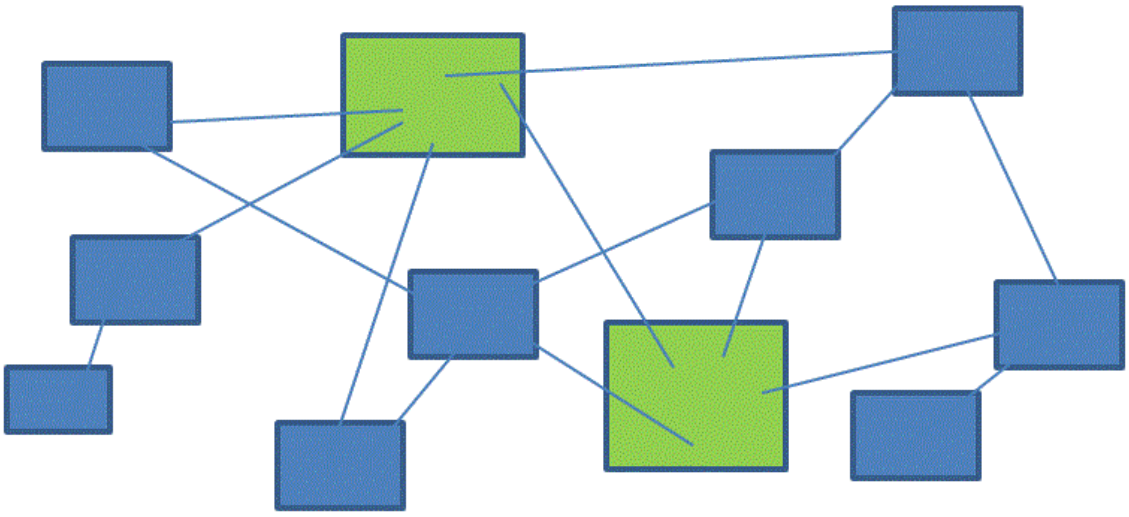
Blockchain Voting System

Vorschlag für ein

System zur Durchführung allgemeiner Wahlen über das Internet

Louis Göttertz

Version 1.4



1. Einleitung

Die größte Herausforderung ist es, das Vertrauen der Bürger zu erwerben. Die Auszählung der Stimmen in einem Wahllokal ist für jeden nachvollziehbar, die Speicherung der Stimme in einem Zentralcomputer nicht. (Johann Hahlen, Bundeswahlleiter und Präsident des Statistischen Bundesamtes, am 18. September 2001 auf dem Deutschen Internet-Kongress in Karlsruhe.)

In vielen westlichen Demokratien sinkt das Vertrauen großer Teile der Bevölkerung in das demokratische System. Die Gründe sind meist vielfältig und in den ökonomischen und sozialen Folgen der Globalisierung zu suchen. Eine Quelle des Misstrauens ist aber auch dort zu suchen, wo für den Bürger undurchschaubare komplexe digitale Prozesse die althergebrachten z.B. bürokratischen Verfahren ersetzen. Dort wo elektronische Wahlverfahren eingesetzt werden sind das Misstrauen der Bevölkerung und der Widerstand gegen die Einführung elektronischer Wahlverfahren meist weit verbreitet (Beispiele siehe https://papierwahl.at/).

Internetwahl-Systeme unterliegen von Gesetzes wegen besonderen hohen Anforderungen. Sie müssen zunächst den allgemeinen Grundsätzen einer demokratischen Wahl genügen und sie müssen vertrauenswürdig sein, sowie technisch zuverlässig funktionieren.

Das Bundesverfassungsgericht urteilte 2009:

„dass der Einsatz elektronischer Wahlgeräte voraussetzt, dass die wesentlichen Schritte der Wahlhandlung und der Ergebnisermittlung vom Bürger zuverlässig und ohne besondere Sachkenntnis überprüft werden können." (Pressestelle Bundesverfassungsgericht, 2009)

Die Tatsache, dass schon die einfachen Wahlcomputer, die – wie man meint – unter kontrollierten Bedingungen hergestellt und betrieben wurden, vom Bundesverfassungsgericht als nicht zulässig gewertet worden sind, betont die hohen Sicherheitsanforderungen, die an Internetwahl-Systeme zu stellen sind, wollen sie ernsthaft bei demokratischen Wahlen eine Alternative zur Briefwahl darstellen.

Die Infrastruktur, um demokratische Regierungswahlen in einem Land abzuhalten, gehört ohne Frage zu den kritischen Infrastrukturen eines Landes, wenn Onlinewahlen

erst einmal etabliert sind. Die Frage, „Wer regiert in welchem Interesse?“ ist eine der wichtigsten Fragen in einer Demokratie. Daher ist das Interesse z.B. an Manipulation einer Parlamentswahl auf Staatsebene möglicherweise groß. Nicht erst das Bekanntwerden des Ausmaßes der Geheimdienstaktivitäten verschiedener Länder[1] beweist, dass mögliche Versuche der Einflussnahme auf Regierungsbildungen durch Wahlmanipulation bei Online-Wahlen längst im Bereich realistischer Bedrohungen liegen. Auch die Berichte über möglicherweise staatlich gelenkte Hackerangriffe gegen ausländische Infrastruktur (meist werden Russland oder China als potentielle Urheber genannt) geben Anlass zur Besorgnis.[2] So gilt es bei einem neuen System für Online-Wahlen nicht nur die bekannten Fehler bisheriger Systeme zu vermeiden, sondern auch ein System zu schaffen, welches für zukünftige, noch unbekannte Bedrohungen gewappnet ist und sie möglichst systematisch vollständig ausschließen kann (Systembedingte Robustheit gegenüber Angriffen).

Während bei einer Wahl mit Stimmzetteln Manipulationen oder Wahlfälschungen unter den Rahmenbedingungen der geltenden Vorschriften jedenfalls nur mit erheblichem Einsatz und einem hohen Entdeckungsrisiko möglich sind, sind Programmierfehler in der Serversoftware, fehlerhafte Implementation oder zielgerichtete Wahlfälschungen durch Manipulation der Software oder der Datenbanken bei Server-Client-Systemen nur schwer erkennbar.

Die bisherigen Internetwahl-Systeme beruhten auf Server-Client Architekturen, die systembedingte Schwachstellen haben:

- Server können sehr leicht fehlerhaft implementiert werden,
- Das Fehlen von angemessenen, sicheren Prozeduren für die alltägliche Wartung/Sicherung der Wahlserver oder das Fehlen von Prozeduren zum Umgang mit Anomalien oder deren Nichtbeachtung z.B. aus Zeit- oder Kostengründen.
- zentrale Server können außerdem von außen z.B. mittels Bot-Netzen angegriffen werden oder
- mittels Schadsoftware kompromittiert werden.

---

[1] Siehe Tagesanzeiger (Schweiz): NSA-Affäre verstärkt Misstrauen in E-Voting, http://www.tagesanzeiger.ch/schweiz/standard/NSAAffaere-verstaerkt-Misstrauen-in-EVoting/story/20525542, 4.1.2013, zuletzt abgerufen 11.06.2016

[2] Siehe den Angriff auf die Bundestags-IT im Mai 2015, der bis heute (Anfang September) nicht abgewehrt werden konnte.

Die Situation für die Clients sieht bisher nicht besser aus: Die Clients könnten auf unsicheren Endgeräten installiert sein oder auch per Schadsoftware kompromittiert werden. Man-in-the Middle-Angriffe können für eine Übermittlung falscher Stimmabgaben verantwortlich sein, oder Wahlentscheidungen ausspionieren. Die Beispiele für Sicherheitslücken der Internetwahl-Systeme bei den Wahlen in Norwegen[3], Norwegen[3], Estland[4] und Australien[5] zeigen, dass es trotz aller Bemühungen, diesen systematischen Problemen Rechnung zu tragen, immer wieder Sicherheitslücken bei Online-Wahlen aufgetreten sind, die die Legitimation dieser Art von Wahlen ernsthaft in Frage stellen. Problematisch ist bei den erwähnten Wahlen außerdem, dass ausschließlich proprietäre Software privater Firmen verwendet wurde, was nicht nur die Überprüfung der Software durch externe, unabhängige Experten erschwert, sondern vor allem auch das Vertrauen der Wähler in das System verhindert.

Nicht nur Verschwörungstheoretikern fehlt das Vertrauen in die Produkte gewinnorientierter oder staatlicher Firmen in diesem höchst sensiblen Bereich: Die Gewährleistung von Anonymität bei gleichzeitiger eindeutiger Identifizierung und Authentifizierung ist daher nicht nur eine starke technische sondern auch eine ebenso starke organisatorische Herausforderung. Die meisten Staaten haben deshalb bisher auf flächendeckenden Einsatz von Online-Wahlsystemen verzichtet: Norwegen hat ein schon gestartetes Projekt zur Onlinewahl mitten in den Vorbereitungen abgebrochen. In den Ländern, in denen über die Einführung von Online-Wahlsystemen nachgedacht wurde, formierte sich oft schon bei Bekanntwerden der Pläne Widerspruch in der Bevölkerung -besonders bei den sonst so internetaffinen sogenannten Netz- und Digital-Rights-Aktivisten.[6]

Neben dem Vertrauen in die Sicherheit und Integrität von Online-Wahlsystemen werden als Nachteile benannt, dass Menschen mit geringen Computerkenntnissen, oder

---

3 Siehe: The rise and fall of Internet voting in Norway, Vortrag auf dem 31. Chaos Computer Congress, URL: https://events.ccc.de/congress/2014/Fahrplan/events/6213.html, zuletzt abgerufen am 11.06.2016.

4 Siehe: Independent Report on E-voting in Estonia: https://estoniaevoting.org/, zuletzt abgerufen am 11.06.2016.

5 Siehe: New South Wales Attacks Researchers Who Found Internet Voting Vulnerabilities, URL: https://www.eff.org/deeplinks/2015/04/new-south-wales-attacks-researchers-who-warned-internet-voting-vulnerabilities, zuletzt abgerufen am 11.06.2016

6 Eine reichhaltige Linksammlung zur Kritik an Online-Wahlen findet man unter: http://papierwahl.at.

ohne Zugang zum Internet benachteiligt würden. Der Kritikpunkt ist zwar nicht ganz unberechtigt, vor allem dort, wo Abstimmungen und Wahlen *ausschließlich* als Wahlen stattfinden sollen. Wenn aber die Onlinewahl zusätzlich zu den bisherigen Wahlmöglichkeiten angeboten wird, sticht dieses Argument weniger.

Die Motivation für die Entwicklung eines neuen Systems für die Durchführung von Online-Wahlen, liegt in den Vorteilen, die ein solches System bietet, sofern Transparenz-, Sicherheits- und organisatorische Probleme zufriedenstellend gelöst sind:

1. Neue Möglichkeiten demokratischer Partizipation, und dadurch Steigerung der politischen Einflussnahme
2. Zeit- und Ortsunabhängigkeit für Wähler bei Stimmabgabe -> Steigerung der Wahlbeteiligung
3. Langfristig Senkung der Kosten für Wahlen

Vor allem der zweite Punkt (Zeit- und Ortsunabhängigkeit der Wähler) erscheint mir sehr entscheidend; diese Funktion wird bei konventionellen Wahlen bisher vor allem durch die Möglichkeit der Briefwahl dargestellt. Bei Verwendung eines Onlinewahlsystems könnten Wahlen auch dort ermöglicht werden, wo konventionelle Wahlen nur unter sehr erschwerten Bedingungen durchgeführt werden können. Die Vorteile der Orts- und Zeitunabhängigkeit würde in Ländern mit schwacher Infrastruktur oder in Ländern, die unter Bürgerkrieg oder Terrorismus leiden, besonders zum Tragen kommen, da gerade dort wo es eine schwache oder zerstörte Infrastruktur gibt, die Nutzung des mobilen Internets in allen Bevölkerungsschichten schon sehr verbreitet ist und schneller zunimmt als in den entwickelten Industrieländern.[7]

Die Blockchain-Technologie, die mit der Erfindung der digitalen Kryptowährung Bitcoin, bekannt wurde, könnte das zentrale Problem der Transparenz und des Vertrauens bei Online-Wahlen lösen und andere Sicherheitsprobleme entschärfen. Im Gegensatz zu den bisher verwendeten Server-Client-Architekturen besteht der Kern der Blockchain-Technologie aus einer mittels Peer-To-Peer-Protokoll verteilten Datenbank, deren Integrität durch einen kryptografischen Hash-Algorithmus sichergestellt wird. Dadurch sind alle Vorgänge in dieser Datenbank für alle

[7] (International Telecommunication Union (ITU), 2015)

Teilnehmer zugänglich und transparent. Bezogen auf ein Wahlsystem hieße das, dass alle Stimmzuweisungen und Stimmabgaben sicher aufgezeichnet würden und jeder Zugriff auf diese Informationen hätte und darüber hinaus die Gültigkeit dieser Informationen gesichert sei. Jeder Wähler kann zum Schluss überprüfen: Wurde meine meine Stimme wie beabsichtigt zugeordnet? Wurde meine Stimme gezählt wie zugeordnet und werden alle Stimmen gezählt?[8]

Die Transparenz, die die Verwendung der Blockchaintechnologie bietet, ist ein Vorteil bezogen auf das Vertrauensproblem, jedoch auch ein Problem für die Durchführung von politischen Wahlen, bei denen u.a. die Anonymität und Geheimhaltung der Wahlergebnisse bis zum Ende der Wahl gewährleistet sein muss. Trotzdem erscheint mir die Blockchain-Technologie aufgrund der Robustheit einer verteilten Anwendung[9] und des enormen Vorteils des Vertrauens in dessen Korrektheit geeignet, als Basistechnologie für ein System zur Durchführung von Wahlen über das Internet vielversprechend, wenn es gelingt die Probleme, die sich z.B. aus der Transparenz der Blockchain ergeben, zu lösen.

Es gibt bereits zahlreiche Weiterentwicklungen von Bitcoin und anderen digitalen Währungen auf Blockchain-Basis, die viel weitergehende Funktionen auch abseits von digitalen Währungen haben und z.B. Intelligente (automatische) Verträge ermöglichen (Smart Contracts), sowie Werkzeuge für Voting, virtuelle Gesellschaften aller Art u.v.m. ermöglichen, deshalb bin ich überzeugt, dass die Blockchain-Technologie auch als Basis für ein „richtiges“ Online-Wahlsystem taugt – ein Wahlsystem, welches politische Wahlen nach demokratischen Standards ermöglicht und damit auch hierzulande – wenn gewollt - umsetzbar wäre.

[8] Vergl.: End-to-End (E2E) Voter-Verifiability (Halderman, 2015)., (Clark, 2011)
[9] Das Bitcoin-Netzwerk funktioniert seit 2009 ohne größere Probleme und Sicherheitslücken.

2. Anforderungen und Probleme eines Blockchain-basierten Wahlsystems

Sicherheit

Integrität

Der Vorteil der jederzeit überprüfbaren Integrität der Informationen in einer Blockchain soll in dem hier vorgeschlagenem Blockchain Voting System (BVS) genutzt werden, um alle relevanten Informationen und Variablen zu speichern, damit kein zusätzlicher Server notwendig ist der z.B. die Stimmzettel für die Clients bereit stellt, oder die Stimmzettel mitsamt der Wahloptionen und Ihrer Codierung auf den unsicheren Clients gespeichert werden müssen. Im BVS sollen sowohl die Daten für die Stimmzettel und Wahloptionen als auch die Stimmen der Wähler (Wahlentscheidungen) in Form von Transaktionen in der Blockchain gespeichert werden.

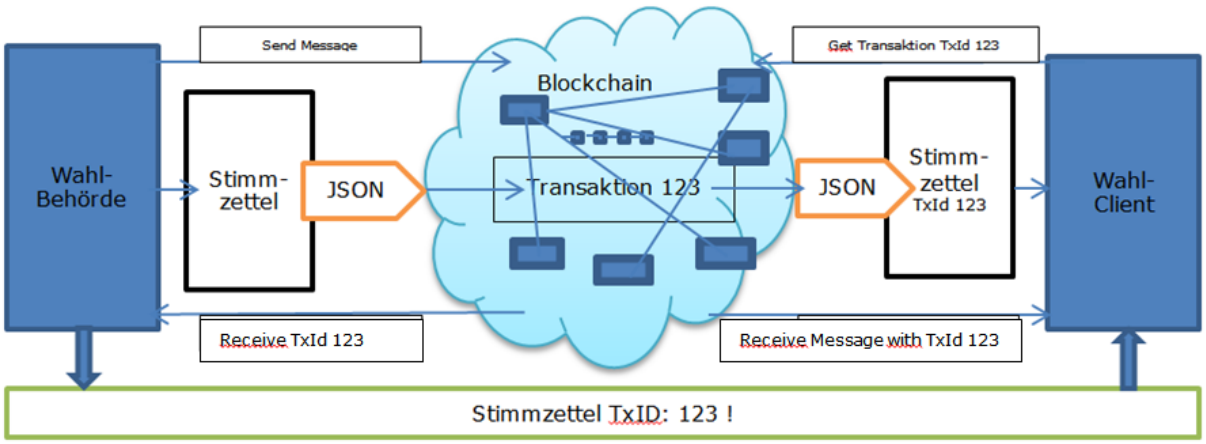


Abbildung 1: Speichern und Lesen der Stimmzettel in bzw. aus der Blockchain.

Im Fall der Stimmzettel, werden diese Daten einfach von den Clients aus der Blockchain geladen. Dazu muss lediglich den Clients die Transaktions-ID (TxId) bekannt sein, die den Stimmzettel enthält. Diese wird vor Beginn der Wahl von den Wahlbehörden bekannt gegeben. Die Gefahr, dass falsche Stimmzettel von einem manipulierten Server geladen werden, ist durch die Blockchain gebannt.[10]

Eine andere Anforderung in Bezug auf die Integrität einer Online-Wahl ist, dass zum Schluss das Wahlergebnis überprüfbar sein muss. Die notwendige Geheimhaltung bei

[10] Vergl.: Decker, et al., 2013

einer Wahl macht es jedoch schwer, ein Verfahren zu entwickeln, welches einerseits einfach für Wählerinnen und Wahlbeobachter zu handhaben ist und andererseits gewährleistet, dass einzelne Stimmen und das Wahlergebnis insgesamt überprüfbar sind. Diese Anforderung wird in der Fachliteratur als End-To-End Verifiability, abgekürzt E2E-V bezeichnet. End-To-End Verifiability wird in der englischen Fachliteratur auf eine kurze Formel gebracht:

The verification objectives can be summarized with the catchphrase, "Cast as intended; recorded as cast; and counted as recorded."[11]

Das bedeutet, dass überprüfbar sein muss:

1. Wurde der beabsichtigte Kandidat gewählt. Wenn beispielsweise die Kandidaten auf den Listen vertauscht würden, könnte ein Wähler unbeabsichtigt die falsche Wahl treffen.
2. Wurde die Stimme so übermittelt und gespeichert, wie gewählt. Durch Manipulationen bei der Übermittlung oder Speicherung können bei einem Online-Wahlsystem
3. wurde die Stimme auch so gewertet wie gespeichert.

Es wäre einfach, diese Anforderungen in einem blockchain-basiertem System zu erfüllen, gäbe es nicht das Wahlgeheimnis, wodurch eine offene Stimmabgabe durch Versenden von Coins oder Assets an Adressen von Kandidaten nicht möglich ist, wenn es um politische Wahlen geht.

Ein Problem dabei ist, dass eine offene Stimmabgabe dazu führt, dass diejenigen, die später wählen, durch Informationen über die bereits abgegebenen Stimmen einen Vorteil haben gegenüber denjenigen, die früher gewählt haben. Die späten Wähler könnten bei ihrer Wahlentscheidung durch diese Informationen beeinflusst werden, um z.B. taktisch zu wählen. Außerdem wäre es den Wählern, die ihre eigenen Adressen ja kennen, zu beweisen, was sie gewählt haben, was mit der Gefahr des Stimmenkaufs verbunden ist (siehe Abschnitt „Geheimhaltung").

Um diesem Problem zu begegnen ist es notwendig, die Stimmabgabe geheim zu halten. Dazu muss die einfachste Möglichkeit für Transaktionen, Stimmen als Assets oder Coins direkt an Adressen von Kandidaten oder Wahloptionen zu senden, ersetzt

[11] Kiniry, et al., 2015 S. 20

werden durch ein Verfahren, das Transaktionen, die die Wahlentscheidungen enthalten diese geheim, das heißt in verschlüsselter Form speichern und an eine „neutrale“ Adresse senden.

[Skizze]

Authentifizierung und Anonymität

Für eine demokratische Wahl muss gewährleistet werden, dass nur berechtigte Wählerinnen ihre Stimme abgeben können und dass jeder die gleiche Anzahl von Stimmen hat. Gleichzeitig muss die Anonymität der abgegebenen Stimmen gewahrt bleiben. Bei einem Peer-To-Peer-Netzwerk auf Basis des Bitcoin-Protokolls ist eine Authentifizierung nicht vorgesehen, jeder kann Teilnehmer in dem Netzwerk werden und alle Transaktionen beobachten. Das sollte sich auch möglichst nicht ändern, da so theoretisch jeder Internetnutzer auch als Wahlbeobachter teilnehmen kann. Stattdessen kann die Eigenschaft eines Blockchain-basierten Netzwerks, über eine native Währung zu verfügen oder auch „Assets“[12] erzeugen zu können, für ein Online-Wahlsystem ausgenutzt werden, um den Wählern ihre Stimmrechte zuzuteilen. Anstatt Stimmzettel auszuhändigen, werden Stimmrechte in Form von digitalen Assets anonymisiert z.B. per Paper Wallet an die Clients der Wähler gesendet, deren Besitz einen Wähler als wahlberechtigt identifiziert.

Geheimhaltung

Ein Online-Wahlsystem muss eine geheime Wahl garantieren. Da eine Online-Wahl unter „unkontrollierten“ Bedingungen stattfindet (nicht im Wahllokal sondern zuhause auf unsicheren Endgeräten), muss außerdem sichergestellt werden, dass kein massenhafter Stimmenkauf, Erpressung etc. technisch ermöglicht wird, ohne dass dies entdeckt wird. Das bedeutet, dass das System beispielsweise nicht offenbaren darf, wie ein bestimmter Wähler gewählt hat.

Widerstandsfähigkeit gegen Erpressung - Coercion resistance und Quittungsfreiheit

[12] Asset (englisch) zu Deutsch „Wert“ als allgemeiner Begriff, bezeichnet in den Blockchain-Protokollen eine Werteinheit, die für beliebige Werte z.B. Aktienanteile, Geld, oder auch Stimmrechte stehen kann. Assets können genau wie die „native Währung“ (z.B. BTC) im Netzwerk transferiert bzw. gehandelt werden.

Das Problem der potentiellen Erpressbarkeit erweitert die Anforderung der bloßen Geheimhaltung: Die Gefahr, dass Stimmen gekauft oder erpresst werden, lässt sich nur verhindern, wenn eine Wählerin keine Möglichkeit hat, zu beweisen, wie sie gewählt hat. Wäre sie dazu in der Lage, könnte ein Erpresser diesen Beleg fordern und sie wäre erpressbar. Eine Anforderung, die deswegen an elektronische Wahlsysteme gestellt wird, wird in der Literatur als „Coercion resistance" - zu Deutsch etwa „Widerstandsfähigkeit gegen Erpressung" bezeichnet.[13] Etwas schwächer formuliert ist in der Literatur die Anforderung der Quittungsfreiheit. Die Quittungsfreiheit besagt einfach, dass ein Wähler keine Information (=Quittung) vom System erhalten darf, wie er gewählt hat, also nicht in der Lage sein darf, die eigene Wahlentscheidung zu überprüfen. Ein möglicher Erpresser darf außerdem auch **ohne Kooperation** der Wählerin keine Möglichkeit haben, eine Verbindung zwischen der Wählerin und ihrer Wahlentscheidung herstellen können dürfen. Um die Anforderungen betreffs der Geheimhaltung und Widerstandsfähigkeit zu erfüllen, ist es notwendig, die Wahlentscheidungen bei der Übertragung in die Blockchain so zu verschlüsseln, dass ein Erpresser keine Möglichkeit hat, vom Opfer oder dem Computer des Opfers einen Schlüssel zur Entschlüsselung der Daten zu bekommen, um Kenntnis über die tatsächliche Wahlentscheidung der Wählerin zu erlangen – sei es mit oder ohne Kooperation der Wählerin. Ein asymmetrisches Verschlüsselungsverfahren stellt sicher, dass eine Entschlüsselung nur mit dem geheimen Schlüssel möglich ist, nicht mit dem Schlüssel des Wahlclients.

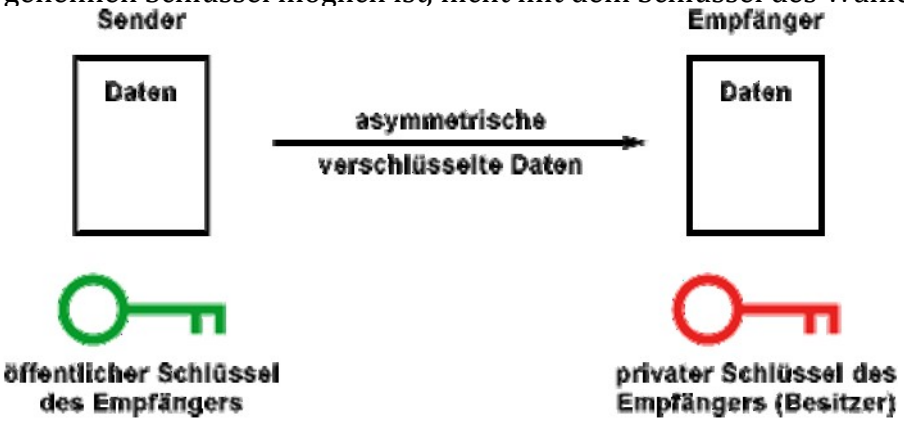


Figure 1

[13] Siehe: Bundeamt für Sicherheit in der Informationstechnik, 2008; Delaune, et al., 2006; Juels, et al., 2005; Ryan, et al., 2009

3. Vorschlag zur Umsetzung der Anforderungen

Das hier vorgeschlagene Blockchain Voting System (BVS)

Keine lokale Datenbank, stattdessen Speichern wahlrelevanten Daten in der Blockchain.

In einem konventionellen Blockchain-basierten Netzwerk ist die Geheimhaltung der Informationen nicht vorgesehen. Die einfache Methode, Coins oder Assets direkt an Kandidatenadressen zu senden, fällt aus, weil jeder der Transaktionen auf die Adressen der Wählerinnen zurückverfolgt werden könnte.

Ich schlage daher ein zweistufiges Verfahren vor, bei dem im ersten Schritt die Stimmabgabe verschlüsselt erfolgt und die Transaktion in der Blockchain dokumentiert wird und im zweiten Schritt die Entschlüsselung auf unabhängigen und **vertrauenswürdigen** Nodes des Blockchain-Netzwerkes erfolgt. Die Wahlentscheidungen könnten direkt als Rohdaten zur Weiterverarbeitung über eine API publiziert werden, um sie schnell verfügbar zu haben; eine bessere Option könnte aber sein, (gleichzeitig) aus für jeden entstandenen Rohdatensätzen wiederum eine Transaktion zu generieren, die ein Asset - gesendet an die entschlüsselte Kandidatenadresse - enthält, um eine öffentliche Überprüfbarkeit der *Einzelergebnisse* zu erleichtern.

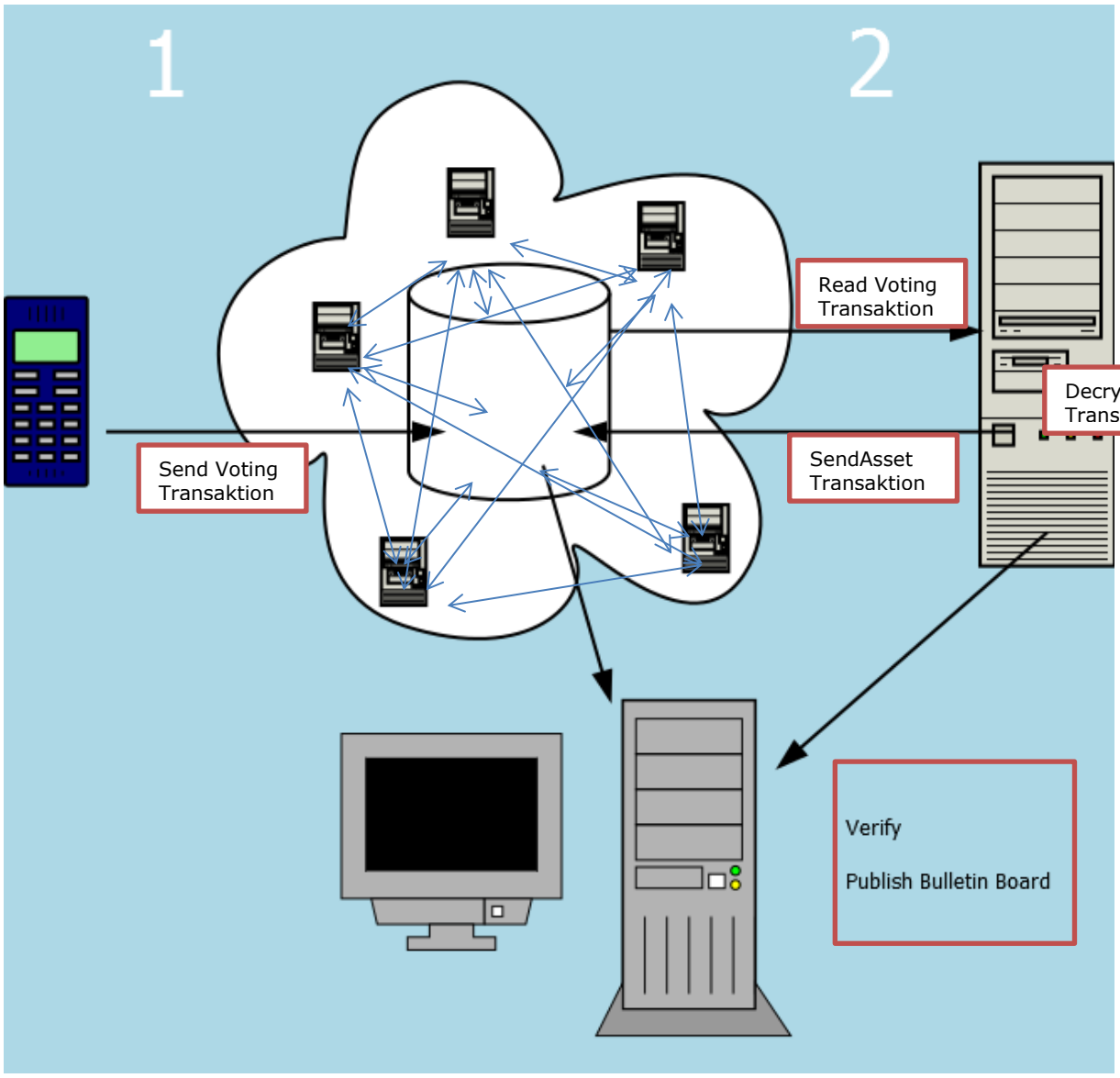


Figure 2

Genauer (Wahlphase):

Im ersten Schritt sendet der Client eine Transaktion, die sein Asset für die Berechtigung zur Stimmabgabe erhält sowie einen Text in den Metadaten, der die **verschlüsselte Wahlentscheidung (Kandidatenadresse)** und eine Prüfziffer enthält. Damit ein Wähler trotz Verschlüsselung die Möglichkeit hat, die E2E-V zu prüfen, schlage ich vor, dass der Wahlclient für jede Wahloption im Client Stimmzettel einen individuellen Code generiert, der neben der verschlüsselten Kandidatenadresse und einer Prüfsumme für beide Angaben zusammen bei der Stimmabgabe übermittelt und auch angezeigt wird. Dieser Code kann nach der

Stimmabgabe in den Transaktionen zur Überprüfung der E2E-V abgefragt werden. Ist der Code korrekt, bedeutet das, dass die richtige Wahloption übermittelt wurde.

[Skizze]

Die Zieladresse der Transaktion ist dabei eine **neutrale** Adresse, die an den Stimmzettel gebunden ist. Wenn alle Stimmen abgegeben wurden, enthält diese neutrale Adresse 100% der Stimmberechtigungen (Assets).

[Skizze]

Im zweiten Schritt nach Ende der Stimmabgabe werden diese Transaktionen, die leicht anhand der neutralen Zieladresse und anhand der Assets zu identifizieren sind, von **besonderen** Nodes aus der Blockchain gelesen und entschlüsselt. Damit stünde schon sehr schnell ein Ergebnis zur Verfügung. Damit die Auswertung jedoch dokumentiert wird und der dritte Punkt der E2E-V erfüllt ist, wird für jede Transaktion ein zufälliges Asset mit dem individuellen Code aus der Stimmabgabe von der neutralen Adresse an die nun entschlüsselte Adresse des jeweiligen Kandidaten gesendet. Nach der Auswertung enthält die neutrale Adresse keine Assets mehr, da diese entsprechend den Wahlentscheidungen auf die Kandidatenadressen verteilt sind, wenn die Auswertung vollständig und korrekt erfolgt.

Die Prüfsummen aus dem ersten und zweiten Schritt jeweils addiert müssen die gleiche Gesamtsumme ergeben, dann ist die Wahl erfolgreich überprüft.

Durch dieses zweistufige Verfahren ist die ursprüngliche Wahlentscheidung für jeden überprüfbar: Jeder kann die Prüfsumme der verschlüsselten Wahlentscheidung aus der Blockchain vergleichen mit der, die bei Stimmabgabe angezeigt wurde und prüfen ob diese bei der Auswertung auch enthalten sind. Kein Wähler ist jedoch dadurch in der Lage zu beweisen, welchen Kandidaten er gewählt hat, da sowohl die verschlüsselte Adresse als auch der individuelle Code nicht einem Kandidaten zugeordnet werden kann, ohne den geheimen Schlüssel zu kennen.

Die Anforderungen der Geheimhaltung werden ebenfalls gewahrt: In der Blockchain ist der Weg der Transaktionen unterbrochen, da im zweiten Schritt der Auswertung ein

zufälliges Asset übermittelt wird und dadurch nicht die Adresse eines Wählers in den Inputs der Kandidatenstimmen enthalten sind.

Die Anonymität der Stimmabgabe kann durch eine Kombination von Maßnahmen erreicht werden:

Die Assets als Stimmberechtigungen werden anonymisiert, in dem zufällige Adressen erzeugt werden, die diese vor Beginn der Wahl enthalten. Diese werden z.B. als „Paper-Wallets“ oder in anderer Form gespeichert und versiegelt. Wählerinnen importieren immer eine **zufällige** Paper Wallet mit den darin enthaltenen Assets für eine bestimmte Wahl. Durch dieses Verfahren bleiben die Adressen anonym und das Wahlgeheimnis in dieser Hinsicht gewahrt.

Anders als bei Bitcoin werden die Adressen nur einmal bei der Stimmabgabe verwendet, so dass eine Zuordnung zwischen bestimmten Clients oder IP-Adressen und den Bitcoin-Transaktionen unwahrscheinlich ist, wenn ein Proxy-Netzwerk ähnlich Tor bei der Stimmabgabe genutzt wird.

4. Wahlphasen

5. Anhang

Abbildungsverzeichnis

Verzeichnis der Methoden (Draft)

Multichain API

Alphabetical list of API commands

In the table below, all optional parameters are denoted in `(round brackets)`.

Command	Parameters	Description
`addmultisigaddress`	`nrequired ["key", ...]`	Creates a pay-to-scripthash (P2SH) multisig address and adds it to the wallet. Funds sent to this address can only be spent by transactions signed by `nrequired` of the specified keys. Each `key` can be a full public key, or an address if the corresponding key is in the node's wallet. (Public keys for a wallet's addresses can be obtained using the `getaddresses` call with `verbose=true`.) Returns the P2SH address.
`addnode`	`ip:port command`	Manually adds or removes a peer-to-peer connection (peers are also discovered and added automatically). The `ip` can be a hostname, IPv4 address, IPv4-as-IPv6 address or IPv6 address. For

		the entire `ip:port` you can also use the part after the `@` symbol of the other node's `nodeaddress`, as given by the `getinfo` command. The `command` parameter should be one of `add` (to manually queue a node for the next available slot), `remove` (to remove a node), or `onetry` (to immediately connect to a node even if a slot is not available). The result can be retrieved via the `getaddednodeinfo` and `getpeerinfo` commands.
`appendrawchange`	`tx-hex address (native-fee)`	Adds a change output to the raw transaction in `tx-hex` given by a previous call to `createrawtransaction`. Any assets or native currency in the transaction inputs which are not claimed in the outputs will be sent to `address`, minus the `native-fee` (which is calculated automatically if omitted). The returned raw transaction can be signed and broadcast to the network using `signrawtransaction` and `sendrawtransaction`.

appendrawdata or appendrawmetadata	tx-hex data-hex|object	Adds a metadata output (using an OP_RETURN) to the raw transaction in tx-hex given by a previous call to createrawtransaction. Raw metadata can be specified in data-hex in hexadecimal form. Alternatively, an object can be passed to represent asset issuance, stream creation or a stream item – see raw transactions for more details. The returned raw transaction can be signed and broadcast to the network using signrawtransaction and sendrawtransaction.
appendrawexchange	tx-hex txid vout {"asset":qty, ...}	Adds to the raw atomic exchange transaction in tx-hex given by a previous call to createrawexchange or appendrawexchange. This adds an offer to exchange the asset/s in output vout of transaction txid for qty units of asset, where asset is an asset name, ref or issuance txid. The txid and vout should generally be taken from preparelockunspent or preparelockunspentfrom.

		Multiple items can be specified within the fourth parameter to request multiple assets. Returns a raw transaction in the `hex` field alongside a `complete` field stating whether the exchange is complete (i.e. balanced) or not. If complete, the transaction can be broadcast to the network using `sendrawtransaction`. If not, it can be passed to a further counterparty, who can call `decoderawexchange` and `appendrawexchange` as appropriate.
`clearmempool`		Removes all unconfirmed transactions from this node's memory pool. This can only be called after `pause incoming,mining`. Successful if no error is returned.

`combineunspent`	`(addresses=*)`
	`(minconf=1)`
	`(maxcombines=1)`
	`(mininputs=10)`
	`(maxinputs=100)`
	`(maxtime=30)`
		Sends transactions to combine unspent outputs (UTXOs) belonging to the same address into a single unspent output, returning a list of txids. This can improve wallet performance, especially for miners in a

		chain with short block times and non-zero block rewards. Set `addresses` to a comma-separated list of addresses to combine outputs for, or `*` for all addresses in the wallet. Only combines outputs with at least `minconf` confirmations, using between `mininputs` and `maxinputs` per transaction. A single call to `combineunspent` can create up to `maxcombines` transactions over up to `maxtime` seconds. See also the `autocombine` runtime parameters.
`create`	`type=stream name open ({"custom-field-1":"x",...})`	Creates a new stream on the blockchain called `name`. For now, always pass the value `"stream"` in the `type` parameter – this is designed for future functionality. If `open` is `true` then anyone with global `send` permissions can publish to the stream, otherwise publishers must be explicitly granted per-stream `write` permissions.
`createfrom`	`from-address type=stream name open`	This works like `create`, but with control over the

	({"custom-field-1":"x",...})	from-address used to create the stream. It is useful if the node has multiple addresses with create permissions.
createkeypairs	(count=1)	Generates one or more public/private key pairs, which are not stored in the wallet or drawn from the node's key pool, ready for external key management. For each key pair, the address, pubkey (as embedded in transaction inputs) and privkey (used for signatures) is provided.
createmultisig	nrequired ["key", ...]	Creates a pay-to-scripthash (P2SH) multisig address. Funds sent to this address can only be spent by transactions signed by nrequired of the specified keys. Each key can be a full hexadecimal public key, or an address if the corresponding key is in the node's wallet. Returns an object containing the P2SH address and corresponding redeem script.
createrawexchange	txid vout {"asset":qty, ...}	Creates a new atomic exchange transaction which

		offers to exchange the asset/s in output `vout` of transaction `txid` for `qty` units of `asset`, where `asset` is an asset name, ref or issuance txid. The `txid` and `vout` should generally be taken from the response to `preparelockunspent` or `preparelockunspentfrom`. Multiple items can be specified within the third parameter to request multiple assets. Returns a raw partial transaction in hexadecimal which can be passed to the counterparty, who can call `decoderawexchange` and `appendrawexchange` as appropriate.
`createrawsendfrom`	`from-address {"to-address":amount,...} (data=[]) (action="")`	This works like `createrawtransaction`, except it automatically selects the transaction inputs from those belonging to `from-address`, to cover the appropriate amounts. One or more change outputs going back to `from-address` will also be added to the end of the transaction.

createrawtransaction	[{"txid":"id","vout":n},...] {"address":amount,...} (data=[]) (action="")	Creates a transaction spending the specified inputs, sending to the given addresses. In Bitcoin Core, each `amount` field is a quantity of the bitcoin currency. For MultiChain, an `{"asset":qty, ...}` object can be used for `amount`, in which each `asset` is an asset name, ref or issuance txid, and each `qty` is the quantity of that asset to send (see native assets). Use `""` as the `asset` inside this object to specify a quantity of the native blockchain currency. The optional `data` array adds one or more metadata outputs to the transaction, where each element is a raw hexadecimal string or object, formatted as passed to `appendrawdata`. The optional `action` parameter can be `lock` (locks the given inputs in the wallet), `sign` (signs the transaction using wallet keys), `lock,sign` (does both) or `send` (signs and sends the transaction). If `action` is `send` the txid is returned. If `action`

		contains `sign`, an object with `hex` and `complete` fields is returned, as for `signrawtransaction`. Otherwise, the raw transaction hexadecimal is returned. See raw transactions for more details on building raw transactions.
`decoderawexchange`	`tx-hex (verbose=false)`	Decodes the raw exchange transaction in `tx-hex`, given by a previous call to `createrawexchange` or `appendrawexchange`. Returns details on the offer represented by the exchange and its present state. The `offer` field in the response lists the quantity of native currency and/or assets which are being offered for exchange. The `ask` field lists the native currency and/or assets which are being asked for. The `candisable` field specifies whether this wallet can disable the exchange transaction by double-spending against one of its inputs. The `cancomplete` field specifies whether this wallet has the assets required to

		complete the exchange. The `complete` field specifies whether the exchange is already complete (i.e. balanced) and ready for sending. If `verbose` is `true` then all of the individual stages in the exchange are listed. Other fields relating to fees are only relevant for blockchains which use a native currency.
`decoderawtransaction`	`tx-hex`	Returns a JSON object describing the serialized transaction in `tx-hex`. For a MultiChain blockchain, each transaction output includes `assets` and `permissions` fields listing any assets or permission changes encoded within that output. There will also be a `data` field listing the content of any `OP_RETURN` outputs in the transaction.
`disablerawtransaction`	`tx-hex`	Sends a transaction to disable the offer of exchange in `tx-hex`, returning the txid. This is achieved by spending one of the exchange transaction's inputs and sending it back to the wallet.

		To check whether this can be used on an exchange transaction, check the `candisable` field of the output of `decoderawexchange`.
`dumpprivkey`	`address`	Returns the private key associated with `address` in this node's wallet. **Use with caution** – any node with access to this private key can perform any action restricted to the address, including granting permissions and spending funds.
`getaddressbalances`	`address (minconf=1) (includeLocked=false)`	Returns a list of all the asset balances for `address` in this node's wallet, with at least `minconf` confirmations. Use `includeLocked` to include unspent outputs which have been locked, e.g. by a call to `preparelockunspent`.
`getaddresses`	`(verbose=false)`	Returns a list of addresses in this node's wallet. Set `verbose` to `true` to get more information about each address, formatted like the output of the `validateaddress` command.

		For more control see the new `listaddresses` command.
`getaddresstransaction`	`address txid (verbose=false)`	Provides information about transaction `txid` related to `address` in this node's wallet, including how it affected that address's balance. Use `verbose` to provide details of transaction inputs and outputs.
`getassettransaction`	`asset txid (verbose=false)`	Retrieves a specific transaction `txid` involving `asset`, passed as an asset name, ref or issuance txid, to which the node must be subscribed. Set `verbose` to `true` for additional information about the transaction.
`getblock`	`hash|height (verbose=1)`	Returns information about the block with `hash` (retrievable from `getblockhash`) or at the given `height` in the active chain. Set `verbose` to `0` or `false` for the block in raw hexadecimal form. Set to `1` or `true` for a block summary including the `miner` address and a list of txids. Set to `2` to `3` to include

		more information about each transaction and its raw hexadecimal. Set to `4` to include a full description of each transaction, formatted like the output of `decoderawtransaction`.
`getblockchainparams`		Returns a list of all the parameters of this blockchain, reflecting the content of its `params.dat` file.
`getblockhash`	`height`	Returns the hash of the block at the given `height`. This can be passed to `getblock` to get information about the block.
`getinfo`		Returns general information about this node and blockchain. MultiChain adds some fields to Bitcoin Core's response, giving the blockchain's `chainname`, `description`, `protocol`, peer-to-peer `port`. There are also `incomingpaused` and `miningpaused` fields – see the `pause` command. The `burnaddress` is an address with no known private key, to which assets can be sent to

		make them provably unspendable. The `nodeaddress` can be passed to other nodes for connecting. The `setupblocks` field gives the length in blocks of the setup phase in which some consensus constraints are not applied.

`getmultibalances`	`(addresses=*)`
	`(assets=*)`
	`(minconf=1)`
	`(includeWatchOnly=false)`
	`(includeLocked=false)`
		Returns a list of balances of the `addresses` in this node's wallet for the specified `assets`, with at least `minconf` confirmations. The `addresses` are specified as a comma-delimited list or array, or `*` for all addresses with non-zero balances. The `assets` are specified as an array of asset names, refs or issuance txids, or `*` for all assets with non-zero balances. Use `includeWatchOnly` to include watch-only addresses (only relevant if `addresses=*`) and `includeLocked` to include unspent outputs which have been locked, e.g. by a call to `preparelockunspent`. The response includes a `total` of the balances shown.

`getnewaddress`		Returns a new address whose private key is added to the wallet.
`getpeerinfo`		Returns information about the other nodes to which this node is connected. If this is a MultiChain blockchain, includes `handshake` and `handshakelocal` fields showing the remote and local address used during the handshaking for that connection.
`getrawtransaction`	`txid (verbose=0)`	If `verbose` is `1`, returns a JSON object describing transaction `txid`. For a MultiChain blockchain, each transaction output includes `assets` and `permissions` fields listing any assets or permission changes encoded within that output. There will also be a `data` field listing the content of any `OP_RETURN` outputs in the transaction.
`getstreamitem`	`stream txid (verbose=false)`	Retrieves a specific item with `txid` from `stream`, passed as a stream name, ref or creation txid, to which the node must be subscribed. Set

		verbose to true for additional information about the item's transaction. If an item's data is larger than the maxshowndata runtime parameter, it will be returned as an object whose fields can be used with gettxoutdata.
gettotalbalances	(minconf=1) (includeWatchOnly=false) (includeLocked=false)	Returns a list of all the asset balances in this node's wallet, with at least minconf confirmations. Use includeWatchOnly to include the balance of watch-only addresses and includeLocked to include unspent outputs which have been locked, e.g. by a call to preparelockunspent.
gettxout	txid vout (unconfirmed=false)	Returns details about an unspent transaction output vout of txid. For a MultiChain blockchain, includes assets and permissions fields listing any assets or permission changes encoded within the output. Set unconfirmed to true to include unconfirmed transaction outputs.

gettxoutdata	txid vout (count-bytes=INT_MAX) (start-byte=0)	Returns the data embedded in output vout of transaction txid, in hexadecimal. This is particularly useful if a stream item's data is larger than the maxshowndata runtime parameter. Use the count-bytes and start-byte parameters to retrieve part of the data only.
getwallettransaction	txid (includeWatchOnly=false) (verbose=false)	Provides information about transaction txid in this node's wallet, including how it affected the node's total balance. Use includeWatchOnly to consider watch-only addresses as if they belong to this wallet and verbose to provide details of transaction inputs and outputs.
grant	addresses permissions (native-amount=0) (comment='') (comment-to='') (start-block=0) (end-block)	Grants permissions to addresses, a comma-separated list of addresses. For global permissions, set permissions to one of connect, send, receive, create, issue, mine, activate, admin, or a comma-

		separated list thereof. For per-asset or per-stream permissions, use the form `entity.issue` or `entity.write,admin` where `entity` is an asset or stream name, ref or creation txid. If the chain uses a native currency, you can send some to each recipient using the `native-amount` parameter. Returns the txid of the transaction granting the permissions. For more information, see permissions management.
`grantfrom`	`from-address to-addresses permissions (native-amount=0) (comment='') (comment-to='') (start-block=0) (end-block)`	This works like `grant`, but with control over the `from-address` used to grant the permissions. It is useful if the node has multiple addresses with administrator permissions.
`grantwithdata` or `grantwithmetadata`	`addresses permissions data-hex|object (native-amount=0) (start-block=0) (end-block)`	This works like `grant`, but with an additional data-only transaction output. To include raw data, pass a `data-hex` hexadecimal string. To publish the data to a stream, pass an object `{"for":stream,"key":"..."`

		,"data":"..."} where stream is as a stream name, ref or creation txid, the key is in text form, and the data is hexadecimal.
grantwithdatafrom or grantwithmetadatafrom	from-address to-addresses permissions data-hex|object (native-amount=0) (start-block=0) (end-block)	This works like grantwithdata, but with control over the from-address used to grant the permissions.
help		Returns a list of available API commands, including MultiChain-specific commands.
importaddress	address(es) (label) (rescan=true)	Adds address (or a full public key, or an array of either) to the wallet, without an associated private key. This creates one or more watch-only addresses, whose activity and balance can be retrieved via various APIs (e.g. with the includeWatchOnly parameter), but whose funds cannot be spent by this node. If rescan is true, the entire blockchain is checked for transactions relating to all addresses in the wallet,

		including the added ones. Returns `null` if successful.
`importprivkey`	`privkey(s) (label) (rescan=true)`	Adds a `privkey` private key (or an array thereof) to the wallet, together with its associated public address. If `rescan` is `true`, the entire blockchain is checked for transactions relating to all addresses in the wallet, including the added ones. Returns `null` if successful.
`issue`	`address name|params qty (units=1) (native-amount=min-per-output) ({"custom-field-1":"x",...})`	Creates a new asset on the blockchain, sending the initial `qty` units to `address`. To create an asset with the default behavior, use an asset name only for `name|params`. For more control, use an object such as `{"name":"asset1","open":true}`. If `open` is `true` then additional units can be issued in future by the same key which signed the original issuance, via the `issuemore` or `issuemorefrom` command. The smallest transactable unit is given by `units`, e.g. `0.01`. If the chain uses a native currency, you can send

		some with the new asset using the `native-amount` parameter. Returns the txid of the issuance transaction. For more information, see native assets.
`issuefrom`	`from-address to-address name|params qty (units=1) (native-amount=min-per-output) ({"custom-field-1":"x",...})`	This works like `issue`, but with control over the `from-address` used to issue the asset. It is useful if the node has multiple addresses with `issue` permissions.
`issuemore`	`address asset qty (native-amount=min-per-output) ({"custom-field-1":"x",...})`	Issues `qty` additional units of `asset`, sending them to `address`. The `asset` can be specified using its name, ref or issuance txid – see native assets for more information. If the chain uses a native currency, you can send some with the new asset units using the `native-amount` parameter. Any custom fields will be attached to the new issuance event, and not affect the original values (use `listassets` with `verbose=true` to see both sets). Returns the txid of the issuance transaction.

issuemorefrom	from-address to-address asset qty (native-amount=min-per-output) ({"custom-field-1":"x",...})	This works like issuemore, but with control over the from-address used.
listaddresses	(addresses=*) (verbose=false) (count=MAX) (start=-count)	Returns information about the addresses in the wallet. Provide one or more addresses (comma-delimited or as an array) to retrieve information about specific addresses only, or use * for all addresses in the wallet. Use count and start to retrieve part of the list only, with negative start values (like the default) indicating the most recently created addresses.
listaddresstransactions	address (count=10) (skip=0) (verbose=false)	Lists information about the count most recent transactions related to address in this node's wallet, including how they affected that address's balance. Use skip to go back further in history and verbose to provide details of transaction inputs and outputs.
listassets	(assets=*)	Returns information about

	(verbose=false) (count=MAX) (start=-count)	assets issued on the blockchain. Pass an asset name, ref or issuance txid in assets to retrieve information about one asset only, an array thereof for multiple assets, or * for all assets. In assets with multiple issuance events, the top-level issuetxid and details fields refer to the first issuance only – set verbose to true for details about each of the individual issuances. Use count and start to retrieve part of the list only, with negative start values (like the default) indicating the most recently created assets. Extra fields are shown for assets to which this node is subscribed.
listassettransactions	asset (verbose=false) (count=10) (start=-count) (local-ordering=false)	Lists transactions involving asset, passed as an asset name, ref or issuance txid, to which the node must be subscribed. Set verbose to true for additional information about each transaction. Use count and start to retrieve part of the list only, with negative start

		values (like the default) indicating the most recent items. Set `local-ordering` to `true` to order transactions by when first seen by this node, rather than their order in the chain.
`listlockunspent`		Returns a list of locked unspent transaction outputs in the wallet. These will not be used when automatically selecting the outputs to spend in a new transaction.
`listpermissions`	`(permissions=*)` `(addresses=*)` `(verbose=false)`	Returns a list of all permissions which have been explicitly granted to addresses. To list information about specific global permissions, set `permissions` to one of `connect`, `send`, `receive`, `issue`, `mine`, `activate`, `admin`, or a comma-separated list thereof. Omit or pass `*` or `all` to list all global permissions. For per-asset or per-stream permissions, use the form `entity.issue`, `entity.write`, `admin` or `entity.*` where `entity` is an asset or stream name, ref or

		creation txid. Provide a comma-delimited list in `addresses` to list the permissions for particular addresses or `*` for all addresses. If `verbose` is `true`, the `admins` output field lists the administrator/s who assigned the corresponding permission, and the `pending` field lists permission changes which are waiting to reach consensus.
liststreamitems	stream (verbose=false) (count=10) (start=-count) (local-ordering=false)	Lists items in `stream`, passed as a stream name, ref or creation txid. Set `verbose` to `true` for additional information about each item's transaction. Use `count` and `start` to retrieve part of the list only, with negative `start` values (like the default) indicating the most recent items. Set `local-ordering` to `true` to order items by when first seen by this node, rather than their order in the chain. If an item's `data` is larger than the `maxshowndata` runtime parameter, it will be returned as an object whose fields can be used with

		gettxoutdata.
liststreamkeyitems	stream key (verbose=false) (count=10) (start=-count) (local-ordering=false)	This works like liststreamitems, but listing items with the given key only.
liststreamkeys	stream (keys=*) (verbose=false) (count=MAX) (start=-count) (local-ordering=false)	Provides information about keys in stream, passed as a stream name, ref or creation txid. Pass a single key in keys to retrieve information about one key only, pass an array for multiple keys, or * for all keys. Set verbose to true to include information about the first and last item with each key shown. See liststreamitems for details of the count, start and local-ordering parameters.
liststreampublisheritems	stream address (verbose=false) (count=10) (start=-count) (local-ordering=false)	This works like liststreamitems, but listing items published by the given address only.
liststreampublishers	stream (addresses=*) (verbose=false) (count=MAX) (start=-count) (local-ordering=false)	Provides information about publishers who have written to stream, passed as a stream name, ref or creation txid. Pass a single address in addresses to retrieve

		information about one publisher only, pass an array or comma-delimited list for multiple publishers, or * for all publishers. Set verbose to true to include information about the first and last item by each publisher shown. See liststreamitems for details of the count, start and local-ordering parameters, relevant only if address=*.
liststreams	(streams=*) (verbose=false) (count=MAX) (start=-count)	Returns information about streams created on the blockchain. Pass a stream name, ref or creation txid in streams to retrieve information about one stream only, an array thereof for multiple streams, or * for all streams. Use count and start to retrieve part of the list only, with negative start values (like the default) indicating the most recently created streams. Extra fields are shown for streams to which this node has subscribed.
listunspent	(minconf=1) (maxconf=999999)	Returns a list of unspent transaction outputs in the

	(["address", ...])	wallet, with between minconf and maxconf confirmations. For a MultiChain blockchain, each transaction output includes assets and permissions fields listing any assets or permission changes encoded within that output. If the third parameter is provided, only outputs which pay an address in this array will be included.
listwallettransactions	(count=10) (skip=0) (includeWatchOnly=false) (verbose=false)	Lists information about the count most recent transactions in this node's wallet, including how they affected the node's total balance. Use skip to go back further in history and includeWatchOnly to consider watch-only addresses as if they belong to this wallet. Use verbose to provide the details of transaction inputs and outputs. Note that unlike Bitcoin Core's listtransactions command, the response contains one element per transaction, rather than one per transaction output.

lockunspent	unlock ([{"txid":"id","vout":n},...])	If unlock is false, locks the specified transaction outputs in the wallet, so they will not be used for automatic coin selection. If unlock is true, it unlocks the specified outputs, or unlocks all outputs if no second parameter is provided.
pause	tasks	Pauses the specified tasks, specified as a comma-delimited list of mining (i.e. generating blocks) and/or incoming (i.e. processing of incoming blocks and transactions). Successful if no error is returned.
ping		Sends a ping message to all connected nodes to measure network latency and backlog. The results are received asynchronously and retrieved from the pingtime field of the response to getpeerinfo.
preparelockunspent	{"asset":qty, ...} (lock=true)	Prepares an unspent transaction output (useful for building atomic exchange transactions) containing qty units of asset, where asset is an asset name, ref or issuance

		txid. Multiple items can be specified within the first parameter to include several assets within the output. The output will be locked against automatic selection for spending unless the optional `lock` parameter is set to `false`. Returns the `txid` and `vout` of the prepared output.
`preparelockunspentfrom`	`from-address {"asset":qty, ...} (lock=true)`	This works like `preparelockunspent`, but with control over the `from-address` whose funds are used to prepare the unspent transaction output. Any change from the transaction is send back to `from-address`.
`publish`	`stream key data-hex`	Publishes an item in `stream`, passed as a stream name, ref or creation txid, with `key` provided in text form and `data-hex` in hexadecimal.
`publishfrom`	`from-address stream key data-hex`	This works like `publish`, but publishes the item from `from-address`. It is useful if a stream is open or the node has multiple addresses with per-stream `write`

		permissions.
resume	tasks	Resumes the specified tasks, specified as in the pause command. Successful if no error is returned.
revoke	addresses permissions (native-amount=0) (comment='') (comment-to='')	Revokes permissions from addresses, a comma-separated list of addresses. The permissions parameter works the same as for grant. This is equivalent to calling grant with start-block=0 and end-block=0. Returns the txid of transaction revoking the permissions. For more information, see permissions management.
revokefrom	from-address to-addresses permissions (native-amount=0) (comment='') (comment-to='')	This works like revoke, but with control over the from-address used to revoke the permissions. It is useful if the node has multiple addresses with administrator permissions.
send or sendtoaddress	address amount (comment='') (comment-to='')	Sends one or more assets to address, returning the txid. In Bitcoin Core, the amount field is the quantity of the bitcoin currency. For

		MultiChain, an `{"asset":qty, ...}` object can be used for `amount`, in which each `asset` is an asset name, ref or issuance txid, and each `qty` is the quantity of that asset to send (see native assets). Use `""` as the `asset` inside this object to specify a quantity of the native blockchain currency. See also `sendasset` for sending a single asset and `sendfrom` to control the address whose funds are used.
`sendasset` or `sendassettoaddress`	`address asset qty (native-amount=min-per-output) (comment='') (comment-to='')`	Sends `qty` of `asset` to `address`, returning the txid. The `asset` can be specified using its name, ref or issuance txid – see native assets for more information. See also `sendassetfrom` to control the address whose funds are used, `send` for sending multiple assets in one transaction, and `sendfrom` to combine both of these.
`sendassetfrom`	`from-address to-address asset qty`	This works like `sendasset`, but with control over the

	(native-amount=min-per-output) (comment='') (comment-to='')	from-address whose funds are used. Any change from the transaction is sent back to from-address. See also sendfrom for sending multiple assets in one transaction.
sendfrom or sendfromaddress	from-address to-address amount (comment='') (comment-to='')	This works like send, but with control over the from-address whose funds are used. Any change from the transaction is sent back to from-address.
sendrawtransaction	tx-hex	Validates the raw transaction in tx-hex and transmits it to the network, returning the txid. The raw transaction can be created using createrawtransaction, (optionally) appendrawdata and signrawtransaction, or else createrawexchange and appendrawexchange.
sendwithdata or sendwithmetadata	address amount data-hex|object	This works like send, but with an additional data-only transaction output. To include raw data, pass a data-hex hexadecimal string. To publish the data to a stream, pass an object

		`{"for":stream,"key":"..."` `,"data":"..."}` where `stream` is as a stream name, ref or creation txid, the `key` is in text form, and the `data` is hexadecimal.
`sendwithdatafrom` or `sendwithmetadatafrom`	`from-address to-address amount data-hex|object`	This works like `sendwithdata`, but with control over the `from-address` whose funds are used. Any change from the transaction is sent back to `from-address`.
`setlastblock`	`hash|height`	Rewinds this node's active chain to `height` or rewinds/switches to another block with `hash`. This can only be called after `pause incoming,mining`. Returns the hash of the last block in the chain after the change.
`signmessage`	`address message`	Returns a base64-encoded digital signature which proves that `message` was approved by the owner of `address` (which must belong to this wallet). The signature can be verified by any node using the `verifymessage` command.

signrawtransaction	tx-hex ([{parent-output},...]) (["private-key",...])	Signs the raw transaction in tx-hex, often provided by a previous call to createrawtransaction and (optionally) appendrawdata and appendrawchange. Returns a raw hexadecimal transaction in the hex field alongside a complete field stating whether the transaction is now completely signed. If complete, the transaction can be broadcast to the network using sendrawtransaction. If not, it can be passed to other parties for additional signing. To create chains of unbroadcast transactions, pass an optional array of {parent-output} objects, each of which takes the form {"txid":txid,"vout":n,"scriptPubKey":hex}. To sign the transaction using (only) private keys which are not in the node's wallet, pass an array of "private-key" strings, formatted as per dumpprivkey.
stop		Shuts down the this

		blockchain node, i.e. stops the `multichaind` process.
`subscribe`	`asset(s)|stream(s) (rescan=true)`	Instructs the node to start tracking one or more `asset(s)` or `stream(s)`. These are specified using a name, ref or creation/issuance txid, or for multiple items, an array thereof. If `rescan` is `true`, the node will reindex all items from when the assets and/or streams were created, as well as those in other subscribed entities. Returns `null` if successful. See also the `autosubscribe` runtime parameter.
`unsubscribe`	`asset(s)|stream(s)`	Instructs the node to stop tracking one or more `asset(s)` or `stream(s)`. Assets or streams are specified using a name, ref or creation/issuance txid, or for multiple items, an array thereof.
`validateaddress`	`address`	Returns information about `address` including a check for its validity.

`verifymessage`	`address signature message`	Verifies that `message` was approved by the owner of `address` by checking the base64-encoded digital `signature` provided by a previous call to `signmessage`. The result is `true` or `false` unless an error occurred.

Wahlclient

Konfiguration, Paper Wallet

Funktionen

1. Import Project, Import Paper Wallet als ein Prozess am besten durch QR-Code

Stimmabgabe

a) Transaktion Stimmabgabe generieren

Funktionen

1. Wahlentscheidung verschlüsseln -> Metadaten erster Abschnitt
   Algorithmus:
2. Client-Prüfziffer für die Kandidaten ermitteln + Individueller Code
   (nachvollziehbar, aber nicht vorhersehbar z.B. aus den aktuellen Börsenkurse eines Indexes -> Metadaten zweiter Abschnitt
3. Metadaten erzeugen

b) Transaktion senden

1. Asset Allocation
2. Asset und Metadaten senden

Evaluation Client

Eigenschaften:

Wallet für die Optionen mit

- jeweils 1 Adresse für jede Option
- Je eine Watch-Only Adresse für Stimmzettel

c) Transaktionen lesen

Für jeden Stimmzettel:

d) Transaktion auswerten

Action: „Proceed“.

Public function proceedAction(string ballot)

Transaktionen = `listaddresstransactions(address)`

Für jede Transaktion:

1. Get Metadata.
2. Decrypt Metadata
3. Generate Transaktion

return void

e) Asset an Kandidatenadresse senden

Für jede neue Transaktion: Send Transaktion

f) Asset-Verteilung auswerten

Für jede Option:

getaddressbalances(Option Adresse)

Client für Election Office

- Projekt anlegen, verwalten, importieren, exportieren
- Walletchecks: Project, Optionen mit getrennten Wallets

Literaturverzeichnis

Ben-Sasson, Eli, et al. 2014. *Zerocash: Decentralized Anonymous Payments from Bitcoin (extended version).* 2014.

Biryuk, Alex, Khovratovic, Dimitry und Pustogarov, Ivan. 2014. Deanonymisation of Clients in Bitcoin P2P Network. [Online] 2014. [Zitat vom: 29. 09 2015.] http://orbilu.uni.lu/bitstream/10993/18679/1/Ccsfp614s-biryukovATS.pdf.

Bundeamt für Sicherheit in der Informationstechnik (BSI). 2008. Common Criteria Protection Profile BSI-CC-PP-0037. *www.bsi.de.* [Online] 1.0, 18. April 2008. [Zitat vom: 09. Juni 2016.] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0037b_pdf.html.

Clark, Jeremy. 2011. Democracy Enhancing Technologies:Toward deployable and incoercible E2E elections. *Université Concordia.* [Online] 2011. [Zitat vom: 15. November 2016.] http://users.encs.concordia.ca/%7Eclark/theses/phd_electronic.pdf.

De Vries, Manon und Bokslag, Wouter. 2016. *Evaluating e-voting: theory and practice.* Department of Information Security Technology, Technical University of Eindhoven. Eindhoven : s.n., 2016. [Dokument]. arXiv:1602.02509v1 [cs.CY] 8 Feb 2016.

Decker, Christian und Wattenhofer, Roger. 2013. *Information Propagation in the Bitcoin Network.* ETH Zurich; Microsoft Research. Zürich : s.n., 2013.

Delaune, Stephanie, Kremer, Steve und Ryan, Mark. 2006. Coercion-Resistance and Receipt-Freeness in Electronic Voting. [Hrsg.] IEEE. *Computer Security Foundations Workshop.* 2006.

Hahlen, Johann. 2001. *Vortrag zum Thema Internetwahlen.* Deutscher Internet-Kongress in Karlsruhe : s.n., 18. September 2001.

Halderman, Alex. 2015. Security Analysis of Estonia's Internet Voting System. [Online] 2015. [Zitat vom: 2. September 2015.] https://estoniaevoting.org/.

International Telecommunication Union (ITU). 2015. The World in Facts and Figures. [Online] 05 2015. [Zitat vom: 09. Juni 2016.] http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf.

Juels, Ari, Catalano, Dario und Jakobsson, Markus. 2005. Coercion-resistant electronic elections. *http://www.arijuels.com.* [Online] 2005. [Zitat vom: 16. November 2016.] http://www.arijuels.com/wp-content/uploads/2013/09/JCJ10.pdf.

Kiniry, Joseph R, et al. 2015. *The Future of Voting.* U.S. VOTE FOUNDATION. s.l. : Galois, 2015.

Luo, Shoufu, Seideman, Jeremy D. und Tsai, Gary. 2016. THE PEOPLE'S CHOICE - A accountable distributed blockchain-based digital voting system. *economist.com.* [Online] 29. September 2016. [Zitat vom: 26. November 2016.] www.economist.com/sites/default/files/cuny.pdf.

New South Wales Electoral Commission. 2014. *iVote® Project- iVote® System Security Implementation Statement.* Sydney : s.n., 2014. Statement.

Pressestelle Bundesverfassungsgericht. 2009. *Verwendung von Wahlcomputern bei der Bundestagswahl 2005 verfassungswidrig.* Karlruhe : s.n., 3. März 2009.

Teague, Vanessa und Halderman, J. Alex. 2015. The New South Wales iVote System:. *CITP Center for Information Technology Policy.* [Online] 22. März 2015. [Zitat vom: 2. September 2015.] http://arxiv.org/pdf/1504.05646v2.pdf.