Cutting the Onion

An Introduction to the Tor Protocol

Emil Engler

GPN 21

8. Juni 2023

Über mich

- Emil Engler (any pronouns)
- Abitur 2023 → Mathestudium ab Oktober am KIT
- FLOSS Contributor seit 2018 (*cURL*, *Rosenpass*, *Tor*)
- BSD > Linux

- `emilengler`
- `@engler@chaos.social`
- `https://emilengler.com`

Onion Routing

Onion Routing? Kann man das essen?

- Alice's IP muss gegenüber Bob geheim bleiben
- Bob's IP darf öffentlich sein
- Der Traffic muss verschlüsselt sein

Der klassische gute Proxy

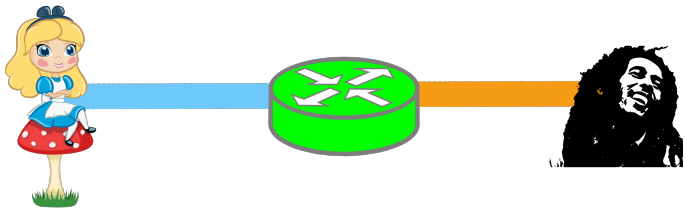

Abbildung: Klassischer guter Proxy

Der klassische gute Proxy

- Bob kennt nicht Alice's IP ✔
- Der Proxy weiß, dass Alice mit Bob kommuniziert ☠
  - Alice muss dem Proxy vertrauen
  - Single point of failure
- Der Proxy kennt alle Kommunikationsdaten ☠

Der klassische böse Proxy

Abbildung: Klassischer böser Proxy

Der klassische böse Proxy

- Der Proxy kennt Alice
- Der Proxy kennt Bob
- Der Proxy kennt sämtliche Daten
- *PENG!*

Der klassische böse Proxy

Abbildung: Jetzt 80% Rabatt mit dem Code *shadowlegends*

Multi-Hop-Routing

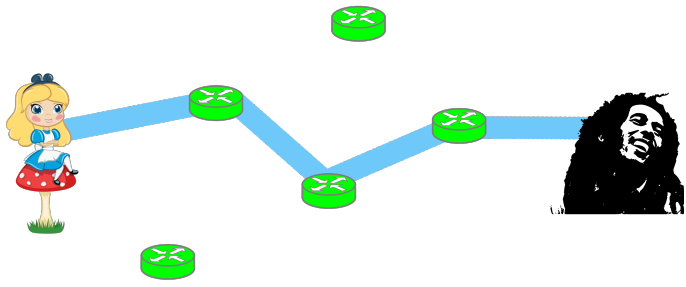

Abbildung: Multi-Hop-Routing ohne Kryptografie

Multi-Hop-Routing

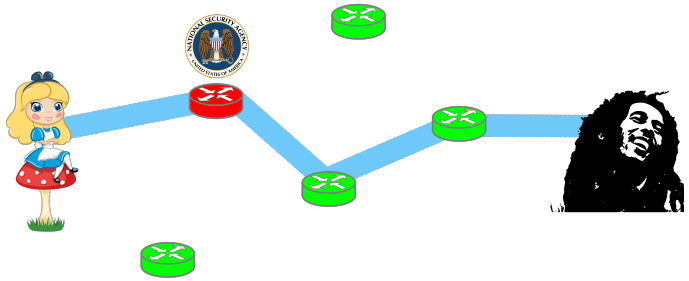

Abbildung: Multi-Hop-Routing ohne Kryptografie

Multi-Hop-Routing

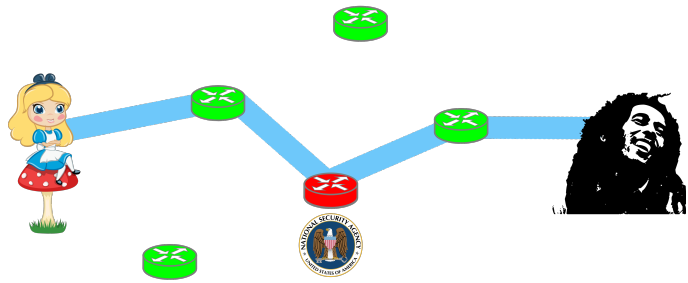

Abbildung: Multi-Hop-Routing ohne Kryptografie

Multi-Hop-Routing

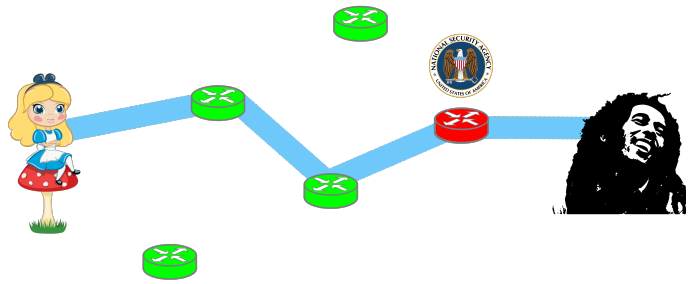

Abbildung: Multi-Hop-Routing ohne Kryptografie

Multi-Hop-Routing

- Bob kennt Alice's IP nicht ✔
- Niemand weiß, dass Alice mit Bob kommuniziert ✔
- Die Daten sind noch immer unverschlüsselt ☠
  - Immer noch irgendwie 💩

Multi-Hop-Routing

Can we do any better?

Abbildung: Miles Glacier Bridge (1984)

Onion Routing

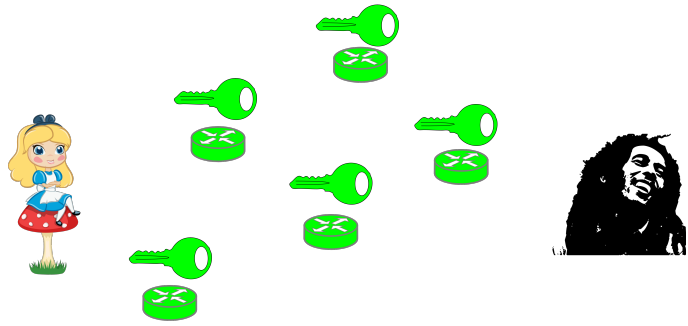

Abbildung: Onion Routing mit PKs

Onion Routing

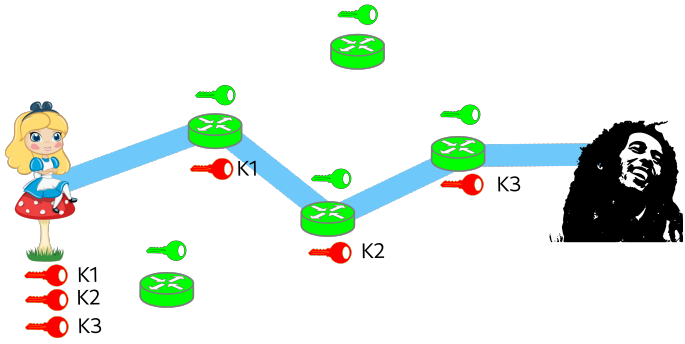

Abbildung: Onion Routing mit Keyexchange

Onion Routing

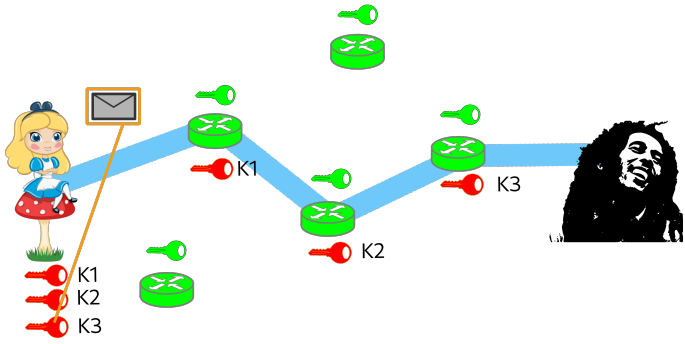

Abbildung: Onion Routing

Onion Routing

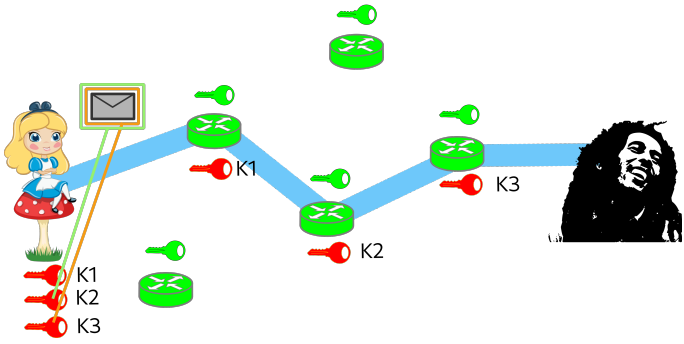

Abbildung: Onion Routing

Onion Routing

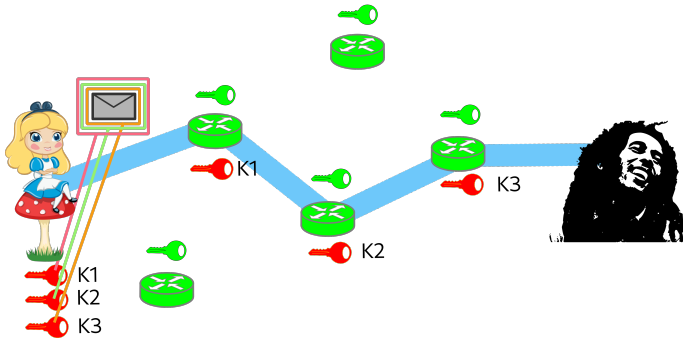

Abbildung: Onion Routing

Onion Routing

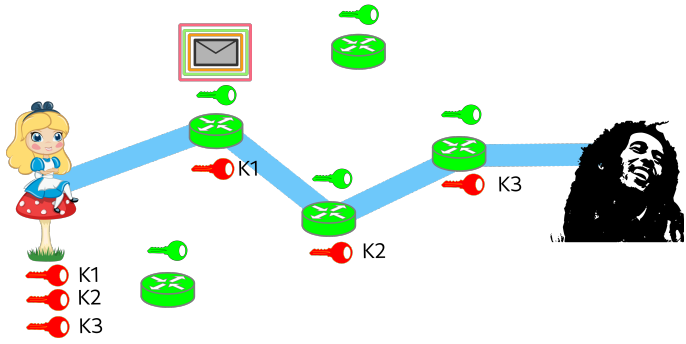

Abbildung: Onion Routing

Onion Routing

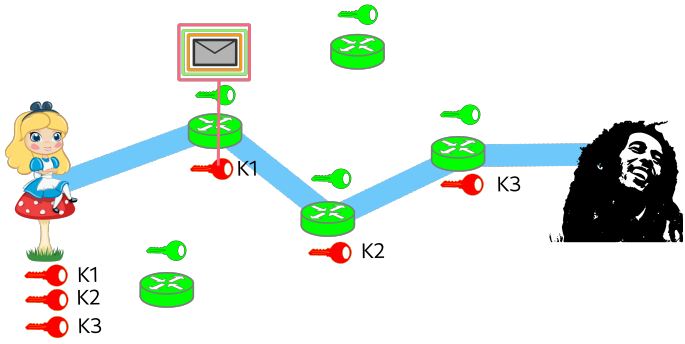

Abbildung: Onion Routing

Onion Routing

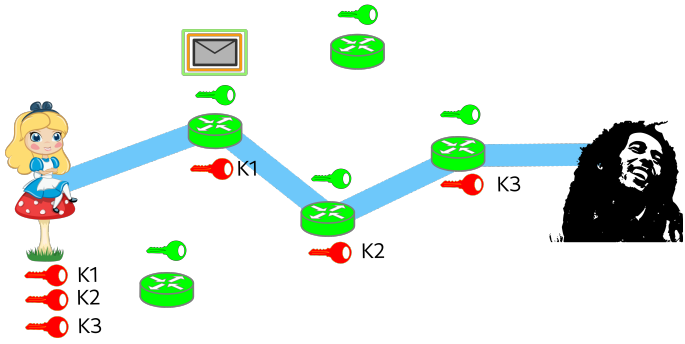

Abbildung: Onion Routing

Onion Routing

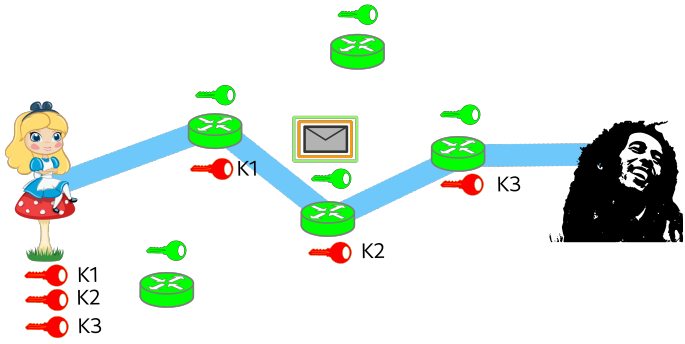

Abbildung: Onion Routing

Onion Routing

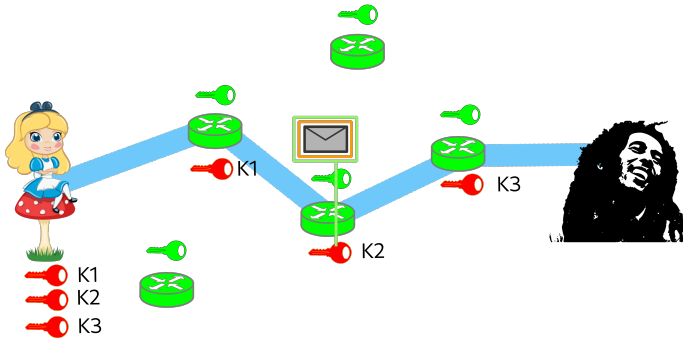

Abbildung: Onion Routing

Onion Routing

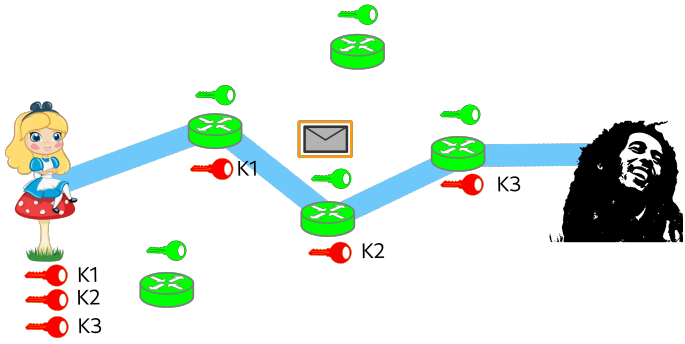

Abbildung: Onion Routing

Onion Routing

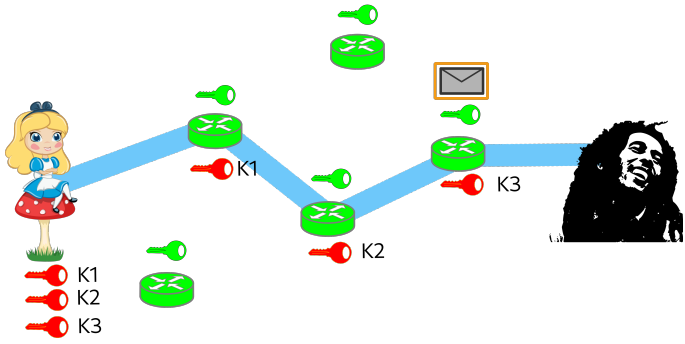

Abbildung: Onion Routing

Onion Routing

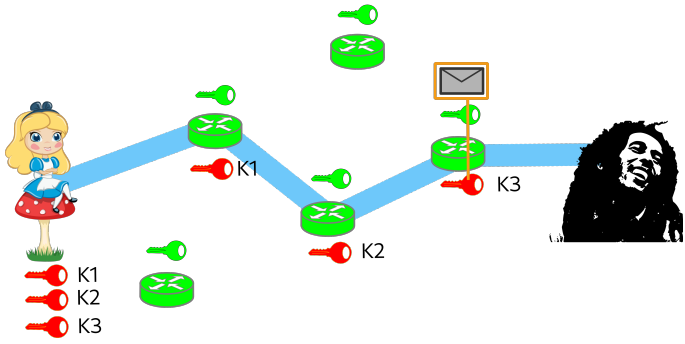

Abbildung: Onion Routing

Onion Routing

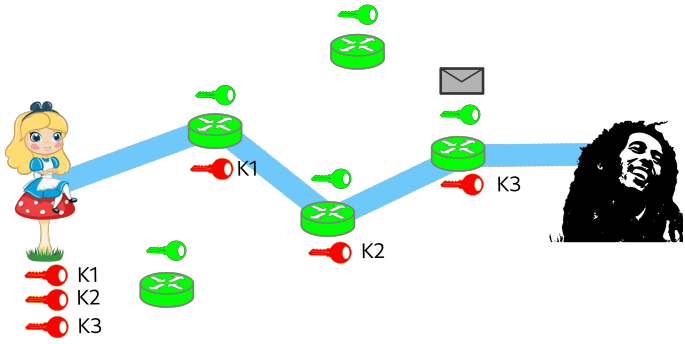

Abbildung: Onion Routing

Onion Routing

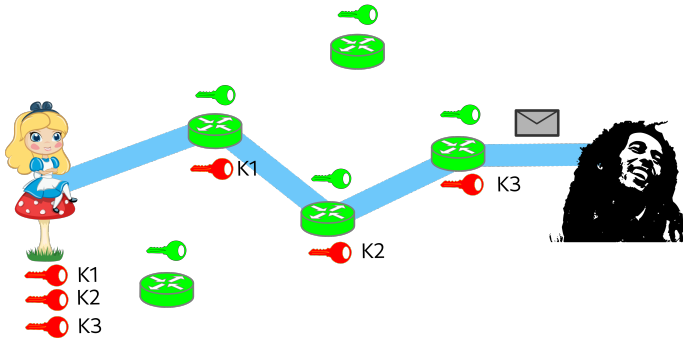

Abbildung: Onion Routing

Onion Routing

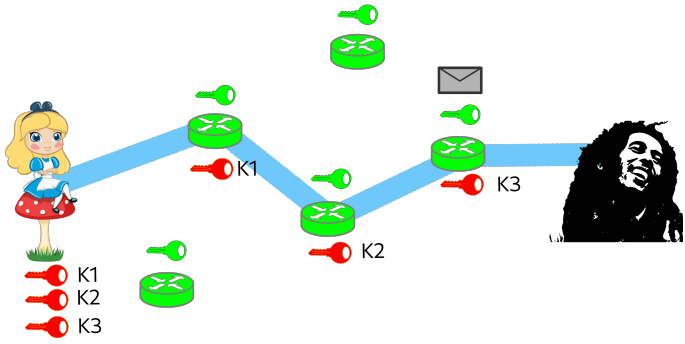

Abbildung: Onion Routing

Onion Routing

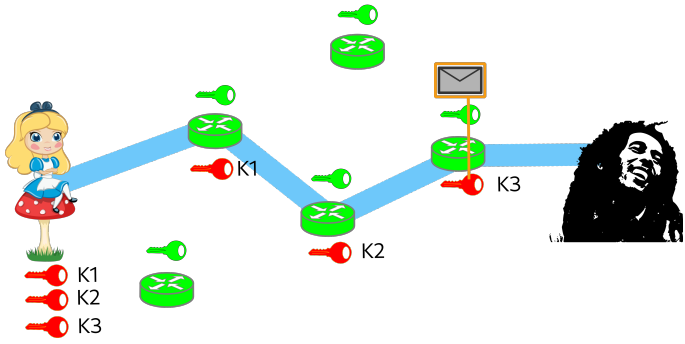

Abbildung: Onion Routing

Onion Routing

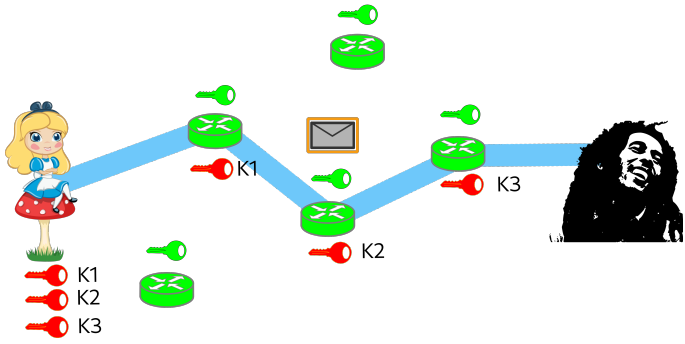

Abbildung: Onion Routing

Onion Routing

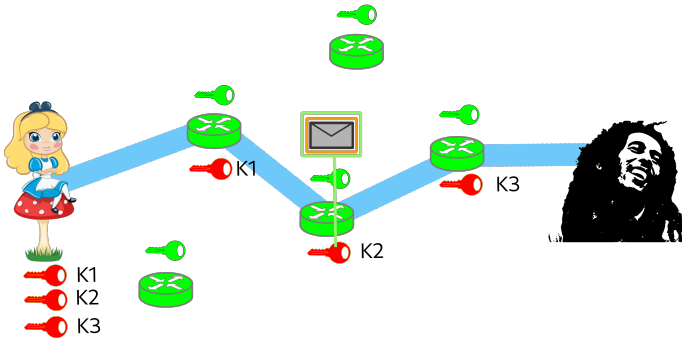

Abbildung: Onion Routing

Onion Routing

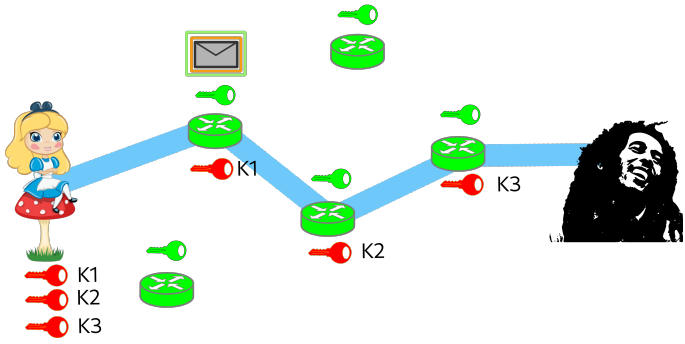

Abbildung: Onion Routing

Onion Routing

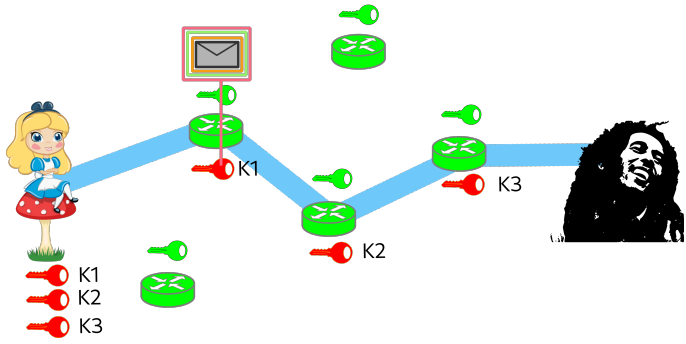

Abbildung: Onion Routing

Onion Routing

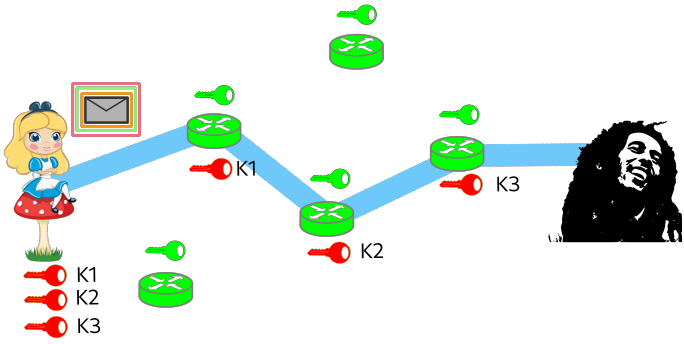

Abbildung: Onion Routing

Onion Routing

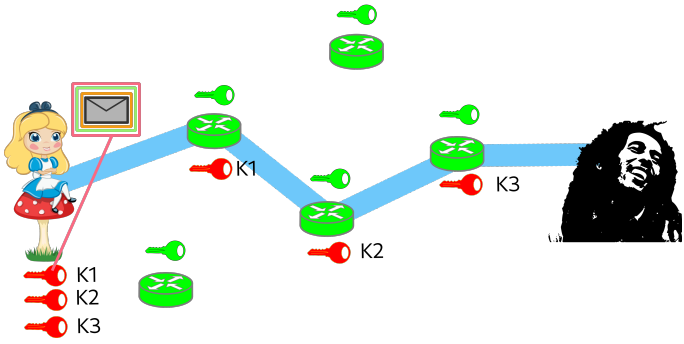

Abbildung: Onion Routing

Onion Routing

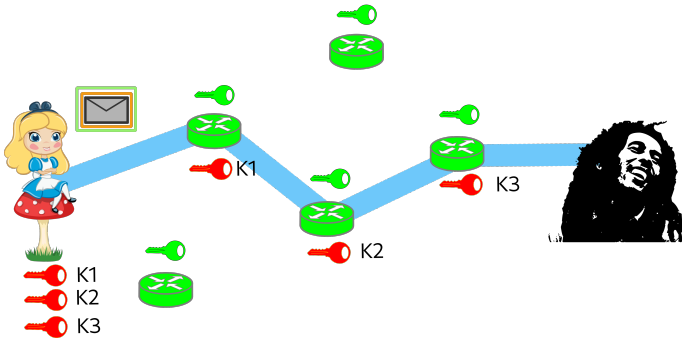

Abbildung: Onion Routing

Onion Routing

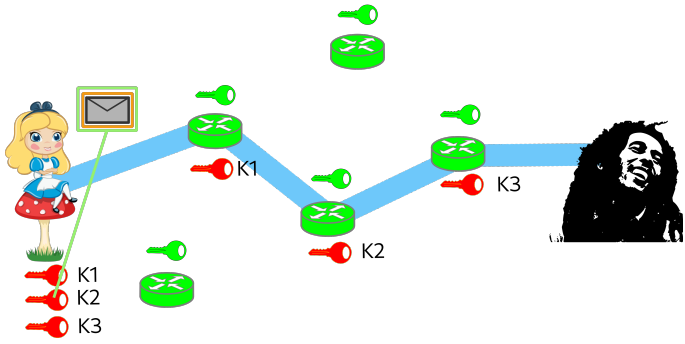

Abbildung: Onion Routing

Onion Routing

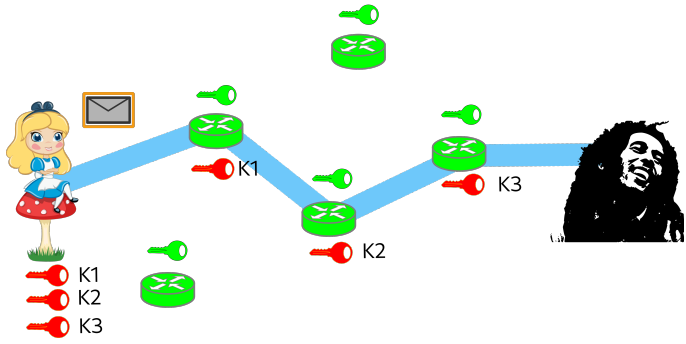

Abbildung: Onion Routing

Onion Routing

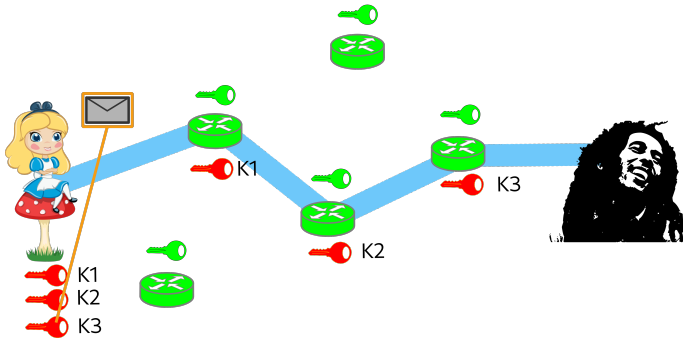

Abbildung: Onion Routing

Onion Routing

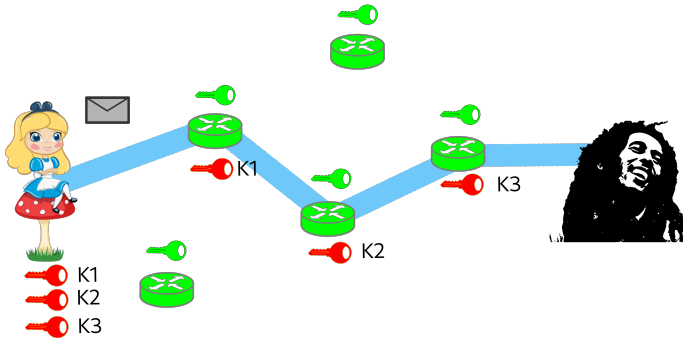

Abbildung: Onion Routing

Das Tor-Protokoll

Das Tor-Protokoll – Was ist Tor?

- Verteiltes Netzwerk zur Anonymisierung von TCP-Traffic
- Clients wählen einen Pfad an Nodes und bauen ein *Circuit*
- Kommunikation findet in festen Paketen (*Cells*) statt

Das Tor-Protokoll – Keys

- Jedes Tor-Node hat eine mehrere öffentliche Keys
  - *RSA1024-* und *Ed25519-Identity-Keys*
  - *Ed25519-Signing-* und *Ed25519-Authentication-Key*
  - *Curve25519* Handshake Key
- Ursprünglich war alles RSA1024, wurde 2012 auf ECC umgestellt
- **Jeder kennt die Public-Keys von jedem**

Das Tor-Protokoll – Cell

- Fundamentaler Message-Type im Tor-Protokoll
- 514-Byte lange Nachricht
  - *Circuit ID* 4 Byte
  - *Command* 1 Byte
  - *Payload* 509 Byte
- Noch keine Verschlüsselung/Onion Routing an dieser Stelle

Das Tor-Protokoll – Cell

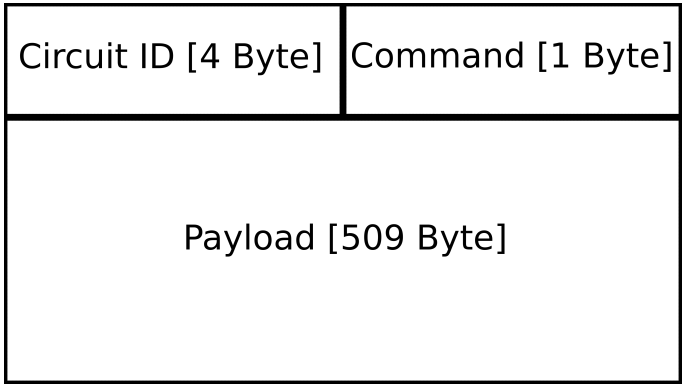

Abbildung: Eine *Cell* visualisiert

Das Tor-Protokoll – Circuit ID

- Identifiziert das *Circuit*
  - Dazu später mehr ...
- Die *Circuit ID* 0 bezieht sich auf die *Connection*
  - Dazu später mehr ...

Das Tor-Protokoll – Command

- Integer welcher der Cell eine Bedeutung gibt
- *Payload* hängt von diesem *Command* ab

- `3 -- RELAY`
- `4 -- DESTROY`
- `7 -- VERSIONS`
- `8 -- NETINFO`
- `10 -- CREATE2`
- `11 -- CREATED2`
- `129 -- CERTS`
- `130 -- AUTH_CHALLENGE`
- `131 -- AUTHENTICATE`

Connection

Connection

- Eine stumpfe TLS-Verbindung zwischen Client-Node bzw. Node-Node
- Verwenden (fast) das selbe Protokoll
- Machen am Anfang einen Handshake
- Besteht aus mehreren *Circuits*

Connection – Client-Node Handshake

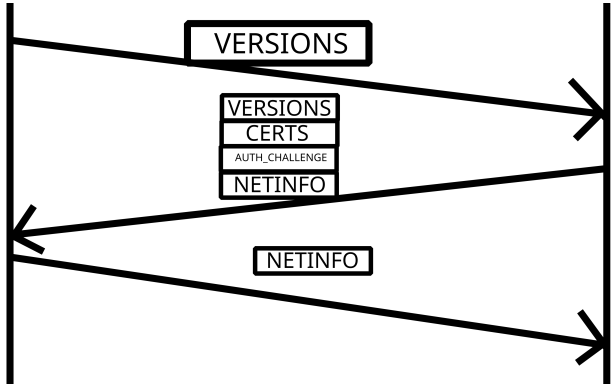

Abbildung: Der Client-Node Handshake

Connection – VERSIONS

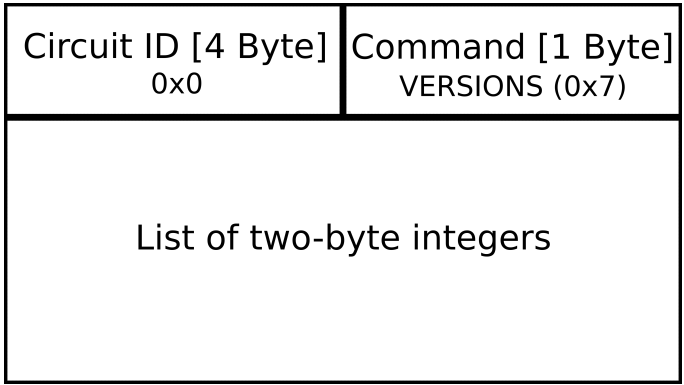

Abbildung: Eine Cell mit dem `VERSIONS`-Command

Connection – CERTS

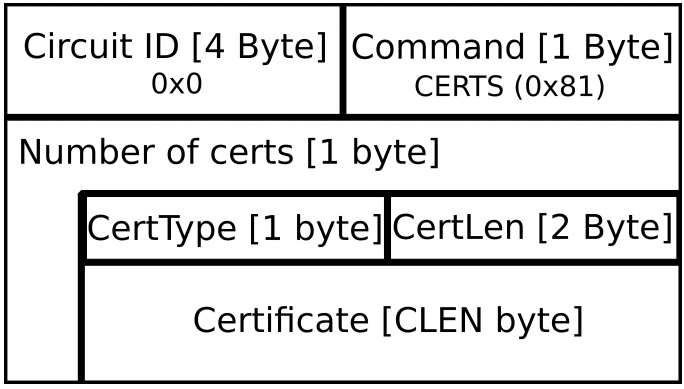

Abbildung: Eine Cell mit dem CERTS-Command

Connection – AUTH_CHALLENGE

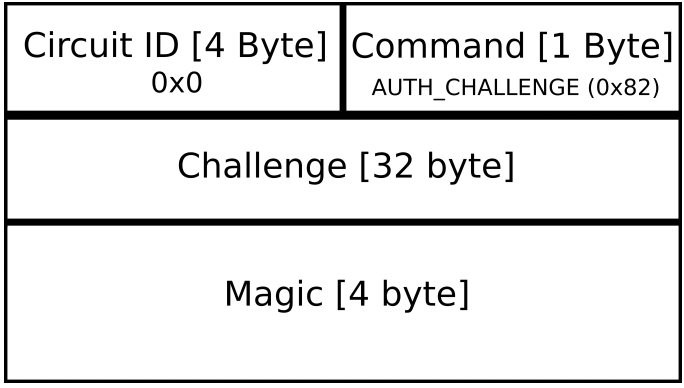

Abbildung: Eine Cell mit dem `AUTH_CHALLENGE`-Command

Connection – NETINFO

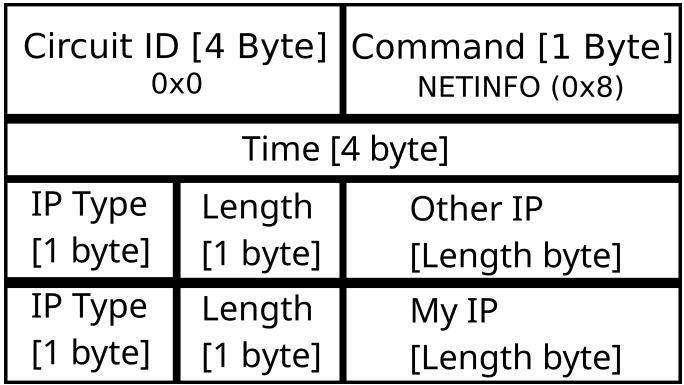

Abbildung: Eine Cell mit dem `NETINFO`-Command

Connection – Client-Node Handshake

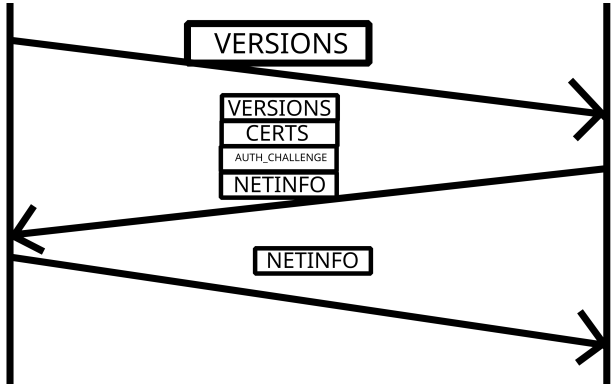

Abbildung: Der Client-Node Handshake

Connection – Node-Node Handshake

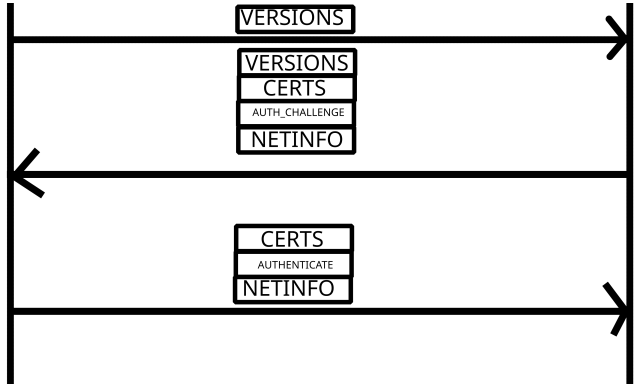

Abbildung: Der Node-Node Handshake

Connection – AUTHENTICATE

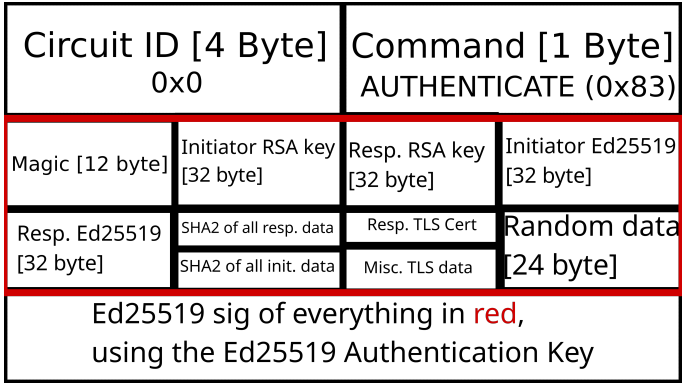

Abbildung: Eine Cell mit dem `AUTHENTICATE`-Command

Connection – Node-Node Handshake

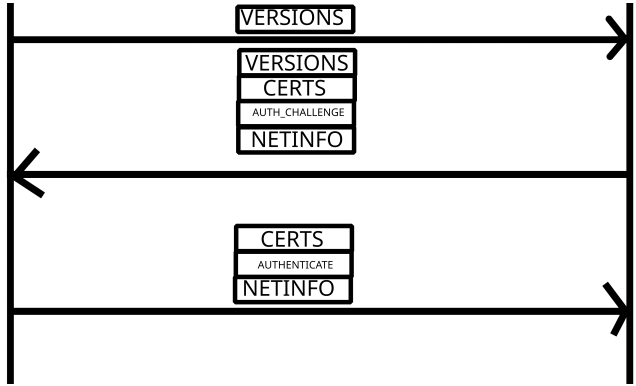

Abbildung: Der Node-Node Handshake

Circuit

Circuit

- Tatsächlicher Pfad im Tor-Netzwerk, bestehend aus drei Nodes
- Ein Tor-Client hat in der Regel mehrere Circuits gleichzeitig
- Sind immer einer Connection zugewiesen
- Werden inkrementell mit Handshakes aufgebaut
  - *Warnung*: Henne-Ei-Problem incoming

Ground Control to Major Tom
Your circuit's dead, there's something wrong
Can you hear me, Major Tom? – David Bowie

Circuit – Handshake mit dem ersten Node

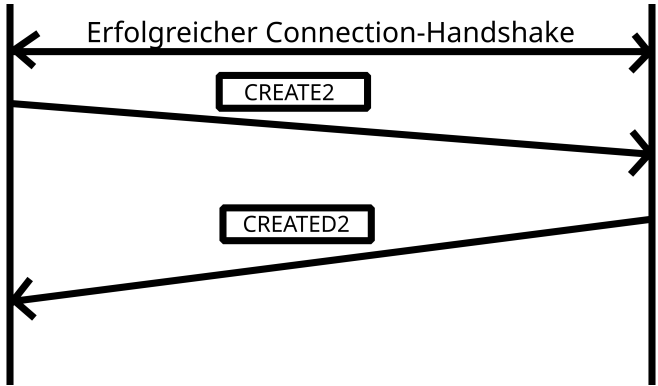

Abbildung: Der Handshake mit dem ersten Hop

Circuit – CREATE2

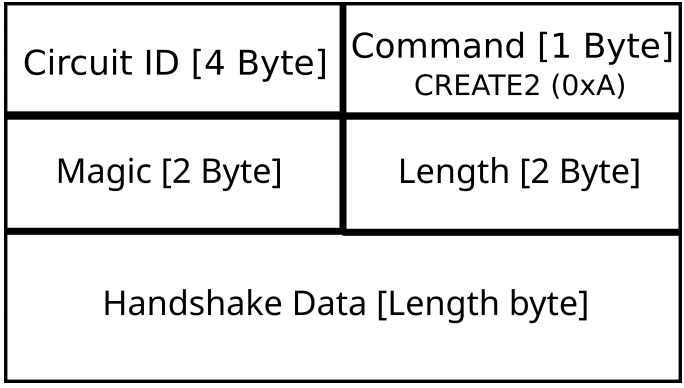

Abbildung: Die `CREATE2`-Cell

Circuit – CREATED2

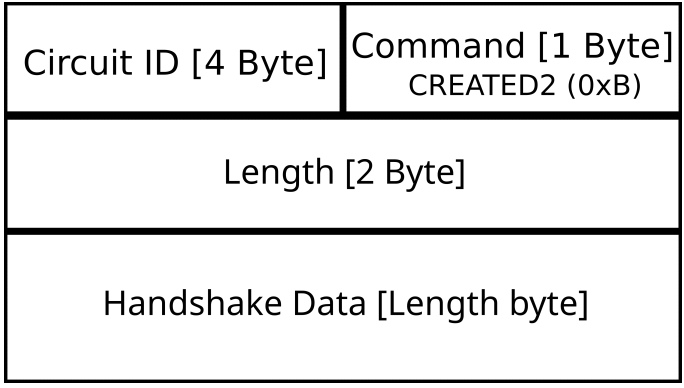

Abbildung: Die `CREATED2`-Cell

Circuit – Handshake mit dem ersten Node

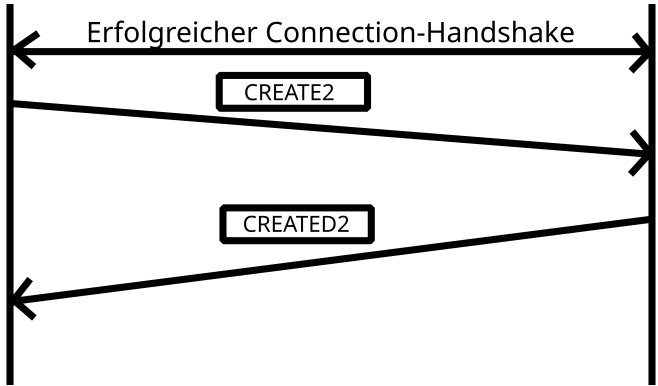

Abbildung: Der Handshake mit dem ersten Hop

----- BEGIN CRYPTOGRAPHY -----

Circuit – Der Circuit-Handshake – Client

i $ID := SHA256(\text{`Public-RSA1024-Identity-Key vom Node`})$

ii $B :=$ Public-Curve25519-Key vom Node

iii $x := KEYGEN(\text{`Curve25519`})$

iv $X := PUBKEY(x)$

Handshake Data := $ID \mid B \mid X$ [84 Byte]

Client erstellt und sendet `CREATE2` Cell mit *Handshake Data* im Payload

Circuit – Der Circuit-Handshake – Node

i $y := KEYGEN(\text{`Curve25519`})$
ii $Y := PUBKEY(y)$
iii $b := PRIVKEY(B)$

- $shared\_secret := ECDH(X, y) \mid ECDH(X, b) \mid ID \mid B \mid X \mid Y \mid MAGIC_1$
- $key\_seed := HMAC\_SHA256(shared\_secret, MAGIC_2)$
- $verify := HMAC\_SHA256(shared\_secret, MAGIC_3)$
- $auth\_input := verify \mid ID \mid B \mid Y \mid X \mid MAGIC_4$

Handshake Reply $:= Y \mid HMAC\_SHA256(auth\_input, MAGIC_5)$
Node erstellt und sendet eine `CREATED2` Cell mit *Handshake Reply* im Payload

Circuit – Der Circuit-Handshake – Client

- $shared\_secret := ECDH(Y, x) \mid ECDH(B, x) \mid ID \mid B \mid X \mid Y \mid MAGIC_1$
- $key\_seed := HMAC\_SHA256(shared\_secret, MAGIC_2)$
- $verify := HMAC\_SHA256(shared\_secret, MAGIC_3)$
- $auth\_input := verify \mid ID \mid B \mid Y \mid X \mid MAGIC_4$

Der Client überprüft nun, ob $HMAC\_SHA256(auth\_input, MAGIC_5)$ identisch mit dem Wert aus `CREATED2` ist

Circuit – Key-Derivation

- Beide Partner haben nun einen Wert für *key_seed*
- Dieser Wert wird als Seed für die HKDF Funktion aus RFC 5869 verwendet

- Byte 0-19: Forward Digest (D_f)
- Byte 20-39: Backward Digest (D_b)
- Byte 40-56: Symmetric Forward Key (K_f)
- Byte 56-72: Symmetric Backward Key (K_b)

----- END CRYPTOGRAPHY -----

Circuit – Henne-Ei-Problem

- Bisher nur ein Hop im Circuit
- Rein logisch müssten wir jetzt über die *Circuit-Extensions* reden
- Dafür müssen wir allerdings die `RELAY` Cell begriffen haben
- ... und diese sind in Circuits mit > 1 Hop leichter zu verstehen

- 🤷

Circuit – Verschlüsseltes RELAY

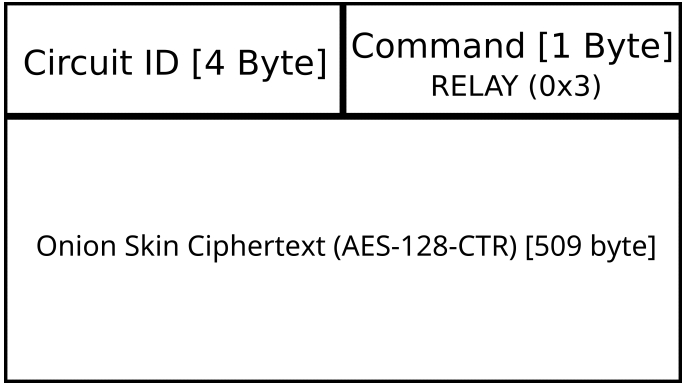

Abbildung: Eine verschlüsselte `RELAY` Cell

Circuit – Unverschlüsseltes RELAY

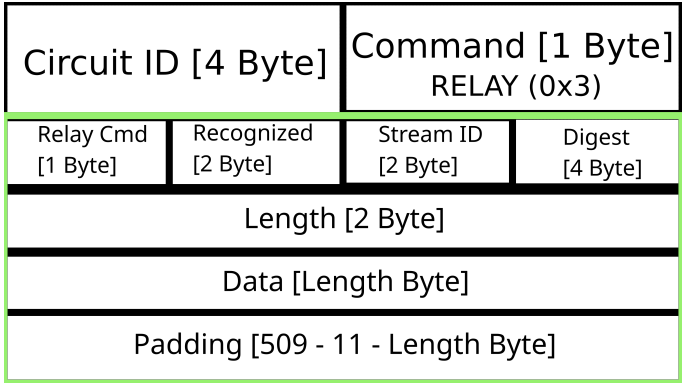

Abbildung: Eine unverschlüsselte `RELAY` Cell

Circuit – RELAY

- Enthält einen *Onion Skin* verschlüsselt mit AES-128-CTR
- Vorwärts:
  - Client verschlüsselt mit den K_f's
  - Vom Letzten bis hin zum Ersten
    - `For I=N...1: encrypt(`$K_f[i]$`)`
  - Jedes Node im Circuit entschlüsselt den Ciphertext:
    - Noch immer verschlüsselt? Weiter an's nächste Node!
    - Kann ich es lesen? Gut, lass es auswerten!
    - `encrypt(`K_f`)`

Circuit – RELAY

- Rückwärts:
  - Jedes Node verschlüsselt mit seinem K_b und schickt es zurück
    - `encrypt(`K_b`); send;`
  - Nur der Client hat alle drei K_b's, um es vollständig zu entschlüsseln
  - Der Client entschlüsselt vom ersten K_b bis hin zum Letzten
    - `For I=1...N: decrypt(`$K_b[i]$`)`

Circuit – To be plain or to be cipher: That's the question

- Problem: Wann weiß ein Node, ob der Payload schon Plain- oder noch Ciphertext ist?
- `Recognized`
  - Muss auf 0 gesetzt sein, wenn es sich um PT handelt
  - Ist mit $P(X) = \frac{1}{2^{16}}$ auch im Ciphertext 0
- `Digest`
  - Der SHA256-Hash aus allen Daten zwischen Client und DIESEM Node
  - Nur die unverschlüsselten Daten fließen in die Berechnung mit ein
  - Der Hash wird mit D_f bzw. D_b initialisiert
  - Keine schöne Lösung, soll bald(TM) ersetzt werden
- Mit $P(X) = \frac{1}{2^{16}} \cdot \frac{1}{2^{32}} = \frac{1}{2^{48}}$ geht diese Methode schief

Circuit – Extending

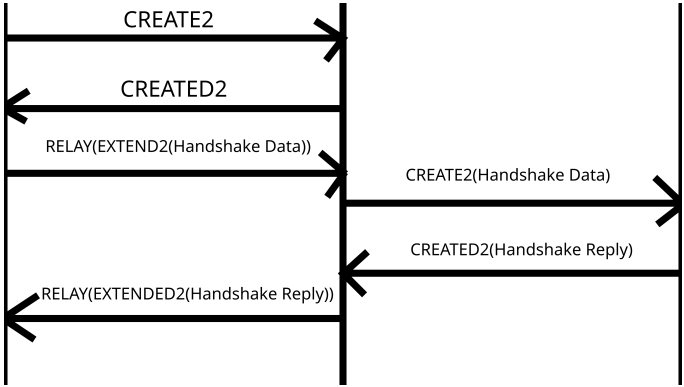

Abbildung: Die Circuit-Extension zum zweiten Node/Hop

Circuit – Extending

- Der Handshake zum n. Hop ist identisch mit dem zum 1. Hop
- Übertragung findet in `RELAY_EXTEND2` / `RELAY_EXTENDED2` Cell statt
  - Vom Aufbau identisch mit `CREATE2`/`CREATED2`
- MITM sind zwecklos und leicht erkennbar, da der Client ja alle Keys kennt

Circuit – Streams

- Ein *Circuit* kann mehrere *Streams* haben
- Ein *Stream* ist eine reale TCP-Verbindung zwischen dem letzten Hop und einem TCP-Server
- *Streams* werden über das *Stream ID* Feld in `RELAY` Cells identifiziert
- Dreischrittig:
  1. Verbindungsaufbau: `RELAY_BEGIN` / `RELAY_CONNECTED`
  2. Datentransfer: `RELAY_DATA`
  3. Verbindungsabbau: `RELAY_END`

Circuit – RELAY_BEGIN

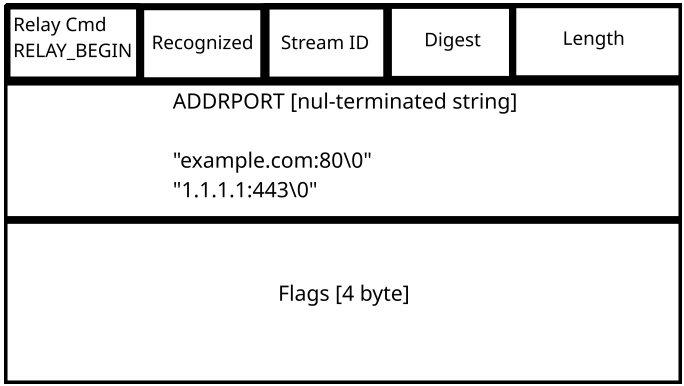

Abbildung: Eine `RELAY_BEGIN` Cell

Circuit – RELAY_CONNECTED

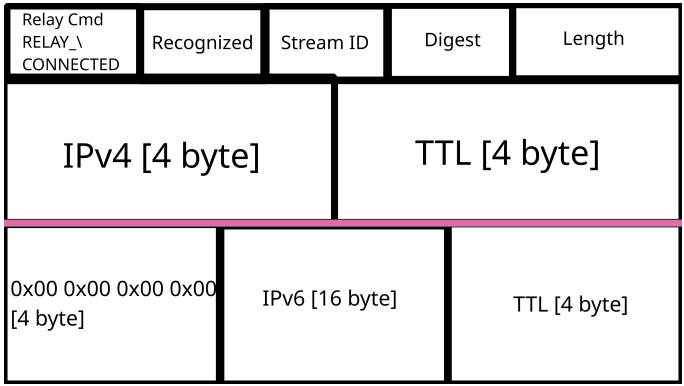

Abbildung: Eine `RELAY_CONNECTED` Cell

Circuit – RELAY_DATA

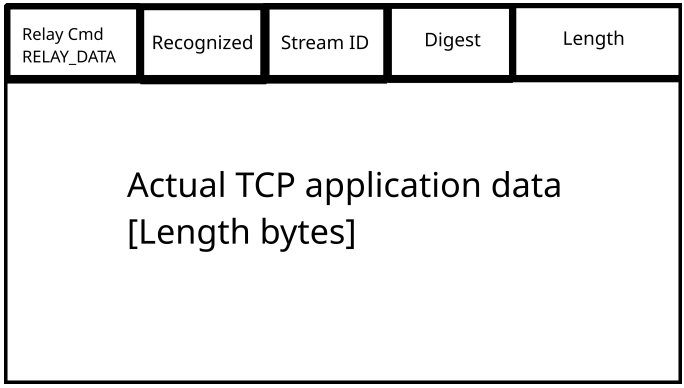

Abbildung: Eine `RELAY_CONNECTED` Cell

Summa summarum

- Tor ist ein sehr umfangreiches und verwirrendes Netzwerkprotokoll
- Es war nicht genug Platz, alles zu behandeln
  - Circuit teardowns, Connection destroys, Legacy, Directory Servers, Hidden Services, ...
- Vereinfachungen an manchen Stellen, da ich nicht den Spec ersetzen möchte

Danke für eure Aufmerksamkeit!