wesentlichen konzentriert sich die E-Voting-Literatur auf mixbasierte Techniken[1] und homomorphiebasierte Techniken.

Mix

Chaum stellte Mixe als einen Baustein für anonyme Kommunikation vor und beschrieb als erster, wie man sie für Wahlen benutzen kann [Cha81]. Seinem Vorschlag folgten anschließend viele Erweiterungen [PIK94; SK95; Oga+97; Abe98; Jak98].

Eine Erweiterung der mixbasierten Lösung wurde kurz nach dem ersten Protokoll von Chaum vorgestellt [Cha88a]. Diese benutzt blinde Signaturen [Cha85] um Wahlberechtigungen auszustellen. Fujioka et al. reduzierte die Komplexität von Chaums Idee um das Schema für größere Wahlszenarien benutzbar zu machen [FOO92]. Viele weitere Artikel erweitern das Fujioka–Okamoto–Ohta Schema [Sak94; Ohk+99; DuR99], später war das Protokoll Vorlage für Implementierungen [CC97; Her97].

[1] bzw. Protokolle, welche ein Anonymisierungsnetz benutzen

Die Grundidee, Mixe für Wahlen zu benutzen, besteht darin ein anonymes schwarzes Brett mit ihnen aufzubauen.[2]. Das Verfahren wird in zwei Phasen aufgeteilt. In einer ersten Phase generieren alle Teilnehmer einen asymmetrischen Schlüssel und veröffentlichen diesen auf dem anonymen schwarzen Brett. Die Wahlbehörde hat hierbei sicherzustellen, dass nur berechtigte Teilnehmer Eingabenachrichten für die Mixe versenden. Jeder Teilnehmer überprüft, ob sein Schlüssel ordentlich durch die Mixe veröffentlicht wurde und kann – im Falle ein Mix verhält sich nicht korrekt – die Aufdeckung verlangen, da noch keine privaten Informationen verschickt werden.

Anschließend, in einer zweiten Phase, verschlüsselt jeder Teilnehmer seine Stimme mit seinem geheimen Schlüssel und veröffentlicht die verschlüsselten Stimmen auf dem anonymen schwarzen Brett. Jeder Teilnehmer kann anschließend alle Stimmen entschlüsseln und das Ergebnis zusammenzählen.

Wie bereits erwähnt wurde, sollte bei einer Umsetzung einer Web 2.0-Terminabstimmung mittels eines E-Voting-Protokolls jeder Teilnehmer ein Wahlobmann sein. Da in dem Protokoll jeder Wahlobmann einen Mix kontrolliert, hat das zur Folge, dass jeder Teilnehmer eine Mix-Instanz darstellt. Daher würde jeder Teilnehmer seine Stimmen asymmetrisch für jeden anderen Teilnehmer verschlüsseln. Da die Verfügbarkeitsvektoren viel Entropie enthalten, muss für jede Verfügbarkeit ein einzelner Wahlzettel abgegeben werden. Werden die Verfügbarkeitsvektoren so außeinandergenommen, so ergibt sich eine Komplexität, die linear von der Anzahl der Teilnehmer *und* von der Anzahl der Zeitpunkte abhängt ($\mathcal{O}(|T| \cdot |U|)$).

Unabhängig von der Berechnungskomplexität besteht das Problem, dass Wahlzettel in einem zweistufigen Verfahren veröffentlicht werden (zwei Phasen), wodurch eine zusätzliche blockierende Protokollrunde[3] entsteht.

Homomorphe Verschlüsselung

Ein Wahlschema auf Basis homomorpher Verschlüsselung wurde zuerst von Cohen und Fischer beschrieben [CF85] und später von Benaloh und Yung erweitert [CBY86]. Einige Schemata auf der Basis homomorpher Verschlüsselung wurden daraufhin in den letzten Jahren vorgestellt [CBT94; SK94; CGS97; HS00; JCJ05].

Die Idee ist, dass alle Stimmen in einer Form verschlüsselt werden die es zulässt, dass die verschlüsselten Werte aggregiert werden können und nur die aggregierte Form entschlüsselt wird. Formal ausgedrückt ist die Verschlüsselungsfunktion $\mathrm{enc}(\cdot)$ ein Homomorphismus, so dass für zwei Operationen $\otimes, \oplus$ gilt: $\mathrm{enc}(x_1) \otimes \mathrm{enc}(x_2) = \mathrm{enc}(x_1 \oplus x_2)$.

Verschlüsselt man beispielsweise eine Nachricht x mit dem ElGamal [EG85] Schlüsselpaar $(\mathsf{pub} = g^{\mathsf{sec}}, \mathsf{sec})$, so ergibt sich:[4]

$$\mathrm{enc}_{\mathsf{pub}}(x) := (g^r, x \cdot (g^{\mathsf{sec}})^r) = (g^r, x \cdot g^{\mathsf{sec} \cdot r}) \tag{4.1}$$

[2] Statt ein anonymes schwarzes Brett mit Hilfe von Mixen aufzubauen kann hier auch das DC-Netz [Cha88b] verwendet werden [PW92]

[3] vgl. Anforderung 4

[4] Auf die Darstellung des Modulo wurde in der Gleichung verzichtet.

Multipliziert man zwei verschlüsselt Nachrichten, welche mit dem selben Schlüssel verschlüsselt wurden, so multiplizieren sich auch deren Klartexte:

$$\begin{aligned}\mathrm{enc}_{\mathsf{pub}}(x_1) \cdot \mathrm{enc}_{\mathsf{pub}}(x_2) &= (g^{r_1}, x_1 \cdot g^{\mathsf{sec}\cdot r_1}) \cdot (g^{r_2}, x_2 \cdot g^{\mathsf{sec}\cdot r_2}) \\ &= \left(g^{(r_1+r_2)}, x_1 \cdot x_2 \cdot g^{\mathsf{sec}\cdot(r_1+r_2)}\right) \\ &= \mathrm{enc}_{\mathsf{pub}}(x_1 \cdot x_2)\end{aligned} \tag{4.2}$$

Um diesen Sachverhalt für Wahlen auszunutzen wird festgelegt, dass die Klartextnachricht x mit einem zweiten Generator h wie folgt kodiert wird:

$$x := h^{\phi},\ \phi := \begin{cases} 1 & \text{Zustimmung} \\ 0 & \text{sonst.} \end{cases} \tag{4.3}$$

Werden unter diesen Bedingungen zwei verschlüsselte Wahlzettel miteinander multipliziert, so erhält man:

$$\begin{aligned}\mathrm{enc}_{\mathsf{pub}}\left(h^{\phi_1}\right) \cdot \mathrm{enc}_{\mathsf{pub}}\left(h^{\phi_2}\right) &= \left(g^{r_1}, h^{\phi_1} \cdot g^{\mathsf{sec}\cdot r_1}\right) \cdot \left(g^{r_2}, h^{\phi_2} \cdot g^{\mathsf{sec}\cdot r_2}\right) \\ &= \left(g^{(r_1+r_2)}, h^{\phi_1+\phi_2} \cdot g^{\mathsf{sec}\cdot(r_1+r_2)}\right) \\ &= \mathrm{enc}_{\mathsf{pub}}\left(h^{\phi_1+\phi_2}\right).\end{aligned} \tag{4.4}$$

Aus diesem Ergebnis kann die Summe der Zustimmungen extrahiert werden, indem der diskrete Logarithmus von $h^{\phi_1+\phi_2}$ gezogen wird. Die Berechnung des diskreten Logarithmus kann zwar nicht allgemein durchgeführt werden, ist jedoch für kleine Zahlen berechenbar.[5]

Hätte eine einzelne Instanz den geheimen Schlüssel sec, so könnte sie nicht nur das Ergebnis sondern auch alle Teilstimmen berechnen. Deshalb werden alle Stimmen mittels einer *Schwellwertverschlüsselung* verschlüsselt. Hier wird der geheime Schlüssel auf mehrere Wahlobmänner verteilt.[6] Um eine Nachricht zu entschlüsseln sind mehrere Wahlobmänner notwendig. Auch die Entschlüsselung wird auf eine Weise durchgeführt, bei der keine einzelne Partei den gesamten Schlüssel erfährt, wodurch auch nach der Ergebnisberechnung die Einzelstimmen geheim bleiben.

Die Schwierigkeit einer so durchgeführten Berechnung liegt darin, dass ein Angreifer in seiner Stimme ϕ andere als die erlaubten Werte kodieren kann. Um das zu verhindern beweist jeder Wähler mittels eines Zero-Knowledge-Beweises, dass er seine Stimme richtig kodiert hat. Die Komplexität eines solchen Beweises ist relativ hoch, es werden abhängig von einem Sicherheitsparameter linear viele diskrete Exponentationen benötigt.

Zusätzlich zum hohen Berechnungsaufwand werden mindestens drei blockierende Protokollrunden benötigt: Zuerst einigen sich alle Teilnehmer auf einen gemeinsamen Schlüssel, anschließend sendet jeder Teilnehmer seine verschlüsselte Verfügbarkeit und im dritten Schritt wird das Resultat von allen Teilnehmern entschlüsselt.[7]

[5] Um die Effizienz hier noch etwas zu verbessern, wird üblicherweise $\phi \in \{1, -1\}$ definiert werden und die Anzahl der Ja/Nein-Stimmen mittels der Anzahl aller abgegebenen Stimmen berechnet.

[6] Auch bei der Konstruktion des geheimen Schlüssels liegt selbiger keiner einzelnen Entität vollständig vor.

[7] Setzt man die Verfahren für E-Voting ein, bestehen diese drei Schritte nicht, da dort die geheimen Schlüssel nur auf wenige Wahlobmänner verteilt sind (vgl. Teilnehmer = Wahlobmann auf Seite 30).

4.1.2. Verteilte Constraint-Optimierung

Ein verteiltes Constraint-Optimierungs-Problem[8] wird gewöhnlich durch

- eine Menge von Agenten,
- eine Menge von Variablen mit zugehöriger Domäne, welche den Agenten zugeordnet sind,
- eine Menge von Kostenfunktionen, welche den Agenten zugeordnet sind und
- eine globale Kostenfunktion

beschrieben. Ziel der Optimierung ist, die globale Kostenfunktion zu minimieren. Hat die globale Kostenfunktion einen Wertebereich von zwei Elementen ($\{\texttt{true}, \texttt{false}\}$), so spricht man von einem verteilten Constraint-Satisfaction-Problem[9].

Zur Lösung eines verteilten Constraint-Optimierungs-Problems wählt ein Agent für die ihm zugeordneten Variablen eine für ihn günstige Variablenbelegung und schickt diesen Vorschlag zusammen mit seinen lokalen Kosten an einen weiteren Agenten. Dieser wählt seine Variablen und berechnet anhand des Vorschlags sowie seiner eigenen Variablen seine lokalen Kosten, welche er wiederum zu einem weiteren Agenten schickt. Dieses Vorgehen wird mehrfach wiederholt, wobei die Agenten auch mehrfach die eigenen Variablen verändern können um die globale Kostenfunktion zu minimieren. In welcher Reihenfolge sich die Agenten festlegen ist Forschungsgegenstand vieler Arbeiten [SSHF00; YH00; LF09; Mod+04; Mah+04; ML04].

Durch die iterative Lösungsfindung der DCSP/DCOP-Protokolle werden nicht alle Informationen direkt veröffentlicht. Untersuchungen, wie viele Informationen bei verschiedenen Algorithmen veröffentlicht werden, wurden von Franzin und Greenstadt durchgeführt [Fra+02; Gre+06].

Das Problem von DCSP/DCOP-Protokollen für eine Anwendung in Web 2.0-Applikationen ist die Anzahl der blockierenden Protokollrunden. Da das Vertrauen nicht auf dritte Instanzen übertragen werden soll, muss der Web-Browser als Agent agieren und mit den anderen Benutzern das Problem lösen. Dadurch würde allerdings für jede Kommunikationsnachricht eine blockierende Protokollrunde entstehen. Um eine Lösung in einer Runde zu erzeugen müssten alle Clients genügend Informationen senden, damit von ihnen keine weitere Eingabe nötig ist. Dadurch geht allerdings der Vorteil verloren, dass weniger Informationen durch eine iterative Lösungsfindung veröffentlicht werden.

Da Web 2.0-Terminabstimmungsapplikationen das Ziel haben über *einzelne* Termine abzustimmen[10] sind die komplexen Problembeschreibungen, welche mit DCSP/DCOP möglich sind[11] hier nicht nötig.

[8] engl. *distributed constraint optimization problem, DCOP*
[9] engl. *distribtuted constraint satisfaction problem, DCSP*
[10] vgl. Definition 4
[11] Es könnten beispielsweise Abflugs- und Landezeiten von Flugzeugen verschiedener Fluggesellschaften unter der Berücksichtigung von Randbedingungen anderer Flughäfen in eine Reihenfolge gebracht werden.

4.1.3. Spezifische Terminplanungsprotokolle

Herlea et al.

Der Artikel von Herlea et al. [Her+01] beinhaltet drei unterschiedliche Herangehensweisen an das Problem. Die erste Lösung basiert auf einer vertrauenswürdigen dritten Partei, welche zwar effizient ist, allerdings einen vertrauenswürdigen Server Administrator erfordert. Die zweite Lösung benutzt eine homomorphe ElGamal-Verschlüsselung, wie sie auch bei E-Voting-Verfahren zur Verwendung kommt. Im Gegensatz zu den genannten E-Voting-Verfahren wird hier nicht die Summe der Stimmen berechnet, sondern ein logisches „∧“ aller verfügbaren Teilnehmer, wodurch nur Zeitpunkte gefunden werden, an denen alle Personen verfügbar sind. Dadurch ist die Flexibilität der Auswahlfunktion[12] nicht besonders hoch. Darüber hinaus benötigt die Berechnung des logischen „∧“ 4 blockierende Protokollrunden was in der Praxis inakzeptabel ist.

Auch das dritte Protokoll stellt nur fest, ob alle Teilnehmer verfügbar sind. Dieses Protokoll ist durch die Benutzung von einfachen XOR-Operationen sehr effizient, benötigt aber so viele blockierende Protokollrunden wie Teilnehmer vorhanden sind.

Bilogrevic et al.

Bilogrevic et al. stellten nach dem in dieser Arbeit vorgeschlagenen Protokoll drei Systeme vor, welche für Terminabstimmung auf mobilen Geräten entworfen sind [Bil+11]. Die Protokolle basieren auf homomorphen Verschlüsselungssystemen (ElGamal, Paillier und Goldwasser-Micali). Die Berechnungskomplexität der Protokolle ist linear abhängig von der Anzahl der Zeitpunkte $\mathcal{O}(|T|)$, das letzte Protokoll ist abhängig von der Anzahl der Zeitpunkte und Teilnehmer ($\mathcal{O}(|T| \cdot |U|)$).

Genau wie das Protokoll von Herlea et al. stellt auch dieses Verfahren nur fest, ob alle Teilnehmer zu einem Zeitpunkt verfügbar sind, was sich negativ auf die Flexibilität der Auswahlfunktion (Anforderung 1) auswirkt.

4.2. Einfaches Schema für protokolltreue Teilnehmer

Für das folgende Protokoll wird von den drei Phasen: Umfrageerstellung, Stimmenabgabe und Ergebnisveröffentlichung ausgegangen, wie sie schon in Abschnitt 2.2.1 auf Seite 9 vorgestellt wurden. Im Folgenden wird der Ablauf jeder Phase einzeln beschrieben.

4.2.1. Umfrageerstellung

Das Anlegen einer Abstimmung funktioniert ähnlich zur Umfragenerstellung bestehender Terminabstimmungsapplikationen. Der Initiator legt die Menge der Startzeitpunkte T und die Menge der Teilnehmer U fest. Mit Festlegung der Teilnehmermenge wird hierbei auch die Anonymitätsmenge festgelegt, in der ein Abstimmender agiert.[13] Ab dieser Stelle soll

[12] vgl. Anforderung 1

[13] Da jeder Teilnehmer die Identität der anderen Teilnehmer feststellen kann, kann ausgeschlossen werden, dass der Initiator eine falsche Anonymitätsmenge vortäuscht. Möchte man die Identität der Teilnehmer

die Annahme gelten, dass die Menge der Teilnehmer fest ist. Wie Teilnehmer dynamisch hinzugefügt und auch entfernt werden können wird in Abschnitt 4.5.1 betrachtet.

Im Folgenden werden unspezifische Teilnehmer mit u und u' angegeben. Werden die Teilnehmer Alice, Bob, Carol oder Mallory genannt, so werden sie mit u_a, u_b, u_c bzw. u_m bezeichnet.

Um die Anonymität der Teilnehmer zu gewährleisten und um nur die Summe der Eingaben zu berechnen, wird Überlagerndes Senden[14] in Gruppen $(\mathbb{Z}_n, +)$ mit $n > 2$ benutzt [Cha88b]. Um sicherzustellen, dass während des Addierens der DC-Netz-Nachrichten kein Überlauf in den Klartextnachrichten entsteht, muss der DC-Netz-Modulus n, welcher die Trägermenge $\mathbb{Z}_n$ definiert, größer als die Anzahl der Teilnehmer $|U|$ sein $(n > |U|)$.

Um in einem DC-Netz zu senden, werden symmetrische Schlüssel benötigt. Für jeden Zeitpunkt, über den abgestimmt wird, wird eine separate DC-Netz-Runde durchgeführt. Daher tauscht jeder Teilnehmer $|T|$ unabhängige, gleichverteilte Zufallszahlen aus $\mathbb{Z}_n$ mit jedem anderen Teilnehmer aus. Ein Schlüssel zwischen den Teilnehmern $u, u' \in U$ für den Zeitpunkt $t \in T$ soll mit $k^t_{u,u'}$ bezeichnet werden. Über die Teilnehmermenge sei im Folgenden eine strenge Totalordnungsrelation $(U, <)$ definiert. Da von einem vollständigen Schlüsselgraphen ausgegangen wird, tauscht jeder Teilnehmer mit jedem der $(|U| - 1)$ anderen Teilnehmer Schlüssel aus. Jeder Teilnehmer u führt folgende Schritte aus:

1. Tausche mit allen Teilnehmern $u' \in U \setminus \{u\}$, eine Zufallszahl $r^t_{u'} \in \mathbb{Z}_n$ für jeden Zeitpunkt $t \in T$ aus.

2. Speichere den DC-Netz-Schlüssel:

$$k^t_{u,u'} := \begin{cases} r^t_{u'} & \text{wenn } u < u' \\ -r^t_{u'} \bmod n & \text{sonst.} \end{cases} \tag{4.5}$$

Anschließend sollte jeder Teilnehmer u eine Anzahl von $(|U| - 1) \cdot |T|$ Schlüssel ausgetauscht haben und eine Schlüsselmatrix der Form

$$\boldsymbol{k}_u := \begin{pmatrix} k^{t_1}_{u,u_1} & \dots & k^{t_{|T|}}_{u,u_1} \\ \vdots & \ddots & \vdots \\ k^{t_1}_{u,u_{|U|-1}} & \dots & k^{t_{|T|}}_{u,u_{|U|-1}} \end{pmatrix}, \{u_1, \dots, u_{|U|-1}\} = U \setminus \{u\} \tag{4.6}$$

besitzen. Jede Zeile dieser Matrix beinhaltet die Schlüssel, die Teilnehmer u mit einem Teilnehmer u' ausgetauscht hat. Dieser Vektor wird mit $\vec{k}_{u,u'}$ bezeichnet. Wie dieser Schlüsselaustausch mittels asymmetrischer Kryptographie durchgeführt werden kann wird in Abschnitt 4.4.4 gezeigt. Bis dahin sei davon ausgegangen, jeder Teilnehmer hätte mit jedem anderen diese Schlüssel ausgetauscht.

---

innerhalb der Teilnehmergruppe geheim halten, so muss dem Initiator vertraut werden und es kann auf Schema 5′ (vgl. Tabelle 3.2 auf Seite 27) zurückgegriffen werden.

[14] auch bekannt als DC-Netz

4.2.2. Stimmenabgabe

In dieser Phase gibt jeder Teilnehmer an, ob er zu den jeweiligen Zeitpunkten, die zur Wahl stehen, verfügbar ist oder nicht. Jeder Teilnehmer u berechnet einen verschlüsselten Verfügbarkeitsvektor $\vec{d}_u$ welcher aus $|T|$ verschlüsselten Verfügbarkeiten d_u^t besteht. Eine verschlüsselte Verfügbarkeit d_u^t wird durch Addieren modulo n der unverschlüsselten Verfügbarkeit $\phi_u^t \in \{0,1\}$ (Definition 2 auf Seite 4) und aller Schlüssel, die der Teilnehmer mit den anderen Teilnehmern ausgetauscht hat, erreicht:

$$d_u^t := \phi_u^t + \sum_{u' \in U \setminus \{u\}} k_{u,u'}^t \mod n. \tag{4.7}$$

Der verschlüsselte Verfügbarkeitsvektor $\vec{d}_u := \left(d_u^{t_1}, \ldots, d_u^{t_{|T|}}\right)$ wird zum Server geschickt.

4.2.3. Ergebnisveröffentlichung

Sobald alle Teilnehmer ihre Stimme abgegeben haben, werden alle verschlüsselten Verfügbarkeitsvektoren den Teilnehmern bekannt gemacht. Jeder Teilnehmer kann die Summe dieser Vektoren berechnen. Da alle Schlüssel paarweise invers in die Berechnung eingegangen sind, ist das Ergebnis die Summe der unverschlüsselten Verfügbarkeitsvektoren. Der Ergebnisvektor ist der Vektor, der an jeder Stelle die Anzahl der verfügbaren Personen eines bestimmten Zeitpunkts enthält:

$$\vec{\sigma} := \sum_{u \in U} \vec{d}_u = \sum_{u \in U} \vec{\phi}_u = \left(\sigma^{t_1}, \ldots, \sigma^{t_{|T|}}\right), \sigma^t \in \{0, \ldots, |U|\}. \tag{4.8}$$

Anhand dieses Ergebnisses kann das Datum gewählt werden, an dem das Ereignis angesetzt wird. Die Auswahlfunktion $\mathcal{S}$ muss hierbei so definiert sein, dass als Eingabe nur die Anzahl der verfügbaren Personen zulässig ist. Dies ist die Flexibilitätseinschränkung, wie sie schon in Abschnitt 3.1.3 diskutiert wurde. Eine Erweiterung, wie eine Präferenzwahl durchgeführt werden kann, wird in Abschnitt 4.5.2 beschrieben.

Da es bei vielen Abstimmungen darauf ankommt, einen Zeitpunkt zu wählen, an dem die Mehrheit der Personen verfügbar ist, wird für den Verlauf dieser Arbeit immer davon ausgegangen, dass $\mathcal{S}_{\max}$ als Auswahlfunktion genommen wird.

Annahme 1 Die Auswahlfunktion $\mathcal{S}$ wählt immer den ersten Zeitpunkt, an dem die meisten Teilnehmer verfügbar sind $\mathcal{S} = \mathcal{S}_{\max}$. □

Der gesamte Kommunikationsablauf ist in Abbildung 4.1 auf der nächsten Seite illustriert.

4.3. Erweiterung des Schemas auf protokollverletzende Angreifer innerhalb der Teilnehmer

Ein Problem des bisher vorgestellten Schemas ist, dass es nicht sicher gegen protokollverletzende Angriffe anderer Teilnehmer ist. In Gleichung (2.1) auf Seite 4 wurde festgelegt, dass

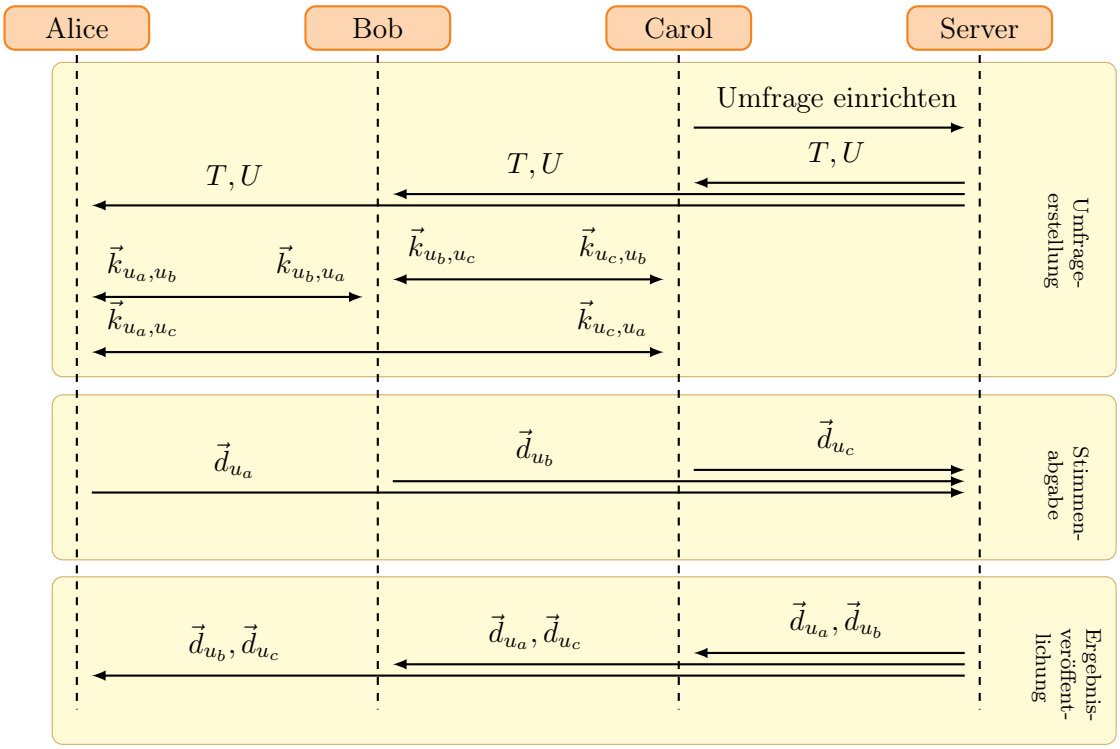


Abbildung 4.1.: Phasen des Protokolls

die unverschlüsselten Verfügbarkeiten und damit die Nachrichten jeder DC-Netz-Runde 0 oder 1 sein müssen ($\phi_u^t \in \{0, 1\}$). Geht man davon aus, dass Teilnehmer protokollverletzend angreifen, so muss gewährleistet werden, dass ein Teilnehmer keinen anderen Wert sendet. Prinzipiell ist es allerdings möglich, alle Werte aus dem Wertebereich des DC-Netzes zu senden ($\phi_u^t \in \mathbb{Z}_n$).

Im Folgenden wird angenommen:[15]

Annahme 2 Es wird davon ausgegangen, dass zu jedem Zeitpunkt höchstens *ein* Angreifer (Mallory, u_m) agiert. □

Sendet ein Angreifer für einen Zeitpunkt t eine -1 und alle anderen Teilnehmer eine 0, so erhält man in der DC-Netz-Summe den Wert $\sum_{u \in U} d_u^t \bmod n = -1 \bmod n$. Bei der üblichen Definition der Modulo-Operation ist das Ergebnis dieser Rechnung $n - 1$. Damit diese Fälle im weiteren Verlauf mit -1 benannt werden können, wird die Modulo-Operation so definiert, dass sie auf die betragskleinste Darstellung eines Elements abbildet:

[15] Eine Betrachtung von mehreren Angreifern befindet sich am Ende von Abschnitt 4.4.1.

Definition 17 (Modulo) Die Modulo-Operation $\mathrm{mod} : \mathbb{Z} \times \mathbb{N} \setminus \{0\} \to \mathbb{Z}$, ist eine Abbildung, die auf die betragskleinste Darstellung eines Elements abbildet:

$$\mathrm{mod}(a, n) = a \bmod n := a - \left\lfloor \frac{a}{n} + 0{,}5 \right\rfloor \cdot n. \tag{4.9}$$

□

Mit dieser Definition ergibt sich die übliche Gleichung $8 \bmod 6 = 2$, sowie die etwas unüblichere $9 \bmod 6 = -3$.

Bisher wurde davon ausgegangen, dass der DC-Netz-Modulus n größer ist, als die Anzahl der Teilnehmer, um einen Überlauf zu vermeiden. Mit dieser Modulo-Definition tritt der Überlauf allerdings schon auf, wenn $\left\lceil \frac{|U|}{2} \right\rceil$ Teilnehmer für einen Zeitpunkt stimmen. Um diesen Überlauf zu vermeiden und auch für nachfolgend erklärte Angriffe mit Werten $\phi_u^t > 1$ keinen Überlauf zu erhalten, wird folgende Annahme getroffen:

Annahme 3 Der DC-Netz-Modulus ist deutlich größer als die doppelte Anzahl der Teilnehmer ($n \gg 2 \cdot |U|$). □

Des Weiteren wird für die nachstehende Beschreibung die Modulo-Operation bei Berechnungen im DC-Netz ($\bmod n$) weggelassen, um die Übersichtlichkeit der Gleichungen zu erhöhen.

Davon ausgehend, dass es prinzipiell möglich ist, jeden Wert aus dem Wertebereich des DC-Netzes zu senden ($\phi_u^t \in \mathbb{Z}_n$), werden nun zwei Angriffe definiert, welche darauf abzielen, die Erfolgswahrscheinlichkeit der Wahl bestimmter Zeitpunkte zu beeinflussen.

Wenn als Terminauswahlfunktion $\mathcal{S}_{\max}$ benutzt wird und alle möglichen Ergebnisvektoren $\vec{\sigma} \in \{0, \ldots, |U|\}^{|T|}$ mit der gleichen Wahrscheinlichkeit auftreten können, so tritt jeder Zeitpunkt $t \in T$ mit der gleichen Wahrscheinlichkeit auf.

Satz 1 *Sendet ein Angreifer zu einem Zeitpunkt Werte kleiner* 0*, so kann er damit die Wahrscheinlichkeit verringern, dass dieser Zeitpunkt von der Auswahlfunktion* $\mathcal{S}_{\max}$ *gewählt wird.*

$$\forall t \in T, u \in U : P\left(\mathcal{S}_{\max}(\vec{\sigma}) = t \,|\, \phi_u^t < 0\right) < P\left(\mathcal{S}_{\max}(\vec{\sigma}) = t\right). \tag{4.10}$$

□

Beispielhaft wird ein Angriff, in dem versucht wird, die Erfolgswahrscheinlichkeit für einen Zeitpunkt zu verringern, wie folgt definiert:

Definition 18 ((−1)-Angriff) Ein (-1)-Angriff bezeichnet den Angriff auf eine Abstimmung, bei der ein Angreifer versucht, eine -1 zu einem Zeitpunkt zu senden:

$$\exists u \in U, t \in T : \phi_u^t = -1. \tag{4.11}$$

□

Im weiteren Verlauf dieser Arbeit wird noch darauf eingegangen, dass (-1)-Angriffe mindestens so schwer zu erkennen sind wie (-2)-Angriffe bzw. $(-\nu)$-Angriffe, weshalb zuerst nur (-1)-Angriffe diskutiert werden.

	t_1	t_2	t_3	t_4
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	−1	0	1
$\sum$	1	1	0	2

(a) (−1)-Angriff auf einen Zeitpunkt

	t_1	t_2	t_3	t_4
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	−1	−1	−1	1
$\sum$	0	1	−1	2

(b) (−1)-Angriff auf mehrere Zeitpunkte

Abbildung 4.2.: Zwei Arten eines (−1)-Angriffs. In beiden Fällen möchte Mallory, dass t_4 gewählt wird. Dargestellt sind die unverschlüsselten Verfügbarkeiten ϕ_u^t.

Beispiele für diesen Angriff sind in Abbildung 4.2 dargestellt. Die Tabelle zeigt zwei Abstimmungen mit drei Teilnehmern und vier Zeitpunkten. Die Werte in den Tabellen zeigen die unverschlüsselten Verfügbarkeiten ϕ_u^t. Mallory versucht die Abstimmung so zu manipulieren, dass t_4 gewinnt. Durch die Anonymisierung der Nachrichten im DC-Netz bleibt Mallorys Angriff vor Alice und Bob verborgen.

Analog zu Satz 1, in der ein Angreifer die Erfolgswahrscheinlichkeit für einen Zeitpunkt verringern wollte, gilt für den umgekehrten Fall:

Satz 2 *Sendet ein Angreifer zu einem Zeitpunkt Werte größer* 1*, so kann er damit die Wahrscheinlichkeit erhöhen, dass dieser Zeitpunkt von der Auswahlfunktion* $\mathcal{S}_{\max}$ *gewählt wird.*

$$\forall t \in T, u \in U : P\left(\mathcal{S}_{\max}\left(\vec{\sigma}\right) = t \,|\, \phi_u^t > 1\right) > P\left(\mathcal{S}_{\max}\left(\vec{\sigma}\right) = t\right). \tag{4.12}$$

□

Es wird analog zu Definition 18 definiert:[16]

Definition 19 ((+2)-Angriff) Ein (+2)-Angriff bezeichnet den Angriff eines Angreifers auf eine Abstimmung, bei der versucht wird, eine +2 zu einem Zeitpunkt zu senden:

$$\exists u \in U, t \in T : \phi_u^t = +2. \tag{4.13}$$

□

Abbildung 4.3 auf der nächsten Seite zeigt eine Beispielabstimmung für diesen Angriff, bei dem Mallory abermals versucht, die Abstimmung so zu manipulieren, dass t_4 gewinnt.

Die vorgestellten Definitionen 18 und 19 stehen hier beispielhaft für ($\pm\nu$)-Angriffe. In Abschnitt 4.4.1 wird kurz gezeigt, dass es mit größeren ν einfacher wird einen Angriff zu erkennen.

Im nächsten Abschnitt wird gezeigt, wie man erkennen kann, dass eine Abstimmung angegriffen wurde (Abschnitt 4.3.1). Abschnitt 4.3.2 zeigt darüber hinaus, wie die Identität des Angreifers herausgefunden werden kann.

[16] Auch hier gilt, dass (+2)-Angriffe mindestens genauso schwer zu erkennen sind wie (+3)-Angriffe bzw. ($+\nu$)-Angriffe.

	t_1	t_2	t_3	t_4
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	0	0	2
$\sum$	1	2	0	3

Abbildung 4.3.: (+2)-Angriff

4.3.1. Angriffe erkennen

Dieser Abschnitt stellt zwei Techniken vor, welche zur Erkennung von (−1)- und (+2)-Angriffen benutzt werden können.

(−1)-Angriff

In Abbildung 4.2 auf der vorherigen Seite wurden bereits Beispiele für (−1)-Angriffe gezeigt. Während der Angriff in Abbildung 4.2(a) unerkannt bleibt, kann der Angriff in Abbildung 4.2(b) erkannt werden. Dort sendet Mallory zu *jedem* von ihr nicht präferierten Zeitpunkt eine −1 ($\phi_{u_m}^{t_1} = \phi_{u_m}^{t_2} = \phi_{u_m}^{t_3} = -1$). Das Ergebnis σ^t sollte für jeden Zeitpunkt zwischen 0 und der Anzahl der Teilnehmer liegen ($\sigma^t \in \{0, \ldots, |U|\}$). Bei t_3 ist das Ergebnis allerdings −1. Um (−1)-Angriffe zu erkennen, wird der Umstand genutzt, dass das Ergebnis für jede Spalte nicht kleiner als 0 sein darf.

Anstatt nur *eine* DC-Netz-Runde für jeden Zeitpunkt zu benutzen, werden *mehrere* simultane Runden durchgeführt. Die eigentliche Verfügbarkeit wird in einer zufällig gewählten Runde gesendet. Alle anderen Runden enthalten eine 0.

Sei ℓ die Anzahl der gleichzeitig stattfindenden DC-Netz-Runden. Jeder Teilnehmer u verteilt für jeden Zeitpunkt t seine Verfügbarkeit ϕ_u^t auf viele Teilstimmen $\varphi_u^{(t,0)}$, …, $\varphi_u^{(t,\ell-1)}$, so dass

1. eine Zufallszahl r gleichverteilt über die Indices der gleichzeitig stattfindenden DC-Netz-Runden gezogen wird ($r \in \mathbb{Z}_\ell$),
2. die Teilstimme mit dem Index r gleich der Verfügbarkeit des Teilnehmers zu diesem Zeitpunkt ist ($\varphi_u^{(t,r)} = \phi_u^t$) und
3. die verbleibenden $\ell - 1$ Teilstimmen 0 sind.

Jede Teilstimme wird in einem separaten DC-Netz verschickt. Statt nur *einen* Schlüssel $k_{u,u'}^t$ für jeden Zeitpunkt $t \in T$ auszutauschen,[17] tauscht jeder Teilnehmer einen Schlüssel $k_{u_a,u_b}^{(t,i)}$ für jeden Zeitpunkt und DC-Netz-Rundenindex $i \in \mathbb{Z}_\ell$ aus. Mit diesen Schlüsseln werden die Teilstimmen für jedes DC-Netz verschlüsselt.[18] Die verschlüsselten Nachrichten

[17] vgl. Schlüsselaustausch Abschnitt 4.2.1
[18] analog zu Gleichung (4.7) auf Seite 36

der Teilnehmer in den einzelnen DC-Netzen werden mit $d_u^{(t,i)}$ bezeichnet. Wenn alle Teilnehmer ehrlich sind, folgen diese Eigenschaften aus der Konstruktion:

Satz 3 *Für jeden Zeitpunkt $t \in T$ und jeden DC-Netz-Rundenindex $i \in \mathbb{Z}_\ell$ gilt: Die Summe aller Teilstimmen sowie die Summe aller DC-Netz-Nachrichten aller Teilnehmer liegt zwischen 0 und der Anzahl der Teilnehmer.*

$$\forall (t,i) \in T \times \mathbb{Z}_\ell : \sum_{u \in U} \varphi_u^{(t,i)} = \sum_{u \in U} d_u^{(t,i)} \in \{0, \ldots, |U|\} \tag{4.14}$$

□

Satz 4 *Für jeden Zeitpunkt $t \in T$ ist die Summe aller Teilstimmen aller Teilnehmer gleich der Summe aller verfügbaren Teilnehmer zu diesem Zeitpunkt.*

$$\forall t \in T : \sigma^t = \sum_{u \in U, i \in \mathbb{Z}_\ell} \varphi_u^{(t,i)} \tag{4.15}$$

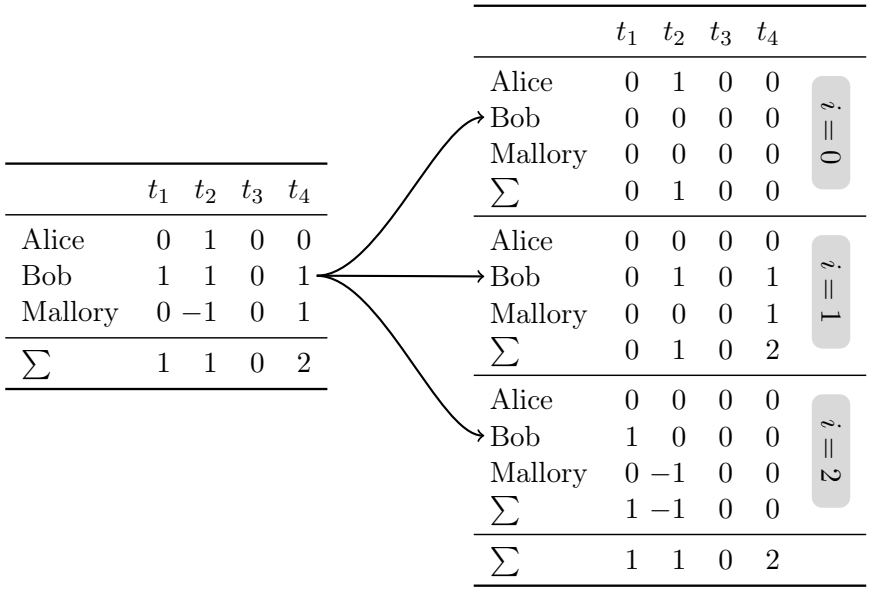
□

In Abbildung 4.2 auf Seite 39 wurde bereits dargestellt, dass ein Angreifer raten muss, für welche Zeitpunkte ein ehrlicher Teilnehmer eine 1 senden wird. Mit der gerade vorgestellten Erweiterung reicht es für Mallory nicht aus, nur die Verfügbarkeit der anderen Teilnehmer zu raten. Darüber hinaus muss sie raten, in welcher DC-Netz-Runde die Verfügbarkeit gesendet wurde. Solange der Index für die DC-Netz-Runde zufällig gewählt und geheim gehalten wurde, ist das allerdings schwierig für eine genügend große Anzahl simultaner DC-Netz-Runden. Eine genauere Betrachtung über die Anzahl der simultanen DC-Netz-Runden, sowie die daraus resultierende Angriffschance ist in Abschnitt 4.4 beschrieben.

Abbildung 4.4 auf der nächsten Seite zeigt eine Beispielabstimmung, in der 3 DC-Netze zum Aufteilen der Stimme benutzt werden ($\ell = 3$). Mallory versucht den Zeitpunkt t_2 mit einer -1 anzugreifen ($\phi_{u_m}^{t_2} = -1$). In der linken Tabelle ist das Schema ohne Stimmenaufteilung wie in Abbildung 4.2(a) zu sehen. Dort würde der Angriff unerkannt bleiben, da alle Elemente des Ergebnisvektors im erlaubten Bereich liegen. Die rechte Tabelle zeigt die gleichen Verfügbarkeitsvektoren aufgeteilt in mehrere Teilstimmen. Für Zeitpunkt t_2 hat Alice hier das erste DC-Netz ($i = 0$) für ihre Verfügbarkeit gewählt ($\varphi_{u_a}^{(t_2,0)} = \phi_{u_a}^{t_2} = 1$), Bob schickt seine Verfügbarkeit im zweiten. Mallorys Angriff wird erkannt, da sie ihn im dritten DC-Netz versucht.

Dieses Verfahren funktioniert nur, wenn davon ausgegangen wird, dass Mallory nicht mit dem Serveradministrator kooperiert. Das Problem ist hierbei, dass Mallory die abgegebenen Stimmen in Erfahrung bringen kann, bevor sie ihre Stimme abgibt. Kooperiert sie mit dem Administrator, so kann sie warten, bis alle anderen Teilnehmer ihre Stimme abgegeben haben, anschließend die einzelnen DC-Netz-Ergebnisse berechnen und ihre Stimme auf diese Ergebnisse anpassen.

Dieser Angriff kann auf mindestens zwei Arten vermieden werden. Zum einen kann dem Server vertraut werden, dass er die Einzelstimmen zurückhält, bis alle Stimmen abgegeben wurden. Die zweite Möglichkeit ist, dass sich alle Teilnehmer mittels eines Commitment-Schemas erst auf eine Stimmabgabe festlegen und anschließend in einer zweiten Runde ihre Stimme schicken. Diese Variante hat den Vorteil, dass keinerlei Vertrauen mehr an den Serveradministrator gestellt werden muss, benötigt allerdings eine zusätzliche blockierende Protokollrunde.



	t_1	t_2	t_3	t_4
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	−1	0	1
$\sum$	1	1	0	2

	t_1	t_2	t_3	t_4	
Alice	0	1	0	0	$i = 0$
Bob	0	0	0	0	
Mallory	0	0	0	0	
$\sum$	0	1	0	0	
Alice	0	0	0	0	$i = 1$
Bob	0	1	0	1	
Mallory	0	0	0	1	
$\sum$	0	1	0	2	
Alice	0	0	0	0	$i = 2$
Bob	1	0	0	0	
Mallory	0	−1	0	0	
$\sum$	1	−1	0	0	
$\sum$	1	1	0	2	

Abbildung 4.4.: Durch Aufteilen des Verfügbarkeitsvektors auf mehrere DC-Netze kann Mallorys Angriff detektiert werden.

(+2)-Angriff

In diesem Abschnitt soll erklärt werden, wie (+2)-Angriffe erkannt werden können. Ein (+2)-Angriff wurde schon in Abbildung 4.3 auf Seite 40 gezeigt. Speziell diesen Angriff hätte Alice erkennen können. Da sie weiß, dass sie selbst 0 gesendet hat, kann das Ergebnis für diese Spalte nicht größer als $|U| - 1$ sein ($\sigma^{t_4} \in \{0, \ldots, |U| - 1\}$). In einer anderen Abstimmung könnte es allerdings sein, dass weder Alice noch Bob für einen Zeitpunkt stimmen, an dem Mallory eine 2 sendet (z. B. t_3 in Abbildung 4.3). In so einem Fall könnten weder Alice noch Bob den Angriff allein entdecken.

Um (+2)-Angriffe zu erkennen, sendet jeder Teilnehmer seine Verfügbarkeit zusätzlich zur normalen Abstimmung in einer Prüfabstimmung. Jeder Teilnehmer u berechnet für jeden Zeitpunkt t eine Prüfverfügbarkeit $\phi_u^{(t,1)}$, zu seiner Verfügbarkeit ϕ_u^t (im Folgenden sei $\phi_u^{(t,0)} := \phi_u^t$), für die gilt:

$$\phi_u^{(t,0)} + \phi_u^{(t,1)} = 1. \tag{4.16}$$

Mit allen Prüfverfügbarkeiten wird eine Abstimmung auf gleiche Weise durchgeführt, wie sie in der normalen Abstimmung durchgeführt wurde. Jede DC-Netz-Runde kann über das Tripel $(t, i, \lambda) \in T \times \mathbb{Z}_\ell \times \{0, 1\}$ eindeutig indiziert werden. Hierbei soll $\lambda = 0$ für DC-Netz-Runden in der normalen Abstimmung stehen ($k_{u,u'}^{(t,i,0)} := k_{u,u'}^{(t,i)}$, $d_u^{(t,i,0)} := d_u^{(t,i)}$, …) und $\lambda = 1$ für Runden in der Prüfabstimmung ($k_{u,u'}^{(t,i,1)}$, $d_u^{(t,i,1)}$, …). Die Menge aller DC-Netz-Runden wird mit Δ bezeichnet ($\Delta := T \times \mathbb{Z}_\ell \times \{0, 1\}$). Jede DC-Netz-Runde kann durch das Tripel $(t, i, \lambda) \in \Delta$ eindeutig beschrieben werden.

Da jeder Teilnehmer in einer der beiden Abstimmungen einmal eine 1 senden muss, gilt:

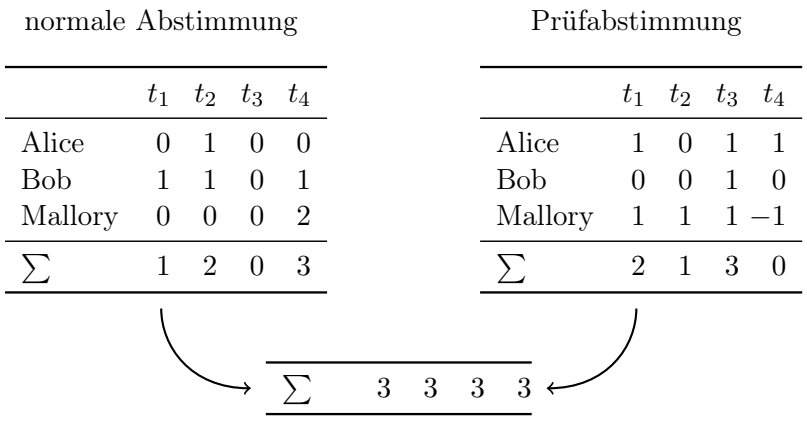


Abbildung 4.5.: Durch die Benutzung einer Prüfabstimmung kann der (+2)-Angriff auf den (−1)-Angriff zurückgeführt werden. Mallory muss eine −1 zum Zeitpunkt t_4 in der rechten Tabelle senden, weil die Summe beider Tabellen sonst nicht gleich der Anzahl der Teilnehmer wäre.

Satz 5 *Die Summe der Elemente der beiden Ergebnisvektoren (von der normalen und von der Prüfabstimmung) ist gleich der Anzahl der Teilnehmer $|U|$.*

$$\forall t \in T : \sum_{u \in U, i \in \mathbb{Z}_\ell} d_u^{(t,i,0)} + \sum_{u \in U, i \in \mathbb{Z}_\ell} d_u^{(t,i,1)} = |U| \tag{4.17}$$

□

Durch das Überprüfen dieser Eigenschaft kann sichergestellt werden, dass jeder seinen Prüfverfügbarkeitsvektor nach Gleichung (4.16) berechnet hat. Will Mallory eine 2 für irgendeinen Zeitpunkt senden, so muss sie eine −1 in der Prüfabstimmung senden. Dieser Angriff kann auf gleiche Weise erkannt werden, wie (−1)-Angriffe auf die normale Abstimmung erkannt werden. Abbildung 4.5 illustriert dieses Verfahren.

4.3.2. Angreifer identifizieren

Bisher wurde gezeigt, wie protokollverletzende Angriffe auf die Integrität einer Abstimmung erkannt werden können. Wird darüber hinaus nichts unternommen, kann ein Angreifer mit diesen Angriffen noch die Verfügbarkeit einer Abstimmung angreifen. Um das zu verhindern, wird eine optionale Identifikationsphase eingeführt, in der ein Angreifer identifiziert werden kann. Da von Abstimmungen in Gruppen ausgegangen wird, in der sich die Teilnehmer kennen, führt die Identifikation eines Angreifers sehr wahrscheinlich zu einem Verlust an Reputation, was die Motivation senkt, einen Angriff durchzuführen.

In dieser Phase wird zuerst überprüft, ob es während der Abstimmung fehlerhafte[19] DC-Netz-Runden gab. Liegen Fehler vor, so werden diese DC-Netz-Runden aufgedeckt. Ist beispielsweise die DC-Netz-Runde $\delta = (t, i, \lambda)$ fehlerhaft, so veröffentlicht jeder Teilnehmer

[19] Fehlerhaft bedeutet hier, dass die jeweilige Runde nicht den Eigenschaften von Satz 3 auf Seite 41 und Satz 5 entspricht. Eine genauere Betrachtung befindet sich in Abschnitt 4.4.1.

u seine $(|U|-1)$ Schlüssel $k^{\delta}_{u,u'}, \forall u' \in U \setminus \{u'\}$. Mit Hilfe dieser Schlüssel können die Einzelstimmen berechnet und überprüft werden, ob diese im zulässigen Wertebereich liegen ($\forall u \in U : \varphi^{\delta}_{u} \in \{0,1\}$).

An dieser Stelle ergibt sich erneut eine Angriffsmöglichkeit, da Mallory (u_m) andere Schlüssel schicken kann, als sie für die Berechnung der DC-Netz-Nachrichten benutzt hat. Hätte sie beispielsweise in der fehlerhaften DC-Netz-Runde $\varphi^{\delta}_{u_m} = -1$ gesendet, so könnte sie diesen Wert verstecken, indem sie einen beliebigen für diese Runde benutzten Schlüssel inkrementiert veröffentlicht. Wenn ihre Stimme mit diesem Schlüssel entschlüsselt wird, sieht es nun so aus, als hätte sie eine 0 und ein anderer Teilnehmer (z. B. Bob, u_b) die Stimme $\varphi^{\delta}_{u_b} = -1$ geschickt. Da Bob aber nicht mit Mallory kooperiert, würde er den richtigen Schlüssel veröffentlichen, mit welchem die richtigen Stimmen ausgerechnet werden können. Dadurch ergibt sich die Situation, dass ein Außenstehender nicht entscheiden kann, wer von den beiden lügt.

Um eine nachträgliche Veränderung von Schlüsseln zu verhindern, legt sich jeder Teilnehmer beim Abstimmen mittels eines Commitment-Schemas auf seinen Schlüssel fest. Zusätzlich zu den Schritten 1 und 2 in Abschnitt 4.2.1 berechnet jeder Teilnehmer u für jede DC-Netz-Runde $\delta \in \Delta$ ein Commitment:

$$\psi^{\delta}_{u,u'} := \text{commit}\left(k^{\delta}_{u,u'}\right), \tag{4.18}$$

welches zusammen mit dem verschlüsselten Verfügbarkeitsvektor zum Server übertragen wird. Die Funktion commit $(\cdot)$ erfüllt hierbei die folgenden zwei Eigenschaften *geheimhaltend* und *bindend*:

Definition 20 (geheimhaltend) Eine Funktion ist geheimhaltend, wenn man unter Kenntnis des Ausgabewertes und ohne Kenntnis des Eingabewertes jedes einzelne Bit des Eingabewertes nur mit einer Wahrscheinlichkeit von $\frac{1}{2}$ raten kann. □

Definition 21 (bindend) Eine Funktion ist bindend, wenn es schwer ist zwei Werte $x \neq x'$ zu finden, für die commit (x) = commit (x') gilt. □

Seien $\mathrm{h}(\cdot)$ eine geheimhaltende kollisionsresistente Hashfunktion und r_1, r_2 zwei Zufallszahlen. Eine weit verbreitete Konstruktion einer Funktion commit $(\cdot)$ auf Basis einer Hashfunktion ist commit $(x) := (\mathrm{h}(x|r_1|r_2), r_1)$ [Sch96; Bis09]. Um den Wert x aufzudecken wird x, r_1, r_2 veröffentlicht, wodurch die korrekte Berechnung des Commitmentwertes überprüft werden kann. Die Zufallszahl r_1 dient hierbei dazu, den Commitmentwert einem bestimmten Protokoll zuzuordnen. r_2 ist für die Geheimhaltung von x notwendig, da die Entropie von x möglicherweise einem Brute-Force-Angriff nicht stand hält.

Im speziellen Szenario von Gleichung (4.18) muss der Schlüssel jedoch ohnehin einem Brute-Force-Angriff standhalten. Deshalb kann auf r_2 in diesem Kontext verzichtet werden. Zusätzlich wird r_1 nicht benötigt, da bei einem geeigneten Datenformat sowie einer Integritätssicherung der Commitments die richtigen Commitmentwerte den Schlüsseln zugeordnet werden können. Das für diesen Anwendungsfall resultierende Schema ist daher:

$$\text{commit}\left(k^{\delta}_{u,u'}\right) := \mathrm{h}(k^{\delta}_{u,u'}) \tag{4.19}$$

Sollte es nötig werden, eine Runde aufzudecken, so kann mit Hilfe des Commitments überprüft werden, ob die richtigen Schlüssel veröffentlicht wurden. Durch die geheimhaltende Eigenschaft wird gewährleistet, dass keine Informationen über die Schlüssel und damit über die Stimme veröffentlicht werden, so lange keine Runden aufgedeckt werden. Möchte Mallory den oben beschriebenen Angriff durchführen, so müsste sie (unter der Annahme $u_m < u_b$) einen Schlüssel finden für den

$$\text{commit}\left(k^{\delta}_{u_m,u_b}\right) = \text{commit}\left(k^{\delta}_{u_m,u_b} + 1\right) \tag{4.20}$$

gilt. Die Lösung für dieses Problem ist identisch zum Brechen der bindenden Eigenschaft.

4.4. Evaluation

In diesem Kapitel wird diskutiert, inwieweit das vorgestellte Schema die in Abschnitt 2.2 aufgestellten Anforderungen erfüllt. Dazu wird jede Anforderung einzeln diskutiert. Am Ende des Kapitels wird die Anforderungsdiskussion nochmal zusammengefasst.

4.4.1. Integrität

Sei $\boldsymbol{d} \in \mathbb{Z}_n^{|U|\times|\Delta|}$ die Matrix aller DC-Netz-Nachrichten mit den Elementen d^{δ}_u für eine DC-Netz-Nachricht ($u \in U, \delta \in \Delta$). Im Folgenden wird wieder getrennt diskutiert, inwieweit Integrität bei (-1)- bzw. $(+2)$-Angriffen gegeben ist.

(-1)-Angriff

Um zu überprüfen, ob ein (-1)-Angriff stattgefunden hat, kann man zwei Fälle unterscheiden:

- Im einfacheren Fall ist das Ergebnis einer DC-Netz-Runde negativ ($\sum_{u\in U} d^{\delta}_u < 0$). Dieser Angriff kann von jedem (auch Außenstehenden) erkannt werden. Dieser Fall soll *„Einfache Erkennung“* genannt werden und trat beispielsweise in Abbildung 4.4 auf Seite 42 auf.

- In einem komplizierteren Szenario haben ein oder mehrere Teilnehmer eine 1 in einer DC-Netz-Runde gesendet, deren Ergebnis 0 ist. Dieser Fall kann auch im einfachen Schema ohne Stimmenaufteilung vorkommen. Bob kann beispielsweise feststellen, dass Mallory bei Zeitpunkt t_1 in Abbildung 4.2(b) angegriffen hat. Um diesen Angriff nachzuweisen muss Bob allerdings beweisen, dass er selbst eine 1 geschickt hat und damit seine Verfügbarkeit für diesen einen Zeitpunkt offenlegen. Bob kann in diesem Fall für sich selbst entscheiden, ob ihm die Geheimhaltung seiner Stimme oder das Entlarven des Angreifers wichtiger ist. Dieser Fall soll mit *„Erkennung mit Anonymitätsverlust“* bezeichnet werden.

Beide vorgestellten Fälle werden nachfolgend einzeln betrachtet.

Einfache Erkennung Dieser Fall war gegeben, wenn das Ergebnis einer DC-Netz-Runde negativ ist ($\sum_{u \in U} d_u^\delta < 0$).

Definition 22 (Globale Abstimmungsverifikation) Die Funktion $\mathcal{V} : \mathbb{Z}_n^{|U| \times |\Delta|} \to \{\texttt{true}, \texttt{false}\}$ zur Verifikation der Korrektheit der Abstimmung hat alle Nachrichten der Abstimmung als Eingabe und gibt `true` zurück wenn die Abstimmung richtig ablief, sonst `false`:

$$\mathcal{V}(\boldsymbol{d}) := \begin{cases} \texttt{true} & \text{wenn } \forall \delta \in \Delta : \sum_{u \in U} d_u^\delta \in \{0, \ldots, |U|\} \\ \texttt{false} & \text{sonst.} \end{cases} \tag{4.21}$$

□

Mallory (u_m) versucht eine -1 am Zeitpunkt t zu senden. Wenn man davon ausgeht, dass μ Teilnehmer für t stimmen, so kann die Wahrscheinlichkeit, dass Mallorys Angriff von $\mathcal{V}(\boldsymbol{d})$ erkannt wird, wie folgt berechnet werden:

$$P\left(\mathcal{V}(\boldsymbol{d}) = \texttt{false} \,|\, \phi_{u_m}^t = -1\right) = \left(\frac{\ell - 1}{\ell}\right)^\mu \tag{4.22}$$

Mit steigendem μ, sinkt die Wahrscheinlichkeit, dass der Angriff erkannt wird. Im für das Erkennen eines Angriffs ungünstigsten Fall senden alle ehrlichen Teilnehmer an allen Zeitpunkten eine 1 ($\mu = |U| - 1$).

Satz 6 *Die untere Grenze für die Wahrscheinlichkeit, einen (−1)-Angriff zu erkennen, ist*

$$P\left(\mathcal{V}(\boldsymbol{d}) = \texttt{false} \,|\, \phi_{u_m}^t = -1\right) \geq \left(\frac{\ell - 1}{\ell}\right)^{|U|-1}. \tag{4.23}$$

□

Geht man davon aus, dass jeder Teilnehmer mit einer Wahrscheinlichkeit größer 0 für einen Zeitpunkt stimmt,[20] so steigt die Wahrscheinlichkeit, dass ein Angriff unerkannt bleibt, mit zunehmenden Teilnehmern. Deshalb sollte die Anzahl der DC-Netz-Runden ℓ in Abhängigkeit von der Anzahl der Teilnehmer $|U|$ gewählt werden.

Sollte Mallory versuchen eine -2 zu senden, steigt die Wahrscheinlichkeit, dass dieser Angriff erkannt wird. In diesem Fall muss sie *zwei* DC-Netz-Runden finden, um ihren Angriff zu verstecken.[21] Um diese Wahrscheinlichkeit zu berechnen, kann das Urnenmodell mit ℓ Kugeln[22] herangezogen werden. Mallory markiert zwei Kugeln mit unterschiedlichen Farben (die zwei DC-Netz-Runden, in denen sie ihre -1 sendet). Für jeden ehrlichen Teilnehmer, der für den angegriffenen Zeitpunkt stimmt[23], wird eine Kugel mit Zurücklegen gezogen. Wenn beide farbige Kugeln mindestens einmal in der Ziehung auftraten, bleibt Mallorys Angriff unerkannt. Der Angriff wird in drei Fällen erkannt:

[20] $\forall u \in U, t \in T : P\left(\phi_u^t = 1\right) > 0$

[21] Mallory kann natürlich auch nur *eine* DC-Netz-Runde wählen, in der sie eine -2 sendet. Wie später ersichtlich sein wird (vgl. Fußnote 25 auf Seite 48), ist die Wahrscheinlichkeit unerkannt zu bleiben allerdings höher, wenn sie ihren Angriff auf zwei unterschiedliche Runden verteilt. Deshalb wird nur dieser Fall betrachtet.

[22] eine Kugel pro DC-Netz-Runde

[23] μ Teilnehmer

c_1: Keine farbige Kugel wurde gezogen.

c_2: Die erste Kugel wurde mindestens einmal gezogen aber nicht die zweite.

c_3: Die zweite, aber nicht die erste, wurde gezogen.

Wie schon in der vorangegangenen Berechnung (vgl. Satz 6 auf der vorherigen Seite) ist es auch hier für den Angreifer am günstigsten, wenn alle ehrlichen Teilnehmer für den angegriffenen Zeitpunkt stimmen ($\mu = |U| - 1$). In diesem Fall erreicht die Erkennungswahrscheinlichkeit ein Minimum. Die Wahrscheinlichkeit für den ersten Fall (c_1) kann wie folgt berechnet werden:

$$P(c_1) = \left(\frac{\ell - 2}{\ell}\right)^{\mu} \geq \left(\frac{\ell - 2}{\ell}\right)^{|U|-1}. \tag{4.24}$$

Die Wahrscheinlichkeit, dass eine Kugel nicht gezogen wird, ist $\left(\frac{\ell-1}{\ell}\right)^{\mu}$. Um die Wahrscheinlichkeit für den zweiten und dritten Fall zu berechnen muss $P(c_1)$ davon abgezogen werden. Damit wird sichergestellt, dass die andere Kugel gezogen wurde:

$$P(c_2) = P(c_3) = \left(\frac{\ell - 1}{\ell}\right)^{\mu} - P(c_1) \geq \left(\frac{\ell - 1}{\ell}\right)^{|U|-1} - P(c_1). \tag{4.25}$$

Zusammengefasst, lässt sich die Wahrscheinlichkeit, dass ein (-2)-Angriff erkannt wird berechnen, indem die Wahrscheinlichkeiten aller drei Fälle addiert werden.

Satz 7 *Die untere Grenze, für die Wahrscheinlichkeit einen (-2)-Angriff zu erkennen, ist*

$$P\left(\mathcal{V}(\boldsymbol{d}) = \texttt{false} \,|\, \phi_{u_m}^t = -2\right) \geq 2 \cdot \left(\frac{\ell - 1}{\ell}\right)^{|U|-1} - \left(\frac{\ell - 2}{\ell}\right)^{|U|-1}. \tag{4.26}$$

□

Verallgemeinert man dieses Problem auf einen $(-\nu)$-Angriff, so erhält man eine Multinomialverteilung.

Erkennung mit Anonymitätsverlust

Definition 23 (Lokale Abstimmungsverifikation mit Beschuldigung) Die Funktion $\mathcal{B} : \mathbb{Z}_n^{|U|\times|\Delta|} \times \mathbb{Z}_n^{(|U|-1)\times|\Delta|} \to \{\texttt{true}, \texttt{false}\}$, die die Korrektheit einer Abstimmung aus Sicht eines Teilnehmers u unter der Bedingung $\mathcal{V}(\boldsymbol{d}) = \texttt{true}$ überprüft, nimmt alle Nachrichten der Abstimmung, sowie die Schlüsselmatrix des Teilnehmers als Eingabe. Sie liefert für einen Teilnehmer u den Wert `true`, wenn die Umfrage aus seiner Sicht korrekt

verlief, d. h. er kann *nicht* beweisen, dass er in einem DC-Netz eine 1 gesendet hat, bei der eine 0 herauskam ($\sum_{u\in U} d_u^\delta = 0$).[24]

$$\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u) := \begin{cases} \texttt{true} & \text{wenn } \neg\exists \delta \in \Delta : \\ & \quad \left(\sum_{u'\in U} d_{u'}^\delta = 0\right) \wedge \\ & \quad \left(d_u^\delta + \sum_{u'\in U\setminus\{u\}} k_{u,u'}^\delta = 1\right) \\ \texttt{false} & \text{sonst.} \end{cases} \tag{4.27}$$

□

Davon ausgehend kann die Funktion, die die Korrektheit einer Abstimmung aus Sicht aller ehrlichen Teilnehmer bei einem Angreifer u_m überprüft, definiert werden.

Definition 24 (Globale Abstimmungsverifikation mit Beschuldigung) Die Funktion $\mathcal{B}_{\text{all}} : \mathbb{Z}_n^{|U|\times|\Delta|} \times \mathbb{Z}_n^{(|U|-1)\times(|U|-1)\times|\Delta|} \to \{\texttt{true}, \texttt{false}\}$, die die Korrektheit einer Abstimmung aus Sicht aller ehrlichen Teilnehmer unter der Bedingung $\mathcal{V}(\boldsymbol{d}) = \texttt{true}$ bestimmt, ist definiert mit

$$\mathcal{B}_{\text{all}}^{u_m}(\boldsymbol{d}, \{\boldsymbol{k}_u : u \in U \setminus \{u_m\}\}) := \begin{cases} \texttt{true} & \text{wenn } \forall u \in U \setminus \{u_m\} : \mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u) = \texttt{true} \\ \texttt{false} & \text{sonst.} \end{cases} \tag{4.28}$$

□

Um einen unerkannten (-1)-Angriff unter diesen Umständen durchzuführen, braucht Mallory mindestens zwei ehrliche Teilnehmer, die ihren Angriff unwissentlich verstecken.[25] Die Wahrscheinlichkeit, dass dieser Angriff erkannt wird, kann durch Addition der Wahrscheinlichkeiten der Fälle, dass weniger als zwei Teilnehmer in der angegriffenen DC-Netz-Runde senden, berechnet werden:

c_1: Keiner sendet eine 1 in der von Mallory angegriffenen DC-Netz-Runde.

c_2: Genau ein Teilnehmer sendet eine 1 in Mallorys Runde.

Für den ersten Fall wäre die Summe der angegriffenen DC-Netz-Runde -1 und daher wäre das Ergebnis von $\mathcal{V}(\boldsymbol{d}) = \texttt{false}$ (vgl. Gleichung (4.23) auf Seite 46). Die untere Schranke[26] für die Wahrscheinlichkeit des zweiten Falls (c_2) ist die Wahrscheinlichkeit, dass $\mathcal{B}_{\text{all}}^{u_m}(\cdot)$ den Wert `false` liefert und kann wie folgt berechnet werden:

Satz 8 *Die Wahrscheinlichkeit, dass es irgendeinen Teilnehmer gibt, der beweisen kann, dass ein (-1)-Angriff auf eine Abstimmung stattgefunden hat, die nicht mit $\mathcal{V}(\boldsymbol{d}) = \texttt{false}$ als falsch identifiziert werden kann, ist beschränkt durch*

$$P\left(\left(\mathcal{B}_{all}^{u_m}(\cdot) = \texttt{false}\right) \mid \phi_{u_m}^t = -1\right) \geq (|U| - 1) \cdot \frac{1}{\ell} \cdot \left(\frac{\ell - 1}{\ell}\right)^{|U|-2}. \tag{4.29}$$

□

[24] Wenn diese Summe kleiner als 0 ist, wurde diese Runde auch angegriffen. Da dieser Angriff allerdings schon von der Funktion $\mathcal{V}(\boldsymbol{d})$ (Gleichung (4.21) auf Seite 46) erkannt werden würde, wird der Fall hier vernachlässigt.

[25] Das sind die gleichen Annahmen, die man braucht, um einen (-2)-Angriff im Fall *Einfache Erkennung* durchzuführen, wenn man nur *eine* DC-Netz-Runde benutzt (vgl. Fußnote 21 auf Seite 46).

[26] Wenn alle $|U| - 1$ ehrlichen Teilnehmer für den angegriffenen Zeitpunkt stimmen.

Bei *einem* Angriff auf nur *einen* Zeitpunkt (vgl. Annahme 2 auf Seite 37) ist es nicht möglich, dass $\mathcal{V}(\boldsymbol{d})$ und $\mathcal{B}_{\text{all}}^{u_m}(\cdot)$ gleichzeitig `false` ausgeben (vgl. Fußnote 24 auf der vorherigen Seite). Daher können Gleichungen (4.23) und (4.29) addiert werden, um die Wahrscheinlichkeit zu berechnen, dass ein (-1)-Angriff erkannt wird, wenn alle Teilnehmer bereit sind, ihre Verfügbarkeit aufzudecken, um einen Angreifer zu identifizieren.

Tabelle 4.1 auf Seite 51 und Abbildung 4.6 auf der nächsten Seite zeigen einige Beispielwerte für diese Formeln. Man kann sehen, dass bei kleineren Abstimmungen mit 5 Teilnehmern eine Aufteilung des Stimmvektors in 10 Teilvektoren genügt, um einen Angriff stark zu erschweren. Die Wahrscheinlichkeit, so einen (-1)-Angriff auf einen Zeitpunkt zu erkennen, ist mindestens 65,6 %. Zusätzlich zu diesen 65,6 %, kann der Angriff mit einer Wahrscheinlichkeit von 29,2 % durch einen Teilnehmer erkannt werden. Wenn alle Teilnehmer bereit sind, ihre Verfügbarkeit aufzudecken, um einen Angreifer zu entlarven, ist die Erkennungswahrscheinlichkeit mindestens 94,8 %. Sollte ein Teilnehmer u mit $\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$ feststellen, dass ein Angriff vorlag, so kann er selbst entscheiden, ob ihm seine Verfügbarkeit für diesen Zeitpunkt wichtiger ist oder ob er diesen Fehler den anderen Teilnehmern anzeigt und damit seine Verfügbarkeit aufdeckt. Daher ist die untere Schranke für die Wahrscheinlichkeit, einen Angriff zu erkennen, ein Wert zwischen den zwei Schranken, welcher davon abhängt, ob der jeweilige Teilnehmer bereit ist, seine Verfügbarkeit für den Zeitpunkt offenzulegen.

Eine weitere Darstellung der Funktionen ist in Abbildung 4.7 auf der nächsten Seite zu sehen. Hier wurde die Erkennungswahrscheinlichkeit festgelegt und die Abhängigkeit zwischen der Anzahl der Teilnehmer und der dafür benötigten Anzahl der DC-Netz-Runden dargestellt. Anhand dieser Abbildung kann beim Erstellen einer Umfrage entschieden werden, wie viele simultane DC-Netz-Runden durchgeführt werden müssen. Geht man beispielsweise davon aus, dass unter Mithilfe ehrlicher Teilnehmer mindestens 90 % Erkennungswahrscheinlichkeit garantiert werden sollen, so benötigt man in einer Umfrage mit 8 Teilnehmern mindestens 13 simultane DC-Netz-Runden.

(+2)-Angriff

Definition 25 (Konsistenzprüfung) Die Funktion $\mathcal{C} : \mathbb{Z}_n^{|U|\times|\Delta|} \to \{\texttt{true}, \texttt{false}\}$, die die Korrektheit der Prüfabstimmung überprüft, ist wie folgt definiert:

$$\mathcal{C}(\boldsymbol{d}) := \begin{cases} \texttt{true} & \text{wenn } \forall t \in T: \\ & |U| = \sum_{u\in U, i\in\mathbb{Z}_\ell, \lambda\in\{0,1\}} d_u^\delta \\ \texttt{false} & \text{sonst.} \end{cases} \tag{4.30}$$

□

Sollte diese Funktion `false` zurückliefern, so ist die Summe beider Abstimmungen kleiner oder größer als die Anzahl der Teilnehmer. Da die Stimme auf mehrere DC-Netze verteilt wurde, brauchen (-1)-Angriffe an dieser Stelle nicht betrachtet werden.

- Wenn die Summe beider Abstimmungen kleiner als die Anzahl der Teilnehmer ist, müssen ein oder mehrere Teilnehmer $\phi_u^{(t,0)} = \phi_u^{(t,1)} = 0$ gesendet haben. Die Anzahl der richtig abgegebenen Stimmen entspricht der Summe beider Abstimmungen.

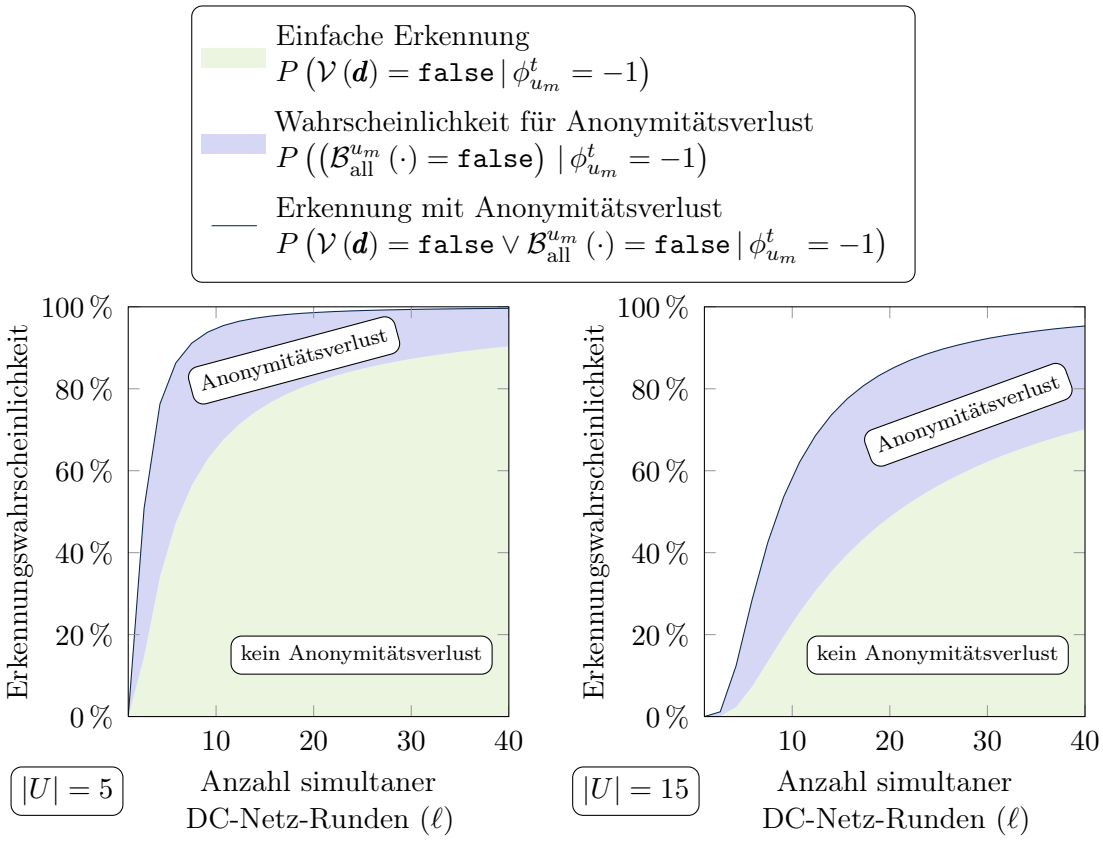


Abbildung 4.6.: Untere Schranken für die Wahrscheinlichkeit, einen (-1)-Angriff zu erkennen

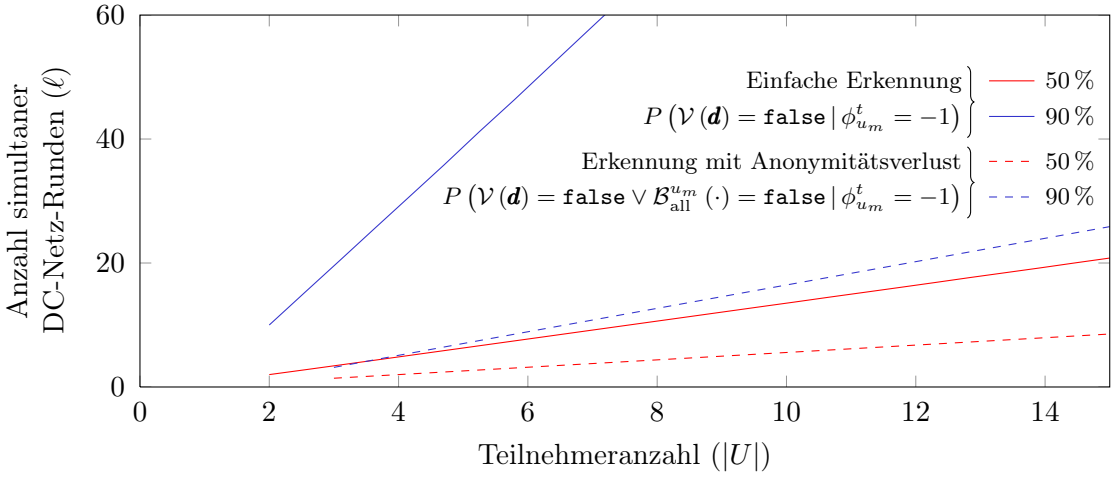


Abbildung 4.7.: Abhängigkeit zwischen der Anzahl der Teilnehmer und der Anzahl der DC-Netz-Runden bei fester Erkennungswahrscheinlichkeit

Tabelle 4.1.: Untere Schranken für die Wahrscheinlichkeit, einen Angriff zu erkennen

ℓ	$\lvert U\rvert$	Einfache Erkennung $P\left(\mathcal{V}\left(\boldsymbol{d}\right) = \texttt{false} \,\middle|\, \phi_{u_m}^t = -1\right)$	Einfache Erkennung $P\left(\mathcal{V}\left(\boldsymbol{d}\right) = \texttt{false} \,\middle|\, \phi_{u_m}^t = -2\right)$	Erkennung mit Anonymitätsverlust $P\left(\mathcal{V}\left(\boldsymbol{d}\right) = \texttt{false} \vee \mathcal{B}_{\text{all}}^{u_m}\left(\cdot\right) = \texttt{false} \,\middle|\, \phi_{u_m}^t = -1\right)$	Wahrscheinlichkeit für Anonymitätsverlust $P\left(\left(\mathcal{B}_{\text{all}}^{u_m}\left(\cdot\right) = \texttt{false}\right) \,\middle|\, \phi_{u_m}^t = -1\right)$
5	15	4,4 %	8,7 %	19,8 % (15,4 %)	
5	5	41,0 %	69,0 %	81,9 % (41,0 %)	
10	15	22,9 %	41,4 %	58,5 % (35,6 %)	
10	5	65,6 %	90,3 %	94,8 % (29,2 %)	
20	15	48,8 %	74,7 %	84,7 % (35,9 %)	
20	5	81,5 %	97,3 %	98,6 % (17,1 %)	

- Ist die Summe beider Abstimmungen größer als die Anzahl der Teilnehmer, so kann die Anzahl der richtig abgegebenen Stimmen nicht so einfach berechnet werden. Trotzdem liegt auch in diesem Fall die Anzahl der verfügbaren Personen zwischen dem Ergebnis der normalen Abstimmung ($\sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,0)}$) und der Anzahl der Teilnehmer minus dem Ergebnis der Prüfabstimmung für diesen Zeitpunkt ($|U| - \sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,1)}$).

Satz 9 *Sollte die Konsistenzprüfung einer Abstimmung fehlschlagen ($\mathcal{C}\left(\boldsymbol{d}\right) = \texttt{false}$), so liegt die Anzahl der verfügbaren Teilnehmer an einem bestimmten Zeitpunkt σ^t in folgendem Bereich:*[27]

$$\sigma^t \in \left\{ \sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,0)}, \ldots, |U| - \sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,1)} \right\}. \tag{4.31}$$

□

Da ein Angreifer seine Stimme allerdings so manipulieren kann, dass der Wertebereich für das Ergebnis immer $\sigma^t \in \{0, \ldots, |U|\}$ ausdrückt, kann damit die Verfügbarkeit der Abstimmung angegriffen werden.

[27] Ohne Beschränkung der Allgemeinheit sei $\sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,0)} \leq |U| - \sum_{u\in U, i\in\mathbb{Z}_\ell} d_u^{(t,i,1)}$.

Mehrere Angreifer

Liegt ein unkoordinierter Angriff mehrerer Teilnehmer auf einen Zeitpunkt vor, so gibt es drei Möglichkeiten:

1. mehrere (-1)-Angriffe,
2. mehrere $(+2)$-Angriffe und
3. sowohl (-1)-Angriffe als auch $(+2)$-Angriffe.

Wird die Konsistenz der Abstimmung beeinflusst ($\mathcal{C}(\boldsymbol{d}) = \texttt{false}$), so ist dies in jedem Fall erkennbar. Wir gehen also davon aus, dass ein Angreifer die Abstimmung so modifiziert, dass die Konsistenzprüfung als Ergebnis `true` liefert. Damit muss unabhängig ob ein (-1)-Angriff oder $(+2)$-Angriff durchgeführt wurde eine -1 in einem DC-Netz versendet werden[28], was möglicherweise über $\mathcal{V}(\boldsymbol{d})$ erkennbar ist.

Tritt einer der beiden ersten Fälle ein (mehrere (-1)-Angriffe oder mehrere $(+2)$-Angriffe), so sieht es für die ehrlichen Teilnehmer so aus, als würde ein Angreifer versuchen die Abstimmung oder Prüfabstimmung mit $\nu | \nu \leq -2 \vee \nu \geq 4$ zu beeinflussen. Dieser Angriff ist einfacher zu erkennen als ein einzelner (-1)- bzw. $(+2)$-Angriff (vgl. Tabelle 4.1).

Tritt der dritte Fall ein und zu einem Zeitpunkt liegen sowohl (-1)-Angriffe als auch $(+2)$-Angriffe vor, so kann es passieren, dass sich beide Angriffe gegenseitig aufheben. In diesem Fall würde ein $(-\nu)$- und ein $(\nu + 1)$-Angriff vorliegen, das Ergebnis würde allerdings so aussehen, als hätten die beiden Angreifer 0 bzw. 1 gesendet. Unentdeckt bleiben die Angriffe nur, wenn die negativen Werte beider Angreifer jeweils von positiven Werten versteckt werden. Die Wahrscheinlichkeit, dass einer der beiden Angriffe entdeckt wird kann nicht mehr mit Gleichung (4.23) berechnet werden, da beide Angreifer jeweils entweder in der Abstimmung oder der Prüfabstimmung eine 2 senden. Dadurch ist die Wahrscheinlichkeit für sie höher, unentdeckt zu bleiben.[29] Da das Ergebnis der Abstimmung allerdings so ist, als wäre es nicht angegriffen, wird hier auf eine genaue Berechnung der Wahrscheinlichkeit verzichtet.

Liegt ein $(-\nu)$-Angriff, und gleichzeitig ein $(+\nu')$-Angriff vor ($\nu \neq \nu' + 1$), so wird das Ergebnis zu Gunsten des Angriffs modifiziert, welcher mehr vom erlaubten Wertebereich abweicht ($\{0, 1\}$). Dieser Angriff wird jedoch mindestens so gut erkannt, wie es in den Sätzen 6 und 8 beschrieben wurde. Ein Beispiel soll das verdeutlichen:

Beispiel 5 Angenommen Mallory will die DC-Netz-Runden eines Zeitpunkts mit $+7$ angreifen, Marvin greift in Runden des selben Zeitpunkts mit -5 an. Mallory muss daher in der Prüfabstimmung eine -6 senden, in der Marvin eine $+6$ sendet.

Für die Berechnung der Wahrscheinlichkeit wurde davon ausgegangen, dass sich die negative Stimme des Angreifers auf $|U| - 1$ ehrliche Teilnehmer verteilt, welche alle eine 1 senden. In diesem Fall gibt es $|U| - 2$ ehrliche Teilnehmer, welche alle eine 1 senden, Marvin, welcher eine 6 sendet, sowie Mallory, die versucht ihre -6 zu verstecken.

[28] entweder in der normalen Abstimmung oder in der Prüfabstimmung
[29] Es ist so, als würden $|U| + 1$ Teilnehmer teilnehmen.

Selbst wenn Mallory in 5 DC-Netz-Runden unerkannt eine -1 senden kann, so benötigt sie für die letzte -1 eine Runde aus den verbleibenden Runden, in denen eine $+1$ gesendet wurde. Verbleibende Runden sind die Runden der ehrlichen Teilnehmer plus Marvins 6 Runden minus Mallories 5 unerkannte Runden: $|U| - 2 + 6 - 5 = |U| - 1$. Für diese letzte Runde gilt also die Berechnung der Wahrscheinlichkeit der Sätze 6 und 8. □

4.4.2. Verfügbarkeit

Um Angriffe auf die Verfügbarkeit der Abstimmung unattraktiv zu machen, wurde eine optionale Identifikationsphase eingeführt, in der herausgefunden werden kann, welcher Teilnehmer die Abstimmung angreift. Dazu wird mit Hilfe der Funktionen $\mathcal{V}(\boldsymbol{d})$, $\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$ und $\mathcal{C}(\boldsymbol{d})$ überprüft, ob ein Angriff stattgefunden hat. Sollte ein Angriff stattgefunden haben, so werden die fehlerhaften DC-Netz-Runden aufgedeckt und damit der Angreifer identifiziert. Da das Aufdecken ein Vertraulichkeitsproblem sein kann, kann auch entschieden werden, auf die Identifikationsphase zu verzichten. Diese Entscheidung muss jedoch vor Durchführung der Abstimmung getroffen und von allen Teilnehmern akzeptiert werden. Wenn jeder Teilnehmer für sich selbst entscheidet, ob er seine Verfügbarkeit offenlegt oder auf Geheimhaltung besteht, würde ein Angreifer immer die Herausgabe seiner Schlüssel, mit der Begründung er hätte seine Daten zu schützen, verweigern.

Da das Aufdecken der DC-Netz-Runden auch ein Vertraulichkeitsproblem für die Teilnehmer darstellt, sollten möglichst wenige DC-Netz-Runden aufgedeckt werden. Näheres dazu wird im folgenden Abschnitt diskutiert.

4.4.3. Vertraulichkeit

Halten sich alle Teilnehmer an das Protokoll, so erfährt jeder die Anzahl der verfügbaren Personen zu den zur Abstimmung stehenden Zeitpunkten. Wie schon in Abschnitt 3.1.3 diskutiert wurde sind das immer noch mehr Informationen als nötig wären, wenn man eine Auswahlfunktion vorher festlegen würde und nur den einen Termin bekannt gibt. Jedoch ist der Informationsgehalt dieses Ergebnisses sehr gering und stellt einen Kompromiss zwischen Flexibilität in der Auswahlfunktion und Vertraulichkeit dar.

Kann ein Angreifer die Einzelstimmen eines Opfers berechnen, so kann er daraus den Verfügbarkeitsvektor des Teilnehmers berechnen. Da die Einzelstimmen mit einer informationstheoretisch sicheren Vernam Chiffre verschlüsselt wurden, bleibt ihm nur die Möglichkeit, die Schlüssel aller anderen Teilnehmer in Erfahrung zu bringen. Das Commitment (vgl. Gleichung (4.18) auf Seite 44) muss die Sicherheit der Vernam Chiffre nicht einschränken, da man sich typischerweise eine der zwei Eigenschaften (geheim oder bindend) informationstheoretisch sicher wählen kann und die andere nur komplexitätstheoretische Sicherheit bietet. Mit der Verbesserung der Kommunikationskomplexität in Abschnitt 4.4.4 wird die Sicherheit allerdings auf komplexitätstheoretisch beschränkt.

Zusätzlich hängt die Geheimhaltung der Nachrichten noch an dem DC-Netz-Index, in dem die Verfügbarkeit gesendet wurde. Wenn Mallory vorhersagen kann, in welcher DC-Netz-Runde die Nachrichten von verschiedenen Teilnehmern gesendet werden, kann sie diese Nachrichten in kleinere Anonymitätsgruppen teilen. Wenn alle Teilnehmer ihre

Nachrichten zufällig über alle DC-Netz-Runden verteilen, benötigt Mallory die Hilfe aller anderen Teilnehmer, um ihr Opfer zu deanonymisieren.

Satz 10 *Wenn alle $|U|$ Teilnehmer ihre DC-Netz-Indices gleichverteilt über die Menge aller DC-Netz-Runden wählen, so erfährt ein passiver Angreifer, der keinerlei Kenntnis über die Auftrittswahrscheinlichkeit der einzelnen Verfügbarkeiten hat, nur die Anzahl der verfügbaren Teilnehmer x. Die Wahrscheinlichkeit, dass ein Opfer zu einem Zeitpunkt verfügbar ist, ist $\frac{x}{|U|}$.*[30] □

Beweis (durch Induktion über ℓ) Ein Angreifer wählt ein beliebiges Opfer und möchte herausfinden, ob es zu einem Zeitpunkt verfügbar ist.

Induktionsanfang ($\ell = 1$): Wird nur ein DC-Netz verwendet, so gilt der Beweis für ein einzelnes DC-Netz, weshalb ein Angreifer nur die Summe aller Eingaben erfährt [Cha88b]. Da jeder Teilnehmer als Eingabe nur eine 0 oder 1 in Abhängigkeit seiner Verfügbarkeit sendet erfährt der Angreifer nur die Summe der verfügbaren Teilnehmer.

Induktionsschritt ($\ell \implies \ell + 1$): Alle $\ell + 1$ DC-Netz-Runden werden in 2 Gruppen geteilt. Die ℓ DC-Netze, für die die Induktionsbehauptung gilt bilden die erste Gruppe. Das Ergebnis dieser Gruppe sei $y \leq x$. Das $\ell + 1$-te DC-Netz bildet die zweite Gruppe mit dem Ergebnis $x - y$.

Für die Berechnung ob das Opfer verfügbar ist (eine 1 gesendet hat) können zwei disjunkte Fälle unterschieden werden:

c_1: Das Opfer hat eine 1 in einer der ersten ℓ DC-Netz-Runden geschickt oder

c_2: in der $\ell + 1$-ten Runde.

Betrachtet man nur die erste Gruppe von DC-Netz-Runden, so ist die Wahrscheinlichkeit, dass das Opfer verfügbar ist

$$P(c_1) = \frac{y}{|U|}. \tag{4.32}$$

Werden nun die Informationen, die man durch die zusätzliche DC-Netz-Runde bekommt mit hinzugezogen, so müssen für das Senden der Verfügbarkeit nur die Teilnehmer in Betracht gezogen werden, welche in den anderen ℓ DC-Netz-Runden noch keine 1 gesendet haben ($|U| - y$ Teilnehmer). Die Wahrscheinlichkeit, dass das Opfer in diesen Runden keine 1 schickt ist $1 - P(c_1)$. Das Ergebnis der letzten Runde ist $x - y$. Die Wahrscheinlichkeit $P(c_2)$, dass das Opfer in der letzten Runde eine 1 schickt ist

$$P(c_2) = (1 - P(c_1)) \cdot \frac{x - y}{|U| - y}. \tag{4.33}$$

Werden $P(c_1)$ und $P(c_2)$ addiert, so erhält man die Wahrscheinlichkeit, dass das Opfer verfügbar ist. Diese Wahrscheinlichkeit beinhaltet die Kenntnis über die einzelnen Summen der Teil-DC-Netze. Hat ein Angreifer keine Kenntnis über die Teil-DC-Netz-Summen, so

[30] Kennt der Angreifer Teile der Eingabe, so müssen die ihm bekannten Verfügbarkeiten von der Wahrscheinlichkeit abgezogen werden.

ist die Wahrscheinlichkeit, dass das Opfer an dem Zeitpunkt verfügbar ist $\frac{x}{|U|}$. Es muss gezeigt werden, dass beide Wahrscheinlichkeiten gleich sind:

$$\begin{aligned} P(c_1) + P(c_2) &= \frac{y}{|U|} + \left(1 - \frac{y}{|U|}\right) \cdot \frac{x-y}{|U|-y} \\ &= \frac{y}{|U|} + \frac{(|U|-y)\cdot(x-y)}{|U|\cdot(|U|-y)} \\ &= \frac{y}{|U|} + \frac{(x-y)}{|U|} \\ &= \frac{x}{|U|}. \end{aligned} \tag{4.34}$$

■

Es wurde bereits erwähnt, dass die Identifikationsphase auch ein Vertraulichkeitsproblem für ehrliche Teilnehmer darstellt. Um die Korrektheit der Umfrage zu überprüfen, testet jeder Teilnehmer die drei Funktionen:

- Verify: $\mathcal{V}(\boldsymbol{d}) \stackrel{?}{=} \texttt{true}$
- Consistency: $\mathcal{C}(\boldsymbol{d}) \stackrel{?}{=} \texttt{true}$
- Blame: $\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u) \stackrel{?}{=} \texttt{true}$.

Die ersten zwei Funktionen kann jeder überprüfen. $\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$ benötigt das Wissen der geheimen Schlüssel und kann daher von jedem Teilnehmer u für seine eigenen Schlüssel durchgeführt werden. Nachfolgend wird für jede Funktion getrennt diskutiert, wie die Identifikation mit möglichst wenig Anonymitätsverlust von statten gehen kann.

Verify

Schlägt die erste der drei Funktionen fehl ($\mathcal{V}(\boldsymbol{d}) = \texttt{false}$), so hat ein Angreifer versucht, in einer DC-Netz-Runde einen negativen Wert zu senden. In diesem Fall wird die DC-Netz-Runde aufgedeckt, deren globale Summe nicht im erlaubten Wertebereich ($\{0, \ldots, |U|\}$) liegt.

Für den Fall, dass mehrere DC-Netz-Runden angegriffen wurden[31], sollte *nur* die Runde aufgedeckt werden, welche die niedrigste Summe hat, da in dieser die Wahrscheinlichkeit am größten ist, dass kein ehrlicher Teilnehmer eine 1 gesendet hat. Ohne möglichen Anonymitätsverlust ehrlicher Teilnehmer ist es nicht entscheidbar, ob es sich um einen oder mehrere Angreifer handelt. Deshalb sollte ein möglicher zweiter Angreifer in einer Folgeabstimmung identifiziert werden.

Definition 26 (Angreiferidentifikation bei Verifikationsfehler) Die Funktion $\mathcal{D}_{\mathcal{V}} : \mathbb{Z}_n^{|U|\times|\Delta|} \to \mathcal{P}(\Delta)$, deren Eingabe alle DC-Netz-Nachrichten sind und die eine Menge von

[31] Das kann entweder passieren, wenn ein Angreifer mehrere DC-Netz-Runden angreift oder wenn mehrere Angreifer eine oder mehrere DC-Netz-Runden angreifen.

DC-Netz-Runden ausgibt, an denen die globale DC-Netz-Summe am niedrigsten ist, ist wie folgt definiert:

$$\mathcal{D}_{\mathcal{V}}\left(\boldsymbol{d}\right) := \left\{ \delta : \sum_{u \in U} d_u^{\delta} = \min_{\delta' \in \Delta} \left\{ \sum_{u \in U} d_u^{\delta'} \right\} \right\}. \tag{4.35}$$

□

Es kann eine beliebige DC-Netz-Runde aus dieser Menge gewählt werden. Es sollte jedoch vor der Abstimmung festgelegt sein, nach welchen Kriterien die Runde gewählt wird (z. B. immer die mit dem kleinsten t, i bzw. λ).

Consistency

Sollte die Konsistenzprüfung für die Prüfabstimmung fehlschlagen ($\mathcal{C}\left(\boldsymbol{d}\right) = \texttt{false}$), so müssten grundsätzlich alle DC-Netz-Runden aufgedeckt werden, für die die Konsistenz fehlschlug:

Definition 27 (Angreiferidentifikation durch Beschuldigung) Die Funktion $\mathcal{D}_{\mathcal{C}} : \mathbb{Z}_n^{|U| \times |\Delta|} \to \mathcal{P}(\Delta)$, deren Eingabe alle DC-Netz-Nachrichten sind und die eine Menge von DC-Netz-Runden ausgibt, an denen die Konsistenzprüfung fehlgeschlagen hat, ist wie folgt definiert:

$$\mathcal{D}_{\mathcal{C}}\left(\boldsymbol{d}\right) := \left\{ \delta = (t, i, \lambda) : i \in \mathbb{Z}_{\ell}, \lambda \in \{0, 1\}, |U| \neq \sum_{u \in U, i' \in \mathbb{Z}_{\ell}, \lambda' \in \{0,1\}} d_u^{(t,i',\lambda')} \right\} \tag{4.36}$$

□

Um zu vermeiden, dass alle Verfügbarkeiten veröffentlicht werden, können die DC-Netz-Runden aus dieser Menge Schritt für Schritt veröffentlicht werden. Sobald der Angreifer gefunden wurde, kann aufgehört werden. Die Reihenfolge, in der aufgedeckt wird, sollte vor der Abstimmung festgelegt sein, den Teilnehmern aber erst bekannt gegeben werden nachdem alle ihre Stimmen abgegeben haben. Eine mögliche Realisierung ist, dass jeder Teilnehmer zusammen mit seiner Stimme ein Commitment für eine Zufallszahl sendet. Wird die Aufdeckung nötig, so werden alle Commitments aufgedeckt und alle Zufallszahlen zu einem Startwert zusammengefasst, welcher dafür benutzt wird, eine Reihenfolge festzulegen. Durch dieses Schema steigt allerdings die Anzahl der blockierenden Protokollrunden an.

In manchen Konstellationen kann es passieren, dass sowohl die Funktion $\mathcal{V}\left(\boldsymbol{d}\right)$ als auch die Funktion $\mathcal{C}\left(\boldsymbol{d}\right)$ das Ergebnis `false` liefern. Da das Ziel der ehrlichen Teilnehmer ist, so wenig wie möglich Runden aufzudecken, um den Angreifer zu finden, sollte erst eine Runde aus der Menge aufgedeckt werden, die mit $\mathcal{D}_{\mathcal{V}}\left(\boldsymbol{d}\right)$ bestimmt wurde.

Auch wenn mehrere Angreifer die Umfrage gleichzeitig angreifen, können beide Funktionen `false` zurückgeben. Da dieser Fall nicht ohne möglichen Anonymitätsverlust davon unterschieden werden kann, ob ein Angreifer mehrere Runden gleichzeitig angreift oder ob es sich um unterschiedliche Angreifer handelt, ist es besser, nur eine Runde aufzudecken und einen möglichen zweiten Angreifer in Folgeabstimmungen zu identifizieren.

Blame

Schlägt für einen Teilnehmer die dritte Funktion fehl, so muss er sich entscheiden, ob er seine Verfügbarkeit aufdeckt, um den Angreifer zu identifizieren, oder ob ihm die Geheimhaltung seiner Stimme am angegriffenen Zeitpunkt wichtiger ist.

Definition 28 (Angreiferidentifikation bei Konsistenzfehler) Die Funktion $\mathcal{D}_{\mathcal{B}} : \mathbb{Z}_n^{|U|\times|\Delta|} \times \mathbb{Z}_n^{(|U|-1)\times|\Delta|} \to \mathcal{P}(\Delta)$, welche die Menge von DC-Netz-Runden ausgibt für die ein Teilnehmer u beweisen kann, dass ein Angriff stattgefunden hat ($\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u) = \texttt{false}$), trotz des Ergebnisses `true` der Funktion $\mathcal{V}(\boldsymbol{d})$, ist wie folgt definiert:

$$\mathcal{D}_{\mathcal{B}}(\boldsymbol{d}, \boldsymbol{k}_u) := \left\{ \delta : \left(\sum_{u' \in U} d_{u'}^{\delta} = 0 \right) \wedge \left(d_u^{\delta} + \sum_{u' \in U \setminus \{u\}} k_{u,u'}^{\delta} = 1 \right) \right\}. \tag{4.37}$$

□

Entscheidet sich der Teilnehmer dafür, den Angreifer identifizieren zu lassen, so muss er sich eine DC-Netz-Runde aus dieser Menge aussuchen, für die er seine Schlüssel offenlegt ($\{k_{u,u'}^{\delta} : u' \in U \setminus \{u'\}\}$). Damit beweist er, dass er an diesem Zeitpunkt eine 1 gesendet hat.

4.4.4. Blockierende Protokollrunden

Das bisher vorgestellte Verfahren geht davon aus, dass jeder Teilnehmer mit jedem anderen Teilnehmer einen symmetrischen Schlüssel austauscht. Würden bei diesem Schlüsselaustausch zwischen jedem Teilnehmer jeweils echte Zufallszahlen über eine gesicherte Verbindung ausgetauscht, so hat das zwar den Vorteil der informationstheoretischen Sicherheit, allerdings würde die Anzahl der Kommunikationsbeziehungen quadratisch mit der Anzahl der Teilnehmer steigen. In diesem Abschnitt soll gezeigt werden, wie der Schlüsselaustausch mittels asymmetrischer Kryptographie gelöst werden kann. Es werden nachfolgend dafür zwei Lösungen vorgestellt, eine mittels eines Schlüsseltransportprotokolls und eine Lösung auf Basis eines Schlüsselvereinbarungsprotokolls.

Anstatt bei jeder Umfrage einen symmetrischen Schlüssel mit jedem anderen Teilnehmer auszutauschen, registriert sich jeder potentielle Teilnehmer einmalig bei einem Schlüsselserver und hinterlegt einen öffentlichen Schlüssel. Das bei dieser Registrierung generierte asymmetrische Schlüsselpaar kann für beliebig viele Folgeabstimmungen verwendet werden.

Da bisher immer nur eine Umfrage betrachtet wurde, nun aber der geheime Schlüssel in vielen Umfragen benutzt wird, wird das DC-Netz-Runden-Tripel (t, i, λ) auf ein Quadrupel $(\mathsf{uuid}, t, i, \lambda)$ erweitert, welches eine für jede Umfrage eindeutige einmalige Umfragen-ID beinhaltet ($\mathsf{uuid} \in \mathsf{UUID}$). Eine DC-Netz-Runde δ ist daher ein Quadrupel, welches Element der Menge $\Delta := \mathsf{UUID} \times T \times \mathbb{Z}_\ell \times \{0, 1\}$ ist. Zusätzlich wurde bisher davon ausgegangen, dass bei der Initialisierung einer Umfrage die Menge der Teilnehmer U, sowie die Menge der Zeitpunkte T festgelegt wird. Um diese über mehrere Umfragen zu unterscheiden wird U^{uuid} bzw. T^{uuid} geschrieben.

Vereinfachter Schlüsselaustausch mittels eines Schlüsseltransportprotokolls

Um einen Schlüsselaustausch durchzuführen, kann der Schlüsseltransportmechanismus 2 des ISO/EIC 11770-3 Standards [ISO11770-3] verwendet werden.[32] Vorbedingung für diesen Mechanismus ist, dass die Teilnehmer öffentliche Schlüssel ausgetauscht haben. Jeder Teilnehmer u registriert sich dafür bei einem Schlüsselserver, indem er einen öffentlichen Schlüssel eines beliebigen asymmetrischen Verschlüsselungssystems hinterlegt. Aus jedem Teilnehmerpaar legt anschließend der Teilnehmer, der zuerst an der Terminabstimmung teilnimmt, symmetrische Schlüssel für die Paarbeziehung fest und schickt diese verschlüsselt und signiert dem anderen Teilnehmer. Um diese Schlüssel-Nachrichten einfacher zu koordinieren, werden sie nicht direkt zu dem anderen Teilnehmer geschickt, sondern zusammen mit dem verschlüsselten Verfügbarkeitsvektor auf dem Abstimmungsserver abgelegt. Dadurch werden die Schlüssel sequenziell in der Reihenfolge festgelegt, in der die Teilnehmer ihre Stimmen abgeben.

Die Kommunikationsschritte dieses Protokolls sind in Abbildung 4.8 auf der nächsten Seite aufgezeigt. In dieser Abbildung ist ein möglicher Ablauf zu sehen, bei dem Carol die erste ist, die ihre Stimme abgibt. Deshalb legt sie mit ihren verschlüsselten Verfügbarkeitsvektoren die Schlüssel für die Paarbeziehung Carol–Alice und Carol–Bob fest ($\vec{k}_{u_c,u_a}, \vec{k}_{u_c,u_b}$). Zu jedem Schlüssel der Schlüsselvektoren berechnet sie ein Commitment (vgl. Gleichung (4.18) auf Seite 44). Die Commitments für die Schlüssel einer Paarbeziehung werden mit dem Commitmentvektor $\vec{\psi}_{u_a,u_b}$ bezeichnet. Carol verschlüsselt die Schlüsselvektoren asymmetrisch für ihre jeweiligen Empfänger und signiert selbige wie im ISO Standard vorgegeben.[33] Das Schlüsselpaket, welches von Carol für Alice bestimmt ist, ist folgendermaßen aufgebaut:[34]

$$u_a, \mathsf{uuid}, \mathrm{enc}_{u_a}\left(u_c, \vec{k}_{u_c,u_a}\right), \vec{\psi}_{u_c,u_a}, \mathrm{sig}_{u_c}\left(u_a, \mathsf{uuid}, \mathrm{enc}_{u_a}\left(u_c, \vec{k}_{u_c,u_a}\right), \vec{\psi}_{u_c,u_a}\right). \quad (4.38)$$

Im ISO Standard wird ein Zeitstempel oder eine Zählvariable vorgeschlagen, um zu verhindern, dass alte Schlüssel wiederverwendet werden. In Abweichung zum ISO Standard wird das hier durch die Umfragen-ID (uuid) sichergestellt.

Da Alice erst nach Carol abstimmt, teilt ihr der Server den Schlüsselvektor mit, den Carol für die Paarbeziehung (Alice, Carol) festgelegt hat. Anhand der Signatur kann Alice erkennen, ob Dritte angegriffen haben. Anschließend überprüft sie, ob die Commitments zu den Schlüsseln passen, um Angriffe von Carol auszuschließen. Weil Bob noch nicht abgestimmt hat, generiert Alice für die Paarbeziehung (Alice, Bob) neue Schlüssel und berechnet diese auf gleiche Weise, wie es bei Carol erklärt wurde.

Um die Anzahl der benötigten Signaturen zu reduzieren, kann jeder Teilnehmer, statt jedes Schlüsselvektor–Commitmentvektor–Paar einzeln zu signieren, alle Paare insgesamt signieren. Damit die Teilnehmer die Signatur überprüfen können, muss der Server dann allerdings jedem Teilnehmer alle Schlüsselvektor–Commitmentvektor–Paare mitteilen, auch diese, die nicht für die jeweilige Person bestimmt sind.

[32] Das Protokoll ist in Anhang D beschrieben.

[33] Diese Verschlüsselung und Signatur ist in Abbildung 4.8 aus Platzgründen nicht dargestellt.

[34] Sei $\mathrm{enc}_u(x)$ die für Teilnehmer u verschlüsselte Nachricht x und $\mathrm{sig}_u(x)$, eine digitale Signatur von Teilnehmer u für die Nachricht x.

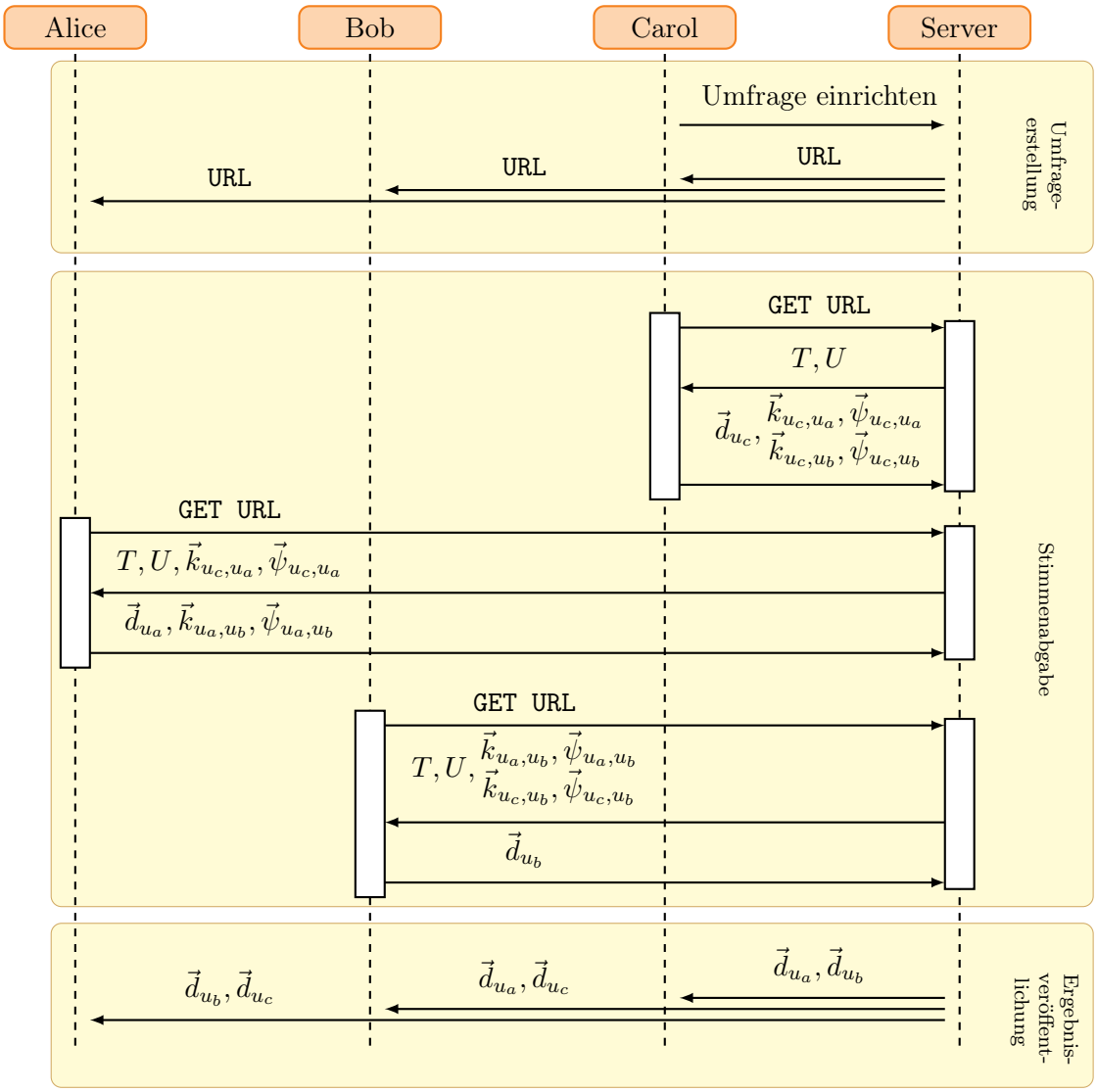


Abbildung 4.8.: Kommunikationsschritte des Protokolls, wenn ein Schlüsseltransportprotokoll verwendet wird. Die Darstellung beinhaltet nur, welche Daten zwischen den Entitäten ausgetauscht werden. Insbesondere wurde auf die Darstellung der Verschlüsselung und der Signatur aus Lesbarkeitsgründen verzichtet. Die verschlüsselten Verfügbarkeitsvektoren ($\vec{d}_u$) sind jeweils von ihren Sendern digital signiert. Die Schlüsselvektoren der Paarbeziehungen ($\vec{k}_{u,u'}$) sind für den jeweils anderen Partner asymmetrisch verschlüsselt. Die verschlüsselten Schlüsselvektoren, sowie die Commitments ($\vec{\psi}_{u,u'}$) sind von ihren Sendern digital signiert. Ein Beispiel für eine solche Nachricht ist in Gleichung (4.38) auf der vorherigen Seite dargestellt. Die `URL` ist die URL, unter welcher die Umfrage aufgerufen werden kann.

Vereinfachter Schlüsselaustausch mittels eines Schlüsselvereinbarungsprotokolls

Eine weitere Möglichkeit, den Schlüsselaustausch zu vereinfachen, ist die Benutzung eines Schlüsselvereinbarungsprotokolls. Im Folgenden soll diese Möglichkeit für das Diffie–Hellman-Schlüsselvereinbarungsprotokoll [DH76] vorgestellt werden.

Sei q ein Modulus und g ein Generator für das Diffie–Hellman-Schlüsselvereinbarungsprotokoll. Jeder Teilnehmer u registriert sich in drei Schritten:

1. Hole den Diffie–Hellman-Modulus q und den Diffie–Hellman-Generator g vom Schlüsselserver.

2. Generiere eine Zufallszahl und speichere sie als geheimen Schlüssel sec_u.

3. Berechne den öffentlichen Schlüssel $\mathsf{pub}_u := g^{\mathsf{sec}_u} \bmod q$ und veröffentliche ihn auf dem Schlüsselserver.

Nach diesen Schritten hat jeder Teilnehmer die Möglichkeit, sich mit Hilfe des öffentlichen Schlüssels eines anderen Teilnehmers auf ein gemeinsames Diffie–Hellman-Geheimnis zu einigen.

Die symmetrischen Schlüssel werden aus den Diffie–Hellman-Geheimnissen berechnet. Jeder Teilnehmer u berechnet daher für jeden anderen Teilnehmer $u', u \neq u'$:

$$\mathsf{dh}_{u,u'} := (\mathsf{pub}_{u'})^{\mathsf{sec}_u} \bmod q \tag{4.39}$$

Für jede DC-Netz-Runde $\delta \in \Delta$ wird ein Schlüssel $k^{\delta}_{u,u'}$ wie folgt berechnet:

$$k^{\delta}_{u,u'} := \mathrm{KDF}\left(\delta, \mathsf{dh}_{u,u'}\right) \tag{4.40}$$

Die Funktion KDF ist hierbei eine Funktion, welche eine $(\mathsf{dh}_{u,u'})$-pseudozufällige Ausgabe erzeugt.

Definition 29 ((seed)-Pseudozufall) Eine Funktion $\mathcal{F}(x, \mathsf{seed}) = r$ erzeugt eine *(*seed*)-pseudozufällige* Ausgabe, wenn es unter Kenntnis beliebig vieler Tupel

$$\mathcal{K} = \{(x, r) : \mathcal{F}(x, \mathsf{seed}) = r\} \tag{4.41}$$

keinen polynomialen statistischen Test gibt, der die Ausgabe r für ein neues x, $(x, r) \notin \mathcal{K}$ von einer echten Zufallszahl gleicher Länge mit einer besseren Wahrscheinlichkeit als $\frac{1}{2}$ unterscheiden kann. □

Die Menge $\mathcal{K}$ ist für unsere Funktion KDF die Menge aus älteren Umfragen bekannter Quadrupel

$$\mathcal{K} = \left\{ \left(\delta, u_a, u_b, k^{\delta}_{u_a,u_b}\right) : \delta = (\mathsf{uuid}, t, i, \lambda) \in \Delta, u_a, u_b \in U^{\mathsf{uuid}}, u_a \neq u_b \right\}, \tag{4.42}$$

für welche durch mögliche Aufdeckung die Schlüssel bekannt sind.

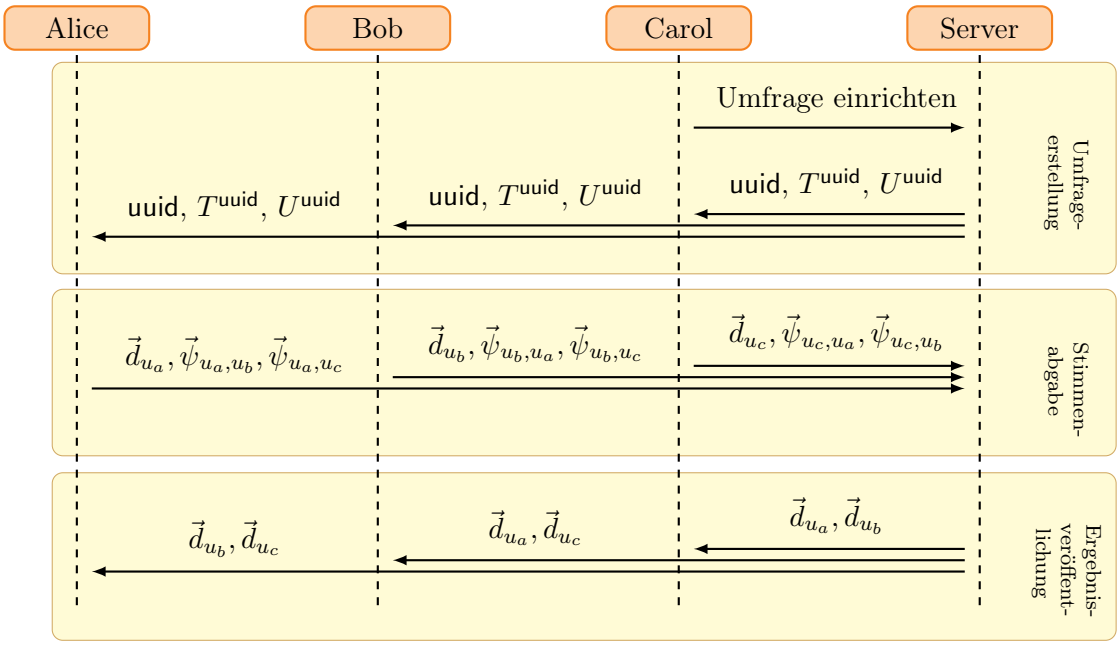


Abbildung 4.9.: Kommunikationsschritte des Protokolls, wenn ein Schlüsselvereinbarungsprotokoll verwendet wird

Eine mögliche Funktion, die diese Eigenschaft erfüllt, ist eine symmetrische Entschlüsselungsfunktion $\text{decr}^{\text{sym}}_{\mathsf{key}}(\mathsf{ciphertext})$, welche Angriffen widersteht, bei denen der Schlüsseltext adaptiv gewählt werden kann.[35] Die Schlüssel werden also wie folgt berechnet:

$$k^{\delta}_{u,u'} := \text{decr}^{\text{sym}}_{\mathsf{dh}_{u,u'}}(\delta)\,. \tag{4.43}$$

Der Kommunikationsablauf des Protokolls ist in Abbildung 4.9 dargestellt.

Zusammenfassung: Blockierende Protokollrunden

Mit den vorgestellten Erweiterungen ist der Kommunikationsaufwand nicht höher als bei vergleichbaren Web 2.0-Applikationen. Es werden die minimale Anzahl blockierender Protokollrunden mit Registrierung benötigt. Damit die Notwendigkeit der Registrierung für wenigstens einen Teil der Nutzer entfällt, kann auf existierende Schlüsselstrukturen (z.B. PGP oder X.509) zurückgegriffen werden. Werden existierende Schlüsselstrukturen verwendet, so muss der Schlüsselaustausch in der Regel mit dem vorgestellten Schlüsseltransportprotokoll durchgeführt werden. Ein Implementierungsproblem bei der Nutzung vorhandener Schlüssel ist, dass es für Web-Browser in JavaScript keine API gibt, die

[35] Die Sicherheit gegen aktiv adaptive Angriffe wird hier benötigt, da ein Initiator, wenn er mehrere aufeinanderfolgende Umfragen organisiert, Einfluss auf die Zeitpunkte T hat.

einen Zugriff auf lokale PGP oder SSL Operationen erlauben.[36] Ein weiteres Problem des Schlüsseltransportprotokolls ist, dass eine Wettlaufsituation (engl. *race condition*) zwischen den Teilnehmern entsteht, die von der Implementierung gelöst werden muss.

Im weiteren Verlauf der Arbeit wird nur das Schlüsselvereinbarungsprotokoll betrachtet.

4.4.5. Reaktionszeit

Da sich die Reaktionszeit über den Berechnungsaufwand und den Kommunikationsaufwand definiert, werden beide Teile im Folgenden getrennt betrachtet.

Berechnungsaufwand

Wie schon erwähnt wurde, agiert der Server nur als schwarzes Brett und veröffentlicht alle Nachrichten, nachdem die letzte Stimme abgegeben wurde. Um zu entscheiden, wann die letzte Nachricht eingegangen ist, und um externe Angriffe abzuwenden, muss die digitale Signatur jeder Nachricht überprüft werden. Serverseitig ist daher die Überprüfung $|U|$ digitaler Signaturen nötig.

Die clientseitigen Berechnungen werden im Folgenden nach den Phasen der Umfrage getrennt betrachtet.

Bevor ein Teilnehmer an Umfragen teilnehmen kann, muss er einen Schlüssel bei einem Schlüsselserver hinterlegen. Dafür ist bei jedem Teilnehmer eine diskrete Exponentation ($\mathsf{pub}_u = g^{\mathsf{sec}_u} \bmod q$) sowie die Berechnung eines Signaturschlüssels nötig. Da dieser Schlüssel allerdings für alle Folgeabstimmungen verwendet werden kann, ist der Aufwand vernachlässigbar.

Um die Umfrage zu erstellen, legt der Initiator die Menge der Teilnehmer und die Menge der Zeitpunkte fest. Um diese vor Modifikationen zu schützen, müssen sie digital signiert werden. In der Umfrageerstellungsphase ist daher die Berechnung einer digitalen Signatur notwendig.

In der Abstimmungsphase wird zuerst die Signatur des Umfrageninitiators überprüft. Anschließend muss für jede DC-Netz-Runde ein Schlüssel (Gleichung (4.43) auf der vorherigen Seite) sowie das zugehörige Commitment (Gleichung (4.18) auf Seite 44) berechnet werden. Die dabei aufwendigste Operation ist die Berechnung des Diffie–Hellman-Geheimnisses (Gleichung (4.39) auf Seite 60). Da das Diffie–Hellman-Geheimnis für jede DC-Netz-Runde gleich ist, muss jeder Teilnehmer u die Berechnung einmal für jeden anderen Teilnehmer $u', u \neq u'$ durchführen. Um Berechnungen in Folgeumfragen zu sparen, kann ein einmal berechnetes Diffie–Hellman-Geheimnis für spätere Abstimmungen gespeichert werden. Darüber hinaus muss jeder Teilnehmer $(2 \cdot \ell \cdot |T| \cdot |U|)$ symmetrische Entschlüsselungen sowie Commitments berechnen. Um die Stimme gegen Angriffe dritter zu schützen, ist abschließend noch die Berechnung einer digitalen Signatur notwendig.

[36] Eine Option wäre natürlich den privaten Schlüssel, welcher sonst für die E-Mail-Verschlüsselung benutzt wird, dem Browser mitzuteilen, um anschließend die Operationen in JavaScript auszuführen. In diesem Fall setzt sich allerdings der Benutzer der Gefahr aus, dass böswilliger JavaScript Code den geheimen Schlüssel ausliest und damit nicht nur die Sicherheit der Terminabstimmungsapplikation sondern auch die des E-Mail-Verkehrs gefährden kann.

Tabelle 4.2.: Berechnungsaufwand für die Durchführung einer Umfrage

	Abstimmung		Identifikation*
	Teilnehmer	Server	Teilnehmer
diskrete Exponentation	$\lvert U\rvert - 1$		
digitale Signatur berechnen	1		
digitale Signatur überprüfen	1	$\lvert U\rvert$	$\lvert U\rvert - 1$
symmetrische Entschlüsselung	$2 \cdot \ell \cdot \lvert T\rvert \cdot \lvert U\rvert$		
Commitment	$2 \cdot \ell \cdot \lvert T\rvert \cdot \lvert U\rvert$		$\geq \lvert U\rvert - 1$**

* nur im Fehlerfall
** in Abhängigkeit des Fehlers

Stellt sich in der Verifikationsphase heraus, dass eine DC-Netz-Runde fehlerhaft war, so wird eine Aufdeckung der fehlerhaften Runde durchgeführt. Um an dieser Stelle externe Angreifer auszuschließen, müssen die Signaturen der Teilnehmernachrichten überprüft werden. Es ist abhängig vom Fehler, wie hoch der Berechnungsaufwand ist. Mindestens[37] muss jeder Teilnehmer $(|U| - 1)$ Commitments überprüfen.

Tabelle 4.2 fasst den Berechnungsaufwand nochmal zusammen. Es ist ersichtlich, dass die Anzahl der asymmetrischen Operationen (diskrete Exponentation und digitale Signaturen) unabhängig von der Anzahl der Zeitpunkte $|T|$ ist. Werden die gemeinsamen Diffie–Hellman-Geheimnisse zwischen den Teilnehmern für Folgeumfragen gespeichert, so sinkt der Berechnungsaufwand weiter.

Kommunikationsaufwand

Die Anzahl der zu übertragenden Nachrichten steigt im vorgestellten Schema nur unwesentlich. Die meisten zusätzlichen Daten werden in der Stimmenabgabe übertragen. Hier schickt jeder Teilnehmer zusätzlich zu seinem Verfügbarkeitsvektor die Commitments. Die Größe der Daten hängt vom DC-Netz-Modulus ab. Die folgende Beispielrechnung verdeutlicht die Menge der übertragenen Daten.

Beispiel 6 Angenommen im DC-Netz wird mit 8 Bit großen Zahlen gerechnet ($n = 256$) und es wird eine Terminabstimmung mit 5 Teilnehmern über 16 Zeitpunkte durchgeführt ($|U| = 5, |T| = 16$). Die Anzahl simultaner DC-Netze pro Zeitpunkt ist 20 ($\ell = 20$).

Für diese Abstimmung müsste jeder Teilnehmer in der Stimmenabgabe einen Verfügbarkeitsvektor der Länge 16 spezifizieren. Jedes Element des Vektors wird über 20 DC-Netze verteilt. Zusätzlich werden 20 Prüf-DC-Netze verwendet. Die Anzahl der DC-Netze ist also $|\Delta| = |T \times \mathbb{Z}_\ell \times \{0,1\}| = 16 \cdot 20 \cdot 2 = 640$, für den Verfügbarkeitsvektor $\vec{d}_u$ werden insgesamt $16 \cdot 20 \cdot 2 \cdot 8\,\text{Bit} = 640\,\text{Byte}$ benötigt.

[37] wenn nur eine Runde aufgedeckt wird, d. h. $\mathcal{V}(\boldsymbol{d}) = \texttt{false}$ oder $\mathcal{B}_{\text{all}}^{u_m}(\boldsymbol{d}, \{\boldsymbol{k}_u : u \in U \setminus \{u_m\}\}) = \texttt{false}$

Angenommen, das Commitmentschema verwendet 160 Bit große Commitments[38]. Die Menge der zu übertragenden Daten für einen Commitmentvektor zwischen zwei Teilnehmern $\vec{\psi}_{u,u'}$ wäre in so einem Fall $|\Delta| \cdot 160\,\text{Bit} = 12{,}5\,\text{kByte}$.

Insgesamt müssen ein Verfügbarkeitsvektor und $(|U| - 1)$ Commitmentvektoren übertragen werden. In diesem Beispiel sind das $640\,\text{Byte} + 4 \cdot 12{,}5\,\text{kByte} = 50{,}625\,\text{kByte}$. Diese Datenmengen sind im Web durchaus üblich und sorgen für keine erhöhten Reaktionszeiten. □

4.4.6. Installationsaufwand

Für das vorgestellte Schema ist keine zusätzliche Installation notwendig. Alle Operationen können in JavaScript implementiert werden.

Wird mehr Wert auf die Minimierung der blockierenden Protokollrunden gelegt und sollen bestehende Schlüsselstrukturen wiederverwendet werden, so könnte man verlangen, dass Nutzer sich Programme wie GnuPG installieren und damit Nachrichten vom Server entschlüsseln. Um die Applikation weiterhin vom Web-Browser aus bedienen zu können, müssten dann allerdings noch Browsererweiterungen wie z.B. FireGPG vorausgesetzt werden, die eine JavaScript API bereitstellen, um auf GnuPG zuzugreifen.

4.4.7. Flexibilität

Wie schon in Abschnitt 3.1.3 diskutiert wurde, ist nicht mehr jede Auswahlfunktion möglich, weil nur noch die Summe aller Verfügbarkeitsvektoren bekannt ist.

4.5. Erweiterungen

In diesem Abschnitt werden mögliche Erweiterungen des Schemas vorgestellt, welche dazu dienen sollen, die Benutzbarkeit des Schemas zu verbessern.

4.5.1. Teilnehmer dynamisch hinzufügen/entfernen

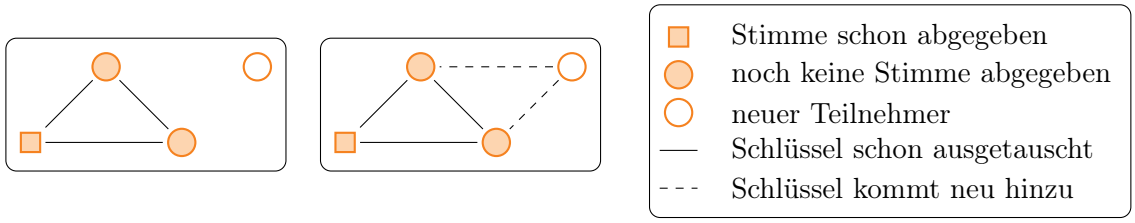
Die Menge der Teilnehmer wird am Anfang der Umfrage festgelegt und wurde bisher als fest angenommen. In der Praxis kann sich das als Problem erweisen, da es sein kann, dass nach Beginn der Umfrage Teilnehmer hinzugefügt werden sollen, die am Anfang nicht bedacht wurden. Zusätzlich kann es passieren, dass bestimmte Teilnehmer nicht mehr abstimmen sollen, die ursprünglich für das Abstimmen in der Umfrage konfiguriert wurden. Im Folgenden werden beide Fälle getrennt besprochen.

Teilnehmer dynamisch hinzufügen

Sollen Teilnehmer hinzugefügt werden nachdem eine Umfrage gestartet wurde, so kann es sein, dass einige Teilnehmer ihre Stimme zum Zeitpunkt des Hinzufügens schon abgegeben haben. Statt für seine Stimme mit allen anderen Teilnehmern Schlüssel auszutauschen[39],

[38] z. B. bei einer Realisierung mit SHA-1

[39] bzw. zu berechnen



(a) Bevor ein Teilnehmer hinzugefügt wurde

(b) Nachdem der Teilnehmer hinzugefügt wurde

Abbildung 4.10.: Schlüsselgraph vor und nach dem dynamischen Hinzufügen von Teilnehmern

tauscht der neu hinzugefügte Teilnehmer nur mit den Teilnehmern Schlüssel aus, die noch keine Stimme abgegeben haben. Für die Teilnehmer, die ihre Stimme schon abgegeben haben ändert sich nichts. Alle Teilnehmer, die ihre Stimme abgeben nachdem der neue Teilnehmer hinzugefügt wurde, beziehen den symmetrischen Schlüssel mit dem neuen Teilnehmer in ihre Berechnung mit ein. Abbildung 4.10 illustriert, wie sich der Schlüsselgraph durch das dynamische Hinzufügen von Teilnehmern ändert.

Teilnehmer dynamisch entfernen

Analog zum Hinzufügen von Teilnehmern kann es sinnvoll sein, Teilnehmer aus Umfragen zu entfernen, die ursprünglich mit abstimmen sollten. Normalerweise kann die Summe erst berechnet werden, wenn alle Teilnehmer ihre Stimme abgegeben haben. Fehlt die Stimme von nur einem Teilnehmer, so ist die gesamte Terminfindung blockiert.

Da alle anderen Teilnehmer die symmetrischen Schlüssel des zu entfernenden Teilnehmers kennen, kann dieser vor Beendigung der Umfrage ohne sein zutun entfernt werden. Jeder Teilnehmer u_i, der einen Schlüssel mit dem zu entfernenden Teilnehmer u benutzt hat, muss der Reduktion der Anonymitätsmenge zustimmen, indem er die $|\Delta|$ Schlüssel k^{δ}_{u,u_i} veröffentlicht, die er mit Teilnehmer u teilt.[40] Mit diesen Schlüsseln kann ein Ersatzvektor[41] $\vec{d}_u$ berechnet werden. Die Elemente des Vektors ergeben sich wie folgt:

$$d^{\delta}_{u} = 0 - \sum_{u_i \in U \setminus \{u\}} k^{\delta}_{u,u_i}. \tag{4.44}$$

Mit diesem Ersatzvektor kann das Protokoll wie bereits beschrieben durchgeführt werden.

Kann ein Angreifer die Nachrichten anderer Teilnehmer zurückhalten, so kann diese Erweiterung als Angriff benutzt werden. Dafür könnte Mallory Bobs Nachricht zurückhalten und alle anderen Teilnehmer davon überzeugen, Bob zu entfernen. Schafft sie das, so

[40] Der Schlüsselgraph muss hier nicht unbedingt vollständig sein, da es sein kann, dass ein Teilnehmer dynamisch hinzugefügt wurde.

[41] Der Ersatzvektor steht für den verschlüsselten Verfügbarkeitsvektor des zu entfernenden Teilnehmers u mit $\forall t \in T : \phi^t_u = 0$.

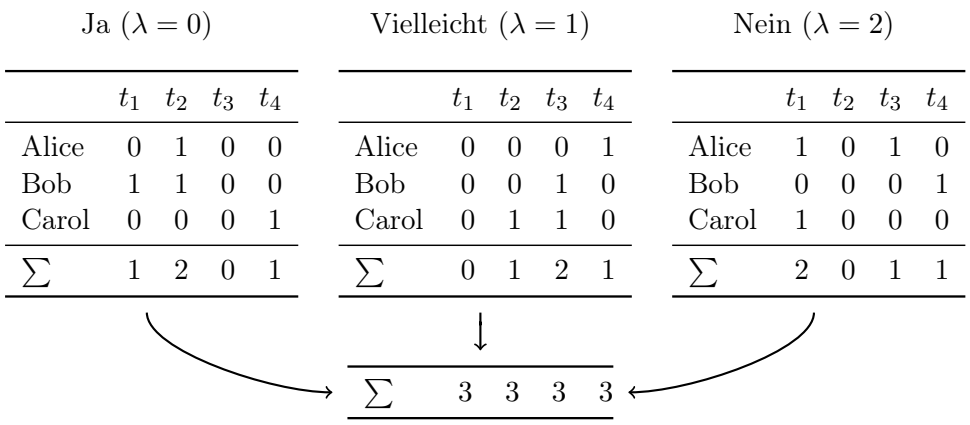


Abbildung 4.11.: Beispiel für eine Ja/Nein/Vielleicht-Umfrage

erhält sie Bobs Schlüssel und kann seinen gesamten Verfügbarkeitsvektor entschlüsseln. Damit kann sie Bob nicht nur daran hindern seine Präferenzen zu äußern, zusätzlich wird die Vertraulichkeit seiner Daten kompromittiert.

4.5.2. Präferenzwahl

In Terminabstimmungsapplikationen ist es häufig möglich, mehr als nur „Ja“ und „Nein“ zu einem Termin anzugeben. Oft hat man die Option einer Ja/Nein/Vielleicht-Umfrage[42], manchmal aber auch vier Optionen.[43] Mit dem hier eingeführten Schema sind prinzipiell nur Ja/Nein-Umfragen möglich.

Das Schema lässt sich jedoch relativ einfach auf ein η-Options Schema erweitern.

Beispiel 7 Angenommen Alice u_a ist sich unsicher, ob sie zum Zeitpunkt t_4 verfügbar ist. Im Schema, wie es bisher spezifiziert ist, hätte sie sich entscheiden müssen, ob sie für diesen Zeitpunkt stimmt ($\phi_u^{(t_3,0)} = 1, \phi_u^{(t_3,1)} = 0$) oder dagegen ($\phi_u^{(t_3,0)} = 0, \phi_u^{(t_3,1)} = 1$). Statt eine Verfügbarkeit $\phi_u^{(t,0)}$ und eine Prüfverfügbarkeit $\phi_u^{(t,1)}$ zu spezifizieren, können in einem abgewandelten Schema 3 Optionen spezifiziert werden, wobei folgende Semantik gegeben wird:

$\phi_u^{(t,0)} = 1$ ... Teilnehmer ist verfügbar,
$\phi_u^{(t,1)} = 1$ ... Teilnehmer ist vielleicht verfügbar und
$\phi_u^{(t,2)} = 1$ ... Teilnehmer ist nicht verfügbar.

Alice kann nun ausdrücken, dass sie evtl. verfügbar ist: $\phi_u^{(t_3,0)} = 0, \phi_u^{(t_3,1)} = 1, \phi_u^{(t_3,2)} = 0$. Abbildung 4.11 zeigt eine Illustration für eine Ja/Nein/Vielleicht-Umfrage. □

[42] z. B. bei Doodle [Näf]: „Ja-Nein-Wennsseinmuss-Umfrage“
[43] z. B. bei moreganize [FS]: Bevorzugt, Ok, Noch unsicher, Absage

Statt eine Verfügbarkeit $\phi_u^{(t,0)}$ und eine Prüfverfügbarkeit $\phi_u^{(t,1)}$ zu spezifizieren, werden η Optionen $\phi_u^{(t,\lambda)} \in \{0,1\}, \lambda \in \mathbb{Z}_\eta$ angegeben. Für alle Optionen zu einem Zeitpunkt gilt die Eigenschaft:

$$\sum_{\lambda \in \mathbb{Z}_\eta} \phi_u^{(t,\lambda)} = 1. \tag{4.45}$$

Dadurch lässt sich äquivalent zu Satz 5 ableiten:

Satz 11 *Die Summe der Elemente aller Ergebnisvektoren ist gleich der Anzahl der Teilnehmer $|U|$.*

$$\forall t \in T : \sum_{u \in U, i \in \mathbb{Z}_\ell, \lambda \in \mathbb{Z}_\eta} d_u^{(t,i,\lambda)} = |U|. \tag{4.46}$$

□

Durch Überprüfung von Gleichung (4.46) kann hier wiederum sichergestellt werden, dass kein (+2)-Angriff stattgefunden hat.

4.5.3. Stimmenupdate

Terminkalender ändern sich und mit ihnen die Verfügbarkeiten der Teilnehmer. Da es immer eine gewisse Zeit von der Abgabe der eigenen Stimme bis zur Veröffentlichung des Ergebnisses dauert, ist es ganz natürlich, dass bisweilen der Wunsch besteht, die abgegebene Stimme anzupassen bzw. eine neue verschlüsselte Verfügbarkeit anzugeben und die Alte ungültig zu machen.

Ein Stimmenupdate ist unter den gegebenen Anforderungen nicht ganz unkritisch. Das folgende Beispiel soll das verdeutlichen.

Beispiel 8 Seien $d_{u_a}^t, d_{u_b}^t, d_{u_c}^t$ drei verschlüsselte Verfügbarkeiten von den drei Teilnehmern u_a, u_b, u_c für einen Zeitpunkt t. Angenommen Teilnehmer u_b möchte seine Stimme zu einem Zeitpunkt neu abgeben $\left(\overline{d_{u_b}^t}\right)$.

Da wir von einem Schema ausgehen, welches die minimale Anzahl blockierender Protokollrunden hat, ist es ohne Nutzerinteraktion immer möglich, aus den drei Stimmen $d_{u_a}^t, d_{u_b}^t, d_{u_c}^t$ die Summe der Verfügbarkeiten zu berechnen. Genauso ist es allerdings möglich, aus den drei Stimmen $d_{u_a}^t, \overline{d_{u_b}^t}, d_{u_c}^t$ die Summe der Verfügbarkeiten zu berechnen. Damit kann über die Differenz beider Summen die Stimme von Teilnehmer u_b berechnet werden. Diese Eigenschaft ist unabhängig von dem hier vorgestellten Schema. □

Es ist klar, dass diese Differenz von jeder Entität berechnet werden kann, wenn ein Teilnehmer seine Stimme ändern will, nachdem die Umfrage abgeschlossen und damit das erste Ergebnis schon veröffentlicht ist.

Will ein Teilnehmer seine Stimme ändern bevor die Umfrage abgeschlossen ist, so sollte der Server die alte verschlüsselte Verfügbarkeit nicht veröffentlichen, um nicht jedem die Möglichkeit der Differenzbildung zu geben. Das verlangt jedoch Vertrauen in den Server, da dieser die Verfügbarkeit des Teilnehmers durch Differenzbildung berechnen kann.

Eine Möglichkeit, das Vertrauen in den Server zu reduzieren, ohne eine zusätzliche Protokollrunde einzuführen, wird im Folgenden vorgestellt. Es sei u_b der Teilnehmer,

welcher seine Stimme ändern will, bevor alle Teilnehmer ihre Stimme abgegeben haben. U_1 ist die Menge aller Teilnehmer außer u_b, die ihren verschlüsselten Verfügbarkeitsvektor schon zum Server übertragen haben, und U_2 ist die Menge aller Teilnehmer, die noch nicht gewählt haben ($u_b \notin U_1 \cup U_2, U_1 \setminus U_2 = \emptyset, \{u_b\} \cup U_1 \cup U_2 = U$). In dem Moment, in dem der Teilnehmer seine neue Stimme abgibt, tauscht er neue Schlüssel mit allen Teilnehmern aus, die ihre Stimme noch nicht abgegeben haben (U_2). Die daraus resultierenden Schlüssel werden mit $\overline{k^{\delta}_{u_b,u}}, u \in U_2$ bezeichnet. Anschließend berechnet der Teilnehmer seine neue Stimme wie folgt:

$$\overline{d^{\delta}_{u_b}} = \varphi^{\delta}_{u_b} + \sum_{u \in U_1} k^{\delta}_{u_b,u} + \sum_{u \in U_2} \overline{k^{\delta}_{u_b,u}}. \tag{4.47}$$

Ein Teilnehmer $u_c \in U_2$, der seine Stimme später abgibt berechnet seine Stimme nach folgender Formel:

$$d^{\delta}_{u_c} = \varphi^{\delta}_{u_c} + \overline{k^{\delta}_{u_c,u_b}} + \sum_{u \in U \setminus \{u_b\}} k^{\delta}_{u_c,u}. \tag{4.48}$$

Ist der Server vertrauenswürdig und veröffentlicht die alte Stimme nicht, so ändert sich nichts an der Sicherheit des Verfahrens. Kooperieren Angreifer mit dem Server und können somit Kenntnis über die alte Stimme erlangen, so ist das Verfahren noch so sicher, als wären alle Schlüssel $k^{\delta}_{u_b,u}, u \in U_1$ veröffentlicht (daher: das Verfahren ist sicher, solange ein Teilnehmer aus U_2 vertrauenswürdig ist).

Wird statt Schlüsselaustausch das Schlüsselvereinbarungsprotokoll verwendet, so wird ein zusätzlicher Wert benötigt, der in die Schlüsselerzeugungsfunktion KDF mit einbezogen wird (vgl. Gleichung (4.40) auf Seite 60). Dies könnte eine Zufallszahl, Counter oder Zeitstempel sein, den derjenige Teilnehmer u festlegt, der von zwei Teilnehmern u, u' zuerst seine Stimme abgibt. Anschließend hat dieser Teilnehmer u die Möglichkeit die Zufallszahl oder den Counter zu erneuern, so lange der andere Teilnehmer u' noch keine Stimme abgegeben hat. Eine Erneuerung des Wertes bewirkt, dass ein anderer symmetrischer Schlüssel für beide Teilnehmer generiert wird.

Sei $\mathsf{ts}_{u,u'}$ der Zeitstempel, ab dem der Schlüssel zweier Teilnehmer u und u' nicht mehr verändert wird. Zu dem Zeitpunkt, an dem u seine Stimme abgeben möchte, berechnet sich der Zeitstempel $\mathsf{ts}_{u,u'}$ mit Teilnehmer u' wie folgt:

$$\mathsf{ts}_{u,u'} = \begin{cases} \text{jetzt} & \text{wenn } u' \text{ noch nicht abgestimmt hat,} \\ \mathsf{ts}_{u',u} & \text{sonst.} \end{cases} \tag{4.49}$$

Für jede DC-Netz-Runde $\delta \in \Delta$ wird ein Schlüssel $k^{\delta}_{u,u'}$ wie folgt berechnet:

$$k^{\delta}_{u,u'} := \mathrm{KDF}\left(\delta, \mathsf{ts}_{u,u'}, \mathsf{dh}_{u,u'}\right). \tag{4.50}$$

5. Implementierung

In diesem Kapitel wird die während der Arbeit entstandene Implementierung vorgestellt. Dabei wird erst auf den Teil der Implementierung eingegangen, der ohne clientseitige JavaScript-Berechnungen auskommt (Abschnitt 5.1). Die dort gezeigten Implementierungen sind Umsetzungen von Verfahren aus Tabelle 3.1 auf Seite 21. In Abschnitt 5.2 werden die Implementierungen vorgestellt, die clientseitige Berechnungen mit JavaScript benötigen, welches Verfahren aus Tabelle 3.2 auf Seite 27 sind.

Die Implementierung ist sowohl in der Benutzung als auch im Sourcecode frei[1] verfügbar. Sie kann unter `https://dudle.inf.tu-dresden.de` benutzt und heruntergeladen werden.

Abschnitt 5.3 geht gesondert auf Probleme ein, die sich bei der Implementierung von Kryptographie mit JavaScript ergeben.

5.1. Verfahren mit vertrauenswürdigem Serveradministrator

5.1.1. YATA – Yet Another Terminabstimmungsapplikation

Die Basis der Implementierung bildet eine Terminabstimmungsapplikation, wie sie schon mehrfach im Web verfügbar ist. Leider fand sich zu Beginn der Arbeit keine quelloffene Implementierung unter den verfügbaren Applikationen. Heute gibt es weitere Implementierungen [Sol; Dij; Bor; Avi], welche zeitgleich oder nach der vorliegenden Implementierung gestartet wurden.

Um eine Terminabstimmung durchzuführen, werden die Phasen so durchlaufen, wie es in Abschnitt 2.2.1 erklärt wurde. Zuerst wird eine Umfrage erstellt, wofür ein Titel für die Umfrage festgelegt werden muss (Abbildung 5.1 auf der nächsten Seite). Optional kann eine URL gewählt werden, über welche die Umfrage erreichbar ist. Wählt man für die URL eine Zeichenkette mit hoher Entropie und behandelt diese vertraulich, so kann die URL gleichzeitig als Passwort dienen, welches zum Abstimmen (Schreibzugriff) und zum Anschauen des Ergebnisses (Lesezugriff) nötig ist (Schema 2 in Tabelle 3.1 auf Seite 21). Wird eine einfach zu merkende URL gewählt, so kann man auf die Zugriffskontrolle verzichten (Schema 0).

Im nächsten Schritt bei der Erstellung der Umfrage müssen die Zeitpunkte festgelegt werden, für welche die Umfrage durchgeführt wird. Ein Screenshot dieses Interfaces ist in Abbildung 5.2 auf der nächsten Seite zu sehen.

Wurde die notwendige Konfiguration durchgeführt, so kann die Umfrage-URL an alle Teilnehmer verteilt werden. Jeder Teilnehmer äußert seine Verfügbarkeiten zu den zur Abstimmung stehenden Terminen. Haben genügend Teilnehmer abgestimmt, kann ein Termin festgelegt werden. Das Interface, mit welchem sowohl die Verfügbarkeiten spezifiziert

[1] Frei im Sinne der Definition der Free Software Foundation, Lizenz: AGPLv3.

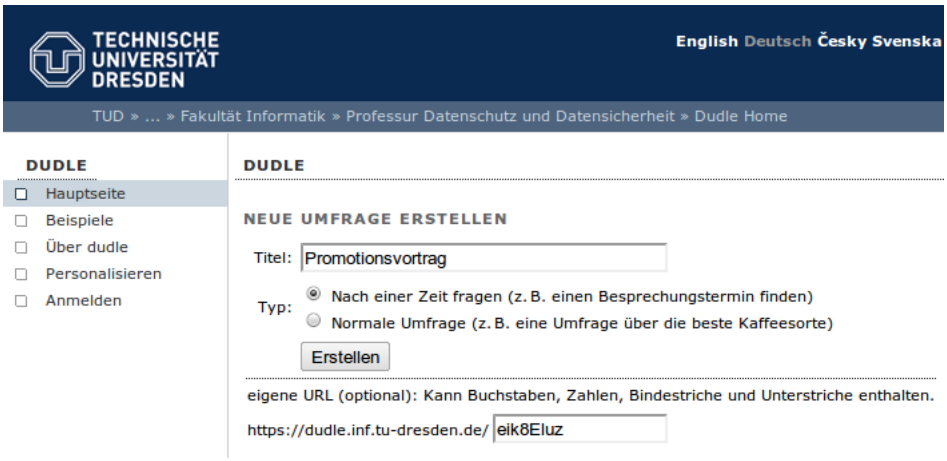


Abbildung 5.1.: Interface zum Erstellen einer Dudle-Umfrage. Wird für die optionale URL eine geheime Zeichenkette mit hoher Entropie gewählt, so kann die URL gleichzeitig als Passwort angesehen werden.

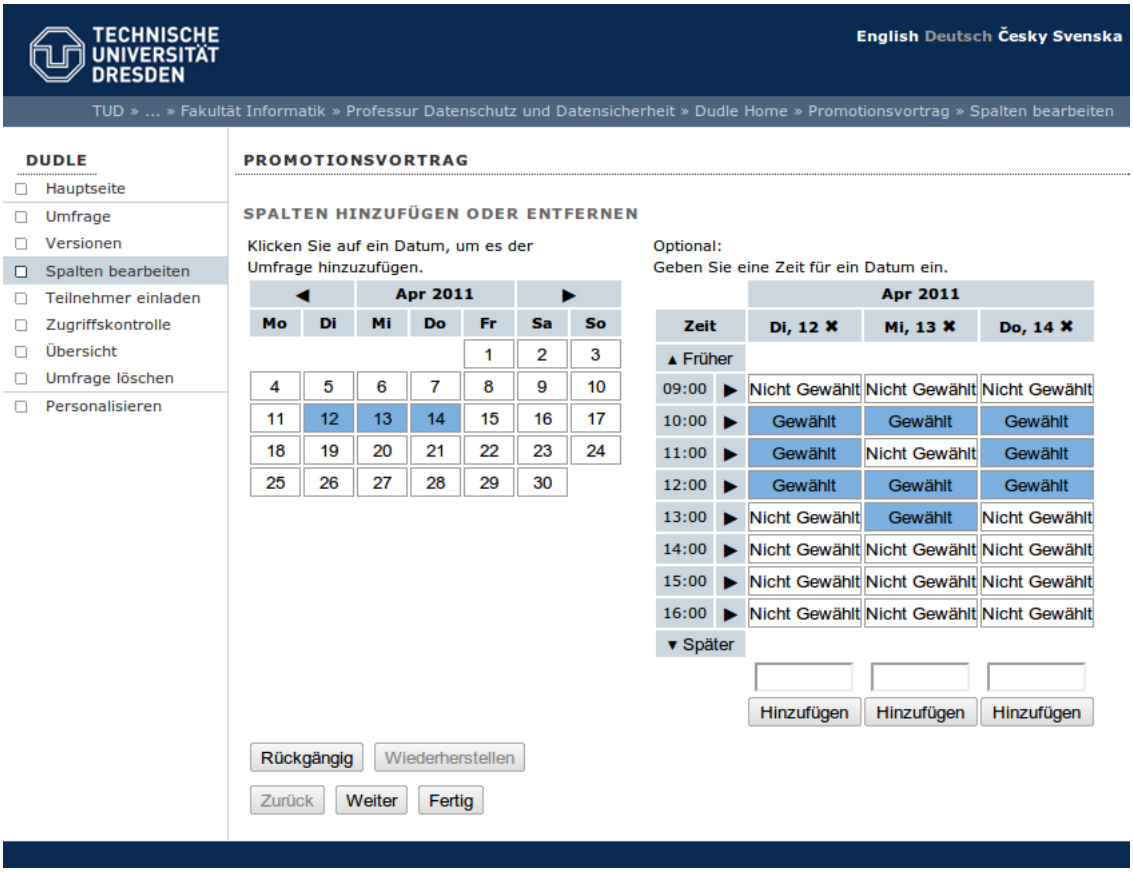


Abbildung 5.2.: Konfigurieren der Zeitpunkte einer Terminabstimmung

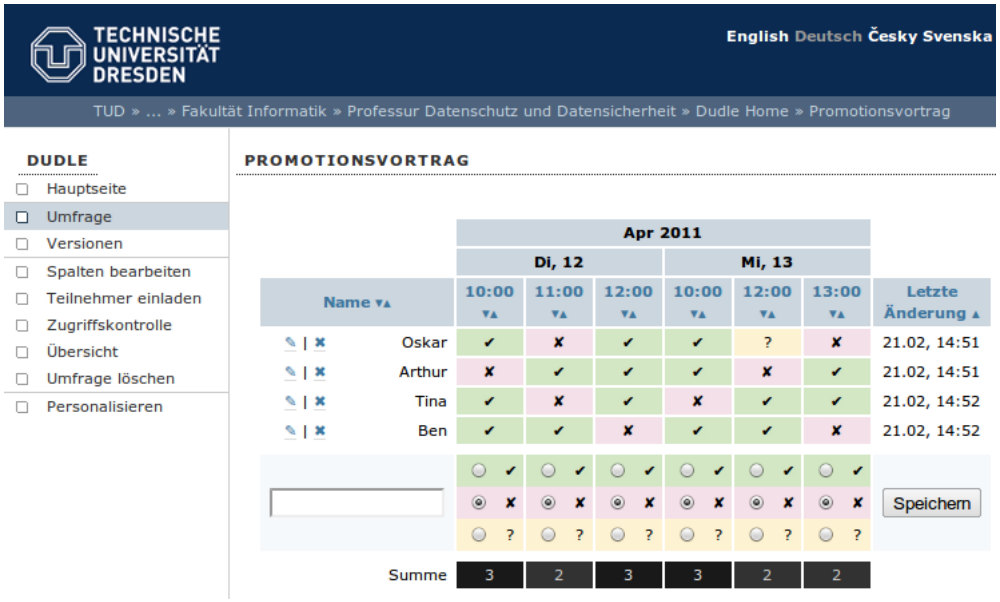


Abbildung 5.3.: Interface einer Terminabstimmung

werden als auch die aggregierten Werte eingesehen werden können, ist in Abbildung 5.3 zu sehen.

Wie vorher schon bemerkt wurde, gibt es bereits einige Implementierungen, die sich mit dem Problem der Terminabstimmung befassen. Damit auch die Implementierung ohne besonderen Zugriffsschutz einen Mehrwert bzgl. mehrseitiger Sicherheit bietet, wurde in dieser Implementierung auf die Benutzung von JavaScript verzichtet. Diese Restriktion bedeutet zwar Mehraufwand in der Implementierung sowie ein trägeres Interface, lohnt sich jedoch aus Sicht des Datenschutzes. So ist es ohne JavaScript schwerer, einen Teilnehmer gegen seinen Willen zu identifizieren bzw. zu reidentifizieren. JavaScript bietet zahlreiche Möglichkeiten, Informationen über den Rechner des Benutzers zu sammeln z. B. Bildschirmauflösung, installierte Schriften, besuchte Seiten, Browserplugins. . . . Kombiniert man diese Informationen mit denen, die durch die Entropie des Verfügbarkeitsvektors gegeben sind, wird eine Identifikation unter Umständen sehr leicht. Zahlreiche Demonstrationen, wie JavaScript missbraucht werden kann, finden sich bereits online [Ele; Eck10; Kam10].

Am stärksten macht sich die Einschränkung des Verzichts auf JavaScript bei der Umfrageerstellung bemerkbar, wenn die Zeitpunkte konfiguriert werden müssen (vgl. Abbildung 5.2 auf der vorherigen Seite). Hier muss – aus Sicht des Programmierers – der Zustand des Monats- sowie Uhrzeitnavigators zwischen jedem Klick am Server gehalten werden, damit der Zustand nach Auswahl eines neuen Datums so dargestellt werden kann, wie er vorher gewesen ist. Aus Sicht des Benutzers findet hier zwischen jedem Klick ein Server-round-trip statt, weshalb die Benutzung bei einer Internetverbindung mit hoher Latenz unangenehm träge werden kann.

Promotionsvortrag

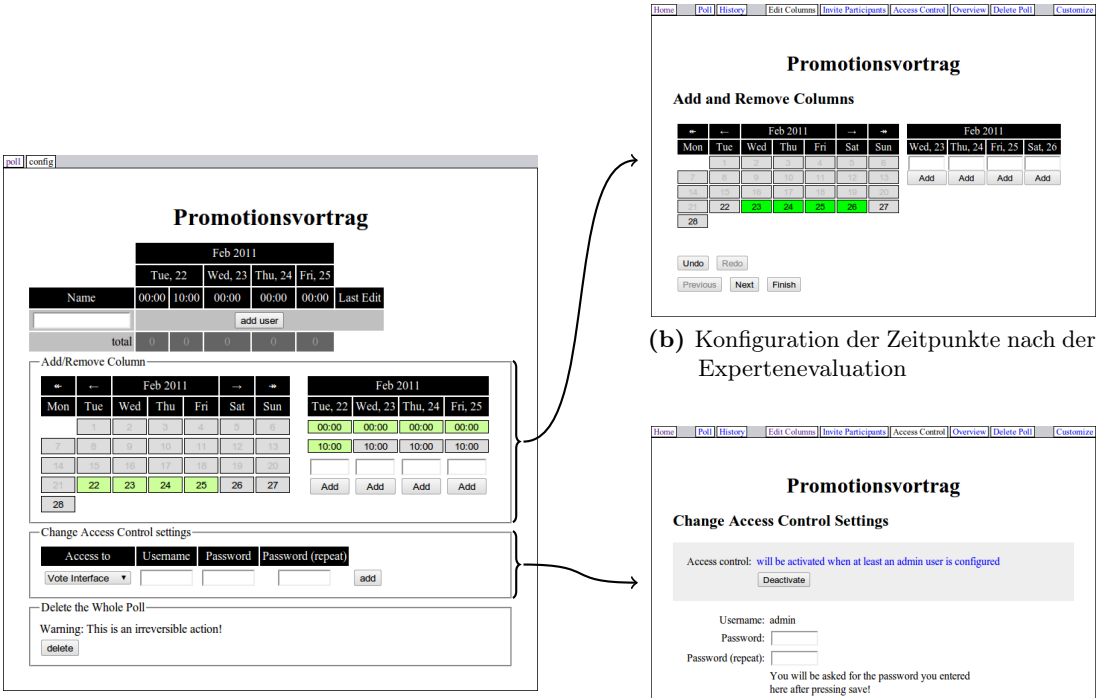
		Apr 2011						
		Di, 12			Mi, 13			
Name ▾▴		10:00 ▾▴	11:00 ▾▴	12:00 ▾▴	10:00 ▾▴	12:00 ▾▴	13:00 ▾▴	Letzte Änderung ▴
✎ | ✖	Oskar	✔	✘	✔	✔	?	✘	21.02, 14:51
✎ | ✖	Arthur	✘	✔	✔	✔	✘	✔	21.02, 14:51
✎ | ✖	Tina	✔	✘	✔	✘	✔	✔	21.02, 14:52
✎ | ✖	Ben	✔	✔	✘	✔	✔	✘	21.02, 14:52
[_____]		( ) ✔ (*) ✘ ( ) ?	( ) ✔ (*) ✘ ( ) ?	( ) ✔ (*) ✘ ( ) ?	( ) ✔ (*) ✘ ( ) ?	( ) ✔ (*) ✘ ( ) ?	( ) ✔ (*) ✘ ( ) ?	[Speichern]
Summe		3	2	3	3	2	2	

Abbildung 5.4.: Abstimmungsinterface in einem Textbrowser (hier: w3m)

Ein weiterer Vorteil, welcher sich durch den Verzicht von JavaScript ergibt, ist, dass sehr wenige Anforderungen an den Browser des Nutzers gestellt werden. Ein Beispiel ist in Abbildung 5.4 gegeben, wo das gleiche Interface von Abbildung 5.3 zu sehen ist, welches allerdings von dem Textbrowser w3m gerendert wurde.

Da diese Implementierung von Grund auf neu begonnen wurde, fand eine Evaluation der Oberfläche statt, welche in mehreren Iterationen zur Verbesserung der Benutzbarkeit geführt hat. Die erste Verbesserung fand in Form einer Expertenevaluation statt. Hierfür wurde das Interface von Mitarbeitern des *Center for Usability Research & Engineering (CURE), Wien* bewertet und in mehreren Iterationen verändert. Die Bewertungen sowie Iterationen fanden innerhalb einer Programmierwoche vor Ort statt, in der tägliches Feedback stattfand und ausschließlich am Interface gearbeitet wurde. Die Hauptveränderung innerhalb dieser Evaluation war die Umstellung der Konfiguration und Umfrageerstellung in eine durch einen Dialog-Assistent geführte Erstellung. Vorher wurden alle Einstellungen zu einer Umfrage in einer einzelnen Seite durchgeführt. Des Weiteren wurde eine Undo/Redo-Funktionalität hinzugefügt, die es ermöglicht, Fehler in der Konfiguration zu beheben. Zusätzlich zu dieser Funktionalität wurde darauf geachtet, dass der Zurück-Knopf des Browsers den Zustand der Konfiguration so zurücksetzt, dass keine unerwarteten Veränderungen auftreten. Abbildung 5.5 zeigt drei Screenshots der Umgestaltung. In Abbildung 5.5(a) ist die Konfiguration zu sehen, wie sie vor der Expertenevaluation durchgeführt wurde, Abbildung 5.5(b) und Abbildung 5.5(c) zeigen die Dialoge nach der Evaluation.

Zwei Monate nach Abschluss der Expertenevaluation wurde eine weitere Evaluation durchgeführt. Diese fand an der TU Dresden innerhalb der Fakultät Informatik statt. Es wurde fünf Probanden eine Aufgabe gestellt, welche sie mit der Think-Aloud-Methode [Lew82] lösen sollten. Um den Test mit möglichst wenig Informatik-affinen



(a) Vor der Expertenevaluation: Ein Konfigurationsdialog mit allen Optionen

(b) Konfiguration der Zeitpunkte nach der Expertenevaluation

(c) Zugriffskontrolleinstellungen nach der Expertenevaluation

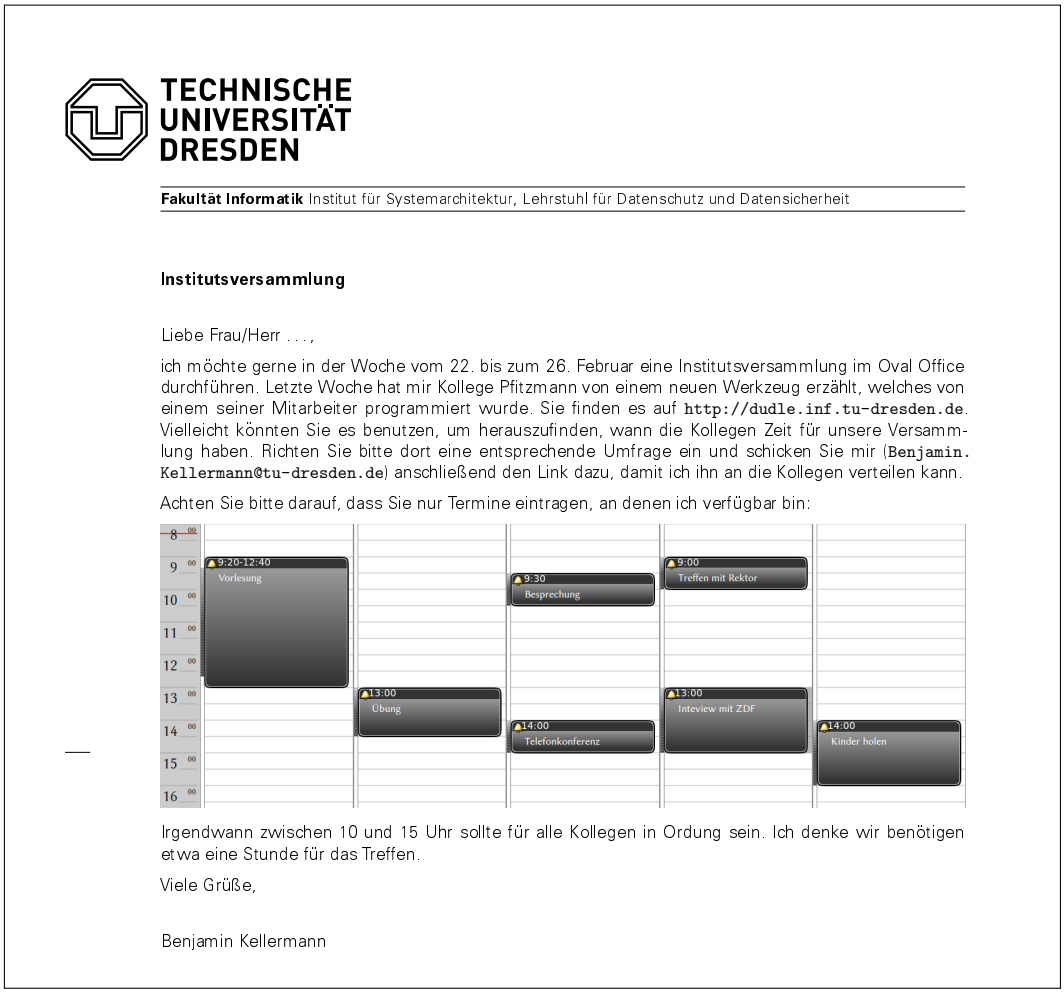
Abbildung 5.5.: Einfluss der Expertenevaluation auf das Konfigurationsinterface. Die Konfiguration wurde von einem monolithischen Dialog in eine geführte Konfiguration (Dialog-Assistent) geändert. Außerdem wurde eine Undo/Redo-Funktionalität eingebaut (zu sehen in Abbildung (b) unter dem Datumsnavigator).

Probanden durchzuführen, wurde die Aufgabe an fünf Sekretärinnen von fünf unterschiedlichen Professuren gestellt. Die Aufgabe, findet sich in Abbildung 5.6.

Der erste Test wurde nach zwei Probanden abgebrochen, da das Interface komplett in englischer Sprache verfasst war und dadurch große Verständnisprobleme bestanden. Die Sprache des Benutzerinterfaces wurde daraufhin mittels gettext[2] lokalisiert und der Test wurde nochmal mit allen Probanden durchgeführt. Als Ergebnis der Evaluation wurde die Freitexteingabe für einzelne Zeiten durch eine Liste mit Zeiten als Vorschlag ersetzt. Zusätzlich wurden kurze Handlungsanweisungen zwischen den Überschriften und den Eingabemasken eingefügt. Das Ergebnis entspricht der Applikation in ihrem aktuellen Zustand und war schon in Abbildung 5.2 auf Seite 70 zu sehen.[3]

[2] `http://www.gnu.org/software/gettext`, zu letzt abgerufen am 18. März 2011

[3] Hier ist zusätzlich noch zu erkennen, dass die CSS-Stylesheets verglichen mit Abbildung 5.5 verändert wurden.



TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Lehrstuhl für Datenschutz und Datensicherheit

Institutsversammlung

Liebe Frau/Herr . . . ,

ich möchte gerne in der Woche vom 22. bis zum 26. Februar eine Institutsversammlung im Oval Office durchführen. Letzte Woche hat mir Kollege Pfitzmann von einem neuen Werkzeug erzählt, welches von einem seiner Mitarbeiter programmiert wurde. Sie finden es auf `http://dudle.inf.tu-dresden.de`. Vielleicht könnten Sie es benutzen, um herauszufinden, wann die Kollegen Zeit für unsere Versammlung haben. Richten Sie bitte dort eine entsprechende Umfrage ein und schicken Sie mir (`Benjamin.Kellermann@tu-dresden.de`) anschließend den Link dazu, damit ich ihn an die Kollegen verteilen kann.

Achten Sie bitte darauf, dass Sie nur Termine eintragen, an denen ich verfügbar bin:

Irgendwann zwischen 10 und 15 Uhr sollte für alle Kollegen in Ordung sein. Ich denke wir benötigen etwa eine Stunde für das Treffen.

Viele Grüße,

Benjamin Kellermann

Abbildung 5.6.: Aufgabe, welche fünf Sekretärinnen der Fakultät Informatik vorgelegt wurde. Diese Aufgabe sollte mit der Think-Aloud-Methode gelöst werden.

Um eine Erweiterbarkeit der Applikation zu gewährleisten, wurde ein Mechanismus implementiert, der es erlaubt andere CSS-Stylesheets, sowie Plug-Ins zu schreiben.

Abbildung 5.7 zeigt eine Statistik über die Webseitenzugriffe sowie die Anzahl der durchgeführten Umfragen[4] im Zeitraum Dezember 2009 bis August 2011. Anhand dieser Statistik ist zu sehen, dass die Implementierung trotz etablierter Lösungen eine zunehmende Akzeptanz bei Benutzern findet.

[4] Von den ursprünglich 2064 Umfragen wurden die gelöscht, die weniger als 3 Teilnehmer ($|U| < 3$) oder weniger als 3 Zeitpunkte ($|T| < 3$) enthielten. Außerdem wurden Umfragen entfernt, in denen insgesamt weniger als 10 Änderungen durchgeführt wurden. Die verbleibenden 666 Umfragen, werden als „ernsthafte“ Umfragen angesehen.

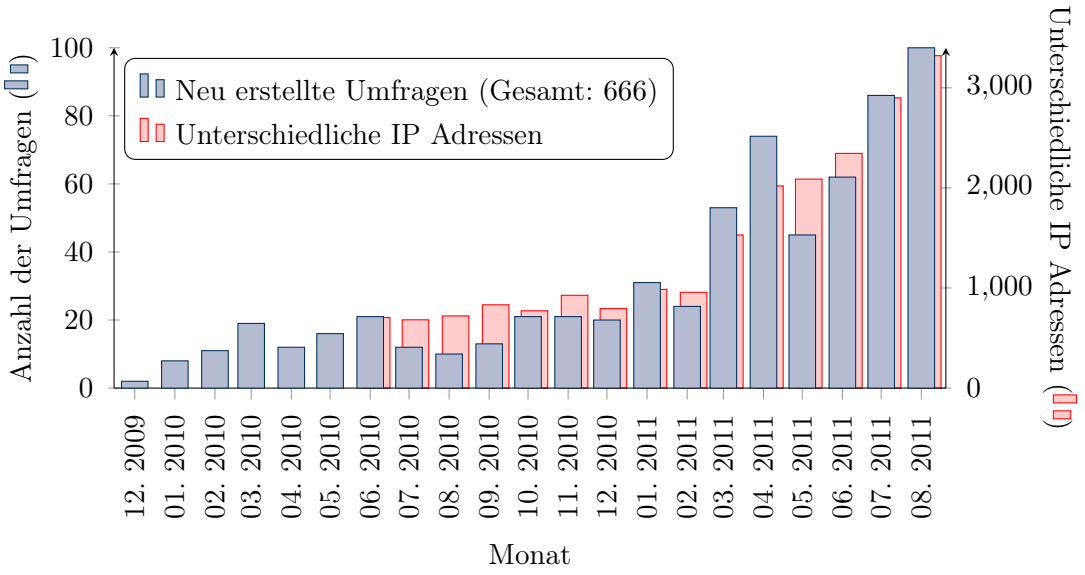


Abbildung 5.7.: Anzahl der Webseitenzugriffe sowie durchgeführten Umfragen von Dezember 2009 bis August 2011

Tabelle 5.1.: Verschiedene Möglichkeiten, Integritätsschutz gegenüber den anderen Teilnehmern der Umfrage zu erreichen. Vollständiges Vertrauen genießt nur der Abstimmungsserver, im ersten Fall zusätzlich der Netzwerkbetreiber.

	Universelle Überprüfbarkeit durch	Passwort von	Registrierung
4a	E-Mail (kein Netzwerkbetreiberschutz)	Server	keine
4b	E-Mail, PGP-Fingerprint	Server	PGP-Schlüsselserver
3a	E-Mail, Serverbeglaubigung	Teilnehmer	Abstimmungsserver
3b	Benutzernamen, Serverbeglaubigung	Teilnehmer	Abstimmungsserver

5.1.2. Schutz gegenüber dem Netzwerkbetreiber

Die erste Form, Integritätsschutz gegenüber den anderen Teilnehmern der Umfrage zu erreichen war, ein unterschiedliches Passwort an jeden Teilnehmer zu verschicken (vgl. Schema 3 und 4 in Tabelle 3.1 auf Seite 21). Während diese Form in Grundzügen in bestehenden Applikationen implementiert ist, so ist sie jedoch aus Sicht der mehrseitigen Sicherheit nicht vollständig umgesetzt. Insbesondere bieten die Anbieter teilweise keine Verschlüsselung der Kommunikation zum Umfragenserver. Einen vertraulichen E-Mail-Versand ermöglicht keine Applikation.

Um der Anforderung des dritten und vierten Schemas der Tabelle 3.1 gerecht zu werden wurde diese Art der Zugriffskontrolle in das System integriert. Es gibt dabei 4 Varianten sich zu authentifizieren, zwei implementieren Schema 3 und zwei Schema 4. Tabelle 5.1 zeigt eine Übersicht über die vier Implementierungen.

Während des Anlegens einer Umfrage ist es möglich, Teilnehmer in Form von E-Mail-Adressen einzuladen. Das Abgeben der Stimme für diese Teilnehmer ist passwortgeschützt. Jede E-Mail-Adresse die eingetragen wurde bekommt ein Passwort in Form eines Links[5] per E-Mail geschickt. Diese Implementierung entspricht Schema 4.

Um Integritätsschutz gegenüber dem Netzwerkbetreiber zu erreichen, versucht der Server einen PGP-Schlüssel von einem Schlüsselserver herunterzuladen. Der Initiator der Umfrage bestätigt hierbei, dass der zum Schlüssel zugehörige Fingerprint zu dem jeweiligen Teilnehmer gehört. Zusätzlich ist es jedem Teilnehmer möglich, den Fingerprint der anderen Teilnehmer im Abstimmungsinterface zu vergleichen. Diese Verifikation ist in Form des Tooltipps, welcher sich an jedem Namen aufrufen lässt sichtbar. In Tabelle 5.1 ist diese Implementierung mit 4b gekennzeichnet.

Besitzt ein Teilnehmer keinen PGP-Schlüssel, so wird die E-Mail unverschlüsselt verschickt. In diesem Fall muss dem Netzwerkbetreiber bzgl. Integrität und Vertraulichkeit vertraut werden[6] (4a in Tabelle 5.1).

Möchte ein Teilnehmer dem Netzwerkbetreiber nicht Vertrauen und trotzdem keinen PGP-Schlüssel benutzen, so hat er die Möglichkeit, sich vorher am Server zu registrieren. Diese Variante stellt Schema 3 dar und wurde auf zwei unterschiedliche Arten implementiert: mit Benutzung einer E-Mail-Adresse (3a) und ohne (3b).

In Variante 3b registriert sich ein Benutzer am Umfragenserver und legt dabei einen eindeutigen Benutzernamen fest. Da die Passwörter SSL-verschlüsselt übertragen werden,[7] braucht dem Netzwerkbetreiber an dieser Stelle nicht vertraut werden. Die anderen Teilnehmer müssen in diesem Fall wissen, unter welchem Benutzernamen sich dieser Teilnehmer registriert hat. Bei der Abstimmung können sie überprüfen, ob der Benutzername mit dem des ihnen als registriert bekannten übereinstimmt.

Variante 3a benutzt die E-Mail-Adresse als Benutzernamen. Die E-Mail-Adresse wird hierbei vom Umfragenserver auf Korrektheit validiert:

1. Der Benutzer bekommt an diese E-Mail-Adresse eine URL zugeschickt, welche ein Einmalpasswort enthält.

2. Nach Aufrufen der URL legt der Benutzer sein Passwort fest.

Die Benutzung einer E-Mail-Adresse hat den Vorteil, dass sich die anderen Umfragenteilnehmer nicht den Benutzernamen merken müssen, sondern anhand der E-Mail-Adresse den Benutzer erkennen können. Um einen wirksamen Integritätsschutz gegenüber dem Netzwerkbetreiber zu erreichen, muss den anderen Teilnehmern allerdings bekannt sein,

[5] z. B. `https://dudle.inf.tu-dresden.de/umfrage/?password=aeB1IeKa`

[6] Da die E-Mail meist mittels einer Verbindungsverschlüsselung zum E-Mail-Provider ausgeliefert wird, und diese wiederum mittels einer Verbindungsverschlüsselung abgerufen werden kann, kann es sein, dass hier nur der E-Mail-Provider statt des ganzen Netzwerkbetreibers als Angreifer in Frage kommt.

[7] Um an dieser Stelle eine authentische Verbindung zum Webserver herzustellen, wurde über den DFN-Verein ein SSL-Zertifikat beantragt, welches eine Zertifikatskette bis zu einem Zertifikat der Deutschen Telekom besitzt. Der Schlüssel zum Überprüfen des Telekom Root Zertifikats wird mit allen gängigen Browsern mitgeliefert. Durch Auslieferung der Zertifikatskette mittels des Webservers kann überprüft werden, dass der Schlüssel zum Verschlüsseln der SSL-Verbindung zu dem DNS-Eintrag des Servers gehört.

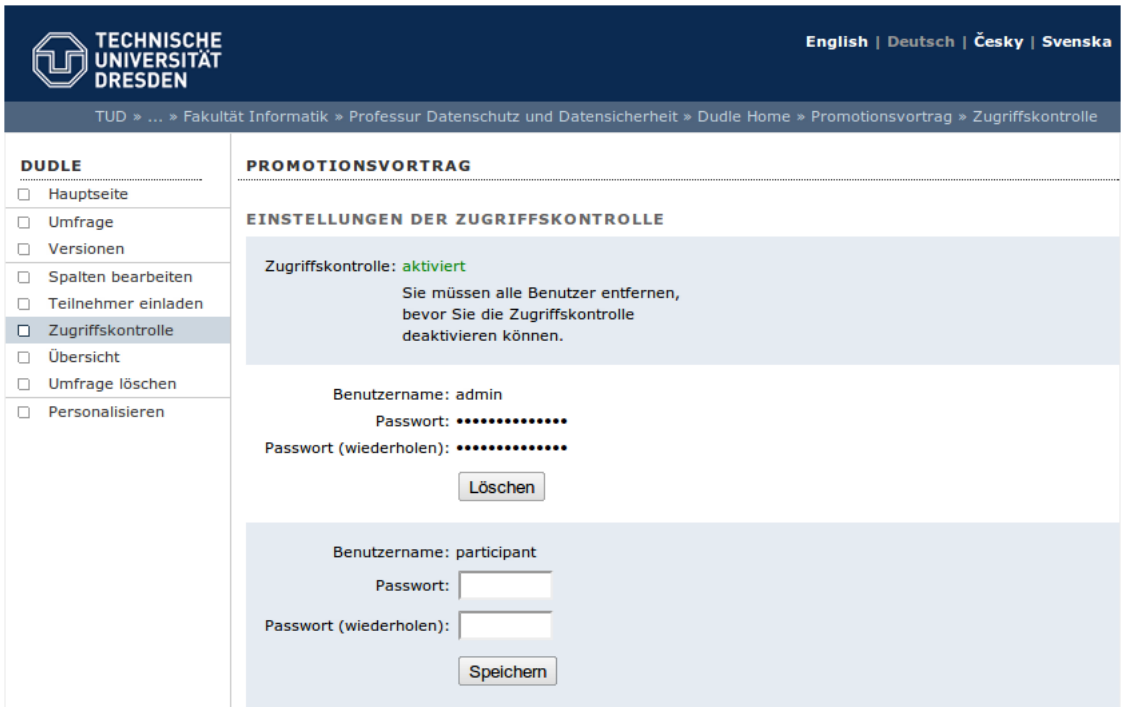


Abbildung 5.8.: Interface zur Konfiguration eines Abstimmungspassworts

dass dieser Nutzer unter seiner E-Mail-Adresse registriert ist, da der Netzwerkbetreiber andernfalls die Möglichkeit hat, sich im Moment der Umfrageerstellung zu registrieren und die E-Mail zurückzuhalten. [8]

Sobald mindestens ein Teilnehmer die URL zur Umfrage unverschlüsselt zugeschickt bekommt, muss dem Netzwerkbetreiber bzgl. Vertraulichkeit vertraut werden. Dieses Vertrauen kann über ein zusätzliches Passwort umgangen werden, welches für jede Umfrage konfiguriert werden kann. Abbildung 5.8 zeigt diese Eingabemaske. Im Unterschied zum Teilnehmerpasswort sowie der URL der Umfrage wird das hier konfigurierte Passwort nicht automatisch vom System an die Teilnehmer verschickt. Die vertrauliche Übertragung obliegt daher dem Initiator der Umfrage.

5.2. Verfahren ohne vertrauenswürdigem Serveradministrator

Die Implementierungen, welche in diesem Abschnitt vorgestellt werden, setzen im Gegensatz zu den Implementierungen aus Abschnitt 5.1 alle voraus, dass kryptographische Berechnungen in der Applikation auf Clientseite durchgeführt werden können. Für diese Berechnungen benutzen die Implementierungen JavaScript. Auf einzelne Probleme

[8] Wie schon in Fußnote 6 auf der vorherigen Seite angemerkt, kann es sein, dass hier nur der E-Mail-Provider selbst als Angreifer in Frage kommt.

in der Benutzung von JavaScript für Kryptographie in Web 2.0-Applikationen wird in Abschnitt 5.3 gesondert eingegangen.

5.2.1. Symmetrische Verschlüsselung, symmetrische Authentifikation

Um das notwendige Vertrauen in den Serveradministrator zu reduzieren, wurde in Abschnitt 3.2.1 vorgeschlagen, das Abstimmungspasswort als Schlüssel für eine symmetrische Verschlüsselung sowie Authentikation zu benutzen (Schema 2′). Dieses Schema wurde in Form eines Plug-Ins für das bestehende System implementiert. Die Implementierung fand im Rahmen eines studentischen Praktikums von Ole Rixmann statt.

Das Plug-In modifiziert die Passwortabfrage, wie sie in Abbildung 5.8 zu sehen war. Anstatt das Passwort an den Server zu senden, wird es benutzt um alle Daten mit AES in JavaScript zu verschlüsseln und zu authentifizieren. Als Schlüsselerzeugungsfunktion wird hierbei PBKDF2 aus dem RFC2898 [RFC2898] benutzt.[9] Dieser Schlüssel wird anschließend für AES256-CCM benutzt, wodurch Verschlüsselung und Authentifikation zugleich erreicht wird. Für die Implementierung wurde die Stanford JavaScript Crypto Library [SHB09] verwendet.

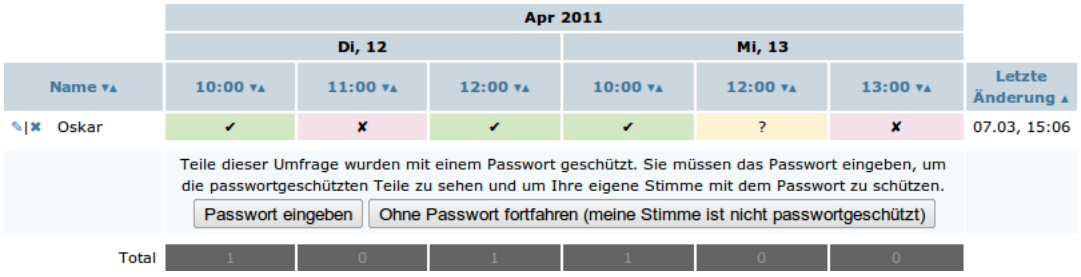
Abbildung 5.9 auf der nächsten Seite zeigt Screenshots der Implementierung. Zu sehen ist die gleiche Abstimmung wie sie schon in Abbildung 5.3 zu sehen war. Im Unterschied zu dem Interface dort werden die Stimmen, welche verschlüsselt abgegeben wurden, erst nach Passworteingabe entschlüsselt und angezeigt. Anhand des Schloss-Icons neben den Namen kann man erkennen, welche Teilnehmer ihre Stimmen symmetrisch verschlüsselt sowie authentifiziert abgegeben haben. Das Siegel deutet auf eine digitale Signatur hin und ist Bestandteil der Implementierung von Abschnitt 5.2.2.

Um die Passworteingabe zu erleichtern, wird dem Nutzer eine Möglichkeit gegeben, eine spezielle URL aufzurufen. Hierbei wird der Fragment-Teil einer URL benutzt, welcher normalerweise für Document-Anchor benutzt wird. Ein Document-Anchor ist ein Verweis auf ein bestimmtes Element in einem HTML Dokument. Er ist normalerweise dafür gedacht, nach einem Seitenaufruf zu diesem Element im Dokument zu springen. Der Document-Anchor wird in einer URL im Fragment-Teil angegeben. Abbildung 5.10 auf der nächsten Seite zeigt ein Beispiel für eine URL mit Fragment-Teil. Der Host-Teil sowie der Path-Teil von dieser URL werden bei einem GET-Request zum Server geschickt. Alles was nach dem Doppelkreuz (`#`) steht ist der Fragment-Teil der URL. Der Fragment-Teil wird *nicht* zusammen mit dem GET-Request zum Server übertragen. Daher kann er als Angabe für sensible Daten benutzt werden.

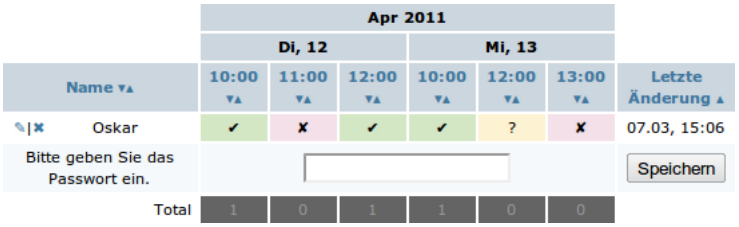
Wird während der Umfragenkonfiguration ein Passwort für die symmetrische Verschlüsselung angegeben (Abbildung 5.8 auf der vorherigen Seite), so wird dieses für die Session im Web Storage gespeichert.[10] Bei Beendigung der Konfiguration wird dem Umfrageninitiator eine Zusammenfassungsseite angezeigt. Diese zeigt die Umfrage-URL an, welche an die Teilnehmer verteilt werden soll und beinhaltet zusätzlich einen `mailto:`-Link, welcher

[9] Mit dem Passwort und einer Zufallszahl (salt [MT79]) wird ein key stretching [Kel+98] um 10 Bit durchgeführt.

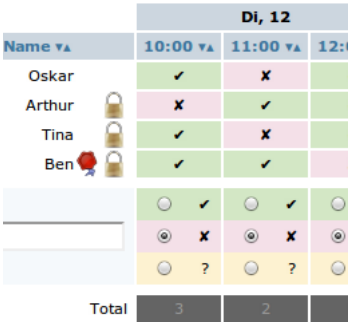
[10] Auf die Schlüsselspeicherung wird in Abschnitt 5.3.1 näher eingegangen.



(a) Frage, ob mit oder ohne Passwort fortgefahren werden soll



(b) Aufforderung zur Passworteingabe. Ohne Eingabe des Passworts können nur die unverschlüsselt abgegebenen Stimmen angezeigt werden.



(c) Nach Passworteingabe verhält sich das Interface genauso wie in einer unverschlüsselten Abstimmung. Anhand der Schlosssymbole kann ein Benutzer sehen, wessen Stimme symmetrisch verschlüsselt/authentifiziert abgegeben wurde. Das Siegel zeigt eine digitale Signatur an.

Abbildung 5.9.: Interface der Abstimmung, wenn symmetrische Verschlüsselung und/oder digitale Signaturen verwendet werden

`https://dudle.inf.tu-dresden.de/umfrage/#passwd=rvoj4pej`

Scheme — Host — Path — Fragment

```
1 $ telnet -z ssl dudle.inf.tu-dresden.de 443
2 GET /umfrage/ HTTP/1.0
3 Host: dudle.inf.tu-dresden.de
```

Abbildung 5.10.: Aufbau einer URL (nach RFC3986 [RFC3986]) und ihre Zerlegung bei einem GET-Request. Der Fragment-Teil der URL ist *nicht* Teil des Requests

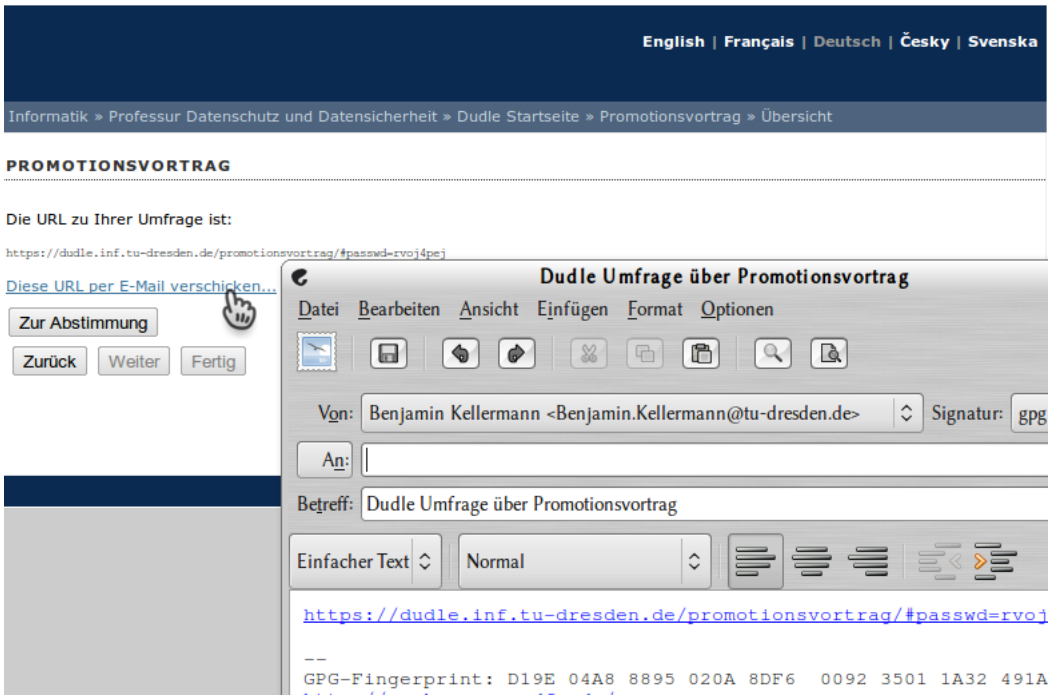


Abbildung 5.11.: Anhängen des Passworts an die URL in der Übersichtsseite. Wird die URL mit dem Passwort aufgerufen, so wird die Passworteingabe (vgl. Abbildungen 5.9(a) und 5.9(b)) übersprungen.

dem Initiator helfen soll, Einladungen zu verschicken. Hinter dem `mailto:`-Link verbirgt sich ein E-Mail-Template mit der URL zur Umfrage.

Hat der Initiator ein Passwort für die symmetrische Verschlüsselung gewählt und dieses wurde im Web Storage gespeichert, so wird das Passwort aus dem Web Storage ausgelesen und der URL angehängt. Rufen die Teilnehmer anschließend die URL auf, welche das Passwort enthält, so wird der Passwortdialog (vgl. Abbildung 5.9(b)) übersprungen. Da das Passwort als Document-Anchor kodiert ist, wird es nicht zum Server übertragen. Abbildung 5.11 zeigt die Übersichtsseite sowie das E-Mail-Template.

Mithilfe dieser Erleichterung verhält sich das Interface für einen Teilnehmer nicht anders als bei einer gewöhnlichen Umfrage wie sie für Schema 2 erklärt wurde. Genau wie dort bekommt der Nutzer einen Link zugeschickt, hinter dem sich eine Umfrage verbirgt.

5.2.2. Symmetrische Verschlüsselung, digitale Signatur

Damit Teilnehmer die Verfügbarkeiten der anderen Teilnehmer nicht mehr unbemerkt ändern können, wurde vorgeschlagen, die Nachrichten digital zu signieren. Es wurden zwei Schemata vorgestellt, welche mit $3'$ und $4'$ bezeichnet wurden. In dieser Arbeit wurde Schema $3'$ im Rahmen eines studentischen Praktikums von Ole Rixmann und Deborah Schmidt in Form eines Plug-Ins implementiert.

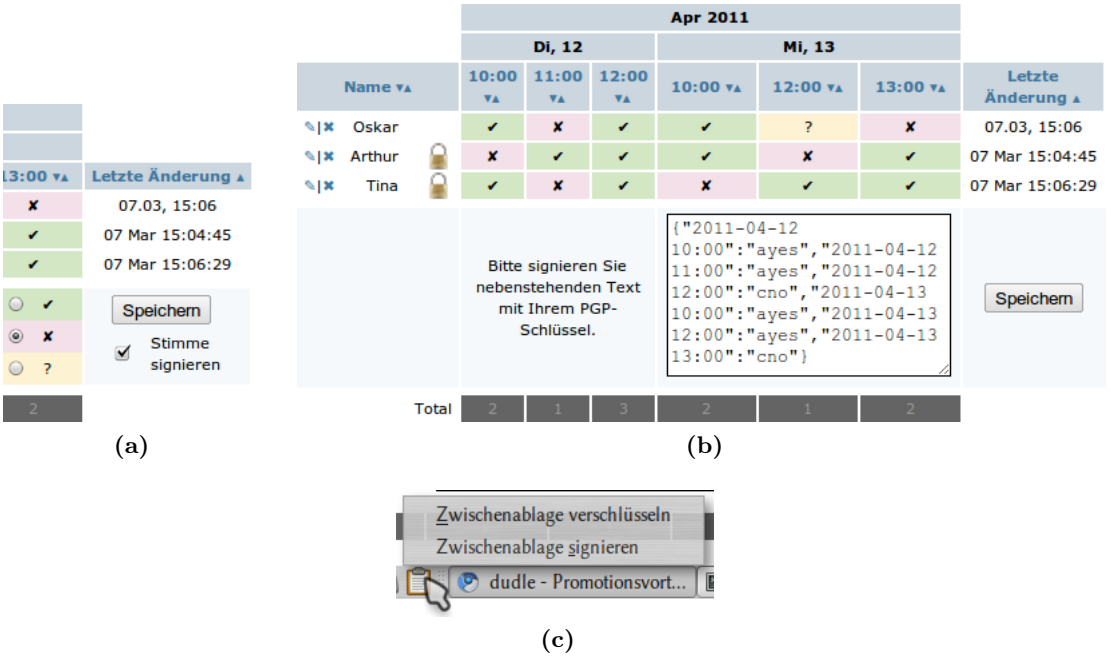


Abbildung 5.12.: Interface, um dem Verfügbarkeitsvektor eine digitale Signatur anzuhängen

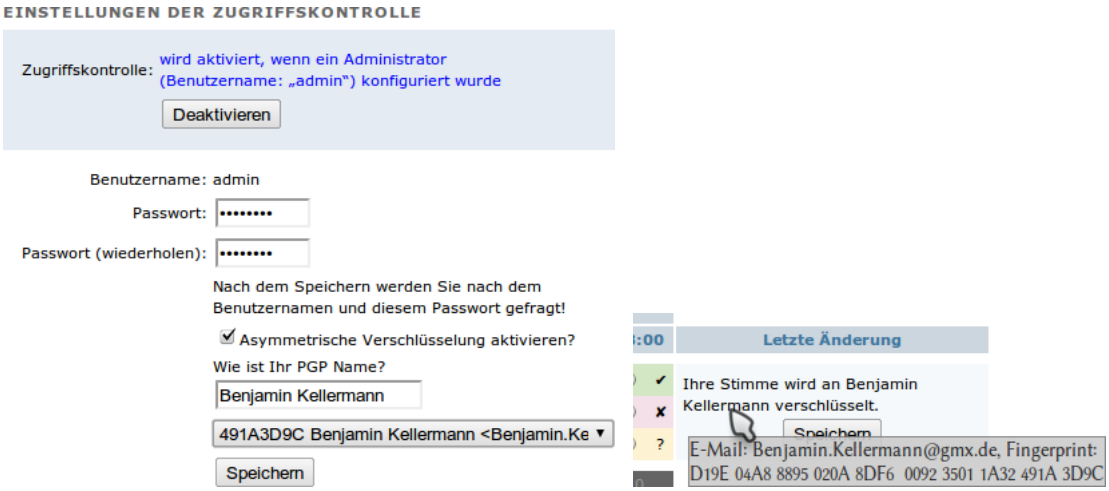
Bei der Stimmenabgabe ermöglicht dieses Plug-In die Signatur seiner Stimme. Dafür bekommt ein Nutzer seine Stimme in maschinenlesbarem Format angezeigt[11] und wird dazu aufgefordert selbige mittels PGP zu signieren (vgl. Abbildung 5.12(b)). Es gibt bereits unzählige Applets, die die nötige Funktionalität für solche Zwecke bereitstellen.[12] Ein Screenshot für das Seahorse-Applet ist in Abbildung 5.12(c) zu sehen.

Zusammen mit den abgegebenen Stimmen werden die evtl. abgegebenen Signaturen beim Anschauen der Ergebnisse vom Server geladen. Die Überprüfung der Signatur erfolgt automatisch mit JavaScript. Anhand eines Siegel-Icons kann ein Teilnehmer sehen, welche Verfügbarkeitsvektoren mit welchem Schlüssel signiert sind. In Abbildung 5.9(c) ist so ein Siegel zu sehen. Würde der Nutzer mit der Maus über das Siegel fahren, so sieht er, mit welchem Schlüssel die Stimme signiert wurde.[13]

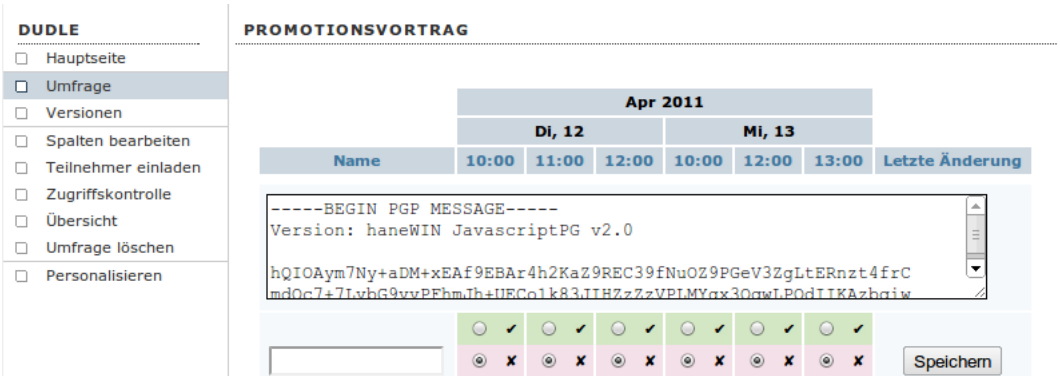
5.2.3. Asymmetrische Verschlüsselung an den Initiator

Wird nur dem Initiator und nicht den anderen Teilnehmern einer Umfrage vertraut, so wurde in Abschnitt 3.2.2 vorgeschlagen, die Verfügbarkeitsvektoren asymmetrisch an den Initiator zu verschlüsseln. Hierfür wurde ein Plug-In für die bestehende Umfrageapplikation geschrieben, welches eine asymmetrische Verschlüsselung mittels JavaScript vornimmt.

[11] hier als JSON-String
[12] z. B. Seahorse, kgpg, kleopatra, gpa, FireGPG, Cryptophane, GnuPG-Shell
[13] Hier wäre auch der Fingerprint des Schlüssels sichtbar.



(a) Konfiguration der Zugriffskontrolle **(b)** Hinweis vor dem Abstimmen



(c) Abruf der Verfügbarkeitsvektoren

Abbildung 5.13.: Interface der Abstimmung mit asymmetrischer Verschlüsselung

Dieses Plug-In wurde im Rahmen eines studentischen Praktikums von Martin Sachse, Oliver Höhne und Robert Bachran geschrieben.

Das Plug-In setzt voraus, dass der Initiator vor Erstellung der Umfrage einen öffentlichen Schlüssel bei einem PGP-Schlüsselserver hochgeladen hat. Theoretisch ist die Benutzung von PGP für dieses Plug-In nicht nötig, die Entscheidung PGP zu benutzen wurde nur getroffen, damit das Schlüsselmanagement nicht neu implementiert werden muss.

Das Plug-In fügt eine neue Option in den Zugriffskontrolldialog (vgl. Abbildung 5.8 auf Seite 77) ein, welcher bei Erstellung der Umfrage vom Initiator durchlaufen wird. Der modifizierte Dialog ist in Abbildung 5.13(a) zu sehen. Sobald der Initiator die Checkbox „Asymmetrische Verschlüsselung aktivieren?“ anwählt, erscheint ein Eingabefeld, in dem der Name des Initiators eingegeben werden kann. Das Plug-In durchsucht einen PGP-Schlüsselserver nach dem eingegebenen Namen und bietet die dort hochgeladenen Schlüssel zur Auswahl für die asymmetrische Verschlüsselung an.

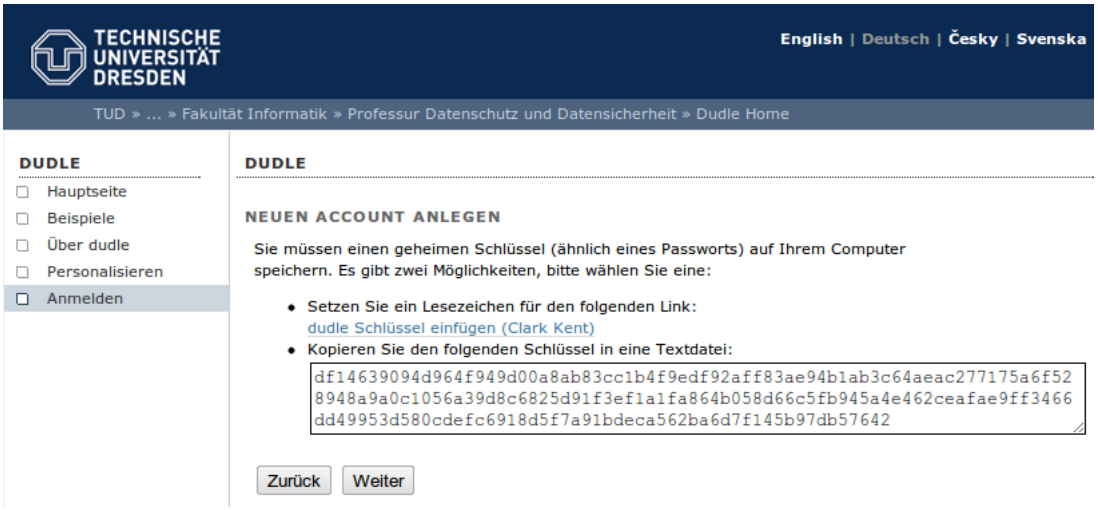


Abbildung 5.14.: Interface zur Schlüsselgenerierung beim Registrieren am Umfragenserver

Stimmt ein Teilnehmer nun in einer so konfigurierten Umfrage ab (vgl. Abbildung 5.3 auf Seite 71), so wird seine Stimme automatisch asymmetrisch mit dem öffentlichen Schlüssel des Umfrageninitiators verschlüsselt an den Server geschickt.[14]

Möchte der Initiator das Ergebnis der Umfrage anschauen, so kann er anschließend die verschlüsselten Verfügbarkeitsvektoren vom Umfragenserver abrufen. Abbildung 5.13(c) zeigt einen Screenshot dieses Abrufs. Es ist ein Textfeld mit der verschlüsselten Nachricht zu sehen. Diese Nachricht muss mittels PGP entschlüsselt und der Inhalt des Textfeldes mit dem entschlüsselten Inhalt ausgetauscht werden. An dem Textfeld sorgt ein JavaScript-onChange-Handler dafür, dass das Ergebnis in der Tabelle aktualisiert wird, sobald der entschlüsselte Verfügbarkeitsvektor in das Feld eingegeben wurde. Für das entschlüsseln der Nachricht kann ein PGP-Applet verwendet werden (vgl. Abbildung 5.12(c)).

5.2.4. Minimal benötigtes Vertrauen in alle Entitäten

Das Verfahren, welches am wenigsten Vertrauen in die beteiligten Entitäten benötigt, basierte auf einer homomorphen Verschlüsselung der Verfügbarkeitsvektoren. Um diese durchzuführen, wurde ein symmetrischer Schlüssel für jedes Teilnehmerpaar benötigt, welcher mittels einer Diffie–Hellman Schlüsselvereinbarung berechnet werden konnte. Für die Schlüsselvereinbarung muss sich jeder Teilnehmer einmalig am Abstimmungsserver registrieren. Hierbei wird ein Schlüsselpaar generiert. Der öffentliche Schlüssel wird zum Server geschickt, während der Benutzer aufgefordert wird, sich den geheimen Schlüssel zu speichern. Dieser Schritt ist in Abbildung 5.14 zu sehen, auf die Schlüsselspeicherung wird in Abschnitt 5.3.1 nochmals genauer eingegangen.

[14] Damit sich der Teilnehmer sicher sein kann, dass seine Stimme mit dem richtigen Schlüssel verschlüsselt wird, kann der Fingerprint des Initiators überprüft werden (vgl. Abbildung 5.13(b)).

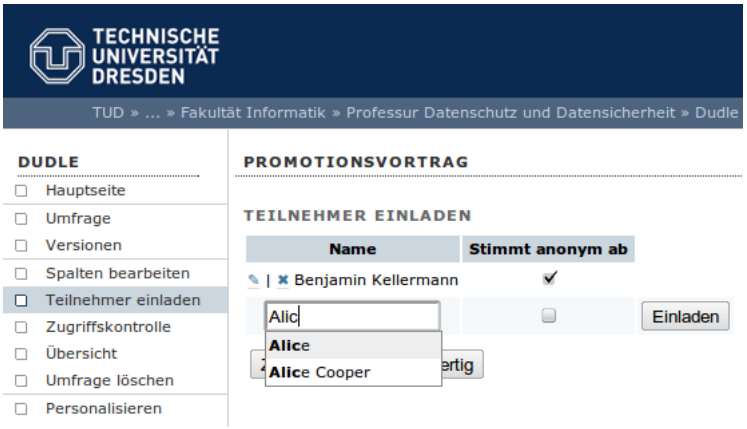


Abbildung 5.15.: Interface zum Einladen von Teilnehmern zur anonymen Abstimmung

Die Erstellung einer Umfrage findet auf gleichem Weg statt, wie die Erstellung der normalen Umfrage. Der einzige Unterschied ist, dass der Initiator hier die Menge der Teilnehmer spezifizieren muss, die initial an der Umfrage teilnehmen soll. Dafür wurde der Schritt, in dem Teilnehmer eingeladen werden, um eine Spalte ergänzt, in der angegeben werden kann, ob der jeweilige Teilnehmer anonym abstimmen soll oder nicht. Abbildung 5.15 zeigt dieses Interface. Hier sieht man auch, dass eine Benutzernamensvervollständigung implementiert wurde, um bei der Auswahl der Benutzer zu helfen. Wird die Checkbox angewählt aber ein Name eingegeben, zu dem kein Schlüssel auf dem Server hinterlegt ist, so weigert sich das Programm fortzufahren.

Will ein Teilnehmer abstimmen, der für die anonyme Abstimmung konfiguriert wurde, so muss er als erstes seinen privaten Schlüssel eingeben. Dies ist in Abbildung 5.16 zu sehen.

Nachdem er diesen eingegeben hat, werden die Präferenzen wie gewohnt ausgewählt. Im Unterschied zu den bisher gezeigten Abstimmungsimplementierungen ist hier nur noch eine einfache ja/nein Eingabe mittels einer Checkbox möglich. Die Option für „vielleicht“ („?“), wie sie in Abbildung 5.3 auf Seite 71 zu sehen war, entfällt.

Haben alle Teilnehmer abgestimmt, werden automatisch alle verschlüsselten Verfügbarkeitsvektoren heruntergeladen und das Ergebnis berechnet. Das Ergebnis sieht dann so aus, wie es schon in Abbildung 5.3 zu sehen war mit dem Unterschied, dass keine Teilnehmerstimmen sichtbar sind.

Das dynamische Hinzufügen und Entfernen von Teilnehmern wurde so implementiert, wie es in Abschnitt 4.5.1 beschrieben ist. Wird ein Teilnehmer vom Umfrageninitiator hinzugefügt, so ist für die Teilnehmer kein Unterschied in der Benutzung ersichtlich. Alle Teilnehmer, die nach dem Zeitpunkt abstimmen, zu dem ein neuer Teilnehmer hinzugefügt wurde, berechnen einen symmetrischen Schlüssel mit diesem und beziehen den neuen Schlüssel mit in ihre Berechnung ein.

Im Gegensatz zum Hinzufügen kann jeder Teilnehmer vorschlagen, dass ein anderer Teilnehmer entfernt werden soll. Möchte ein Teilnehmer einen anderen entfernen, so wird

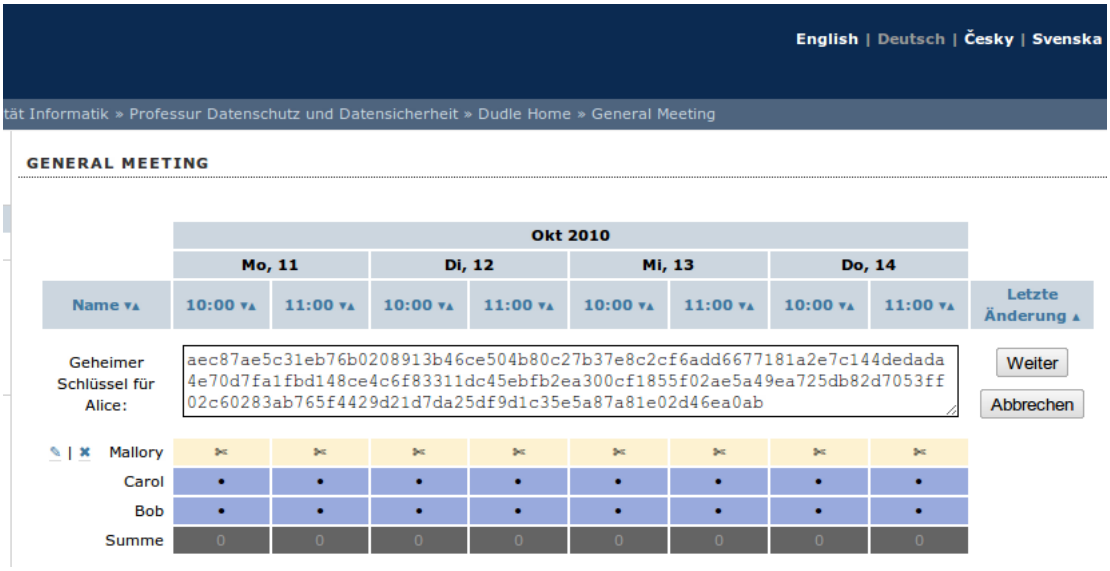


Abbildung 5.16.: Interface, welches zur Schlüsseleingabe während der Abstimmungsphase auffordert

er nach seinem geheimen Schlüssel gefragt, wie es auch beim Abstimmen der Fall ist. Aus diesem geheimen Schlüssel werden anschließend die Rundenschlüssel berechnet, welche zum Server übertragen werden.

Wurde ein Teilnehmer zum Entfernen vorgeschlagen, so ist das in Form eines Scherensymbols sichtbar (vgl. Mallory in Abbildung 5.16). Gibt ein Teilnehmer seine Stimme ab, nachdem ein anderer Teilnehmer zum Entfernen vorgeschlagen wurde, so wird der Abstimmende im Verlauf seiner Stimmenabgabe gefragt, ob er dem Entfernen zustimmt. Abbildung 5.17 auf der nächsten Seite zeigt diese Frage. Hier ist auch zu sehen, welche Teilnehmer dem Entfernen schon zugestimmt haben.

Versucht ein Teilnehmer die Umfrage anzugreifen, so konnte dies durch verschiedene Überprüfungen erkannt werden (vgl. Abschnitt 4.4.1). In der aktuellen Implementierung sind zwei der drei Überprüfungen implementiert: $\mathcal{V}(\boldsymbol{d})$ und $\mathcal{C}(\boldsymbol{d})$. Abbildung 5.18 auf Seite 87 zeigt einen Screenshot von einer Abstimmung in der eine Manipulation vorlag. Hier hat ein Teilnehmer versucht drei Spalten der Umfrage anzugreifen. In der Spalte des 2. Januars lag ein (-1)-Angriff vor, in der Spalte des 4. Januars ein $(+2)$-Angriff. Diese beiden Fehler wurden mit $\mathcal{V}(\boldsymbol{d}) = \texttt{false}$ festgestellt. In der Spalte des 3. Januars wurden die Werte für die Prüfabstimmung nicht konsistent zu denen der normalen Abstimmung gesendet. Hier war $\mathcal{C}(\boldsymbol{d}) = \texttt{false}$. Die dritte Überprüfung ($\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$) sowie die Aufdeckung wurden noch nicht implementiert.

Das Interface wurde mehrfach mittels der Think-Aloud-Methode evaluiert. Hierbei stellte sich für das Abstimmungsinterface (vgl. Abbildung 5.16) als Problem heraus, dass es möglich ist, gleichzeitig anonym sowie nicht-anonym abzustimmen. Was als Feature gedacht war entpuppte sich bei dieser Evaluation als Problem, da es oft passierte, dass die

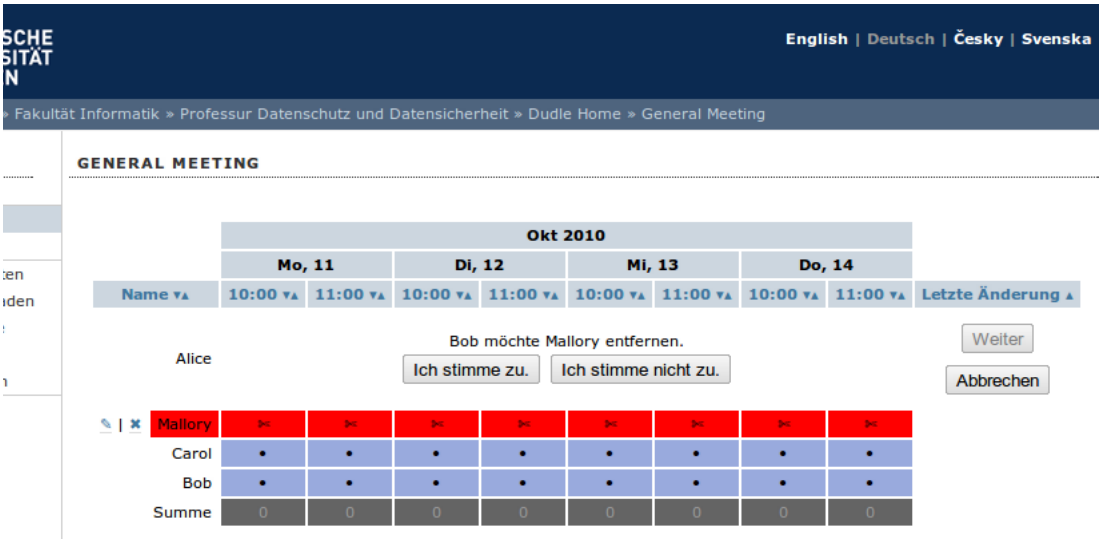


Abbildung 5.17.: Interface, welches über das dynamische Entfernen eines Teilnehmers informiert. Bob möchte Mallory aus der Umfrage entfernen und hat deshalb die mit Mallory gemeinsamen symmetrischen Schlüssel zum Umfrageserver geschickt. Da Alice erst später abstimmt, wird sie bei der Stimmenabgabe gefragt, ob sie dieser Entfernung zustimmt und auch ihre mit Mallory gemeinsamen Schlüssel offenlegt. Die Umfrage kann nur abgeschlossen werden, wenn alle Teilnehmer dem Entfernen zustimmen oder Mallory selbst mit abstimmt.

Nutzer ihre Stimme versehentlich nicht-anonym abschickten. Um das Feature zu erhalten, jedoch diesen Fehler zu erkennen wurde eine Warnung eingebaut, welche angezeigt wird wenn der Nutzer versucht nicht-anonym unter einem Namen abzustimmen, welcher für die anonyme Umfrage konfiguriert ist.

Das Interface zum Registrieren eines neuen Nutzers (vgl. Abbildung 5.14), bei welchem der Diffie–Hellman-Schlüssel generiert wird, wurde mehrfach überarbeitet. Für die Speicherung des Schlüssels wird hier ein Bookmarklet benutzt, wie es in Abschnitt 5.3.1 erläutert wird. Da diese Art der Schlüsselspeicherung sowie das zugehörige Interface in dieser Arbeit neu erarbeitet wurde konnte auf kein gängiges Interface-Design-Pattern zurückgegriffen werden. Größtes Problem war, dass nahezu alle Testpersonen auf den Link klickten, statt ein Lesezeichen von ihm zu erstellen. Erst als eine browserspezifische Hilfe, sowie eine überprüfende zweite Schlüsseleingabe eingebaut wurden, waren die Benutzungsergebnisse hier befriedigend.

Die letzte Evaluation wurde vom Center of Usability Research, Wien durchgeführt. Diese wurde mit 16 Teilnehmern durchgeführt, welche aus allen Berufsschichten (7 mit abgeschlossenem Studium, 9 ohne) und allen Altersschichten (zwischen 23 und 57 Jahren) kamen. Jeder Teilnehmer musste hierbei 4 Aufgaben mit der Think-Aloud-Methode durchführen:

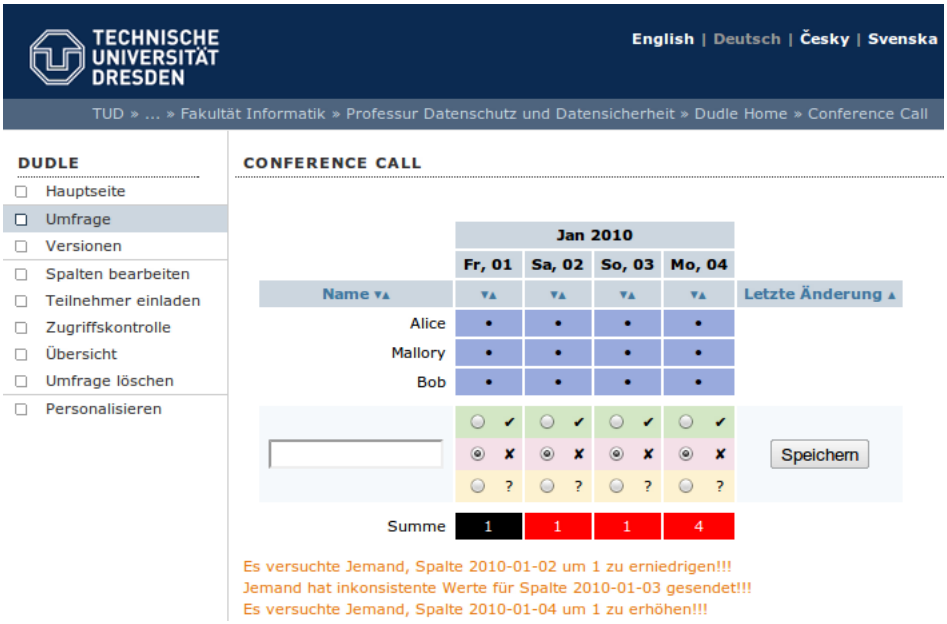


Abbildung 5.18.: Interface, welches bei Erkennung eines Angriffs angezeigt wird. In der Spalte des 2. Januars lag ein (-1)-Angriff vor, in der Spalte des 4. Januars ein $(+2)$-Angriff. Diese beiden Angriffe wurden mit $\mathcal{V}(\boldsymbol{d}) = \texttt{false}$ festgestellt. In der Spalte des 3. Januars wurden die Werte für die Prüfabstimmung nicht konsistent zu denen der normalen Abstimmung gesendet, was mit $\mathcal{C}(\boldsymbol{d}) = \texttt{false}$ festgestellt werden konnte.

1. An einer Umfrage ohne Zugangskontrolle teilnehmen (Stimmenabgabe und Ergebnisveröffentlichung von Schema 0),

2. sich für eine anonyme Umfrage registrieren (Registrierung von Schema 7′),

3. eine anonyme Umfrage anlegen (Umfrageerstellung von Schema 7′) und

4. an einer anonymen Umfrage teilnehmen (Stimmenabgabe und Ergebnisveröffentlichung von Schema 7′).

Diese Aufgaben wurden in einer kurzen Geschichte eingebettet, welche in Abbildung 5.19 zu sehen ist. Das Ergebnis dieser Evaluation war gut, alle bis auf einen Probanden konnten alle Aufgaben durchführen.[15] Kleinere Benutzungsprobleme konnten identifiziert werden, welche teilweise schon in das Interface eingeflossen sind.

Trotz, dass die Aufgaben gelöst werden konnten, verstanden die Benutzer nicht, warum Schlüssel für eine anonyme Umfrage notwendig sind. Eine Lösung für dieses Problem

[15] Der eine Proband scheiterte schon an der ersten Aufgabe, in der er Probleme hatte die Radiobuttons zum Abstimmen zu erkennen (vgl. Abbildung 5.3 auf Seite 71). Der Testleiter hatte bei diesem Probanden generell den Eindruck, dass dieser sehr wenig Erfahrung im Umgang mit Computern hatte.

ist Passwörter zu benutzen, von welchem kryptographische Schlüssel generiert werden. Mehrere Ansätze hierzu werden in Abschnitt 5.3.1 diskutiert.

5.3. Kryptographie mit JavaScript

5.3.1. Schlüsselspeicherung

Ein wesentliches Problem bei der Implementierung von kryptographischen Methoden mit JavaScript ist die Speicherung des geheimen Schlüssels für kryptographische Operationen. Es gibt mehrere Stellen, an denen hierbei ein geheimer Schlüssel gespeichert werden kann:

- Gedächtnis des Nutzers
- Zentral auf einem Server
- Passwortmanager
- Web Storage
- Dateisystem
- Lesezeichen

Der Schlüssel sollte[16] den Rechner des Nutzers nicht verlassen, um die Sicherheit des kryptographischen Verfahrens zu gewährleisten. Im Folgenden wird auf jede Möglichkeit einzeln eingegangen.

Gedächtnis des Nutzers

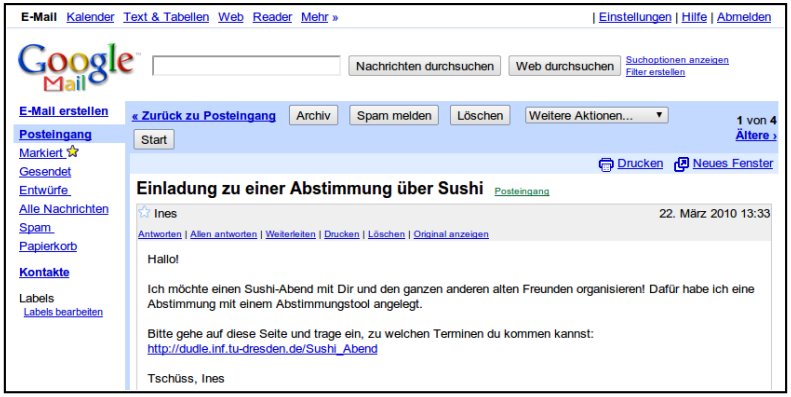
Eine Applikation hat immer die Möglichkeit, den Nutzer darum zu bitten, sich einen Schlüssel zu merken. Die gängigste Art ist, dass sich der Nutzer ein Passwort wählen darf oder ihm ein Passwort generiert wird. Soll aus diesem Passwort anschließend ein Schlüssel für ein kryptographisches System gewonnen werden, so bedarf es einer Passwort-basierten Schlüsselerzeugungsfunktion[17]. Zwei bekannte Schlüsselerzeugungsfunktionen für symmetrische Systeme sind im RFC2898 spezifiziert [RFC2898].

In asymmetrischen Systemen werden häufig bestimmte Kriterien gestellt, die ein Schlüssel erfüllen muss. Schlüsselerzeugungsfunktionen für solche Systeme wurden von IEEE vorgeschlagen [IEEE1363.2].

Die **Vorteile** sind, dass der Schlüssel prinzipiell gegen Angriffe auf den Computer des Nutzers sicher aufbewahrt sind und dass ein Nutzer an anderen Rechnern die gleiche Funktionalität vorfinden wie an seinem eigenen.

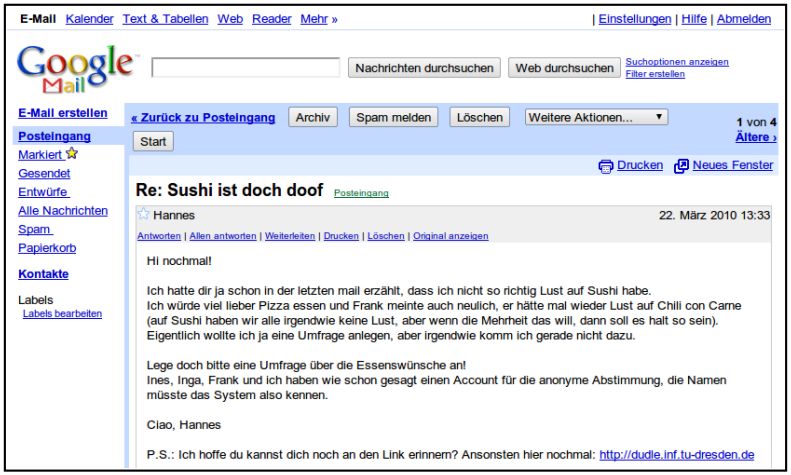
Der **Nachteil** ist, dass die Entropie eines Passworts, welches sich Nutzer merken müssen, normalerweise sehr gering ist [AS99; Yan+04], was sie schwach gegen Brute-Force-Angriffe macht.[18] Werden von anderen Applikationen bestehende Passwörter wiederverwendet,

[16] Eine Ausnahme stellt hier die Methode der Speicherung „Zentral auf einem Server“ dar.
[17] engl. *password-based key derivation function*
[18] Um dieses Problem zumindest teilweise zu umgehen kann key stretching [Kel+98] verwendet werden.



(a) Einladung zu einer normalen Abstimmung

(b) Aufforderung, sich ein Schlüsselpaar anzulegen, um anschließend anonym abstimmen zu können



(c) Aufforderung, eine anonyme Abstimmung zu erstellen

(d) Einladung, an einer anonymen Abstimmung teilzunehmen

Abbildung 5.19.: Aufgaben, die 16 Nutzer zur Evaluation der Oberfläche bekamen

kann man mit einem Angriff auf das schwächere beider Systeme das andere System brechen. Zusätzlich können Benutzbarkeitsprobleme entstehen, da Passwörter vergessen werden können.

Zentral auf einem Server

Schlüssel können zentral auf einem Schlüsselserver gespeichert und vor Gebrauch in einem Kryptosystem abgerufen werden. Damit der Schlüssel bzw. das kryptographische System nicht seine Funktion verliert, muss hier natürlich eine Form der Zugangskontrolle für die Schlüssel implementiert werden. Dafür gibt es zwei Möglichkeiten.

Die erste Möglichkeit ist, den Benutzer nach einem Passwort zu fragen, welches vom Server überprüft wird, bevor der Schlüssel freigegeben wird. Der Server muss hierbei natürlich vertrauenswürdig sein, da er alle Schlüssel kennt.

Die zweite Möglichkeit ist, die Schlüssel verschlüsselt zu speichern und das Passwort des Benutzers als Schlüssel für ein symmetrisches Verschlüsselungssystem zu verwenden. In diesem Fall hat der Server nur noch die Möglichkeit einen Schlüsseltext-Angriff durchzuführen. Neben dem Server kann allerdings noch jeder andere einen solchen Angriff versuchen, da jeder die Möglichkeit hat, das verschlüsselte Passwort abzurufen.

Werden beide Möglichkeiten kombiniert, hat das den Vorteil, dass nicht jeder einen Angriff auf den verschlüsselten Schlüssel durchführen kann. Um ein Passwort zu überprüfen, benötigt der Server Informationen über selbiges.[19] Wird das selbe Passwort für Verschlüsselung sowie Zugriffskontrolle benutzt, so muss dieser Informationsverlust durch höhere Entropie des Passworts ausgeglichen werden. Um einem Informationsverlust des Verschlüsselungspassworts ganz zu umgehen, müssen zwei Passwörter benutzt werden, was wiederum Mehraufwand für den Benutzer bedeutet.

Vorteile dieser Methode sind, ähnlich zur vorher vorgestellten Methode „Benutzung des Gedächtnisses“, dass die Schlüssel von jedem Rechner aus abgerufen werden können. Genau wie dort muss sich der Benutzer ein Passwort merken, die Speicherung findet daher nicht ausschließlich beim Server statt. Die Methode, einen Schlüssel auf einem Server zu speichern, kann allerdings performanter sein, als die Benutzung einer Schlüsselerzeugungsfunktion (besonders, wenn, wie häufig bei asymmetrischen Systemen notwendig, bestimmte Anforderungen an den Schlüssel gestellt werden).

Die **Nachteile** des Teilvertrauens in den Schlüsselserver wurden schon erwähnt. Dazu kommen noch die gleichen Probleme, die sich durch die Benutzung eines Passworts ergeben.

Passwortmanager

Der Passwortmanager eines Web-Browsers ist der generische Ort, um Geheimnisse des Benutzers zu speichern. Er ist per Definition ein sicherer Speicher, der Programmierer braucht sich daher nicht um Probleme wie XSS oder andere Angriffe zu kümmern. Schutz gegen diese Angriffe soll vom Web-Browser erreicht werden.

[19] Meistens liegt das Passwort in Form eines gespeicherten Hashwertes vor.

Ein Problem ist allerdings, dass keine JavaScript-API existiert, um etwas im Speicher des Passwortmanagers abzulegen oder etwas daraus zu lesen. Der Vorteil, keine Möglichkeit des Zugriffs zu haben, ist, dass böshafter JavaScript-Code geheimzuhaltende Daten nicht lesen kann. Der Nachteil ist, dass gutwilliger JavaScript-Code auch keinen Zugriff darauf hat.

In dieser Arbeit wurde eine Möglichkeit herausgefunden, wie es ohne JavaScript-API möglich ist, Daten im Passwortmanager zu speichern bzw. selbige wieder auszulesen, welche im Folgenden vorgestellt wird.

Browser sind darauf ausgelegt, Passwörter zu speichern, sobald sie von einer Webseite zur Authentifikation benötigt werden. Das folgende Codeschnipsel zeigt ein Beispiel. Ein Browser würde dieses Schnipsel als ein typisches Login-Formular darstellen:

```
1 <form>
2   <input name="username" type="text" />
3   <input name="password" type="password" />
4   <input type="submit" />
5 </form>
```

Es werden drei Inputfelder dargestellt, eins für den Benutzernamen, ein Passwortfeld und ein Button, auf den der Nutzer klicken kann, um das Formular abzuschicken. Sobald der Nutzer den Button zum Abschicken klickt, würde sich der Passwortmanager einschalten, um das Passwort für einen späteren Gebrauch lokal zu speichern. Das Problem, was sich hierbei ergibt, ist, dass mit dem POST-Request der Inhalt des Inputfeldes von Zeile 2 und der Inhalt des Inputfeldes von Zeile 3 als key-value-Paar `username:...` sowie `password:...` zum Server übertragen wird. Da der Schlüssel den Rechner des Nutzers allerdings nicht verlassen sollte, ist diese Lösung nicht akzeptabel.

Mit einem Trick kann dieser Sendevorgang umgangen werden. Lässt man das `name`-Attribut für das Inputfeld weg, so wird auch kein key-value-Paar zum Server übertragen. Die Passwortmanager von Internet Explorer, Firefox und Safari merken sich aber trotzdem den Inhalt aller Felder, auch die, die nicht übertragen wurden. Ein Codeschnipsel, welches für die Speicherung eines Passworts benutzt werden könnte, sieht also wie folgt aus:[20]

```
1 <form>
2   <input name="username" type="text" />
3   <input type="password" />
4   <input type="submit" />
5 </form>
```

Wird die Seite später wieder aufgerufen, so setzt der Passwortmanager des Browsers das Passwort in das zugehörige `<input>` Feld ein, welches mittels JavaScript ausgelesen werden kann.

Zusammenfassend sind die **Vorteile** der per Definition sichere Speicher sowie die einfache Benutzung auf Nutzerseite.

[20] Zu Beachten ist, dass immer mindestens zwei `<input>`-Felder, eins mit `type="text"` und eins mit `type="password"` vorhanden sein müssen. Andernfalls speichern mindestens Internet Explorer und Safari das Passwort nicht im Passwortmanager.

Der **Nachteil** dieser Methode ist ihre Zuverlässigkeit. Zuerst funktioniert sie nicht in allen Browsern.[21] Darüber hinaus können Nutzer den Passwortmanager explizit deaktivieren und somit die Speicherung verhindern, obwohl die Methode prinzipiell in einem Browser funktionieren sollte. Außerdem ist unklar, ob die Methode in Zukunft funktionieren wird, da die Browser ihr Verhalten ändern könnten.

Web Storage

Der Web Storage ist ein vom W3C spezifizierter Speicherbereich, welcher über JavaScript zugreifbar ist. Die Spezifikation ist momentan noch als „Working Draft“ gekennzeichnet, die API wird aber schon von vielen Web-Browsern implementiert.

Der Web Storage kann ähnlich wie Cookies benutzt werden. Cookies werden bei jedem Request als Header mit zum Server übertragen. Beim Web Storage ist das nicht der Fall, weshalb sich der Speicher prinzipiell zum Speichern von Schlüsseldaten eignet.

Vorteilhaft am Web Storage ist, dass es keinerlei Nutzerinteraktion bedarf und somit eine hohe Benutzbarkeit verspricht.

Leider beinhaltet die Spezifikation des Speichers keine Informationen darüber, wie und wo die Daten des Web Storage abgelegt werden. Ein **Nachteil** ist also, dass Angriffe durch Zugriffe auf das lokale Dateisystem sowie den flüchtigen Speicher passieren könnten. Darüber hinaus hat dieser Speicher die gleichen Probleme bei XSS Angriffen, die Cookies auch haben.

Das größte Problem ist allerdings, dass dieser Speicher nicht zuverlässig verfügbar ist. In den in dieser Arbeit durchgeführten Tests wurde der Speicher in manchen Fällen nach Schließen des Browsers oder aber nach einer bestimmten Zeit gelöscht. Diese Beobachtung kann natürlich auch dadurch bedingt sein, dass der Standard noch nicht final und die Implementierungen daher noch nicht sonderlich ausgereift sind. Für die Speicherung eines Schlüssels während einer Session gab es keine Probleme bei den Tests dieser Arbeit.

Dateisystem

Auf das lokale Dateisystem eines Rechners kann zwar nicht direkt mit JavaScript zugegriffen werden, allerdings kann dem Nutzer der Schlüssel in einer geeigneten Kodierung angezeigt werden (z. B. Base16, Base36 oder Base64 [RFC4648]). Der Nutzer muss diesen Schlüssel anschließend händisch in einer Datei abspeichern. Ein Beispiel dafür ist in Abbildung 5.14 auf Seite 83 zu sehen. Hier wird ein Textfeld angezeigt, in dem der Schlüssel als Hexadezimalzahl dargestellt ist. Der Nutzer wird dazu aufgefordert, diesen Schlüssel lokal auf seinem Rechner zu speichern. Für die sichere Aufbewahrung des Schlüssels ist anschließend der Nutzer selbst verantwortlich.

Vorteil dieser Methode ist, dass sie in jedem Fall bei jedem Nutzer funktioniert.

Der **Nachteil** ist allerdings der Aufwand des Nutzers und damit eine schlechtere Benutzbarkeit. Da es ungewöhnlich ist, dass eine Web 2.0-Applikation von ihren Nutzern

[21] Es funktionierte nicht bei Opera und Chrome, was laut Abbildung 2.1 auf Seite 7 ca. 8 % aller Internetnutzer betrifft.

verlangt, Daten in einer Textdatei zu speichern, um sie später einzugeben, muss mehr Erklärungsarbeit geleistet werden und es ist eine höhere Fehlbedienung zu erwarten.

Lesezeichen

Eine letzte Möglichkeit, einen Schlüssel zu speichern, wurde während dieser Arbeit entwickelt. Die Idee ist, ein sogenanntes *Bookmarklet* zu setzen, welches den Schlüssel beinhaltet. Ein Bookmarklet ist hierbei ein Lesezeichen, welches JavaScript-Code enthält. Ein Beispiel soll diese Methode verdeutlichen:

Beispiel 9 Angenommen der zu speichernde Schlüssel des Nutzers sei `0xDEADBEEF`. Ein JavaScript-Codeschnipsel, welches diesen Schlüssel in das Element

```
<input id="key" type="text" />
```

einfügt sieht folgendermaßen aus:

```
document.getElementById('key').value = '0xDEADBEEF'
```

Möchte man nun einen Link erstellen, welcher diesen JavaScript-Code ausführt, so kann man das Pseudoprotokoll `javascript:` benutzen und folgenden Link dem Nutzer anzeigen.[22]

```
1 <a href="javascript:void(document.getElementById('key')
2 .value='0xDEADBEEF')">insert key</a>
```

Dieser muss sich nun von dem Link ein Lesezeichen erstellen. Klickt der Nutzer auf dieses Lesezeichen, so wird der Schlüssel in das Feld `<input id="key">` eingefügt. Von dort kann es von anderem JavaScript-Code ausgelesen und weiter verarbeitet werden.[23] □

In der in Abschnitt 5.2.4 vorgestellten Implementierung wurde diese Art den Schlüssel zu speichern implementiert. Zu sehen ist das Bookmarklet in diesem Abschnitt in Abbildung 5.14 auf Seite 83. In dieser Abbildung wird der Benutzer dazu aufgefordert wird, ein Lesezeichen für das Bookmarklet zu erstellen.

Dieses Interface wurde mit mehreren Testpersonen getestet und mehrfach überarbeitet. Größtes Problem war, dass nahezu alle Testpersonen auf den Link klickten, statt ein Lesezeichen von ihm zu erstellen. Das Ziel des Links (`href`) kann allerdings in diesem Dialog nicht verändert werden, da dann ein falsches Lesezeichen erstellt werden würde. Abhilfe wurde dadurch geschaffen, dem Link noch einen `onClick` Handler zu geben, welcher eine browserspezifische Hilfe anzeigt, wie man ein Lesezeichen erstellen kann. Der endgültige Sourcecode des vorherigen Beispiels sieht daher folgendermaßen aus:

```
1 <a href="javascript:void(document.getElementById('key')
2 .value='0xDEADBEEF')" onClick="help()">insert key</a>
```

[22] Die Funktion `void()` ist hierbei nur nötig, damit das Bookmarklet keinen Rückgabewert hat, was bei manchen Browsern einen Fehler verursacht.

[23] Es ist natürlich genauso möglich, gleich den Funktionsaufruf als Bookmarklet zu speichern, z. B. `executeCryptoFunction("0xDEADBEEF")`.

Die **Vorteile** dieser Methode sind, dass die Benutzung einfacher ist, als eine Speicherung in einer Textdatei. Zusätzlich findet die Speicherung in dem Programm statt (Browser), welches anschließend zum Aufrufen der Webseite benötigt wird.

Nachteilig ist allerdings, dass viele Benutzer Probleme haben, die Methode zu verstehen, was sich in einem höheren Erklärungsaufwand widerspiegelt. Die so gespeicherten Schlüssel sind auf gleiche Weise einem Angriff auf das lokale Dateisystem bzw. den flüchtigen Speicher ausgeliefert wie die Lesezeichen des Nutzers. Ein möglicher Angriff über JavaScript ist dem Autor dieser Arbeit nicht bekannt und scheint unwahrscheinlich, da auch andere Lesezeichen schützenswerten Inhalt haben. Letztendlich kann jedoch nicht ausgeschlossen werden, dass Lesezeichen aus dem Browser über eine Web-Applikation ausgelesen werden könnten (wenn nicht über JavaScript, dann vielleicht über ein Browser-Plug-In o. ä.). In jedem Fall haben viele Browsererweiterungen Zugriff auf die Lesezeichen.

5.3.2. Blockieren der JavaScript-Berechnungen vermeiden

Kryptographische Operationen sind oft sehr rechenintensiv. Die heutigen JavaScript-Interpreter benötigen teilweise mehrere Sekunden, um nur eine kryptographische Operation zu berechnen (vgl. Tabelle 2.1). Die JavaScript-Interpreter aller Browser haben einen Watchdog-Counter, welcher verhindern soll, dass Endlosschleifen in fehlerhaft implementiertem Code den Browser und damit evtl. den ganzen Rechner des Nutzers zum Stillstand bringen. Führt eine Applikation eine bewusst längere Berechnung durch, wie es beim Einsatz von Kryptographie oft vorkommt, so schlägt der Watchdog fälschlicherweise Alarm und versucht die Ausführung des Codes abzubrechen.

Um längere Berechnungen mit JavaScript zu ermöglichen wurde die BigInteger Bibliothek von Wu [Wu] modifiziert. Hierbei wurden Schleifen, welche für den Watchdog zu lange Ausführungszeit benötigten, in rekursive Funktionen umgewandelt. Jeder Schleifendurchlauf ist hierbei in einem rekursiven Funktionsaufruf gekapselt, welcher kurz genug ist, um nicht vom Watchdog beanstandet zu werden. JavaScript hat die Anweisung `window.setTimeout(myFunction, someTime)`, welche eine Funktion asynchron nach `someTime` Millisekunden aufruft. Alle Funktionen, die mit dieser Funktion aufgerufen werden, werden vom Watchdog separat behandelt. Das folgende Beispiel soll die Modifikation verdeutlichen.

Beispiel 10 Angenommen eine Funktion sei zu rechenintensiv um einzeln von einem JavaScript-Interpreter ausgeführt zu werden. Abbildung 5.20(a) zeigt eine solche Funktion. Hier wird in Zeile 12 die Berechnung aufgerufen und in Zeile 13 soll das Ergebnis der Funktion zu einem Server geschickt werden. Die Schleife von Zeile 4–8 wird zu oft ausgeführt, weshalb der Watchdog denkt, diese Funktion befände sich in einer Endlosschleife.

Um dieses Verhalten zu vermeiden, wird der innere Teil der Schleife, welcher kurz genug ist, in vielen Einzelteilen ausgeführt. Da die Ausführung der Funktion anschließend asynchron stattfindet, wird die Funktion um einen Parameter erweitert, in dem eine Callbackfunktion angegeben werden kann. Anstatt am Ende einen Wert zurückzugeben, wird die Funktion diese Callbackfunktion ausführen und den Rückgabewert der Funktion übergeben. Dieser Code ist in Abbildung 5.20(b) dargestellt.

```
function extensiveCalculation(x, y) {
  // Diese Schleife wird zu oft aufgerufen
  // und deshalb vom Watchdog abgebrochen.
  while (reallyReallyOften) {
    // Diese Berechnung ist kurz genug und kann
    // einzeln problemlos aufgerufen werden.
    shortCalculation();
  }
  return foo;
}

var something = extensiveCalculation(a, b);
sendToServer(something);
```

(a) Eine Funktion, welche zu lange rechnet und deshalb vom Watchdog-Timer abgebrochen wird.

```
function extensiveCalculation(x, y, callbackfunc) {
  ...
  // return statement in callback umwandeln
  // return foo;
  callbackfunc(foo);
}

extensiveCalculation(a, b, function(something) {
  sendToServer(something);
});
```

(b) Umwandeln des `return` statements in eine Callback-Funktion

```
function extensiveCalculation(x, y, callbackfunc) {
  function oneRound() {
    if (reallyReallyOften) {
      shortCalculation();
      window.setTimeout(oneRound, 1);
    } else {
      callbackfunc(foo);
    }
  }
  oneRound();
}
```

(c) Umwandeln der `while` Schleife in einen rekursiven Aufruf

Abbildung 5.20.: Vermeidung von blockierenden JavaScript Berechnungen durch Umwandlung einer Funktion in eine asynchrone Berechnung

Tabelle 5.2.: Ausführungszeit für eine diskrete Exponentation in einer Gruppe mit 2^{1024} Elementen mit mehreren Laufzeitumgebungen. Gemessen wurde auf einem Intel Pentium 4 Duo mit 2,8 GHz CPU und 2 GB RAM unter Windows XP SP3. Da Flash und Java jeweils von einer gemeinsamen Laufzeitumgebung ausgeführt werden unterscheiden sich die Zeiten hier nur um wenige Millisekunden.

	 IE8 8.0	 IE7 7.0	 IE6 6.0	 FF 5.0	 Chrome 12.0	 Safari 5.1	 Opera 11.50
[Wu]	4,89 s	5,39 s	5,49 s	0,94 s	0,10 s	1,38 s	0,70 s
[Bai]	8,46 s	13,43 s	13,56 s	0,26 s	0,08 s	0,29 s	0,31 s
FlashPlayer 10.3				≈ 1,0 s			
Java 6 (Build 24)				≈ 0,03 s			
Java Init				≈ 4,40 s (±0,5 s)			

Abschließend wird die `while`-Schleife noch in eine rekursive Funktion umgewandelt, welche sich selbst mittels `setTimeout()` aufruft (Abbildung 5.20(b)).

Hinweis: Die Callbackfunktion `callbackfunc` braucht nicht an die rekursive Funktion `oneRound()` übergeben werden, da die Funktionsdefinition innerhalb der eigentlichen Funktion stattfindet und die Parameter daher für die innere Funktion global sichtbar sind. Aus selbigem Grund ist auch jede andere lokale Variable für die innere Funktion sichtbar und braucht nicht übergeben werden. □

5.3.3. Performanceverbesserung der JavaScript-Berechnungen

In Tabelle 2.1 wurde schon darauf eingegangen, dass einige JavaScript-Interpreter wenig performant sind. Gleichzeitig haben Browser bzw. Betriebssysteme oft andere Laufzeitumgebungen schon installiert. Zwei sehr oft zu findende Laufzeitumgebungen sind die Java-VM sowie der Adobe Flash Player. Daher stellt sich die Frage, wie Ausführungszeiten in diesen Umgebungen sind. Tabelle 5.2 stellt die Ausführungszeiten dieser Laufzeitumgebungen gegenüber. Hier sind die Zeiten der zwei schnellsten JavaScript-BigInteger Bibliotheken einer ActionScript und einer Java Implementierung gegenübergestellt, welche mit dem Adobe Flash Player bzw. der Java-VM ausgeführt wurden. Gerade beim Internet Explorer sieht man, dass es einen erheblichen Geschwindigkeitsvorteil bringt, wenn man Berechnungen in Java ausführt.

Java Applets benötigen eine gewisse Zeit, um sich zu initialisieren. Diese Zeiten sind in der letzten Zeile der Tabelle dargestellt. Möchte man nun beispielsweise zehn diskrete BigInteger-Exponentationen in einer Gruppe mit 2^{1024} Elementen durchführen, so würden mit Internet Explorer 8 und der JavaScript-Bibliothek von Wu allein für die Exponentationen $10 \cdot 4{,}89\,\text{s} = 48{,}9\,\text{s}$ benötigt. Hat der Nutzer jedoch den Flash Player installiert, so

würde er für selbige Berechnung nur $10 \cdot 1{,}02\,\mathrm{s} = 10{,}2\,\mathrm{s}$ benötigen. Mit einem Java-Applet würde die Berechnung trotz der relativ langen Initialisierungszeit von 4,40 s sogar nur $10 \cdot 0{,}03\,\mathrm{s} + 4{,}40\,\mathrm{s} = 4{,}70\,\mathrm{s}$ benötigen.

Für einen Chrome-Nutzer sieht diese Rechnung völlig anders aus. JavaScript ist hier schneller als Flash und bis sich bei ihm die Initialisierungszeit des Java-Applets amortisiert, müssen sehr viele BigInteger Exponentationen durchgeführt werden.

Um die Performancevorteile von Java und ActionScript auszunutzen und gleichzeitig den geringen Installationsaufwand nicht aufzugeben, wurde eine Bibliothek geschrieben, welche den Browser des Nutzers analysiert und versucht die schnellste Bibliothek für die Berechnung der zeitaufwendigen kryptographischen Operationen zu benutzen. Für den Programmierer bietet die Bibliothek ein JavaScript-Interface. Der Programmierer initialisiert die Bibliothek, indem er so genau wie möglich mitteilt, welche Operationen wie oft benötigt werden. Ein Beispiel für eine mögliche Initialisierung sieht wie folgt aus:

```
1 CryptoLibrary.initialize({
2   BigInteger: {
3     modPow: [5, 1024],
4   },
5   AES: [2000, 128],
6   SHA: [2000, 256]
7 });
```

Mit diesen Zeilen wird zum Ausdruck gebracht, dass 5 BigInteger Exponentationen in einer Gruppe mit 2^{1024} Elementen, 2000 AES-128 und 2000 SHA256 Operationen durchgeführt werden sollen. Die Bibliothek beinhaltet eine kleine Datenbank über die erwarteten Ausführungszeiten. Diese wurden auf einem Beispielrechner mit verschiedenen Browsern und Laufzeitumgebungen berechnet.

Bei der Initialisierung prüft die Bibliothek, mit welchem Browser sie es zu tun hat und welche Laufzeitumgebungen zur Verfügung stehen. Anschließend berechnet sie für den jeweiligen Browser für jede vorhandene Laufzeitumgebung die erwartete Ausführungszeit. Stellt sich hierbei heraus, dass es sich lohnt, die Java Virtuelle Maschine zu initialisieren, so wird das getan. Anschließend werden alle Operationen mit der jeweils schnellsten Laufzeitumgebung bzw. Bibliothek ausgeführt.

Die Datenübergabe zwischen JavaScript, Flash und Java erfolgt für den Programmierer vollständig transparent, d. h. als würde er nur mit Daten in JavaScript arbeiten.

5.3.4. JavaScript-Code vertrauen

In vielen der bisher vorgestellten Schemata wurde versucht das Vertrauen zu reduzieren, welches dem Serveradministrator entgegengebracht werden muss. Vertrauensreduktion in den Serveradministrator benötigte hier immer Berechnungen auf Seite des Clients, welche hier mit JavaScript durchgeführt wurden. Problematisch ist an dieser Stelle, dass dem JavaScript-Sourcecode vollständig vertraut werden muss, da er Zugriff auf jegliche Geheimnisse hat, die bei der Berechnung benutzt werden.

Natürlich steht es jedem frei, den JavaScript-Sourcecode vor seiner Ausführung zu lesen, jedoch verfügt nicht jeder Anwender über die dafür nötigen Fähigkeiten bzw. Zeit.

Eine mögliche Lösung für dieses Problem wäre, dass Dritte den Sourcecode lesen und dessen Korrektheit bestätigen. Diese Bestätigung müsste wiederum zusammen mit einer digitalen Signatur an den JavaScript-Code gehängt werden. Um die Glaubwürdigkeit des Codes weiter zu erhöhen, könnte der Sourcecode von mehreren Personen gelesen und bewertet werden.

Das Resultat eines solchen Systems ist ein Reputationssystem, in dem JavaScript-Code bewertet wird. Der Browser überprüft die digitalen Signaturen und stellt dem Nutzer in geeigneter Form dar, wie vertrauenswürdig der jeweilige JavaScript-Code ist, den eine Webseite benutzt. Da so ein System aktuell nicht in Browsern implementiert ist, würde eine Umsetzung in jedem Fall die Installation einer Zusatzsoftware benötigen (z. B. in Form einer Browserextension). Eine vollständige Umsetzung einer solchen Lösung ist kein Bestandteil dieser Arbeit, sollte aber Thema weiterer Forschung sein.

6. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das Problem behandelt einen Termin mit Hilfe des Web 2.0 zwischen mehreren Personen zu finden. Dabei wurde insbesondere auf eine mehrseitig sichere Umsetzung geachtet, was eine Betrachtung verschiedener Schutzziele (insbesondere Vertraulichkeit und Integrität) und der Vertrauenswürdigkeit der beteiligten Entitäten mit einschließt. Es wurden zahlreiche bestehende Web-Applikationen untersucht und in verschiedene Sicherheitsklassen bzgl. des notwendigen Vertrauens eingeteilt.

Ferner wurden mehrere neue Schemata vorgestellt, welche alle das Ziel haben das notwendige Vertrauen in den Serveradministrator zu reduzieren. Neben den Sicherheitsanforderungen wurde bei jedem Schema darauf geachtet, dass es

- effizient genug ist um im Browser in JavaScript ausgeführt zu werden und
- möglichst wenige Protokollrunden benötigt werden, bei denen die Eingaben aller Teilnehmer nötig ist.

Schemata, welche mindestens eine vertrauenswürdige Entität voraussetzen, lassen sich grob in drei unterschiedliche Methoden unterteilen, welche aus den zwei orthogonalen Achsen

1. Art des Kryptosystems (symmetrisch vs. asymmetrisch) und
2. Zweck des Kryptosystems (Vertraulichkeit vs. Integrität)

bestehen. Die drei herausgestellten Methoden sind

1. symmetrische Verschlüsselung/symmetrische Authentikation,
2. symmetrische Verschlüsselung/asymmetrische Authentikation (digitale Signatur) und
3. asymmetrische Verschlüsselung/digitale Signatur.[1]

Neben Methoden, welche mindestens eine vertrauenswürdige Entität benötigen, wurde ein Protokoll entwickelt, in dem ein Teilnehmer ohne Vertrauen in andere Entitäten auskommt. Dieses ähnelt einem E-Voting Protokoll und besteht aus den Schritten:

1. Jeder Teilnehmer verschlüsselt seine Verfügbarkeitsinformationen und sendet die verschlüsselten Informationen an einen Abstimmungsserver.

[1] Die theoretische vierte Methode asymmetrische Verschlüsselung/symmetrische Authentikation wurde nicht betrachtet, da sie keinen Vorteil bzgl. der Benutzbarkeit bietet aber einen Nachteil bzgl. der Sicherheit im Gegensatz zur Methode asymmetrische Verschlüsselung/digitale Signatur.

2. Das verwendete Verschlüsselungsverfahren ist homomorph bzgl. der Addition, wodurch die Summe der verfügbaren Teilnehmer, aber keine einzelne Verfügbarkeit berechnet werden kann.

Um Teilnehmer daran zu hindern, die eigenen verschlüsselten Nachrichten absichtlich falsch zu kodieren, wurde das Protokoll so gestaltet, dass

1. ein Betrug erkannt und
2. ein Betrüger entdeckt werden kann.

Schlussendlich wurden zwei Protokollerweiterungen vorgestellt, mit welchen es möglich ist, Präferenzwahlen durchzuführen und Teilnehmer nach Beginn einer Umfrage dynamisch hinzuzufügen oder zu entfernen.

Um die Benutzbarkeit der einzelnen Schemata zu belegen wurden alle in Form einer Web 2.0-Applikation implementiert. Hierfür wurde ein Webserver aufgesetzt, welcher zur öffentlichen Nutzung freigegeben wurde. Dabei wurden auch Lösungen für Probleme entwickelt, die nicht in theoretischen Protokollen ersichtlich sind. Diese Lösungen umfassen beispielsweise

- das Problem, dass der JavaScript-Interpreter die ausgeführten Berechungen mittels eines Watchdogs überwacht und zu lang dauernde Berechungen beendet,
- die Speicherung kryptographischer Schlüssel im Browser oder
- die Verbesserung der Performance kryptographischer Berechnungen in Web 2.0-Applikationen mittels Java und Flash.

Die Applikation wurde in mehrerern Iterationen von Benutzern ohne technischen Hintergrund evaluiert und deren Benutzbarkeit mehrfach verbessert. Die in den letzen Monaten gestiegenen Zugriffszahlen zeigen die Akzeptanz der Implementierung und belegen deren Benutzbarkeit.

Ein Problem, welches in dieser Arbeit nicht gelöst wurde, ist die Vertrauenswürdigkeit des JavaScript-Codes einer Web 2.0-Applikation. Dieses Problem wurde kurz diskutiert und ein möglicher Lösungsansatz vorgestellt.

Darüber hinaus wurden Benutzbarkeitsprobleme identifiziert, deren Lösung teil zukünftiger Forschung sein kann. Allen voran stellt die Benutzung asymmetrischer Systeme aktuell noch ein Problem für Benutzer ohne technischen Hintergrund dar. Benutzer waren nicht im Umgang mit Schlüsseln vertraut und empfanden keinen Mehrwert in der Benutzung eines kryptographischen Protokolls.

Ein weiterer Aspekt, der noch betrachtet werden kann ist, ob andere Verfahren angewendet werden können, die noch weniger Informationen als die Summe der verfügbaren Teilnehmer veröffentlichen (z. B. nur den Zeitpunkt, an dem die meisten Teilnehmer verfügbar sind). Ein konkretes Verfahren könnte auch beliebige Terminauswahlfunktionen zulassen und anhand dieser nur das Ergebnis mitteilen. Ein solches Verfahren wirft allerdings viele Interfaceprobleme auf. Benutzerstudien könnten hier Aufschluss darüber geben, welche Terminauswahlfunktionen einem Benutzer wie dargestellt werden könnten.

Folgearbeiten sollten sich außerdem mit Terminabstimmungsproblemen zwischen zwei Personen auseinandersetzen. Entsprechende (unsichere) Web 2.0-Applikationen sind in letzter Zeit häufig aufgetaucht (z. B. Doodle: „MeetMe“, Google: „appointment-slot“, `http://tungle.me`).

Abseits von Terminabstimmungsapplikationen sollte auch versucht werden, andere *mehrseitig sichere* Web 2.0-Zero-Footprint-Applikationen zu erstellen. Zur Erstellung dieser kann und sollte auf Techniken aus dieser Arbeit sowie entstandene Bibilotheken zurückgegriffen werden.

Literatur

[12g] 12gurus LLC. *GatherGrid – Find the best time for everyone.* URL: http://www.gathergrid.com (besucht am 03. 11. 2010).

[Abe98] Masayuki Abe. „Universally Verifiable Mix-net with Verification Work Indendent of the Number of Mix-servers“. In: *EUROCRYPT*. Springer, 1998, S. 437–447.

[Adi08] Ben Adida. „Helios: Web-based Open-Audit Voting“. In: *USENIX Security Symposium*. Hrsg. von Paul C. van Oorschot. USENIX Association, 2008, S. 335–348. ISBN: 978-1-931971-60-7.

[AS99] Anne Adams und Martina Angela Sasse. „Users are not the enemy“. In: *Commun. ACM* 42 (12 Dez. 1999), S. 40–46. ISSN: 0001-0782. DOI: 10.1145/322796.322806.

[Avi] Avignon University. *RdvZ*. URL: http://gpl.univ-avignon.fr/rdvz/ (besucht am 18. 07. 2011).

[Bai] Leemon Baird. *BigIntegers in JavaScript, Version 5.4*. URL: http://www.leemon.com/crypto/BigInt.html (besucht am 10. 07. 2010).

[Bil+11] Igor Bilogrevic u. a. „Privacy-preserving activity scheduling on mobile devices“. In: *CODASPY*. Hrsg. von Ravi S. Sandhu und Elisa Bertino. San Antonio, TX, USA: ACM, 2011, S. 261–272. ISBN: 978-1-4503-0466-5. DOI: 10.1145/1943513.1943549.

[Bis09] Joachim Biskup. *Security in Computing Systems: Challenges, Approaches and Solutions*. Springer, 2009. ISBN: 978-3-540-78441-8.

[Bor] Guilhem Borghesi. *STUdS*. URL: https://studs.u-strasbg.fr/index.php (besucht am 18. 07. 2011).

[CBT94] Josh Cohen Benaloh und Dwight Tuinstra. „Receipt-free secret-ballot elections (extended abstract)“. In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. Montreal, Quebec, Canada: ACM, 1994, S. 544–553. ISBN: 0-89791-663-8. DOI: 10.1145/195058.195407.

[CBY86] Josh Cohen Benaloh und Moti Yung. „Distributing the power of a government to enhance the privacy of voters“. In: *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*. Calgary, Alberta, Canada: ACM, 1986, S. 52–62. ISBN: 0-89791-198-9. DOI: 10.1145/10590.10595.

[CC97] Lorrie Faith Cranor und Ron K. Cytron. *Sensus: A security-conscious electronic polling system for the Internet*. 1997. URL: http://citeseer.ist.psu.edu/cranor97sensus.html.

[CF85] Josh D. Cohen und Michael J. Fischer. „A robust and verifiable cryptographically secure election scheme“. In: *SFCS '85: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. Washington, DC, USA: IEEE Computer Society, 1985, S. 372–382. ISBN: 0-8186-0844-4. DOI: 10.1109/SFCS.1985.2.

[CGS97] Ronald Cramer, Rosario Gennaro und Berry Schoenmakers. „A Secure and Optimally Efficient Multi-Authority Election Scheme“. In: *Advances in Cryptology — EUROCRYPT '97*. Hrsg. von Walter Fumy. Bd. 1233. Lecture Notes in Computer Science. Springer, 1997, S. 103–118. DOI: 10.1007/3-540-69053-0_9.

[Cha81] David L. Chaum. „Untraceable electronic mail, return addresses, and digital pseudonyms“. In: *Commun. ACM* 24.2 (1981), S. 84–90. ISSN: 0001-0782. DOI: 10.1145/358549.358563.

[Cha85] David L. Chaum. „Security Without Identification: Transaction Systems to Make Big Brother Obsolete“. In: *Commun. ACM* 28.10 (1985), S. 1030–1044. ISSN: 0001-0782. DOI: 10.1145/4372.4373.

[Cha88a] David L. Chaum. „Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA“. In: *Advances in Cryptology-EUROCRYPT'88*. Bd. 330. Lecture Notes in Computer Science. Davos, Switzerland: Springer, 1988, S. 177–182. ISBN: 0-387-50251-3.

[Cha88b] David L. Chaum. „The dining cryptographers problem: Unconditional sender and recipient untraceability“. In: *Journal of Cryptology* 1.1 (Jan. 1988), S. 65–75. ISSN: 0933-2790 (Print) 1432-1378 (Online). DOI: 10.1007/BF00206326. URL: http://www.springerlink.com/content/m74414x28822u525/.

[Cla+08] Michael R. Clarkson u. a. „Civitas: Toward a Secure Voting System“. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2008, S. 354–368. DOI: 10.1109/SP.2008.32.

[CNM83] Stuart K. Card, Allen Newell und Thomas P. Moran. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1983. ISBN: 0898592437.

[Cru] Matthew Crumley. *JavaScript BigInteger Library Version 0.9*. URL: http://silentmatt.com/biginteger/ (besucht am 10. 07. 2010).

[DH76] Whitfield Diffie und Martin E. Hellman. „New Directions in Cryptography“. In: *IEEE Transactions on Information Theory* IT-22.6 (Nov. 1976), S. 644–654.

[Dij] Sander Dijkhuis. *Pleft*. URL: https://www.pleft.com (besucht am 18. 07. 2011).

[DuR99] Brandon William DuRette. *Multiple Administrators for Electronic Voting. Bachelor's thesis, Massachusetts Institute of Technology*. Mai 1999. URL: http://theory.lcs.mit.edu/~cis/theses/DuRette-bachelors.pdf.

[Eck10] Peter Eckersley. „How Unique Is Your Web Browser?“ In: *Privacy Enhancing Technologies*. Hrsg. von Mikhail J. Atallah und Nicholas J. Hopper. Bd. 6205. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2010, S. 1–18. ISBN: 978-3-642-14526-1. DOI: 10.1007/978-3-642-14527-8.

[EG85] Taher El Gamal. „A public key cryptosystem and a signature scheme based on discrete logarithms“. In: *Proceedings of CRYPTO 84 on Advances in cryptology*. Santa Barbara, California, United States: Springer, 1985, S. 10–18. ISBN: 0-387-15658-5.

[Ele] Electronic Frontier Foundation. *Panopticlick*. URL: https://panopticlick.eff.org/ (besucht am 24. 03. 2011).

[Fin] Ryan Finley. *SurveyMonkey*. URL: http://www.surveymonkey.com (besucht am 03. 11. 2010).

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto und Kazuo Ohta. „A Practical Secret Voting Scheme for Large Scale Elections“. In: *AUSCRYPT*. Hrsg. von Jennifer Seberry und Yuliang Zheng. Bd. 718. Lecture Notes in Computer Science. Springer, 1992, S. 244–251. ISBN: 3-540-57220-1.

[FP97] Hannes Federrath und Andreas Pfitzmann. „Bausteine zur Realisierung mehrseitiger Sicherheit“. In: *Mehrseitige Sicherheit in der Kommunikationstechnik. 2 Bände*. Hrsg. von Günter Müller und Andreas Pfitzmann. Informationssicherheit. Bonn: Addison-Wesley-Longman, 1997, S. 83–104. URL: http://epub.uni-regensburg.de/7407/.

[Fra+02] M. S. Franzin u. a. *Multi-agent Meeting Scheduling with Preferences: Efficiency, Privacy Loss, and Solution Quality*. Techn. Ber. WS-02-13. American Association for Artificial Intelligence, 2002.

[FS] Tom Frey und Dominik Skrobala. *moreganize*. URL: http://moreganize.ch (besucht am 03. 11. 2010).

[Gre+06] Rachel Greenstadt u. a. „Experimental Analysis of Privacy Loss in DCOP Algorithms“. In: *AAMAS*. Hrsg. von Hideyuki Nakashima u. a. New York: ACM, 2006, S. 1424–1426. ISBN: 1-59593-303-4. DOI: 10.1145/1160633.1160899.

[Her+01] Thomas Herlea u. a. „On Securely Scheduling a Meeting“. In: *SEC*. Hrsg. von Michel Dupuy und Pierre Paradinas. Bd. 193. IFIP Conference Proceedings. Kluwer, 2001, S. 183–198. ISBN: 0-7923-7389-8.

[Her97] Mark A. Herschberg. „Secure Electronic Voting Over the World Wide Web“. Magisterarb. Massachusetts Institute of Technology, Mai 1997. URL: http://groups.csail.mit.edu/cis/voting/herschberg-thesis/index.html.

[HS00] Martin Hirt und Kazue Sako. „Efficient receipt-free voting based on homomorphic encryption“. In: *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*. EUROCRYPT'00. Bruges, Belgium: Springer, 2000, S. 539–556. ISBN: 3-540-67517-5. URL: http://portal.acm.org/citation.cfm?id=1756169.1756222.

[IEEE1363.2] IEEE P1363 Working Group. „IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques“. In: *IEEE Std 1363.2-2008* (2009), S. 1–127. DOI: 10.1109/IEEESTD.2009.4773330.

[ISO11770-3] International Organisation for Standardisation. *ISO/IEC 11770-3:2008: Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques*. Norm. Juli 2008.

[Jak98] Markus Jakobsson. „A Practical Mix“. In: *EUROCRYPT*. Bd. 1403. Lecture Notes in Computer Science. Springer, 1998, S. 448–461.

[JCJ05] Ari Juels, Dario Catalano und Markus Jakobsson. „Coercion-resistant electronic elections“. In: *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. WPES '05. Alexandria, VA, USA: ACM, 2005, S. 61–70. ISBN: 1-59593-228-3. DOI: 10.1145/1102199.1102213.

[Kam10] Samy Kamkar. *evercookie – virtually irrevocable persistent cookies*. Okt. 2010. URL: http://samy.pl/evercookie/ (besucht am 24. 03. 2011).

[KB09] Benjamin Kellermann und Rainer Böhme. „Privacy-Enhanced Event Scheduling“. In: *Proceedings of IEEE International Conference on Computational Science and Engineering*. Bd. 3. Conference on Information Privacy, Security, Risk and Trust (PASSAT '09). IEEE/IFIP. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2009, S. 52–59. ISBN: 978-0-7695-3823-5. DOI: 10.1109/CSE.2009.270.

[Kel09] Benjamin Kellermann. „Datenschutzfreundliche Terminplanung“. In: *Proceedings of the 26th Chaos Communication Congress*. Hrsg. von Matthias Mehldau. Chaos Computer Club. Marktstraße 18, 33602 Bielefeld: Art d'Ameublement, Dez. 2009, S. 207–211. ISBN: 978-3-934636-08-8.

[Kel11] Benjamin Kellermann. „Privacy-Enhanced Web-Based Event Scheduling with Majority Agreement“. In: *Proceedings of SEC 2011 – Future Challenges in Security and Privacy for Academia and Industry*. Hrsg. von Jan Camenisch u. a. Bd. 354. IFIP Advances in Information and Communication Technology. Springer, Juni 2011, S. 235–246. ISBN: 987-3-642-21423-3. DOI: 10.1007/987-3-642-21424-0.

[Kel+98] John Kelsey u. a. „Secure Applications of Low-Entropy Keys“. In: *ISW*. Hrsg. von Eiji Okamoto, George I. Davida und Masahiro Mambo. Bd. 1396. Lecture Notes in Computer Science. London, UK: Springer, Sep. 1998, S. 121–134. ISBN: 3-540-64382-6. DOI: 10.1007/BFb0030415.

[Kle+] Albert de Klein u. a. *Sort of Date*. URL: http://sortofdate.com (besucht am 03. 11. 2010).

[Lew82] C. Lewis. *Using the thinking-aloud method in cognitive interface design.* Techn. Ber. IBM Research Report RC 9265. IBM, Yorktown Heights, NY, 1982.

[LF09] Thomas Léauté und Boi Faltings. „Privacy-Preserving Multi-agent Constraint Satisfaction". In: *Proceedings of IEEE International Conference on Computational Science and Engineering.* Bd. 3. Conference on Information Privacy, Security, Risk and Trust (PASSAT '09). IEEE/IFIP. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2009, S. 17–25. ISBN: 978-0-7695-3823-5. DOI: 10.1109/CSE.2009.169.

[Mad] Made Media Ltd. *Diarised.* URL: http://www.diarised.com (besucht am 03. 11. 2010).

[Mah+04] Rajiv T. Maheswaran u. a. „Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling". In: *AAMAS.* IEEE Computer Society, 2004, S. 310–317. ISBN: 1-58113-864-4.

[Mee] Meetomatic Ltd. *Meet-O-Matic: The World's Simplest Meeting Scheduler.* URL: http://www.meetomatic.com (besucht am 03. 11. 2010).

[Mil68] Robert B. Miller. „Response time in man-computer conversational transactions". In: *Proceedings of the fall joint computer conference, part I.* AFIPS '68 (Fall, part I). San Francisco, California: ACM, Dez. 1968, S. 267–277. DOI: 10.1145/1476589.1476628.

[ML04] Roger Mailler und Victor Lesser. „Solving Distributed Constraint Optimization Problems Using Cooperative Mediation". In: *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems.* New York, New York: IEEE Computer Society, 2004, S. 438–445. ISBN: 1-58113-864-4. DOI: 10.1109/AAMAS.2004.249.

[Mod+04] Pragnesh Jay Modi u. a. „ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees". In: *Artificial Intelligence* 161 (2004), S. 149–180.

[MOV97] Alfred J. Menezes, Paul C. van Oorschot und Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press, 1997. ISBN: 9780849385230.

[MT79] Robert Morris und Ken Thompson. „Password Security: A Case History". In: *Communications of the ACM* 22.11 (Nov. 1979), S. 594–597.

[Näf] Michael Näf. *Doodle: easy scheduling.* URL: http://www.doodle.com (besucht am 03. 11. 2010).

[Net] Net Applications. *Browser market share news.* URL: http://marketshare.hitslink.com/browser-market-share.aspx?qprid=2\&qpcal=1\&qptimeframe=M\&qpsp=145 (besucht am 21. 02. 2011).

[New90] Allen Newell. *Unified theories of cognition.* Cambridge, MA, USA: Harvard University Press, 1990. ISBN: 0-674-92099-6.

[Oga+97] Wakaha Ogata u. a. „Fault tolerant anonymous channel“. In: *ICICS*. Hrsg. von Yongfei Han, Tatsuaki Okamoto und Sihan Qing. Bd. 1334. Lecture Notes in Computer Science. Springer, 1997, S. 440–444. ISBN: 3-540-63696-X.

[Ohk+99] Miyako Ohkubo u. a. „An Improvement on a Practical Secret Voting Scheme“. In: *ISW*. Hrsg. von Masahiro Mambo und Yuliang Zheng. Bd. 1729. Lecture Notes in Computer Science. Springer, 1999, S. 225–234. ISBN: 3-540-66695-8.

[PIK94] Choonsik Park, Kazutomo Itoh und Kaoru Kurosawa. „Efficient Anonymous Channel and All/Nothing Election Scheme“. In: *EUROCRYPT*. Lofthus, Norway: Springer, 1994, S. 248–259. ISBN: 3-540-57600-2. DOI: 10.1007/3-540-48285-7_21.

[Pro] Professional Software Engineering Ltd. *agreeAdate*. URL: http://www.agreeadate.com (besucht am 03. 11. 2010).

[PW92] Birgit Pfitzmann und Michael Waidner. *Waidner: Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election; Hildesheimer Informatik-Berichte 3/92,* Techn. Ber. Apr. 1992, S. 7–8.

[RFC2898] Burton S. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898 (Informational). Internet Engineering Task Force, Sep. 2000. URL: http://www.ietf.org/rfc/rfc2898.txt.

[RFC3986] Tim Berners-Lee, Roy Fielding und Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (Standard). Internet Engineering Task Force, Jan. 2005. URL: http://www.ietf.org/rfc/rfc3986.txt.

[RFC4648] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648 (Proposed Standard). Internet Engineering Task Force, Okt. 2006. URL: http://www.ietf.org/rfc/rfc4648.txt.

[RS07] Ronald L. Rivest und Warren D. Smith. „Three voting protocols: ThreeBallot, VAV, and twin“. In: *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*. Boston, MA: USENIX Association, 2007, S. 16–16. URL: http://portal.acm.org/citation.cfm?id=1323111.1323127.

[Sak94] Kazue Sako. „Electronic Voting Scheme Allowing Open Objection to the Tally“. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 77.1 (1994), S. 24–30.

[Sch] Carsten Schmitz. *LimeService*. URL: http://www.limeservice.com (besucht am 03. 11. 2010).

[Sch96] Bruce Schneier. *Applied Cryptography*. 2. Aufl. John Wiley & Sons, Inc., 1996. ISBN: 0-471-11709-9.

[SDV] Andreas Åkre Solberg und DFN-Verein. *Terminplaner*. URL: `http://terminplaner.dfn.de` (besucht am 03. 11. 2010).

[Sha] David Shapiro. *BigInt, a suite of routines for performing multiple-precision arithmetic in JavaScript*. URL: `http://ohdave.com/rsa/BigInt.js` (besucht am 10. 07. 2010).

[SHB09] Emily Stark, Michael Hamburg und Dan Boneh. „Symmetric Cryptography in Javascript“. In: *Computer Security Applications Conference, Annual* 0 (2009), S. 373–381. ISSN: 1063-9527. DOI: 10.1109/ACSAC.2009.42.

[SK94] Kazue Sako und Joe Kilian. „Secure Voting Using Partially Compatible Homomorphisms“. In: *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer, 1994, S. 411–424. ISBN: 3-540-58333-5.

[SK95] Kazue Sako und Joe Kilian. „Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth“. In: *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*. EUROCRYPT'95. Saint-Malo, France: Springer, 1995, S. 393–403. ISBN: 3-540-59409-4. URL: `http://portal.acm.org/citation.cfm?id=1755009.1755052`.

[Sof] Softly Software. *WhenIsGood*. URL: `http://whenisgood.net` (besucht am 03. 11. 2010).

[Sol] Andreas Åkre Solberg. *Foodle*. URL: `https://foodl.org` (besucht am 03. 11. 2010).

[SSHF00] Marius-Calin Silaghi, Djamila Sam-Haroud und Boi Faltings. „Asynchronous Search with Aggregations“. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press / The MIT Press, 2000, S. 917–922. ISBN: 0-262-51112-6.

[Tim] TimeBridge Inc. *TimeBridge | Run Great Meetings*. URL: `http://timebridge.com` (besucht am 03. 11. 2010).

[Tun] Tungle Corporation. *Tungle.me | Scheduling Made Easy*. URL: `http://www.tungle.com` (besucht am 03. 11. 2010).

[WP00] Gritta Wolf und Andreas Pfitzmann. „Charakteristika von Schutzzielen und Konsequenzen für Benutzungsschnittstellen“. In: *Informatik-Spektrum* 23 (3 2000), S. 173–191. ISSN: 0170-6012. DOI: 10.1007/s002870000101.

[Wu] Tom Wu. *BigIntegers and RSA in JavaScript*. URL: `http://www-cs-students.stanford.edu/~tjw/jsbn/` (besucht am 10. 07. 2010).

[Yan+04] Jeff Jianxin Yan u. a. „Password Memorability and Security: Empirical Results“. In: *IEEE Security & Privacy* 2.5 (2004), S. 25–31. DOI: 10.1109/MSP.2004.81.

[YH00] Makoto Yokoo und Katsutoshi Hirayama. „Algorithms for Distributed Constraint Satisfaction: A Review“. In: *Autonomous Agents and Multi-Agent Systems* 3 (2 Juni 2000), S. 185–207. ISSN: 1387-2532. DOI: 10.1023/A:1010078712316. URL: http://portal.acm.org/citation.cfm?id=608603.608652.

A. Entropie eines Verfügbarkeitsvektors

Im Folgenden findet sich eine Beispielrechnung sowie die allgemeine Berechnung der Entropie eines Verfügbarkeitsvektors unter der Annahme, alle Verfügbarkeiten wären voneinander unabhängig.

A.1. Beispiel

Angenommen eine Umfrage hat drei Teilnehmer mit folgenden Verfügbarkeitsvektoren:

	t_1	t_2	t_3	t_4
Alice	0	1	0	0
Bob	1	1	0	1
Carol	0	1	0	1
$\sum$	1	3	0	2

Die Entropie jedes Verfügbarkeitsvektors jeder Person beträgt $4 \cdot \mathrm{ld}\,(2) = 4\,\mathrm{Bit}$, die der ganzen Matrix $3 \cdot 4\,\mathrm{Bit} = 12\,\mathrm{Bit}$. Die Entropie der Summe aller Verfügbarkeitsvektoren beträgt $4 \cdot \mathrm{ld}\,(4) = 4 \cdot 2\,\mathrm{Bit} = 8\,\mathrm{Bit}$. Der Informationsverlust durch die Summation entspricht $12\,\mathrm{Bit} - 8\,\mathrm{Bit} = 4\,\mathrm{Bit}$.

Die Entropie kann weiter verringert werden, wenn nur die Zeitpunkte ausgegeben werden, an denen eine bestimmte Anzahl an Teilnehmern verfügbar ist. Würde der Schwellwert in diesem Beispiel mit 2 Personen angegeben, so würden nur die Spalten von t_2 und t_4 ausgegeben. Die Entropie der Ausgabe würde dann $3 \cdot 2\,\mathrm{Bit} = 6\,\mathrm{Bit}$ betragen, was einem Informationsverlust von 6 Bit entspricht.

A.2. Allgemeine Berechnung

Die Entropie $H\,(\cdot)$ eines Verfügbarkeitsvektors $\vec{\phi}_u$ berechnet sich allgemein wie folgt:

$$\begin{aligned} H\left(\vec{\phi}_u\right) &= H\left(\phi_u^{t_1}, \ldots, \phi_u^{t_{|T|}}\right) \\ &= |T| \cdot \mathrm{ld}\,(2) \\ &= |T|\,\mathrm{Bit}. \end{aligned} \tag{A.1}$$

Die Entropie der Verfügbarkeitsvektoren aller Teilnehmer einer Umfrage ist daher

$$\begin{aligned} H\left(\vec{\phi}_{u_1}, \ldots, \vec{\phi}_{u_{|U|}}\right) &= |U| \cdot |T| \cdot \mathrm{ld}\,(2) \\ &= |U| \cdot |T|\,\mathrm{Bit}. \end{aligned} \tag{A.2}$$

Die Entropie der Summe aller Verfügbarkeitsvektoren einer Umfrage beträgt allgemein

$$H\left(\vec{\sigma}\right) = |T| \cdot \mathrm{ld}\left(|U| + 1\right). \tag{A.3}$$

Der Informationsverlust durch die Aggregation *Summieren* ist daher

$$\begin{aligned} H\left(\vec{\phi}_{u_1}, \ldots, \vec{\phi}_{u_{|U|}}\right) - H\left(\vec{\sigma}\right) &= |U| \cdot |T| \cdot \mathrm{ld}\left(2\right) - |T| \cdot \mathrm{ld}\left(|U| + 1\right) \\ &= |T| \cdot \left(|U|\,\mathrm{Bit} - \mathrm{ld}\left(|U| + 1\right)\right). \end{aligned} \tag{A.4}$$

B. Stimmenabgabe (min. Vertrauen)

Stimmenabgabe(u, uuid)

bekannte Werte:

u ... der Teilnehmer selbst	uuid ... Umfrage-ID
n ... DC-Netz-Modulus	$\left\{\phi_u^t \middle\vert t \in T^{\mathsf{uuid}}\right\}$ ... eigene Verfügbarkeiten
q ... Diffie–Hellman-Modulus	sec_u ... eigener geheimer DH-Schlüssel

hole U^{uuid}, T^{uuid} und ℓ vom Umfrageserver

$u' \in U^{\mathsf{uuid}} \setminus \{u\}$ /* jeder andere Teilnehmer */

- hole $\mathsf{pub}_{u'}$ vom Server /* öffentlicher Diffie-Hellman-Schlüssel */
- $\mathsf{dh}_{u,u'} := (\mathsf{pub}_{u'})^{\mathsf{sec}_u} \bmod q$ /* Diffie-Hellman-Geheimnis */
- $t \in T^{\mathsf{uuid}}$ /* jeder Zeitpunkt */
  - $\phi_u^{(t,0)} := \phi_u^t$ /* normale Abstimmung */
  - $\phi_u^{(t,1)} := 1 - \phi_u^{(t,0)}$ /* Prüfabstimmung */
  - $\lambda \in \{0,1\}$ /* normale + Prüfabstimmung */
    - wähle zufällig $r \in \mathbb{Z}_\ell$ /* wähle DC-Netz */
    - $\varphi_u^{(t,r,\lambda)} := \phi_u^{(t,\lambda)}$ /* setze eigene Verfügbarkeit in das gewählte DC-Netz */
    - $i \in \mathbb{Z}_\ell \setminus \{r\}$ /* verbleibende DC-Netze */
      - $\varphi_u^{(t,i,\lambda)} := 0$
    - $i \in \mathbb{Z}_\ell$ /* alle DC-Netze */
      - $\delta := (t, i, \lambda)$ /* DC-Netz-Runde */
      - $d_u^\delta = \mathtt{undefined}$
        - `true`:
          - $d_u^\delta := \varphi_u^\delta$ /* initialisieren */
        - `false`:
      - $u < u'$
        - `true`:
          - $k_{u,u'}^\delta := \mathrm{decr}_{\mathsf{dh}_{u,u'}}^{\mathrm{sym}}(\delta, \mathsf{uuid})$
        - `false`:
          - $k_{u,u'}^\delta := -\mathrm{decr}_{\mathsf{dh}_{u,u'}}^{\mathrm{sym}}(\delta, \mathsf{uuid}) \bmod n$
      - $d_u^\delta := d_u^\delta + k_{u,u}^\delta \bmod n$ /* addiere Schlüssel auf verschlüsselte Stimme */
      - $\psi_{u,u'}^\delta := \mathrm{commit}\left(k_{u,u'}^\delta\right)$ /* Commitment für Schlüssel */

sende $\left\{d_u^\delta, \psi_{u,u'}^\delta \middle\vert \delta \in \Delta, u' \in U^{\mathsf{uuid}} \setminus \{u\}\right\}$ signiert zum Server

C. Ergebnisveröffentlichung (min. Vertrauen)

Ergebnisveröffentlichung(u)

bekannte Werte:
u ... der Teilnehmer selbst
$\left\{\varphi_u^\delta | \delta \in \Delta\right\}$ ... eigene gesendete Werte

$\left\{d_{u'}^\delta | \delta \in \Delta, u' \in U^{\mathsf{uuid}}\right\}$ vom Server holen `/* alle verschlüsselte Stimmen */`

$U := U^{\mathsf{uuid}}$

$\delta \in \Delta$ `/* jede DC-Netz Runde */`

$\sum_{u' \in U} d_{u'}^\delta \in \{0, \ldots, |U|\}$

`true` | `false`

$\mathcal{V}(\boldsymbol{d})$ schlug fehl

$t \in T^{\mathsf{uuid}}$ `/* jeder Zeitpunkt */`

$|U| = \sum_{u \in U, i \in \mathbb{Z}_\ell, \lambda \in \{0,1\}} d_u^{(t,i,\lambda)}$

`true` | `false`

$\mathcal{C}(\boldsymbol{d})$ schlug fehl

$\delta \in \Delta$ `/* jede DC-Netz Runde */`

$\varphi_u^\delta = 1 \wedge \sum_{u' \in U} d_{u'}^\delta = 0$

`true` | `false`

$\mathcal{B}(\boldsymbol{d}, \boldsymbol{k}_u)$ schlug fehl

$t \in T^{\mathsf{uuid}}$ `/* jeder Zeitpunkt */`

$\sigma^t := \sum_{u' \in U, \delta \in \Delta} d_{u'}^\delta$ `/* Ergebnis */`

Ausgabe: $\left\{\sigma^t | t \in T^{\mathsf{uuid}}\right\}$

D. Schlüsseltransportmechanismus 2 nach ISO/EIC 1170-3

Seien u_a, u_b zwei Teilnehmer, die einen Schlüssel mittels des Schlüsseltransportmechanismus 2 des ISO/EIC 11770-3 Standards [ISO11770-3] austauschen wollen. Sei ts ein Zeitstempel, $\mathrm{enc}_u(x)$ die für Teilnehmer u verschlüsselte Nachricht x und $\mathrm{sig}_u(x)$, eine digitale Signatur von Teilnehmer u für die Nachricht x. Die Nachricht, die Teilnehmer u_a an Teilnehmer u_b schickt ist wie folgt gebildet:

$$u_a \rightarrow u_b : u_b, \mathsf{ts}, \mathrm{enc}_{u_b}(u_a, k_{u_a,u_b}), \mathrm{sig}_{u_a}(u_b, \mathsf{ts}, \mathrm{enc}_{u_b}(u_a, k_{u_a,u_b})) \tag{D.1}$$

Der Zeitstempel ts kann laut Standard durch eine Zählvariable ersetzt werden.