Abschlussbericht Modul Blockchain

Projekt Dead Man Switch - Gruppe 24

Herbstsemester 2018
Hochschule Luzern – Informatik

Julian Bigler - julian.bigler@stud.hslu.ch
Robin Bürgi - robin.buergi@stud.hslu.ch
Dorus Janssens - dorus.janssens@stud.hslu.ch

Inhaltsverzeichnis

Businessplan

automatic release of distributed information in case of your passing

Management Summary

Journalisten und Dissidenten kämpfen auf der ganzen Welt gegen repressive Regime, Korruption und Gewalt. Im Jahr 2017 wurden bei der Ausführung ihrer Tätigkeit mindestens 65 Journalisten und Journalistinnen getötet, 54 entführt oder 326 inhaftiert. [1] Im Jahr 2017 lebten weltweit 50.7 % aller Menschen in hybriden oder autoritären Regimes. [2]
Das Projekt *Dead Man Switch* ermöglicht einem User das dezentralisierte Speichern von kritischen Informationen. Alle Informationen werden mit modernen kryptografischen Verfahren verschlüsselt und abgelegt. Der User muss gegenüber dem System regelmäßig sein Wohlbefinden bestätigen, ansonsten werden seine Daten veröffentlicht.

Business Description

Wir bieten die technische Infrastruktur um das sichere Ablegen der Daten und der garantierten Veröffentlichung zu gewährleisten. Durch die automatische Veröffentlichung kann das System als eine Art Versicherung für an Leib und Leben bedrohte genutzt werden. Potentielle Kunden sind Journalisten, Dissidenten oder Akteure in der Spionage.

Marktübersicht

> *Seventy-one countries suffered net declines in political rights and civil liberties, with only 35 registering gains. This marked the* ***12th consecutive year of decline in global freedom.***
>
> - Freedom House, Freedom in the World 2018 Rapport [3]

Weltweit ist die Demokratisierung in den letzten zwölf Jahren gesunken, die potenzielle Kundenbasis somit gewachsen. Durch die Verwendung einer Blockchain kann global agiert werden und nationale Gesetzgebung weitgehend umgangen werden. Somit kann in einem vollzugsfreien Raum agiert werden.
Es konnten zwei Konkurrenten mit einem funktionsmässig ähnlichen Angebot identifiziert werden. Deadman.io [4] ist ein Hackathon Projekt und hat wie auch Deadmansswitch.net [5] den Nachteil, dass alle Daten zentral und ohne prüfbare Verschlüsselung abgelegt werden.

[1] Jahresbilanz 2017 - Reporter ohne Grenzen (Abgerufen am 08.10.2018) https://www.reporter-ohne-grenzen.de/fileadmin/Redaktion/Presse/Downloads/Jahresbilanz/Jahresbilanz_der_Pressefreiheit_2017.pdf
[2] Democracy Index 2017 - The Economist Intelligence Unit (Abgerufen am 08.10.2018) http://pages.eiu.com/rs/753-RIQ-438/images/Democracy_Index_2017.pdf
[3] Freedom House - Freedom in the World 2018 Rapport (Abgerufen am 08.10.2018) https://freedomhouse.org/report/freedom-world/freedom-world-2018
[4] Deadman.io (Abgerufen am 08.10.2018) http://www.deadman.io/
[5] Deadmansswitch.net (Abgerufen am 08.10.2018) https://www.deadmansswitch.net

Marketing

Um die Anwendung zu vermarkten bietet sich die Zusammenarbeiten mit NGOs wie *Reporter ohne Grenzen* oder *Amnesty International* an, um von deren Reichweite zu profitieren. Da es sich um eine obskure, aber auch faszinierende Anwendung von gehypter Technologie handelt, können auch Plattformen wie Hacker News eine Rolle in der Verbreitung übernehmen. Als initiale Marketing Offensive muss eine Hinrichtung eines Reporters der New York Times verhindert werden. Notfalls auch als Inszenierung.

Design und Umsetzungsplan

Im Paper *Do you need a Blockchain?* [6] werden grundlegende Fragen gestellt, ob eine Blockchain verwendet werden soll. In diesem Fall müssen Daten von mehreren unbekannten Writers persistiert werden. Auf eine zentrale ständig verfügbare “trusted third party” (TTP) möchte verzichtet werden, um einen Single Point of Failure zu umgehen und den Angriffsvektor zu minimieren. Es muss daher eine Permissionless Blockchain wie Ethereum eingesetzt werden. Wegen technischen Limiten einer Blockchain können die Daten selber nicht on-chain gespeichert werden. Als mögliche Lösung kommt das InterPlanetary File System (IPFS) [7] in Frage. Damit können Dateien in einem Peer-to-Peer Netz gespeichert werden. Diese müssen verschlüsselt abgelegt werden. Die dazu benötigten Keys können auch nicht auf der Blockchain abgelegt werden, da alle Informationen auf der Chain öffentlich einsehbar sind. Deshalt muss ein Oracle beigezogen werden, das die eigentliche Key Verwaltung übernimmt. Im Smart Contract bestätigt der User nur seine Anwesenheit mit einem sogenannten Ping. Trifft innerhalb der vorgegeben Frist kein Ping ein, wird der Key durch das Oracle veröffentlicht.

Risiken und Gefahren

Ein mögliches Risiko ist das Oracle. Dieses kann gezielt angegriffen werden. Mögliche Angriffe müssen vorsichtig analysiert werden.
Eine weiter Gefahr ist der Missbrauch des Systems für unerwünscht Zwecke wie bspw. eine Erpressung. Dies ist jedoch wohl nur schwer zu verhindern und muss in Kauf genommen werden.

Finanzen

Dead Man Switch isn't just about software. Dead Man Switch... Dead Man Switch is about people. Dead Man Switch is about innovative technology that makes a difference, transforming the world as we know it, making the world a better place.

- Gavin Belson - Silicon Valley, S01 E01 [8]

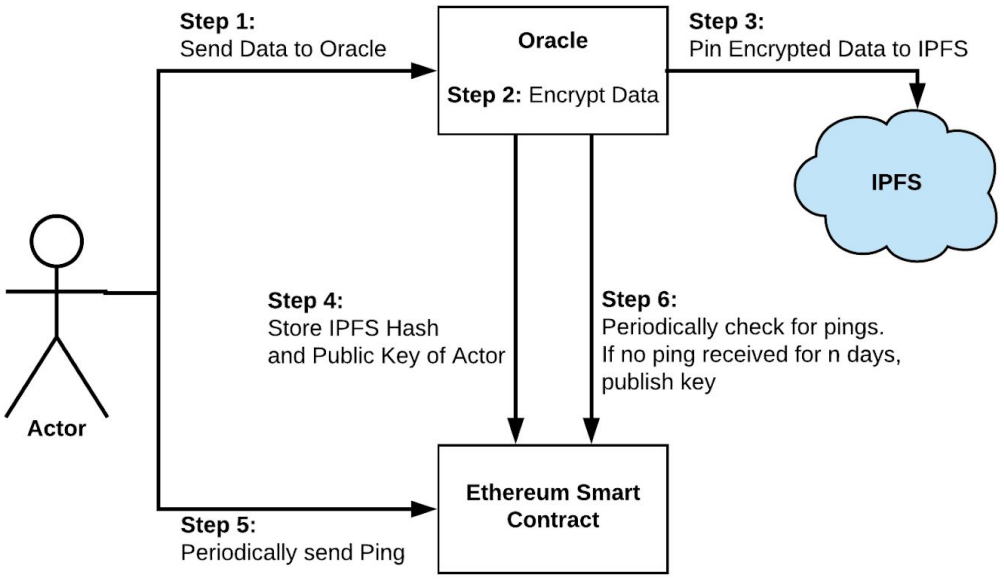
Wir wollen mit diesem Ding kein Geld verdienen. Alle Kosten des Contracts müssen vom User getragen werden und sollten daher minimal gehalten werden.

---

[6] Do you need a Blockchain? - K. Wüst & A. Gervais (Abgerufen am 08.10.2018) https://eprint.iacr.org/2017/375.pdf
[7] InterPlanetary File System (Abgerufen am 08.10.2018) https://ipfs.io/
[8] No Source found. Maybe made up.

Systemarchitektur

Bei der Implementation von diesem Use Case gibt es mehrere Herausforderungen. Wie können die Daten verschlüsselt gespeichert werden, so dass sie nicht manipulierbar sind, dabei aber die Integrität transparent gewährleistet ist? Wie können die Decryption-Keys gespeichert werden, so dass diese von niemandem vor der Veröffentlichung einsehbar sind? Wie kann gewährleistet werden, dass der Ping vom ursprünglichen User stammt und diese nicht manipuliert werden können, bspw. mit einer Replay Attacke?
Um diese Anforderungen zu erfüllen wurde das System in drei Komponenten zerlegt. Einen Ethereum Smart Contract, ein IPFS Deamon und ein Oracle mit einem Web User Interface. Wie diese Systeme untereinander kommunizieren wird im Kapitel Prozesse genauer beleuchtet.



Darstellung 1: Systemübersicht Dead Man Switch

Diese Systemübersicht stellt die Interaktion des Dissidenten, unserem Akteur, mit dem System dar und zeigt alle relevanten Bestandteile. Mit den Steps wird der grobe Ablauf der Interaktionen aufgezeigt.

Smart Contract

Wie im Kapitel Design und Umsetzungsplan grob erläutert, bietet sich für diese Problemstellung eine Permissionless Blockchain an. Die Entschiedene Vorteile einer Umsetzung mit einer Blockchain sind:

- Ping ohne Oracle
  Die Infrastruktur zum persistieren eines Pings ist vollständig verteilt. Keine zentrale Instanz, auch kein Oracle, muss dafür verfügbar sein.

- Einmal publiziert ist der Hash immer öffentlich
  Der Hash der Daten wird in der Blockchain persistiert. Eine nachträgliche Mutation ist dadurch ausgeschlossen und die Integrität der Daten kann nach der Veröffentlichung überprüft werden.
- IPFS File ist öffentlich und validierbar
  Die Daten werden verschlüsselt auf das IPFS gepinnt und sind uneingeschränkt verfügbar.
- Kein Kontrakt mit dem Oracle nach Initialisierung
  Nach der Initialisierung muss nie mehr mit dem Oracle Kontakt aufgenommen werden. Eine Blockierung dieser Applikation ist somit erschwert.
- Authentifizierung von Interaktionen
  Aktionen auf der Blockchain sind grundsätzlich zu einem bestimmten User rückverfolgbar.

Nachteile einer Implementierung auf einer Blockchain sind die zusätzliche Komplexität der Applikation, dass jede Transaktion Geld kostet und durch der längere Laufzeit von On-Chain Transaktionen die Usability reduziert wird.
Die ausgereifteste und am stärksten verbreitete Blockchain dieser Kategorie ist Ethereum. Ethereum bietet die Möglichkeit DAPPs, verteilte Touring komplette Applikationen auf der Blockchain, zu persistieren und auszuführen. Diese sind auch als Smart Contracts bekannt und können mit Solidity, einer von JavaScript inspirierten Sprache, umgesetzt werden. Die durchschnittliche Block Time liegt bei 15 Sekunden. [9] In diesem Anwendungsfall werden aber eher im Bereich von Tagen gearbeitet und die Performance ist somit ausreichend. Deshalb wurde entschieden, den Smart Contract mit Ethereum umzusetzen. Es wird somit das Protokoll und der Consensus Algorithmus von Ethereum verwendet.

IPFS

Um die zu schützenden Daten zu persistieren wird IPFS verwendet. IPFS steht für Interplanetary File System und wird als “Content Addressed, Versioned, P2P File System” beschrieben. [10] Das Oracle fügt die verschlüsselte Datei auf einer laufenden IPFS Node hinzu. Anschliessend ist die Datei global über den dazugehörigen Hash verfügbar. Die hashbasierte Contentadressierung garantiert hierbei die Integrität der Datei. Dieser Hash wird auf die Blockchain geschrieben.
Damit die Daten auf dem IPFS längerfristig persistiert bleiben, müssen diese von mehreren Teilnehmenden gepinnt werden. Die initiale Verteilung wird durch die Interaktion mit einer lokalen IPFS Node gewährleistet.

Oracle

Unsere Systemarchitektur sieht ein Oracle vor, welches für die Initialisierung eines Dead man’s switch zuständig ist. Per Webinterface soll es einem Akteur ermöglicht werden eine

[9] Ethereum BlockTime History (Abgerufen am 18.12.2018) https://etherscan.io/chart/blocktime
[10] IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3) (Abgerufen am 08.10.2018) https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf

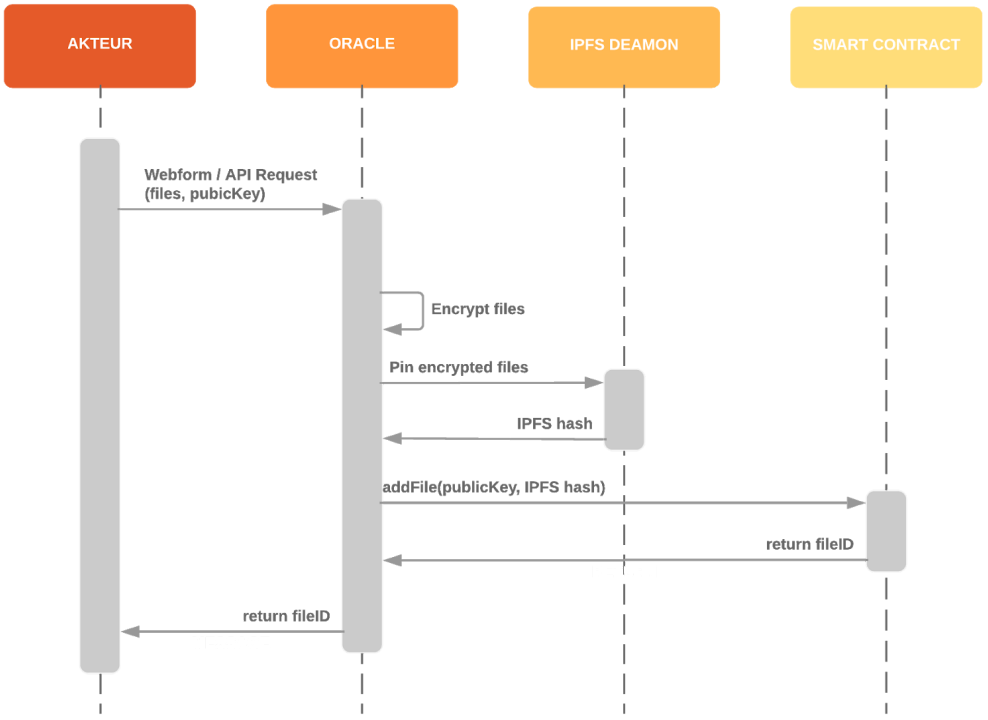
Datei hochzuladen und diese verschlüsseln zu lassen. Die verschlüsselte Datei wird anschliessend vom Oracle zum IPFS Netzwerk hinzugefügt.
Der Key, welcher für die Entschlüsselung der Datei benötigt wird, ist auf dem Oracle gespeichert. Es ist nicht möglich den Key direkt auf der Blockchain zu speichern, da er sonst öffentlich einsehbar wären.
Periodisch wird das Oracle danach die Liste aller noch nicht veröffentlichten Dateien aus der Blockchain lesen und anschliessend prüfen ob Einträge mit abgelaufenen Timestamps existieren. Ist ein Schwellwert erreicht wird der Key der zur Entschlüsselung der jeweiligen Datei benötigt wird auf die Blockchain geschrieben.

Prozesse

In diesem Kapitel wird die Kommunikation zwischen den einzelnen Komponenten und dem Akteur dargelegt. Wir beschränken uns auf die drei wichtigsten Aktionen.

Initialisierung eines Dead Man Switch

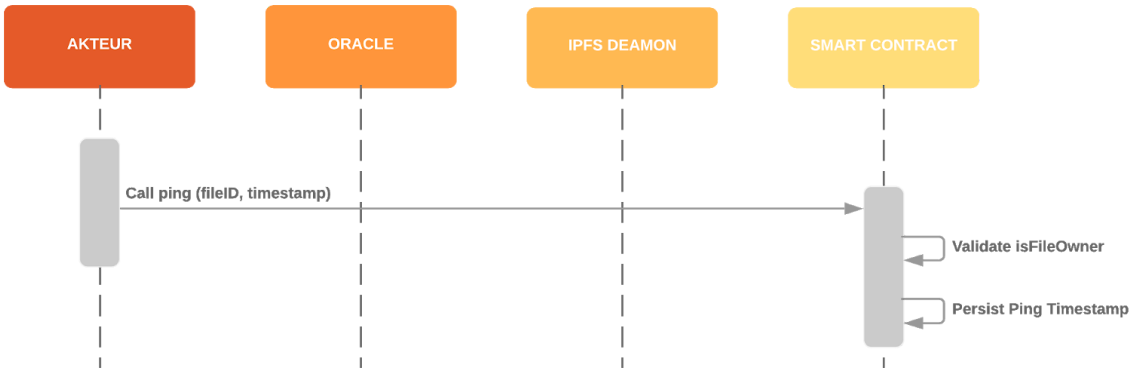
Bei der Initialisierung geht es darum, ein File zu verschlüsseln, zu veröffentlichen und einen entsprechenden Eintrag im Smart Contract zu persistieren. Die Vorbedingung dazu ist, dass ein Oracle läuft und dieses einen Smart Contract initialisiert hat.



Darstellung 2: Sequenzdiagramm - Initialisierung

Über das Web UI oder einen REST Call kann dem Oracle den Auftrag gegeben werden, ein File zu speichern. Dazu muss das File und der Public Key des Akteurs übermittelt werden. Das Oracle steuert nun den Prozess und gibt schlussendlich die ID des Files dem Akteur zurück.

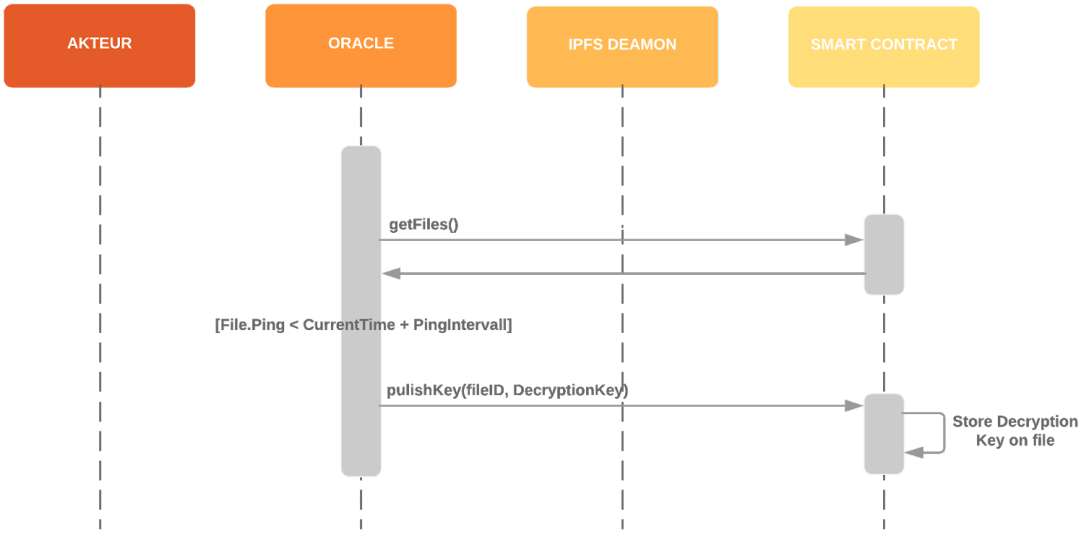
Ping eines Dead Man Switch



Darstellung 3: Sequenzdiagramm - Ping

Mit einem Ping bestätigt der Akteur, dass es ihn noch gibt. Dazu sendet er die FileID und ein UNIX Timestamp. Im Smart Contract ist zur FileID auch die Adresse des Akteurs gespeichert und so kann gewährleistet werden, dass nur dieser Akteur darauf zugreifen kann. Diese Aktion ist eine Call Transaktion an den Smart Contract und es gibt somit kein Kontakt zwischen Akteur und Oracle.

Veröffentlichung der Daten bei ausbleibendem Ping



Darstellung 4: Sequenzdiagramm - Veröffentlichung

Bei der Veröffentlichung ist Augenfällig, dass der Akteur hier keine Rolle spielt. Das Oracle liest periodisch die aktuellen Daten aus dem Smart Contract. Ist ein Ping älter als die aktuelle Zeit, wird der Key auch auf im Contract gespeichert. Ist noch gar kein Ping vorhanden, wird dieser Eintrag ignoriert. Das hat der Vorteil, dass der User dadurch erst validiert wird und der Contract somit auch erst nach der Initialisierung aktiviert werden kann. Aktuell wird alle dreissig Sekunden überprüft ob ein File abgelaufen wird. Sollte das Oracle einmal nicht verfügbar sein, bspw. durch ein DoS-Angriff, wird die Veröffentlichung zwar verzögert aber nicht langfristig verhindert.

Implementation

Zu Beginn wurde eine erste Version des Smart Contracts umgesetzt, da uns dieser Teil des Projekts am bekanntesten war und wir daher das daher als grösstes Risiko einschätzen. Der Kern der Implementation des Smart Contracts besteht aus einem Array eines Structs, welches ein File repräsentiert. Bei der Initialisierung wird ein neues File Objekt angelegt. Darin wird die Adresse des Owners, in diesem Fall der Public Key des Akteurs, und der IPFS Hash persistiert. Das files Array ist öffentlich lesbar.

```
struct File {
    address fileOwner;
    string ipfsHash;
    string key;
    string ping;
}
mapping (uint => File) public files;
```

Der Smart Contract bietet zur Manipulation dieser Daten die folgenden drei Methoden an.
Der Contract DeadSwitch wurde mit der aktuellsten Solidity Version 0.5.1, Stand 16.12.2018, implementiert. [11]

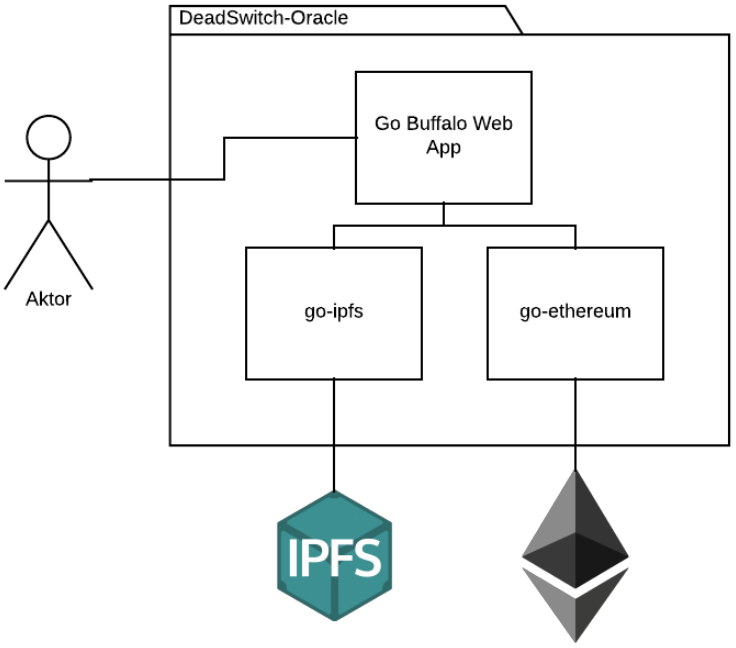
```
pragma solidity ^0.5.1;
contract DeadSwitch {
   [..]
   function addFile(string memory _ipfsHash, address _fileOwner)
      public onlyCreator returns (uint fileId) {[..]}

   function pubishKey(string memory _key, uint _fileID)
      public onlyCreator{[..]}

   function ping(string _ping, uint _fileID)
      public onlyFileOwner(_fileID) {[..]}
}
```

Speziell hervorzuheben sind die beiden Access Modifier *onlyCreator* und *onlyFileOwner*.
Für *onlyCreator* wird im Konstruktor des Contracts die Adresse des Erstellers des Contracts persistiert. Somit kann nur der Ersteller, d.h. das Oracle, die Methoden *addFile* oder *publishKey* ausführen.
Der *onlyFileOwner* ist ein Access Modifier der sicherstellt, dass nur der Besitzer des Files darauf einen Ping speichern kann. Das wird dadurch ermöglicht, dass die Adresse der Besitzers direkt auf dem entsprechenden File gespeichert wird. Die Berechtigung für eine Datenmanipulation ist dadurch dynamisch direkt auf den Daten selber gespeichert.

[11] Solidity (Abgerufen am 18.12.2018) https://solidity.readthedocs.io/en/v0.5.1/

```
modifier onlyFileOwner(uint _fileId) {
   require(files[_fileId].fileOwner == msg.sender);
   _;
}
```

Parallel dazu wurde ein PoC für die Funktionen des Oracles umgesetzt. Der technische Aufbau des Oracles sieht folgendermassen aus:



Darstellung 5: Aufbau Oracle

Für die Implementation wurde die Programmiersprache Go verwendet. Diese wurde eingesetzt, da sie exzellent für die Interaktion mit IPFS und Ethereum geeignet ist. Die beiden offiziellen Implementationen von IPFS (go-ipfs[12]) und Ethereum (go-ethereum[13]) sind in golang implementiert und können somit direkt aus dem Oracle Code aufgerufen werden.

Der wichtigste Teil des Oracles bildet die Interaktion mit der Ethereum Blockchain, resp. dem Smart Contract. Um aus dem Go Code direkt mit dem Smart Contract zu interagieren wird dieser zuerst von Solidity nach Go Code mit folgendem Befehl kompiliert[14].

```
abigen --sol=deadManSwitch.sol --pkg=main --out=DeadManContract.go
```

[12] GitHub go-ipfs (Abgerufen am 18.12.2018) https://github.com/ipfs/go-ipfs
[13] GitHub Go Ethereum (Abgerufen am 18.12.2018) https://github.com/ethereum/go-ethereum
[14] Interacting with Ethereum Smart Contracts using Go (Abgerufen am 18.12.2018) https://zupzup.org/eth-smart-contracts-go/

Das daraus resultierende Code File “DeadManContract.go” beinhaltet eine Funktion welche es erlaubt den Contract zu deployen.

```
_, transaction, contract, err = DeployDeadSwitch(auth, sim)
```

Sobald der Contract deployed ist und eine Instanz davon verfügbar ist, können die einzelnen Funktionen “RPC mässig” direkt aus dem Go Code aufgerufen werden. So kann zum Beispiel ein File Eintrag ausgelesen werden

```
file, err := contract.Files(&bind.CallOpts{
      From: auth.From,
}, id)
```

Der zweite kritische Teil der Applikation ist die Interaktion mit IPFS, respektive die verschlüsselung der Daten und das anschliessende hinzufügen der Daten zum IPFS-Netzwerk.

Die Daten werden mit AES-256 verschlüsselt und anschliessend der lokalen IPFS Node hinzugefügt.

```
// upload and pin to IPFS
ipfsHash, err := sh.Add(bytes.NewReader(ed))
```

Sobald die Datei zu IPFS hinzugefügt wurde kann der Hash und der entsprechende User in den Smart Contract aufgenommen werden. Folgendes ist der kritische Ausschnitt aus dieser Funktion:

```
fmt.Println("Add ipfs hash", ipfsHash, "to blackchain...")
_, err := contract.AddFile(&bind.TransactOpts{
      From:     auth.From,
      Signer:   auth.Signer,
      GasLimit: 230000000,
}, ipfsHash, toAddress)
if err != nil {
      log.Println(err)
      return errors.WithStack(err)
}
fmt.Println("Mining...")
sim.Commit()
```

Da wir uns auf einer von go-ethereum simulierten Blockchain bewegen, darf nicht vergessen werden den nächsten Block mit sim.Commit() zu “minen”.

Im Hintergrund überprüft das Oracle immer wieder ob ein Deadman Switch getriggert werden soll, weil schon zu lange kein Ping mehr abgesetzt wurde. Dies wurde mit einer im

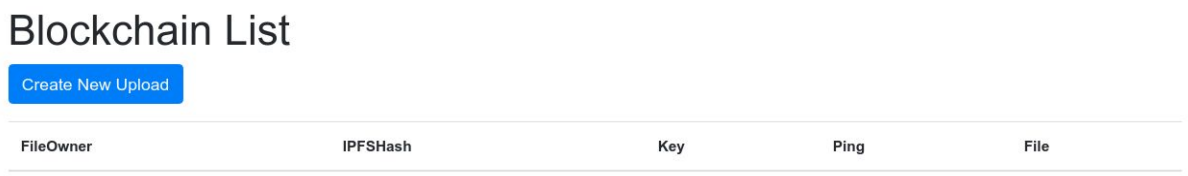
Hintergrund laufenden goroutine[15] bewerkstelligt. Sollte ein Ping zu alt sein veröffentlicht das Oracle den Schlüssel zur entsprechenden Datei auf der Blockchain.

```
func writeKeyToBlockchain(fileID int, key string) error {
	transaction, err := contract.PubishKey(&bind.TransactOpts{
		From:   auth.From,
		Signer: auth.Signer,
	}, key, big.NewInt(int64(fileID)))

	sim.Commit()
	fmt.Println("Gas used to write key to blockchain: ",
transaction.Cost())
	return err
}
```

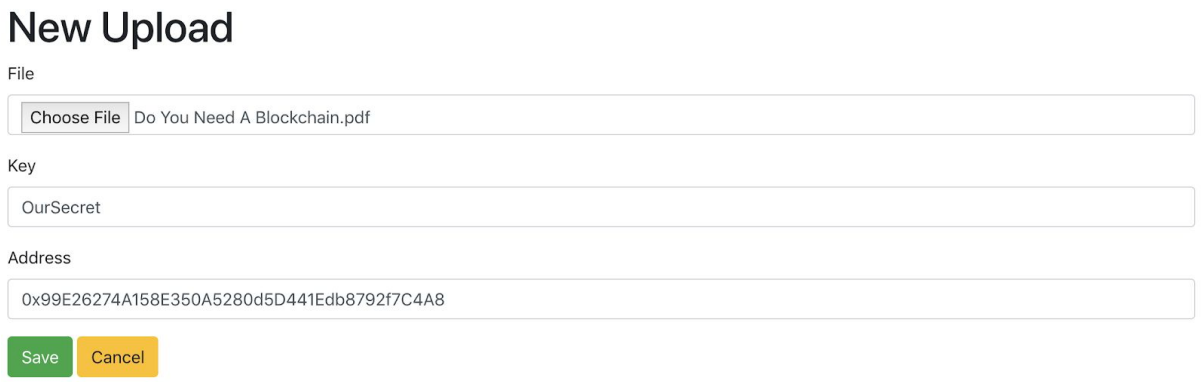
Sobald der Schlüssel in die Blockchain geschrieben wurde kann die AES verschlüsselte Datei eigens entschlüsselt werden oder bequem über das Oracle heruntergeladen werden.

Benutzeroberfläche

Die Benutzeroberfläche ist bewusst minimalistisch gehalten. Der erste Printscreen zeigt die Ausgangslage. Aktuell ist kein File im Contract gespeichert.

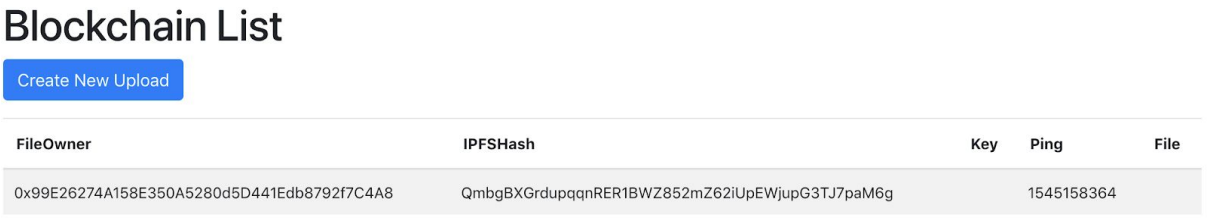


Klickt man auf “Create New Upload” wird das folgende Formular angezeigt. Ein Akteur kann hier die Datei, ein Key und sein Public Key angeben.

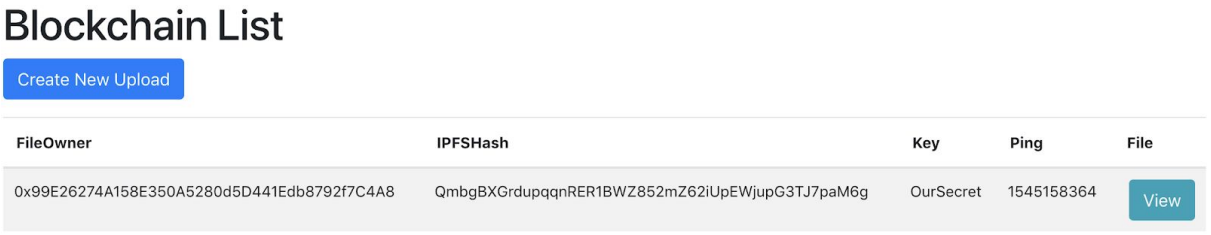


[15] Go by Example: Goroutines (Abgerufen am 18.12.2018) https://gobyexample.com/goroutines

Sobald die Datei hochgeladen, verschlüsselt und ein Eintrag auf dem Smart Contract erfolgreich angelegt wurde, wird ein Eintrag in der Liste angezeigt. Sowohl der Key wie auch ein Ping sind zu diesem Zeitpunkt noch nicht vorhanden. Sobald der erste Ping eingetroffen ist, wird dieser in der Liste angezeigt.



Läuft der Ping ab, wird der Key von dem Oracle publiziert. Ist der Key vorhanden wird ein Button angezeigt, über welchen die Datei direkt entschlüsselt wird und heruntergeladen werden kann.



Kosten

Alle Transaktionskosten, bis auf den Ping, werden vom Oracle getragen. Das ist ein Sicherheitslücke, da jemand beliebig viele Files auf Kosten des Oracles hochladen kann. Die Transaktionskosten wurden mit Remix ermittelt. [16] Die Kosten halten sich wie geplant in Grenzen.

Action	**Cost in Gas**	**Transaction Cost ***	**Interval**
Create Contract	909’926	$ 0.190	Once
AddFile	96’854	$ 0.020	Once per File
PublishKey	43’646	$ 0.009	Once per File
Ping	44’035	$ 0.009	Min. Once per Ping Interval

* Berechnet mit ethgasstation.info bei einem Gas Price von 2.4 Gwei [17]

[16] Remix - Solidity IDE (Abgerufen am 18.12.2018) https://remix.ethereum.org
[17] ETH Gas Station (Abgerufen am 18.12.2018) https://ethgasstation.info

Evaluation

Der Use Case wurde mit M. Langhart durchgespielt. Beim Einfügen des Public Keys und beim Ausführen des Pings musste er unterstützt werden. Der Business Case konnte von Herr Langhart nachvollzogen werden, aber berechtigte Kritik am User Interface wurde ausgeübt.

Fazit

Die umgesetzte Lösung erfüllt die zu Beginn skizzierte Lösung als Proof of Concept. Wir sind mit dem Resultate zufrieden und halten diesen Use Case auch nach näherer Prüfung als sinnvoll. Der Fokus wurde auf die Interaktion mit der Ethereum Blockchain und IPFS gelegt.
Die Skalierbarkeit des Systems ist gegeben, da die Pings direkt über die Blockchain umgesetzt werden und das Oracle nicht beanspruchen. Auch werden die Dateien verteilt abgelegt, wobei der Speicher mit zusätzlichen IPFS Nodes erweitert werden kann oder von Pin Anbietern gebrauch gemacht werden kann.
Die Interaktion zwischen dem Web Framework gobuffalo, IPFS und der Ethereum Blockchain hat aufgrund der gleichen Programmiersprache sehr gut funktioniert. Die Kompilierung des Solidity Codes zu Golang stellte Funktionsaufrufe des Smart Contract so zur Verfügung, als wäre es ein Golang Package.
Ein offenes Problem sind die Gas Kosten für die Aktionen des Oracles. Diese werden nicht wie geplant auf den User abgewälzt und ein bösartiger Akteur könnte das Oracle pleite gehen lassen. Um diese Problem zu lösen, müsste bei der Initialisierung eines Files ein Gas Betrag mitgegeben werden.
Der Source Code ist öffentlich auf Github (https://github.com/eduDorus/dead-man-switch) verfügbar.

Ausblick

Eine bereits angedachte Erweiterung, welche jedoch noch nicht implementiert werden konnte ist die Verteilung des Oracles. Eine solide Lösung wäre hierbei der Einsatz eines Zero Knowledge Protokoll zum Teilen der Secrets. [18]
Zudem wäre interessant eine incentive basierte Erweiterung des Oracles zu entwickeln. Die Oracles sollten hierbei einen finanziellen Ansporn haben, die verschlüsselten Daten auf IPFS gepinnt zu belassen und der dazugehörige Decryption Key nicht unerwünscht zu veröffentlichen. Eine weitere Erweiterung wäre eine Verbesserung der Nutzerführung und das Hinzufügen einer Landing Page.

[18] Secret Sharing DAOs: The Other Crypto 2.0 (Abgerufen am 18.12.2018)
https://blog.ethereum.org/2014/12/26/secret-sharing-daos-crypto-2-0/