Technische Hochschule Rosenheim

Übersicht BITE: Bitcoin Lightweight Client Privacy using Trusted Execution

Maximilian Bundscherer
Fakultät Informatik

SoSe 2020

Bitcoin ist eine Kryptowährung, basierend auf der Blockchain-Technologie. Um am Bitcoin-Netzwerk partizipieren zu können, benötigt man Zugriff auf eine Bitcoin-Node. Die Nodes lassen sich grob in zwei Arten unterteilen: Die Full-Node speichert die Blockchain vollständig, im Gegensatz zur Leightweight-Node, die die Blockchain nur partiell speichert. Da die vollständige Bitcoin-Blockchain aktuell ungefähr 269,82 GB (Stand März 2020) umfasst, wird auf mobilen Endgeräten, zum Beispiel Smartphones, häufig auf ein Wallet mit integrierter Lightweight-Node gesetzt, die mit einer Full-Node über die Simplified Payment Verification Methode synchronisiert.

Auf der USENIX Security Konferenz im Jahr 2019 wurde aufgezeigt, dass Lightweight-SPV-Clients mit Bloomfilter die Identität des Nutzers nicht ausreichend verschleiern. Daher wurde ein System namens BITE vorgestellt, das die Privatsphäre der Nutzer besser berücksichtigen könnte. Die Idee ist grundsätzlich die Verwendung von Trusted Execution Environments wie Intel SGX. Diese sollen auf einer Full-Node betrieben werden, um dort einen Dienst für Lightweight-Nodes anzubieten.

Diese Arbeit gibt eine einführende Übersicht über das Thema Bitcoin und die Bitcoin-Blockchain, um anschließend das System BITE und die zugrundeliegenden Techniken zu erläutern und evaluieren und Full-Node-Betreiber dazu zu motivieren, Techniken aus BITE in ihre Full-Node zu integrieren.

Inhaltsverzeichnis

Abkürzungsverzeichnis

P2P Peer-to-Peer

SPV Simplified Payment Verification

ORAM Oblivious RAM

UTXO Unspent Transaction Output

FPR False Positive Rate

TEE Trusted Execution Environment

CMOV Conditional Move

BITE Bitcoin Lightweight Client Privacy using Trusted Execution

1 Einleitung und Motivation

Bitcoin ist eine Kryptowährung, basierend auf der Blockchain-Technologie. Die ursprüngliche Idee von Bitcoin war es, ein Bezahlsystem zu schaffen, das ohne die Kontrolle von Banken auskommt [Nak08]. Im Gegensatz zu Fiatgeld, zum Beispiel Euro oder Dollar, setzt Bitcoin auf ein dezentralisiertes System [Ant15, C.1][Fil20, S.33].

Um die Daten dezentral verwalten zu können, wird auf die sogenannte Blockchain-Technologie gesetzt. Eine Blockchain ist eine kontinuierlich fortlaufende Liste mit Datensätzen, die mittels kryptographischer Verfahren miteinander verbunden sind [Ant15, C7]. Die Blöcke hängen also von deren jeweiligen Vorgängern ab. Eine Blockchain kann über mehrere Knoten, zum Beispiel Server, verteilt werden und kann somit als verteiltes Datenbanksystem interpretiert werden.

Um am Bitcoin-Netzwerk partizipieren zu können, benötigt man Zugriff auf einen Bitcoin-Knoten, bezeichnet als Node. Dieser Zugriff erfolgt häufig über ein sogenanntes Bitcoin-Wallet. Bitcoin-Wallets verwalten die öffentlichen und privaten Schlüssel des Nutzers und haben ebenfalls häufig eine eigene Bitcoin-Node für Transaktionen oder Abfragen integriert.

Eine Node kann dabei verschiedene Aufgaben übernehmen, zum Beispiel die Validierung einer Transaktion. Um diese Aufgaben übernehmen zu können, ist es notwendig, die Blockchain entweder vollständig oder partiell zu speichern. Die Bitcoin-Nodes lassen sich daher grob in zwei Arten unterteilen:

- **Full-Nodes**: Diese Art von Node speichert vollständig die Bitcoin-Blockchain. Das benötigt aber sehr viel Speicher (ca. 270 GB), mehr dazu im Abschnitt 2.
- **Lightweight-Nodes**: Diese Art von Node speichert die Bitcoin-Blockchain nur partiell. Genauer gesagt werden oft nur die Block Header (mehr dazu im Abschnitt 2.1) und die dazugehörigen Hash-Baum-Pfade (mehr dazu im Abschnitt 2.3) gespeichert. Das benötigt bedeutend weniger Speicher.

Da die Bitcoin-Blockchain für jeden öffentlich einsehbar ist, stellt die Privatsphäre im Hinblick auf die Bekanntgabe der Identität des Nutzers einen wichtigen Aspekt dar: Mit der Identität ist es möglich, den Transaktionsverlauf und das Guthaben eines Nutzers zur errechnen bzw. einzusehen.

Auf mobilen Endgeräten, zum Beispiel Smartphones, wird aufgrund des hohen Speicherverbrauchs häufig auf ein Wallet mit integrierter Lightweight-Node gesetzt[1], die mit einer Full-Node über die Simplified Payment Verification (SPV) Methode synchronisiert. Dies hat aber den Effekt, dass der Nutzer einer Full-Node vertrauen muss und einen Teil seiner Privatsphäre aufgibt, mehr dazu im Abschnitt 2.4. Die SPV Methode wurde daher um die sogenannten Bloomfilter erweitert, um die Identität des Nutzers besser zu verschleiern, mehr dazu im Abschnitt 2.4.

Auf der USENIX Security Konferenz im Jahr 2019 wurde von Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame und Srdjan Capkun aufgezeigt, dass Lightweight-SPV-Clients mit Bloomfilter die Identität des Nutzers nicht ausreichend

1 Auch auf Desktop-Computern wird häufig aufgrund von Bequemlichkeit oder fehlendem technischen Verständnis auf ein Wallet mit integrierter Lightweight-Node gesetzt.

verschleiern. Daher stellten diese ein System namens Bitcoin Lightweight Client Privacy using Trusted Execution (BITE) vor, das die Privatsphäre der Nutzer besser berücksichtigen könnte [Mat19].

Diese Arbeit gibt eine einführende Übersicht über das Thema Bitcoin und die Bitcoin-Blockchain, um anschließend das System BITE und die zugrundeliegenden Techniken zu erläutern und evaluieren und Full-Node-Betreiber dazu zu motivieren, Techniken aus BITE in ihre Full-Node zu integrieren.

2 Bitcoin - Eine Übersicht

Bitcoin ist eine dezentrale Kryptowährung, basierend auf der Blockchain-Technologie. Um am Bitcoin-Netzwerk partizipieren zu können, benötigt man Zugriff auf einen Bitcoin-Knoten, bezeichnet als Bitcoin-Node. Die Bitcoin-Nodes kommunizieren untereinander und es gibt keine zentrale Steuerung, deshalb wird hier von einem Peer-to-Peer (P2P)-Netzwerk gesprochen.

Da bei einem solchem dezentralem Netzwerk die Transaktionen unter den Teilnehmern selbst durchgeführt werden können, ist es möglich, das Geld, zum Beispiel aufgrund von Synchronisierungslatenzen, mehrfach auszugeben. Man spricht in diesem Fall vom sogenannten Double-Spending-Problem. Um das zu verhindern, wurde ein Konsens-Algorithmus wie Proof-of-Work implementiert, der konfliktbehaftete Blöcke mit ihren Transaktionen im Zeitverlauf nach definierten Regeln auflöst [Fil20, S.32].

Ein Proof-of-Work-Algorithmus zeichnet sich dadurch aus, dass ein Ergebnis sehr schwer bzw. rechenintensiv zu berechnen ist, aber das Ergebnis ohne großen Aufwand nachgeprüft werden kann [Yan20, S.131]. Dieses Verfahren wird von sogenannten Minern durchgeführt und wird nicht in dieser Arbeit behandelt, da das nicht zum Verständnis von BITE beitragen würde.

Eine Transaktion im Bitcoin-Netzwerk besteht immer aus Inputs und Outputs, die sich auf eine Adresse und einen Betrag beziehen. Ein Input referenziert stets einen Output einer vorhergehenden Transaktion, der als Unspent Transaction Output (UTXO) zuvor noch nicht als Input referenziert worden ist [Fil20, S.42]. Vereinfacht lässt sich also sagen, eine UTXO repräsentiert im Bitcoin Netzwerk eine Geldmünze, die noch ausgegeben werden kann. Das Guthaben einer Bitcoin-Adresse ist die Summe seiner UTXOs. Mehre UTXOs ergeben zusammen ein UTXO-Set.

Die vollständige Erläuterung von Bitcoin und der zugrundeliegenden Technologien würde den Rahmen dieser Arbeit sprengen, daher soll zumindest eine Basis für Abschnitt 3 geschaffen werden.

2.1 Die Bitcoin-Blockchain

Die Bitcoin-Blockchain ist eine kontinuierlich fortlaufende Liste mit Datensätzen, die mittels kryptographischer Verfahren miteinander verbunden sind [Ant15, C7]. Die Blöcke hängen also von deren jeweiligen Vorgängern ab. Die vollständige Bitcoin-Blockchain umfasst aktuell (Stand März 2020) ungefähr 269,82 GB [Liu].

Da die Datensätze, bezeichnet als Blöcke, mittels kryptographischer Verfahren über eine Block-Prüfsumme (Hash) miteinander verbunden sind, ist eine Manipulation innerhalb der fortlaufenden Liste fast ausgeschlossen. Die Manipulation eines Blocks würde sich auf die weiteren Blöcke auswirken und somit einer vollständigen Neuberechnung der jeweiligen Block-Prüfsummen der auf den manipulierten Block folgenden Blöcke bedürfen. Da bei einer sinnvollen Verteilung der Daten die jeweiligen Block-Prüfsummen mehreren (unabhängigen) Knoten bekannt sind, würden die nicht manipulierten Knoten den manipulierten Knoten bei einem Abgleich der Prüfsummen als nicht vertrauenswürdig einstufen. Es ist möglich, aber unwahrscheinlich, dass manipulierte Blöcke dieselbe Block-Prüfsumme aufweisen wie ein nicht manipulierter Block.

Aufbau eines Blocks in der Bitcoin-Blockchain

Ein Block aus der Bitcoin-Blockchain besteht aus folgenden Elementen [Ant15, C7]:

- **Block Size (4 bytes)**: Die Größe des Blocks in Bytes.
- **Block Header (80 bytes)**: Der Header des Blockes. Wird im folgendem Abschnitt erläutert.
- **Transaction Counter (1-9 bytes VarInt)**: Anzahl der Transaktionen, die durchgeführt werden sollen.
- **Transactions (Variable size)**: Die Transaktionen selbst.

Aufbau des Block Headers eines Blockes

Der Block Header besteht aus den folgenden Elementen [Ant15, C7]:

- **Version (4 bytes)**: Versionsnummer, um auf Software und Protokoll-Änderungen reagieren zu können.
- **Previous Block Hash (32 bytes)**: Eine Referenz zur Prüfsumme vom Vorgänger-Block (Parent Block).
- **Merkle Root (32 bytes)**: Eine Prüfsumme über alle Transaktionen innerhalb des Blockes. Die Berechnung wird im Abschnitt 2.3 genauer erläutert.
- **Timestamp (4 bytes)**: Die geschätzte Zeit der Erstellung des Blocks im UNIX-Timestamp Format.
- **Difficulty Target (4 bytes)**: Beschreibt die Komplexität der Berechnung des Proof-of-Work-Algorithmus für diesen Block. Dieses Feld ist für das Mining von Relevanz, daher wird in dieser Arbeit nicht weiter darauf eingegangen.
- **Nonce (4 bytes)**: Ein Zähler für den Proof-Of-Work-Algorithmus. Dieses Feld ist ebenfalls für das Mining von Relevanz, daher wird in dieser Arbeit nicht weiter darauf eingegangen.

Der erste Block und das Einfügen eines Blocks

Da bei der Bitcoin-Blockchain die Blöcke kryptographisch von ihren Vorgängern abhängen, muss mindestens ein Block existieren, um weitere Blöcke einfügen zu können. Der erste Block der Bitcoin-Blockchain wird als Genesis Block bezeichnet und wurde im Jahr 2009 erstellt[2] [Ant15, C7].

Eine Bitcoin-Node parst beim Einfügen eines Blocks zum Beispiel folgenden Inhalt [Ant15, C7]:

```
{
"size" : 43560,
"version" : 2,
"previousblockhash" :
"0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
"merkleroot" :
"5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
"time" : 1388185038,
"difficulty" : 1180923195.25802612,
"nonce" : 4215469401,
"tx" : [
"257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",

#[... many more transactions omitted ...]

"05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
]
}
```

2.2 Aufgaben und Arten von Nodes

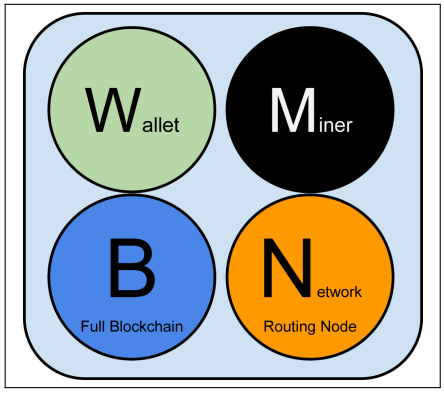

Abbildung 1: Visualisierung der vier Aufgaben von Bitcoin-Nodes im Allgemeinen [Ant15, C6]

Die Bitcoin-Nodes lassen sich grob in zwei Arten unterteilen: Die **Full-Nodes** speichern eine

2 Der erste Block lässt sich unter dem Hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f auffinden.

vollständige Kopie der Blockchain und die **Lightweight-Nodes** speichern nur einen Teil der Blockchain, nämlich nur die Block Header und die dazugehörigen Hash-Baum-Pfade.

Die Bitcoin-Nodes haben im Bitcoin-Netzwerk verschiedene Aufgaben, die sich in die vier folgenden Oberpunkte zusammenfassen lassen [Ant15, C.6], vergleiche Abbildung 1:

- **Wallet**: Speichert und verwaltet die privaten Schlüssel und Adressen des Nutzers, um Transaktionen und Abfragen auslösen zu können. In dieser Abbildung ist das Wallet ein Teil der Node.
- **Miner**: Erzeugen durch das Mining neue Blöcke (UTXO) und damit neue Bitcoins. Auf das Mining wird in dieser Arbeit nicht eingegangen.
- **(Full) Blockchain**: Speichert die Blockchain partiell oder vollständig.
- **Network**: Sorgt für die Kommunikation innerhalb des P2P-Netzwerkes (Routing).

Abbildung 2: Aufgaben der Bitcoin-Node *Reference Client (Bitcoin Core)* [Ant15, C6]

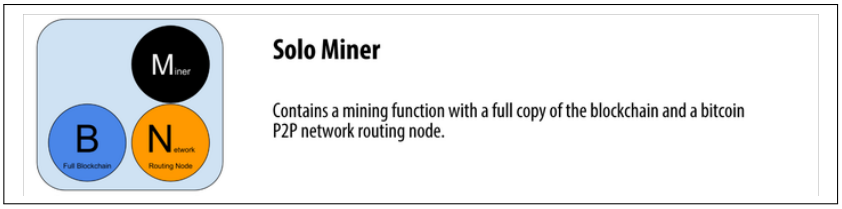

Abbildung 3: Aufgaben der Bitcoin-Node *Solo Miner* [Ant15, C6]

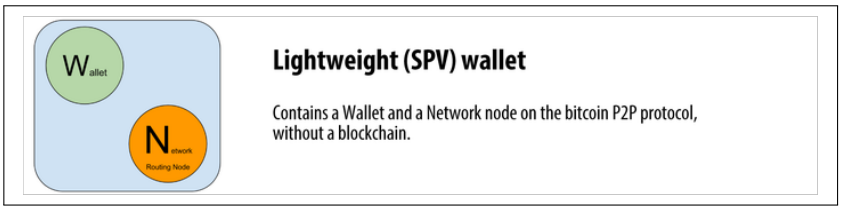

Abbildung 4: Aufgaben der Bitcoin-Node *Lightweight (SPV) Wallet* [Ant15, C6]

Mit diesen unterschiedlichen Aufgaben lassen sich verschiedene Arten von Nodes erzeugen. Einige wichtige Beispiele [Ant15, C.6]:

- **Reference Client (Bitcoin Core)**: Diese Art von Node folgt der Referenzimplementierung von Bitcoin, bezeichnet als Bitcoin Core. Bei dieser Art handelt es sich um einen komplett unabhängigen Knoten. Es handelt sich hierbei um eine Full-Node. Vergleiche Abbildung 2.

- **Solo Miner**: Diese Art ist hauptsächlich für das Minen von Bitcoins ausgelegt. Es kann sich hierbei um eine Full-Node oder eine Lightweight-Node handeln. Vergleiche Abbildung 3.
- **Lightweight (SPV) Wallet**: Diese Art wird häufig von mobilen Endgeräten verwendet. Es handelt sich hierbei um eine Lightweight-Node. Die Synchronisierung mit einer Full-Node erfolgt über das SPV-Protokoll. Vergleiche Abbildung 4.

2.3 Prüfsummen aus Hash-Bäumen (Merkle Trees)

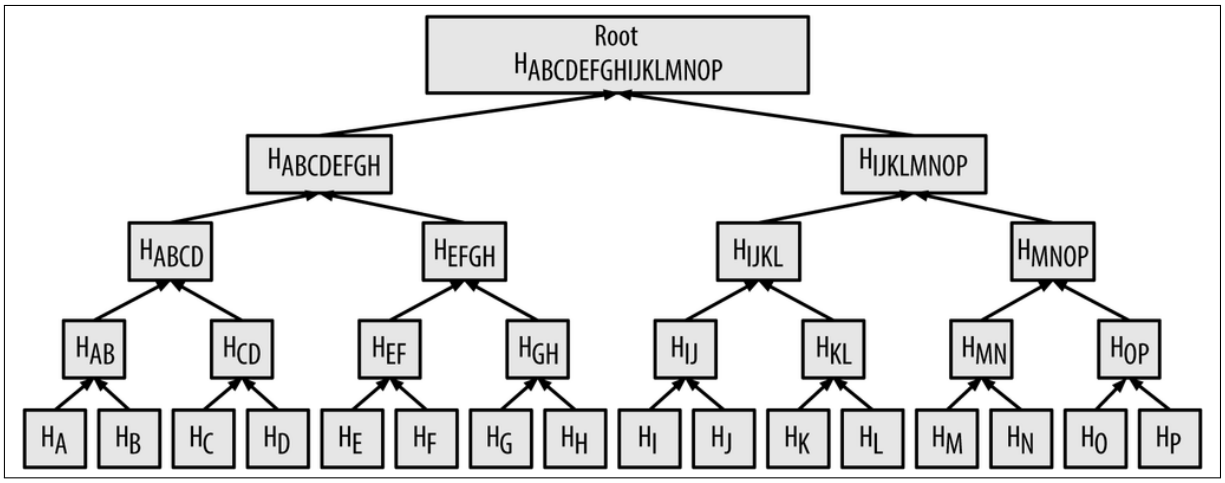

Abbildung 5: Eine abstrakte Übersicht eines Hash-Baums [Ant15, C7]

Wie bereits im Abschnitt 2.1 erläutert, enthält jeder Block Header eine Prüfsumme über alle Transaktionen innerhalb des Blocks. Diese Prüfsumme lässt sich über einen sogenannten Hash-Baum berechnen und findet sich dort in der Wurzel wieder [Ant15, C7].

Um Hash-Bäume besser verstehen zu können, wird anhand von Abbildung 5 ein abstrakter Hash-Baum und die Berechnung der Prüfsumme erläutert:

In der untersten Ebene werden die Prüfsummen der Transaktionen als Blätter dargestellt (H_A, H_B, ..., H_P). Die Transaktionen werden über den Algorithmus SHA256 gehasht. Der Hash-Algorithmus wird hier zweimal angewendet, daher wird das Hash-Verfahren hier genauer als double SHA256 bezeichnet. Ein Beispiel für H_A könnte *a3* und für H_B könnte *b5* sein. Die Hashes der Transaktionen werden in H_{AB} konkateniert und anschließend noch einmal gehasht. Aus dem obigen Beispiel müsste man erst *a3* und *b5* konkatenieren (*a3b5*) und anschließend hashen. Dieses Prinzip setzt sich bis zur Wurzel durch. Die Prüfsumme lässt sich nun an der Wurzel ablesen.

2.4 Simplified Payment Verification (SPV)

Lightweight-Nodes speichern die Blockchain nur partiell und nicht vollständig. Um trotzdem Transaktionen durchführen zu können oder das Guthaben zu einer Adresse einsehen zu können, synchronisieren sich Lightweight-Nodes mit einer oder mehreren Full-Nodes über die SPV Methode.

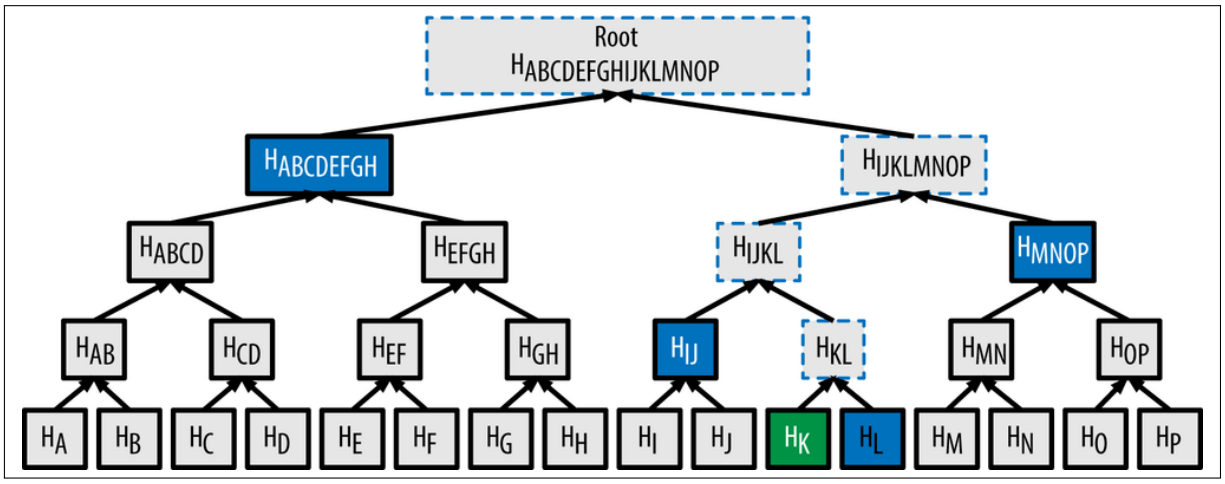

Abbildung 6: Ein abstraktes Beispiel einer Transaktion innerhalb eines Hash-Baums [Ant15, C7]

Für Lightweight-Nodes sind die Hash-Bäume von besonderer Relevanz, da diese nicht über die gesamte Blockchain verfügen [Ant15, C7]. Die Abbildung 6 zeigt abstrakt einen solchen Hash-Baum, der für SPV verwendet wird:

- Die SPV-Node kennt die gewünschte Transaktion K und kann damit die zugehörige Prüfsumme H_K berechnen. Diese wird hier **grün** dargestellt.
- Um zu überprüfen, ob die Transaktion durchgeführt worden ist, muss die SPV-Node die **blau** dargestellten Prüfsummen von einer vertrauenswürdigen Full-Node abfragen.
- Um nun zu überprüfen, ob die Transaktion erfolgreich war, berechnet die SPV-Node die Prüfsummen ausgehend von H_K bis zur Wurzel. Die berechneten Prüfsummen werden hier **blau-gestrichelt-umrandet** dargestellt. Der Wurzel-Hash wird anschließend mit einer (anderen) Full-Node verglichen. Falls die Ergebnisse übereinstimmen, ist die Transaktion erfolgreich durchgeführt worden.

Der triviale Ansatz wäre, die Full-Node mit der öffentlichen Adresse des Nutzers zu kontaktieren, woraufhin die Full-Node die Transaktionen für den Client filtert und diesem zusendet. Dies hat aber zum Nachteil, dass dann die Full-Node die Identität des Nutzers kennt (zum Beispiel über die IP-Adresse des Clients und deren Zuweisung zur öffentlichen Adresse).

Erweiterung von SPV um Bloomfilter

Um die Identität eines Nutzers durch die Nutzung einer SPV-Lightweight-Node besser verschleiern zu können, wurde das SPV-Protokoll um die Bloomfilter erweitert.

Um die Bloomfilter besser verstehen zu können, werden die Bloomfilter an einem 16-Bit-Bloomfilter-Beispiel erläutert:

- Abbildung 7 zeigt einen leeren Bloomfilter mit einer Größe von 16-Bit. Dieser wird mit *0* initialisiert. Die Hash-Funktionen K_1, K_2 und K_3 nehmen in diesem Beispiel die öffentliche Adresse des Nutzers entgegen und haben einen Ziel-Bereich von *1 - 16*.

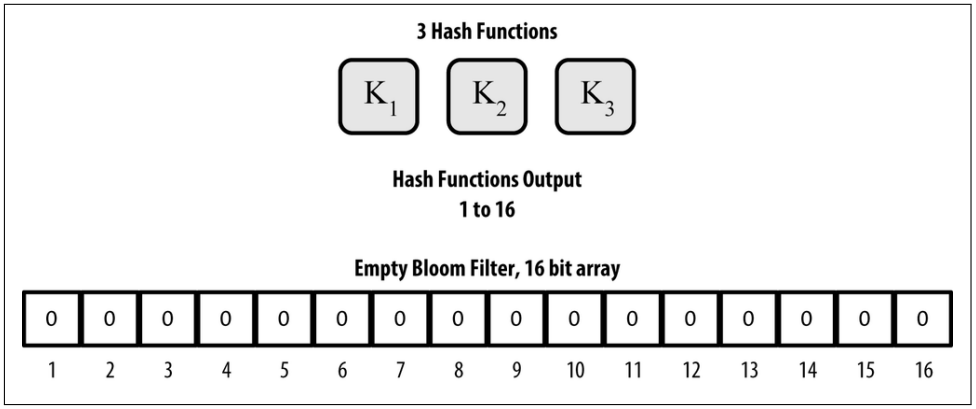

Abbildung 7: Leer initialisierter Bloomfilter [Ant15, C6]

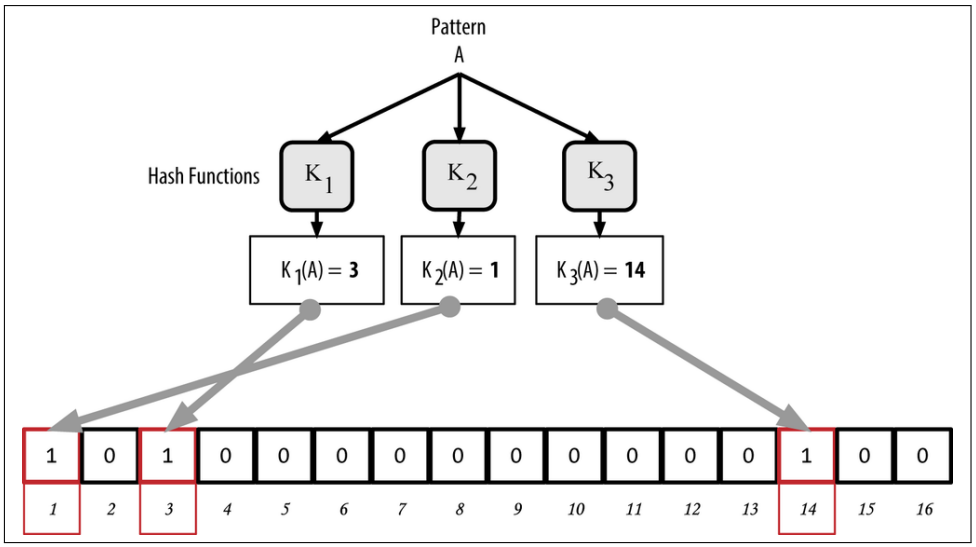

Abbildung 8: Bloomfilter nach Anwendung von Pattern auf eine Adresse [Ant15, C6]

- Abbildung 8 zeigt einen Bloomfilter nach der Anwendung eines Pattern A. Die Pattern definieren, wie die jeweiligen Hash-Funktionen definiert sind. Es können mehrere Pattern (hintereinander) angewendet werden. Die Pattern müssen auf der Lightweight-Node und auf der Full-Node übereinstimmen. Die Pattern werden in diesem Beispiel nun auf die öffentliche Adresse des Nutzer angewendet. Die Hash-Funktionen K_1, K_2 und K_3 errechnen anschließend die Stellen, an der der Bloomfilter auf *1* gesetzt wird.
- Dieser Bloomfilter wird anschließend der Full-Node übermittelt. Dass Hash-Funktionen kollidieren können, ist in diesem Fall von Nutzen, da unterschiedliche öffentliche Adressen auf den gleichen Bloomfilter kommen können:
  Die Full-Node wendet die gleichen Pattern auf alle Transaktionen an und filtert somit die angefragten Transaktionen aus (+ Transaktionen die nicht gefragt waren, das kann aber die Full-Node aufgrund der Kollision nicht unterscheiden). Diese werden nun der Lightweight-Node zugestellt. Diese kann nun ihre spezifischen Transaktionen filtern und damit das Guthaben berechnen.

Vereinfacht gesagt führt die Nutzung eines Bloomfilters zu einer höheren Privatsphäre, da die Full-Node nicht genau sagen kann, welche Adresse gemeint ist und der Lightweight-Node mehr Daten sendet, als diese bräuchte. In diesem Kontext wird häufig von einer sogenannten False Positive Rate (FPR) gesprochen. Diese beschreibt hier das Verhältnis von gefragten Adressen und ungefragten Adressen in der Antwort einer Full-Node, wobei eine FPR von 0% beschreibt, dass nur gefragte Adressen in der Antwort enthalten sind (womit aber die Identität direkt offen gelegt wird), und eine FPR von 100%, dass nur ungefragte Adressen in der Antwort enthalten sind.

Je nach eingesetztem Bloomfilter und Kombinationen variiert dabei die FPR und die Komplexität der Berechnung. Das bedeutet, dass die Auswahl eines geeigneten Bloomfilters eine Abwägungssache zwischen Geschwindigkeit und Privatsphäre darstellt. Die derzeit verwendeten Bloomfilter sind nicht ausreichend, um die Privatsphäre angemessen zu bewahren [Mat19].

3 BITE

Auf der USENIX Security Konferenz im Jahr 2019 wurde von Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame und Srdjan Capkun aufgezeigt, dass Lightweight SPV Clients mit Bloomfilter (vergleiche Abschnitt 2.4), die Identität des Nutzers nicht ausreichend verschleiern. Daher stellen diese ein System namens BITE vor, das die Privatsphäre der Nutzer besser berücksichtigen könnte [Mat19].

BITE wurde in zwei Varianten vorgestellt [Mat19]:

- V1 namens **Scanning Window**
- V2 namens **Oblivious Database**

3.1 Einführende Begriffe

Um im Rahmen dieser Arbeit auf BITE eingehen zu können, ist es zuerst notwendig, die verwendeten Begriffe zu erläutern.

3.1.1 Oblivious RAM (ORAM)

Die Idee hinter Oblivious RAM (ORAM) ist es, die Privatsphäre der Clients bei der Kommunikation zu ausgelagerten Daten selbst bei einem verschlüsselten Protokoll zu steigern. Hierfür kann nicht nur die Netzwerk-Kommunikation betrachtet werden, sondern auch die Kommunikation zwischen Prozessor und Arbeitsspeicher.

Da selbst bei einem verschlüsselten Zugriff Spuren (zum Beispiels Zugriffszeiten und Zugriffsreihenfolge) hinterlassen werden, ist es möglich, über statistische Verfahren Rückschlüsse auf die Identität des Clients und die Daten selbst zu ziehen [Ste14].

Der Begriff ORAM ist nicht eindeutig definiert und wird von verschiedenen Autoren unterschiedlich verwendet. ORAM wurde erstmals von Oded Goldreich im Jahr 1987 verwendet und bezog sich hauptsächlich auf die Frage, wie man ein Programm ausführen kann, ohne dass das Programm von außen analysiert werden kann[3]. Das wichtige hierbei: Die Eingabe und Ausgabe des Programms bleiben gleich, der Kontrollfluss verändert sich aber.

Im Rahmen der Arbeit wird der Begriff ORAM so definiert, dass der Zugriff auf Ressourcen, wie Daten, durch permanentes Mischen von Zugriffen und (erneutem) Verschlüsseln von Daten weniger Spuren hinterlässt. Diese Definition folgt der Definition von Goldreich [Ste14]. Dadurch wird zwar der Zugriff auf die Daten aufwendiger, erhöht aber die Privatsphäre des Clients.

3.1.2 Trusted Execution Environment (TEE) anhand von Intel SGX

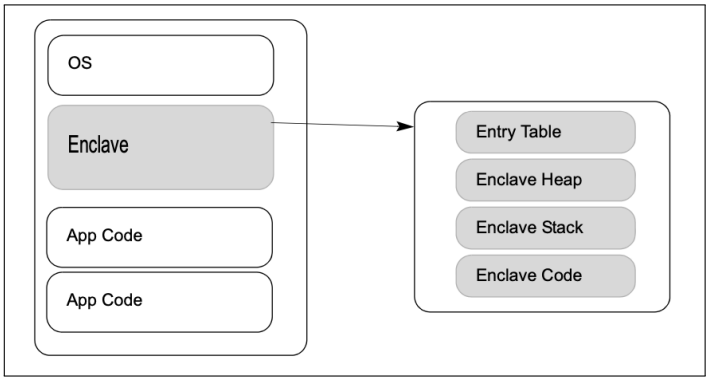

Abbildung 9: Eine Intel SGX Enklave mit eigenem (virtuellem) Adressraum [Int16]

Eine Trusted Execution Environment (TEE), zum Beispiel eine Smartcard oder die ARM Trustzone, stellt eine sichere (isolierte) Laufzeitumgebung zur Verfügung. Diese Arbeit geht auf die TEE SGX der Firma Intel ein, da diese bei BITE verwendet wird.

Die Firma Intel definiert Software Guard Extensions (Intel SGX) als Sammlung von Instruktionen und Mechanismen für den Speicherzugriff, als Erweiterung der Prozessoren, die der Intel-Architektur folgen [Int16]. Mit der Erweiterung lassen sich sogenannte Enklaven erstellen.

3 siehe `https://dl.acm.org/doi/10.1145/28395.28416`

Die Enklaven lassen sich als eigenen geschützten Bereich, mit zum Beispiel eigenem Adressraum, sehen, die vom Prozessor vor fremden Zugriffen geschützt werden (vergleiche Abbildung 9). Der Schutz umfasst sogar das Zugriffsverbot von privilegierten Prozessen. Dadurch kann nicht einmal das Betriebssystem selbst die in der Enklave laufenden Anwendungen analysieren.

Im Rahmen der Arbeit kann aufgrund der Komplexität nicht auf die Funktionsweise von Intel SGX eingegangen werden. Intel selbst hat die Idee von Intel SGX in acht Punkten zusammengefasst [Int13], wovon folgende für diese Arbeit relevant sind:

- Intel SGX erlaubt es den Entwicklern, kritische Daten vor unberechtigten Zugriffen oder Modifikationen von höher privilegierten Prozessen zu schützen.
- Intel SGX erlaubt es der Applikation, auf vertrauliche Informationen bzw. Ressourcen zuzugreifen, ohne dass das Betriebssystem über einen Scheduler mit einem Resource-Manager die Applikation steuern kann.
- Intel SGX erlaubt es Entwicklern, einen Bereich für die Applikation zu schaffen, der nicht von anderen kontrolliert werden kann. Da der Bereich unabhängig vom Betriebssystem ist, können dort auch isolierte Applikationen installiert werden.
- Durch die Enklaven ist es möglich, einen sicheren Bereich zu schaffen, der nicht von Hackern oder bereits vorhandener Schadsoftware auf dem Host-Betriebssystem kompromittiert werden kann.

Intel SGX ist nicht dazu gedacht, Schutz gegen physische Angriffe zu bieten [Int16] [Mat19] und ist auch anfällig für Angriffe auf physische Ressourcen wie Prozessor-Cache [Bra17]. Immer wieder neu aufkommende CPU-Sicherheitslücken verringern zusätzlich den Schutz von SGX Enklaven, wie zuletzt auch die Zeitung *heise* aufzeigte [Man20].

3.1.3 CMOV/Raccoon

Es ist möglich, den Kontrollfluss eines Programms zu analysieren, und damit auch, zu verstehen, was ein Programm gerade tut. Um die Analyse des Kontrollflusses zu erschweren, werden bedingte Sprünge über den Assembler-Befehl Conditional Move (CMOV) implementiert: Wenn die im Opcode angegebene Bedingung erfüllt ist, wird der Quelloperand in den Zieloperanden geschrieben. Falls es sich beim Quelloperanden um einen Speicheroperanden handelt, wird dieser unabhängig von der Bedingung gelesen [PDF].

CMOV und ORAM können miteinander kombiniert werden, um die Kontrollfluss-Analyse noch einmal deutlich zu erschweren. Raccoon stellt eine Kombination dieser Möglichkeiten dar [Ran15].

3.2 Zielsetzung und Ansätze

Das übergeordnete Ziel der Lösung BITE lässt sich in drei Punkte unterteilen [Mat19]:

1. **Privatsphäre**: Die Lightweight-Clients sollten in der Lage sein, zu überprüfen, ob deren Transaktionen auf der Blockchain durchgeführt worden sind, oder abzufragen, wie

viel Guthaben diese noch besitzen. Dazu benötigen sie eine Full-Node, vergleiche Abschnitt 2.4. Die Full-Node soll nicht in der Lage sein, Rückschlüsse auf die Identität des Nutzers zu ziehen.

2. **Vollständigkeit**: Der Verifikationsprozess sollte sicherstellen, dass keine gültigen Transaktionen fehlen oder ausgelassen werden.

3. **Performance**: Die Geschwindigkeit und der Speicherbedarf sollten besser als oder gleich der aktuellen Lightweight-Clients sein.

3.2.1 Die Kernidee und ein trivialer Ansatz

Die Idee ist grundsätzlich die Verwendung von TEEs wie Intel SGX, vergleiche Abschnitt 3.1.2. Diese sollen auf einer Full-Node betrieben werden, um dort einen Dienst für Lightweight-Nodes anzubieten[Mat19].

Eine triviale Möglichkeit TEE zu nutzen, wäre, dass die Lightweight-Node ihren privaten Schlüssel mit einer Full-Node innerhalb einer Enklave teilt. Die Enklave kann nun im Namen des Clients handeln. Diese Lösung ist aber in der Praxis nicht vertretbar, da kompromittierte Enklaven (zum Beispiel durch einen Hacker-Angriff oder durch den physischen Zugang zum Server selbst) den privaten Schlüssel des Nutzers verraten oder ungewollt mit dessen Identität handeln könnten [Mat19].

3.2.2 Grundsätzliche Vorzüge von BITE

Falls ein Client überprüfen will, ob seine Transaktion durchgeführt worden ist, oder der Client sein Guthaben abrufen möchte, verbindet sich die Client-BITE-Lightweight-Node mit einer BITE-Full-Node über eine sicheren Kanal und teilt der Enklave mit, an welchen Adressen der Client interessiert ist[Mat19].

Die Enklave stellt alle angefragten Informationen aus der lokal gespeicherten Blockchain bereit und übermittelt diese dem Client. Dabei verlässt der private Schlüssel des Clients niemals den Client selbst [Mat19].

In BITE stellt der Client eine TLS-Verbindung zu der Enklave her. Das war bei bisherigen Lightweight-Clients mit SPV-Unterstützung nicht der Fall. Das stellt bereits eine Verbesserung dar [Mat19].

Die bestehenden Aufgaben einer Full-Node, vergleiche Abschnitt 2.2 bleiben durch BITE unberührt. Daher kann BITE als Erweiterung zu einer Full-Node betrachtet werden [Mat19].

3.3 Die angenommenen Modelle hinter BITE

Um im Abschnitt 4 das System BITE evaluieren und das System besser verstehen zu können, werden zunächst die angenommen Modelle der Veröffentlichung von BITE erläutert.

3.3.1 Das Systemmodell

Die Abbildung 10 beschreibt das System-Modell von BITE: Ein BITE-Leightweight-Client LC_n kommuniziert mit einer Enklave E_j auf einer BITE-Full-Node FN_j. Die Enklave führt

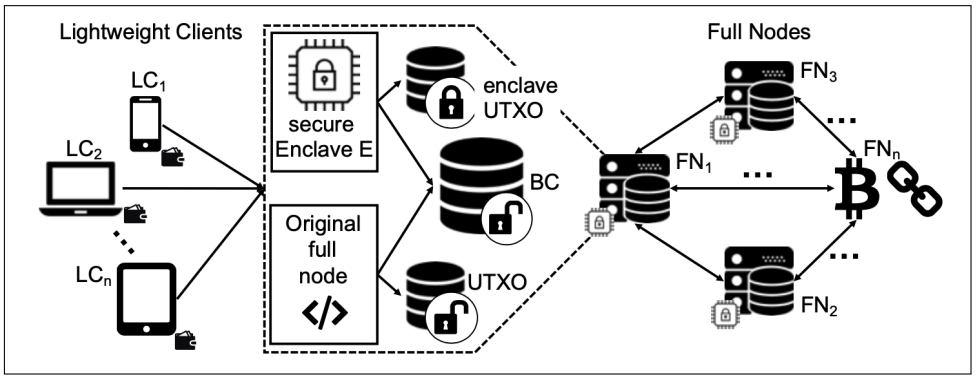

Abbildung 10: Das System-Model von BITE [Mat19]

die Client-Abfragen auf einem speziellen UTXO-Set innerhalb der Enklave durch. Dafür muss die BITE-Full-Node vorher die gesamte Blockchain *BC* aus dem P2P-Netzwerk herunterladen und lokal verwalten. BITE verwendet als TEE Intel SGX. In Intel SGX ist der Speicher einer Enklave auf 128 MB limitiert. Zwar wird das Auslagern von Daten (Swapping) unterstützt, stellt jedoch keine praktikable Lösung dar, da bereits 2009 die gesamte Bitcoin-Blockchain ungefähr 200 GB umfasste und das dazugehörige UTXO-Set 2,8 GB. Daher verwaltet die Enklave innerhalb der Enklave ein spezielles und reduziertes UTXO-Set [Mat19].

3.3.2 Annahmen über den Angreifer

Der in der BITE-Veröffentlichung beschriebene Angreifer kontrolliert die gesamte Full-Node, inklusive deren privilegierte Prozesse. Es könnte sich hierbei sowohl um einen Administrator als auch einen Hacker handeln.

Da der Angreifer das Betriebssystem kontrolliert, kann er [Mat19]:

- Enklaven (neu) einplanen oder neu starten,
- mehrere Instanzen starten,
- Nachrichten zwischen den Enklaven beeinflussen und auswerten (Lesen, Blockieren, Verzögern oder Manipulieren).

In der BITE-Veröffentlichung beschriebene Angreifer kann nicht [Mat19]:

- Hardware Restriktion von Intel SGX umgehen,
- den Kontrollfluss einer Enklave sinnvoll auswerten (zwar sind Sicherheitslücken in Intel SGX bekannt, aber durch verschiedene Ansätze wie ORAM, CMOV wird die Analyse deutlich erschwert - vergleiche Abschnitt 3.1.1 und 3.1.3),
- kryptographische Primitive wie Verschlüsselung und Signaturen umgehen.

3.3.3 Herausforderungen bei BITE

In der BITE-Veröffentlichung wird zwischen zwei verschiedenen Herausforderungen unterschieden [Mat19]:

- **Leakage through external accesses**: Da der Angreifer das Betriebssystem kontrollieren kann, kann dieser Zugriffmuster, zum Beispiel auf Dateien oder Datenbanken, feststellen. Obwohl es möglich ist, externe Daten nur für die Verwendung innerhalb von Enklaven zu verschlüsseln, ist es zum Beispiel möglich, über die Auswertung der externen Zugriffe Rückschlüsse auf die Identität des Clients zu ziehen. Auch durch die Auswertung der Antwortgrößen von einer Full-Node zu einer Lightweight-Node in Kombination mit der Anzahl der gescannten Blöcke auf der Blockchain ist es möglich, Rückschlüsse auf die Identität des Clients zu ziehen.
- **Leakage through side channels**: TEEs wie Intel SGX sind anfällig für sogenannte Seitenkanalattacken, vergleiche Abschnitt 3.1.2: Da sich die Enklaven und das Betriebssystem diverse Ressourcen wie Prozessor-Cache teilen, ist es über zum Beispiel Monitoring möglich, Rückschlüsse auf die Identität des Clients zu ziehen

3.4 BITE V1: Scanning Window

Diese erste Variante von BITE, genannt **Scanning Window**, kann als Erweiterung zu SPV ohne Bloomfilter, siehe Abschnitt 2.4, betrachtet werden.

Abbildung 11 zeigt abstrakt die **Initialisierung und die fortlaufende Funktionsweise** von BITE V1 auf [Mat19]:

- **Bereich a**: Bei der Initialisierung verbindet sich eine Full-Node FN_j mit dem Bitcoin-Netzwerk (Schritt *a-1*) und lädt die gesamte Blockchain herunter (Schritt *a-2*). In ähnlicher Weise wird auch die lokale Blockchain aktualisiert, zum Beispiel falls neue Blöcke über das P2P-Netzwerk empfangen werden.
- **Bereich b**: Ein Lightweight-Client LC_i kennt durch die Installation mit Konfiguration bereits einen gültigen (in der Vergangenheit liegenden) Block Header. Wenn der Client das erste Mal gestartet wird, lädt er aus dem P2P-Netzwerk alle neueren Block Header herunter und überprüft dabei zwei Punkte: (1) Haben alle Blöcke den Proof-of-Work-Algorithmus bestanden? und (2) Führen die neuen Block Header wieder zu dem Block Header, der aus der Konfiguration bekannt ist?

Sobald der Client nun mit dem P2P-Netzwerk synchronisiert ist, speichert dieser nun eine kleine Anzahl an neuesten Block Headern. Der Client kann sich selbst aktualisieren, wenn dieser periodisch oder vor einer Transaktion bzw. einer Abfrage die letzten Block Header herunterlädt. Der Client benötigt folglich nicht so viel Speicher, da dieser nicht die gesamte Blockchain speichern muss. [Mat19].

Aus der Abbildung 11 lässt sich auch entnehmen, **wie mit Client-Anfragen umgegangen wird** [Mat19]:

- **Schritt 1**: Ein Lightweight-Client LC_i verbindet sich mit einer Enklave E_j auf einer Full-Node FN_j und gleicht dabei seinen Stand ab und verifiziert diesen mit den Daten aus der Enklave.
- **Schritt 2**: Falls die Verifizierung erfolgreich war, verbinden sich der Client LC_i und die Enklave E_j über einen verschlüsselten Kanal, in diesem Fall TLS.

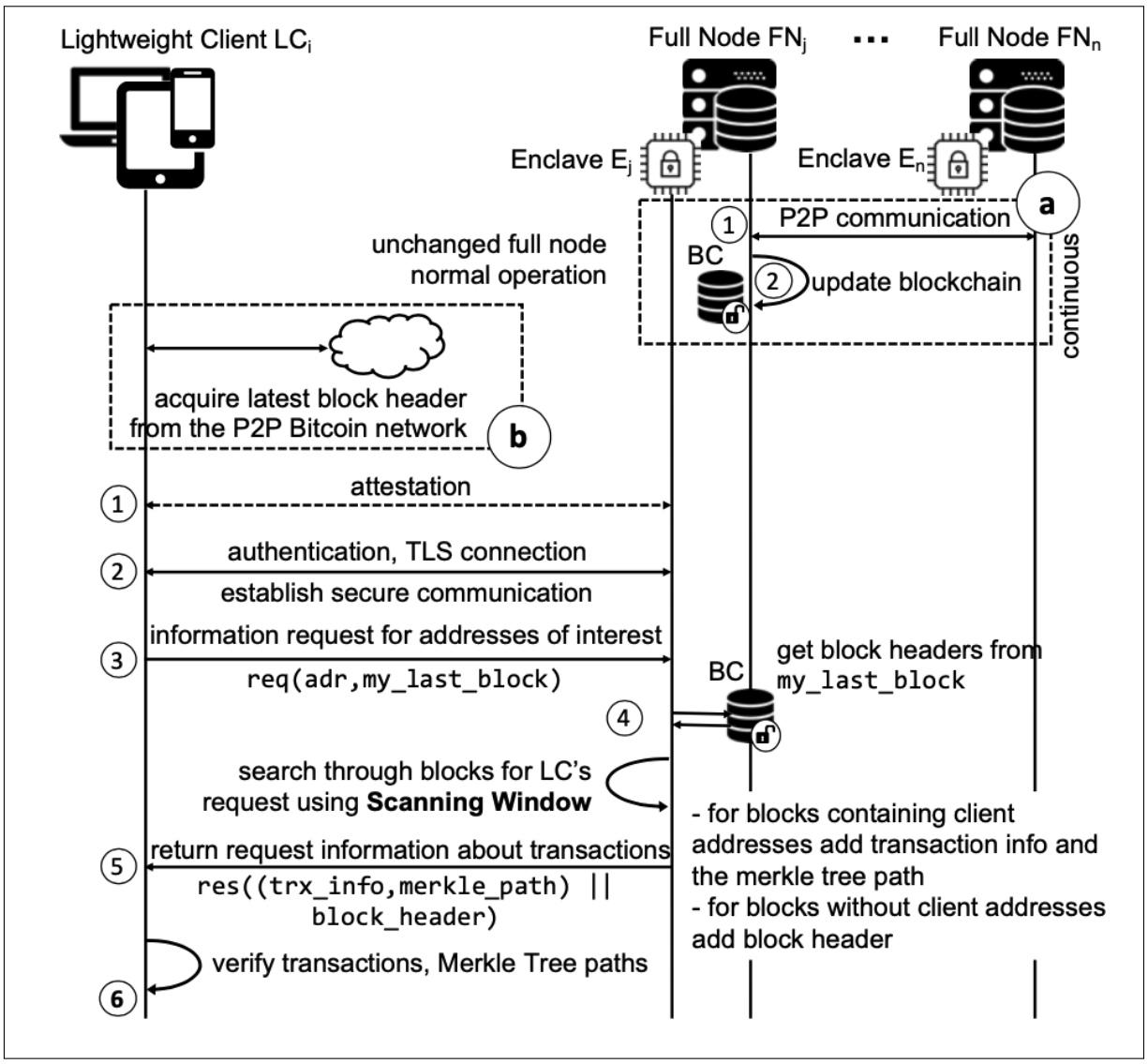

Abbildung 11: Übersicht BITE V1 [Mat19]

- **Schritt 3**: Der Client LC_i sendet nun eine Anfrage über die angefragten Adressen. Diese Anfrage beinhaltet zusätzlich eine Block-Nummer, die angibt, wie tief innerhalb der Blockchain nach passenden Transaktionen gesucht werden soll. Diese Nummer wird bei der ersten Transaktionsüberprüfung auf die Nummer des Blocks gesetzt, bei der der Nutzer ungefähr anfangen hat, Bitcoin zu verwenden. Bei den nächsten Prüfungen wird die Nummer aus der vorherigen Interaktion mit einer Full-Node verwendet.
- **Schritt 4**: Eine Enklave E_j scannt nun die lokale Kopie der Blockchain auf die angeforderte Adresse und den angegeben Bereich. Diese Scan-Technik schützt vor *Leakage through external accesses* und wird im folgendem Abschnitt 3.4.1 genauer erläutert.
- **Schritt 5**: Für die Blöcke, die Client-Adressen enthalten, werden die vollständigen Transaktionsinformationen und die entsprechenden Prüfsummen in die Antwort der Enklave E_j aufgenommen. Für Blöcke, die nicht die Client-Adresse enthalten, werden nur die Block Header hinzugefügt.
- **Schritt 6**: Nun kann der Client LC_i zwei Punkte überprüfen, ob diese zu deren Datenstand passen: (1) Stimmen die empfangenen Block Header mit meinem Stand überein? und (2) Stimmen die empfangenen Transaktionen und Hash-Baum-Pfade mit den Block Headern überein? Bei einem positiven Ergebnis setzt der Client auf die letzte verifizierte Block-Nummer und schließt die Verbindung zur Enklave.

3.4.1 Scan-Technik

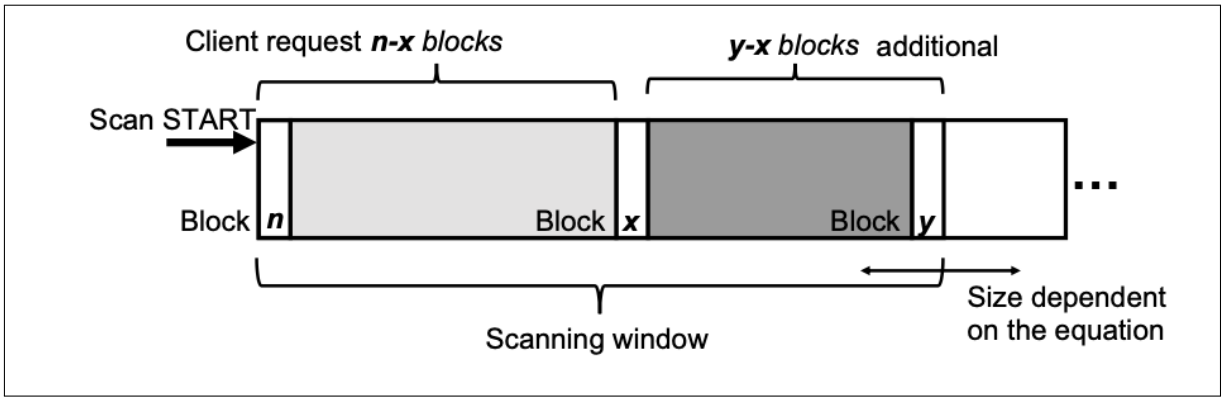

Abbildung 12: BITE V1 - Scan-Technik (Scanning-Window) [Mat19]

Um gegen *Leakage through external accesses* vorzugehen, vergleiche Abschnitt 3.3.3, setzt diese Variante auf eine besondere Scan-Technik der Blockchain, um Spuren zu reduzieren: Ohne BITE ist es möglich, über die Antwortgröße (definiert durch die Transaktionen, die den Client betreffen), und über das Wissen über der Anzahl der gescannten Blöcke Annahmen über die Identität des Clients zu treffen [Mat19].

Die Idee hinter dem Scan-Schema von BITE V1 ist grundsätzlich, das Verhältnis zwischen der Antwortgröße und die Anzahl der gescannten Blöcke zu verstecken. Abbildung 12 zeigt das Konzept dahinter [Mat19]:

Die Enklave scannt vom neuesten Block n ausgehend in Richtung x. Die Anzahl der gescannten Blöcke definiert der Client. Transaktionen, die den Client betreffen, werden in die Antwort r aufgenommen. Die Gesamtgröße der Antwort r wird durch den Schwellwert t geteilt. Der Schwellwert t gibt die maximale Antwortgröße pro Block an.

Durch die Anzahl der gescannten Blöcke ergibt sich eine maximale Antwortgröße von $r = (n\text{-}x) * t$. Falls r die maximale Antwortgröße übersteigt, scannt die Enklave weiter (in diesem Beispiel bis y), bis das Kriterium wieder erfüllt ist. Falls r kleiner ist als die maximale Antwortgröße, wird die Antwort künstlich mit unpassenden Transaktionen aufgefüllt.

Um den Rahmen dieser Arbeit nicht zu sprengen, wird auf die Berechnung von t nicht weiter eingegangen: "The exact size of the threshold is empirically determined"[Mat19].

3.4.2 Schutz gegen Leakage through side channels

Die oben beschriebenen Maßnahmen schützen vor *Leakage through external accesses* aber nicht vor *Leakage through side channels*: Es ist weiterhin möglich, über zum Beispiel Prozessor-Cache-Monitoring, Rückschlüsse auf die Identität des Client zu ziehen, vergleiche Abschnitt 3.1.2 und 3.3.3.

Daher entschied man sich bei BITE V1 auf Kosten der Leistung die Sicherheitsfunktionen einer Enklave zu verbessern: Um gegen eine Auswertung durch Zeitmessung beim Scannen von Blöcken vorzugehen, werden nicht nur die Prüfsummen zu den angefragten Transaktionen, sondern zu sämtlichen Transaktionen berechnet. Außerdem verwendet BITE CMOV, vergleiche Abschnitt 3.1.3, als Wrapper um das eigentliche Programm, um den Kontrollfluss schwerer analysierbar zu machen. Dabei sieht zum Beispiel ein Zugriff auf die Antwort r immer wie eine Kopie von Daten aus, auch wenn die Antwort nur gelesen wird.

3.5 BITE V2: Oblivious Database

Die zweite Variante von BITE, genannt **Oblivious Database**, stellt einen komplett neuen Verifizierungsprozess für Lightweight-Clients bereit.

Abbildung 13 zeigt abstrakt die **Initialisierung und die fortlaufende Funktionsweise** von BITE V2 auf [Mat19]:

- **Bereich a**: Bei der Initialisierung verbindet sich eine Full-Node FN_j mit dem Bitcoin-Netzwerk (Schritt *a-1*) und lädt die gesamte Blockchain herunter (Schritt *a-2*). In ähnlicher Weise wird auch die lokale Blockchain aktualisiert, zum Beispiel falls neue Blöcke über das P2P-Netzwerk empfangen werden.
- **Bereich b**: Bei der Initialisierung einer Enklave E_j wird innerhalb der Enklave ein spezielles UTXO-Set erstellt, ähnlich wie bei der Erstellung eines normalen UTXO-Sets. Wichtig hierbei ist, dass dieses spezielle UTXO-Set nur innerhalb der Enklave entschlüsselbar ist und auch nur über den ORAM-Algorithmus Abfragen durchführt. Das spezielle UTXO-Set ist nach Clientanfragen indexiert, um einen schnellen Zugriff zu gewährleisten und wird fortlaufend aktualisiert, zum Beispiel falls neue Blöcke über das P2P-Netzwerk empfangen werden. Das spezielle UTXO-Set wird im folgenden Abschnitt 3.5.1 genauer erläutert.

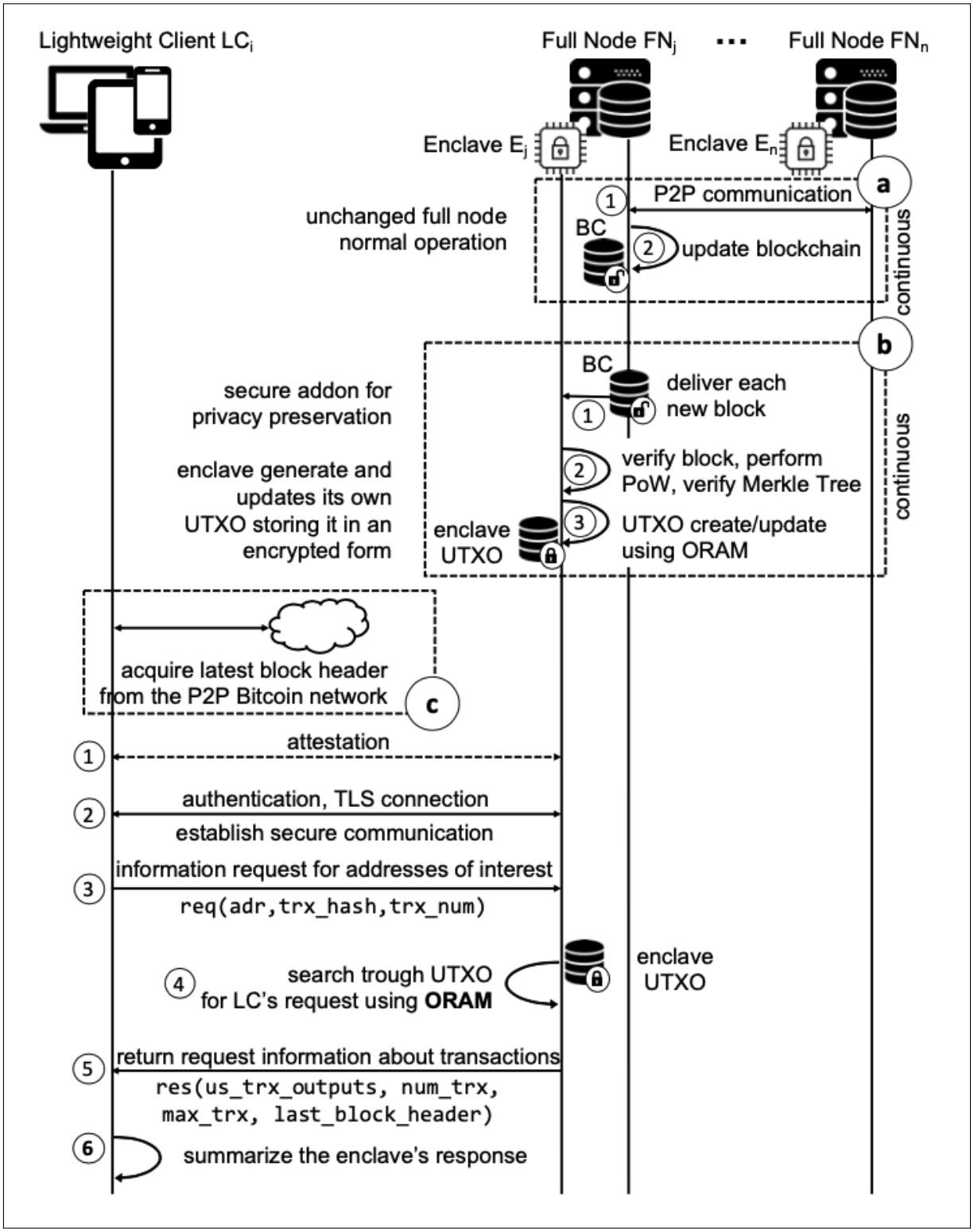

Abbildung 13: Übersicht BITE V2 [Mat19]

- **Bereich c**: Wie bei der erste Variante werden die neuesten Block Header vom Client selbst dem P2P-Netzwerk entnommen, vergleiche Abschnitt 3.4.

Sobald der Client nun mit dem P2P-Netzwerk synchronisiert ist, speichert dieser nun eine kleine Anzahl an neuesten Block Headern. Der Client kann sich selbst aktualisieren, wenn dieser periodisch oder vor einer Transaktion bzw. einer Abfrage die letzten Block Header herunterlädt. Der Client benötigt folglich nicht so viel Speicher, da dieser nicht die gesamte Blockchain speichern muss. [Mat19].

Aus der Abbildung 13 lässt sich auch entnehmen, **wie mit Client-Anfragen umgegangen wird** [Mat19]:

- **Schritt 1**: Ein Lightweight-Client LC_i verbindet sich mit einer Enklave E_j auf einer Full-Node FN_j und gleicht dabei seinen Stand ab und verifiziert diesen mit den Daten aus der Enklave.
- **Schritt 2**: Falls die Verifizierung erfolgreich war, verbinden sich der Client LC_i und die Enklave E_j über einen verschlüsselten Kanal, in diesem Fall TLS.
- **Schritt 3**: Der Client LC_i sendet nun eine Anfrage über die angefragten Adressen. Diese Anfrage beinhaltet zusätzlich eine Prüfsumme über die letzten bekannten Transaktionen und deren Anzahl.
- **Schritt 4**: Die Enklave E_j scannt nun ihr spezielles UTXO-Set über den ORAM-Algorithmus auf die angeforderte Adresse. Das spezielle UTXO-Set wird im folgenden Abschnitt 3.5.1 genauer erläutert.
- **Schritt 5**: In der Antwort von der Enklave E_j, werden die aktuelle Anzahl der übermittelten Transaktionen und die maximale Anzahl der insgesamt verfügbaren Transaktionen aufgenommen. Falls die Werte übereinstimmen, weiß der Client, dass dieser nun vollständig synchronisiert ist. Andernfalls muss der Client LC_i mit der Synchronisierung fortfahren, kann dabei auch auf andere Enklaven E_j bzw. Full-Nodes FN_j zurückgreifen. In der Antwort von der Enklave E_j, wird noch die Prüfsumme des letzten bekannten Blocks aufgenommen. Der Client LC_i kann über diese Prüfsumme und mit Prüfsummen aus dem P2P-Netzwerk überprüfen, ob die Enklave E_j über den letzten gültigen Block verfügt.
- **Schritt 6**: Nun kann der Lightweight-Client LC_i die Antworten aggregieren und selbst durch den Abgleich der Prüfsummen aus dem P2P-Netzwerk überprüfen. Die Enklave E_j garantiert die Vollständigkeit der aktuellen gescannten Blockchain. Dadurch wird der Rechenaufwand für den Client reduziert. Abschließend wird die Verbindung zur Enklave E_j getrennt.

Hinweis zum Schritt 5: Die Größe der Antwort ist im Vorhinein definiert, um die Auswertung durch die Antwortgröße zu verhindern. Falls die vorher definierte Antwortgröße unterschritten wird, wird die Antwort künstlich mit unpassenden Transaktionen aufgefüllt. Bei einer Überschreitung der Antwortgröße wird die Antwort entsprechend reduziert [Mat19].

3.5.1 Das spezielle UTXO-Set

Um gegen *Leakage through external accesses* vorzugehen, vergleiche Abschnitt 3.3.3, setzt diese Variante auf ein spezielles UTXO-Set, um Spuren zu reduzieren:

Auf dieses UTXO-Set wird ausschließlich über den ORAM-Algorithmus zugegriffen. Der eingesetzte Algorithmus ist *ORAM Path* [Mat19]. Um den Rahmen der Arbeit nicht zu sprengen, wird dieser Algorithmus nicht genauer erläutert, lässt sich aber unter [Ste14] einsehen. Die grundsätzliche Idee von ORAM wurde im Abschnitt 3.1.1 erläutert.

Die Datenbank hinter diesem UTXO-Set wird auf der Festplatte verteilt und mit zufälligen Werten initialisiert. Nicht alle Teile werden sinnvoll genutzt, was die Erstellung von Zugriffsmustern erschwert. Wenn nun neue Blöcke hinzugefügt werden sollen, verifiziert die Enklave zunächst den Proof-of-Work-Algorithmus, vergleiche Abschnitt 2. Dann fasst diese alle Transaktionen pro Adresse zusammen und indexiert diese danach. Dabei greift diese immer wieder auf das UTXO-Set zu und verbindet somit neue Transaktionen mit bestehenden [Mat19].

3.5.2 Schutz gegen Leakage through side channels

Die oben beschriebenen Maßnahmen schützen vor *Leakage through external accesses*, aber nicht vor *Leakage through side channels*: Es ist weiterhin möglich, über zum Beispiel Prozessor-Cache-Monitoring, Rückschlüsse auf die Identität des Client zu ziehen, vergleiche Abschnitt 3.1.2 und 3.3.3.

Daher entschied man sich bei BITE V2 auf Kosten der Leistung die Sicherheitsfunktionen einer Enklave zu verbessern: Auch wenn eine Adresse nicht in einem Festplatten-Bereich vorkommt, wird der vollständige Bereich in den Arbeitsspeicher kopiert. Von außen sieht also jeder Zugriff wie eine Kopie aus. Außerdem verwendet BITE CMOV, vergleiche Abschnitt 3.1.3, als Wrapper um das eigentliche Programm, um den Kontrollfluss schwerer analysierbar zu machen. Dabei sieht zum Beispiel ein Zugriff auf das UTXO-Set immer wie eine Kopie von Daten aus, auch wenn die Antwort nur gelesen wird [Mat19].

4 BITE Evaluation

Die BITE-Veröffentlichung definiert selbst drei Ziele: **Privatsphäre**, **Vollständigkeit** und **Performance**, vergleiche Abschnitt 3.2. Die hinter dem System angenommenen Modelle lassen sich im Abschnitt 3.3 einsehen.

Diese Evaluation unterteilt sich in zwei Bereiche: Die **Sicherheitsanalyse** evaluiert, ob die Ziele Privatsphäre und Vollständigkeit erreicht worden sind. Die **Performance Analyse** evaluiert, ob das Ziel Performance erreicht worden ist.

4.1 Sicherheitsanalyse

Dieser Teil der Evaluation bezieht sich auf die Ziele: **Privatsphäre** und **Vollständigkeit**, vergleiche Abschnitt 3.2. Die Herausforderungen *Leakage through external accesses* & *Leakage through side channels* lassen sich dem Abschnitt 3.3.3 entnehmen.

4.1.1 Leakage through external accesses

Bei dieser Herausforderung kann der Angreifer über Zugriffsmuster, zum Beispiel auf Datenbanken, oder zum Beispiel über die Auswertung der Antwortgröße in Kombination mit der Anzahl der gescannten Blöcke Rückschlüsse auf die Identität des Clients ziehen, vergleiche Abschnitt 3.3.3. Die zwei Versionen von BITE werden getrennt betrachtet.

V1 - Scanning Window

Bei dieser Variante werden komplette Blöcke gescannt und nicht nur auf einzelne Adressen innerhalb der Blöcke zugegriffen. Die Spuren werden dadurch reduziert. Das konstante Verhältnis zwischen der Anzahl der gescannten Blöcke und der Antwortgröße reduziert ebenfalls Spuren. Der Angreifer kann dadurch nur auf die Anzahl der Blöcke schließen, auf die zugegriffen wird, aber nicht, welche Transaktionen von Relevanz sind [Mat19].

Aktuelle Bitcoin-Ansätze implementieren keine dieser Verfahren und keine sichere Verbindung über zum Beispiel TLS, daher stellt BITE V1 bezüglich des Ziels Privatsphäre eine Verbesserung dar.

V2 - Oblivious Database

Um Spuren zu reduzieren, werden die UTXO-Zugriffe über den ORAM-Algorithmus realisiert. Der Angreifer kann die Enklave als Black-Box benutzen und die geschriebenen Daten beeinflussen, indem er zum Beispiel modifizierte Blöcke an die Enklave sendet und die von der Enklave gesendeten Werte abfängt. Das stellt jedoch kein Risiko dar, da der Angreifer nichts aus den Antworten ableiten kann, da jede Antwort eine konstante Größe aufweist und die Verbindung zu anderen Clients über TLS verschlüsselt ist [Mat19].

Aktuelle Bitcoin-Ansätze implementieren keine dieser Verfahren und keine sichere Verbindung über zum Beispiel TLS, daher stellt BITE V2 bezüglich des Ziels Privatsphäre eine Verbesserung dar.

4.1.2 Leakage through side channels

Bei dieser Herausforderung kann der Angreifer über Seitenkanalattacken, zum Beispiel über das Monitoring über gemeinsam genutzte Ressourcen, Rückschlüsse auf die Identität des Clients ziehen, vergleiche Abschnitt 3.3.3. Die zwei Versionen von BITE werden getrennt betrachtet.

Die meisten Seitenkanalattacken auf Intel SGX Umgebungen liefern Datenzugriffs-Muster und Kontrollflussspuren unvollständig und erfordern viele Wiederholungen, um das Rauschen in den Daten zu bereinigen [Bra17] [Mat19]. Da in BITE der Client mit einer Full-Node über TLS kommunziert, können die Anfragen nicht einfach wiederholt werden (TLS schützt auch vor Replay-Attacken). Außerdem sind die Enklaven selbst zustandslos und gegen Roll-Back geschützt. Der Angreifer kann zwar selbst einen eigenen Client entwickeln, das bringt diesem gegenüber dem BITE-Referenz-Client aber keinen Vorteil. Aus diesen Gründen sind Seitenkanalattacken gegen BITE im Allgemeinen schwieriger [Mat19].

V1 - Scanning Window

Die Erweiterung zu der Version V1, siehe Abschnitt 3.4.2, stellt einen zusätzlichen Schutz gegen *Leakage through side channels* dar, ohne diese Erweiterung wäre die Version nur besser gegen *Leakage through external accesses* geschützt.

Dieser zusätzliche Schutz geht auf Kosten der Leistung und schützt verstärkt gegen Seitenkanalangriffen, durch die Erschwerung der Kontrollflussanalyse durch CMOV und die Berechnung aller Prüfsummen auch zu ungefragten Blöcken [Mat19].

Aktuelle Bitcoin-Ansätze implementieren keine dieser Verfahren, daher stellt BITE V1 bezüglich dem Ziel Privatsphäre eine Verbesserung dar.

V2 - Oblivious Database

Die Erweiterung zu der Version V2 siehe Abschnitt 3.5.2 stellt einen zusätzlichen Schutz gegen *Leakage through side channels* dar, ohne diese Erweiterung wäre die Version nur besser gegen *Leakage through external accesses* geschützt.

Dieser zusätzliche Schutz geht auf Kosten der Leistung und schützt verstärkt gegen Seitenkanalangriffe durch die Erschwerung der Kontrollflussanalyse durch CMOV und zum Beispiel das Kopieren aller (auch unzutreffenden) Bereiche der Festplatte in den Arbeitsspeicher [Mat19].

Aktuelle Bitcoin-Ansätze implementieren keine dieser Verfahren, daher stellt BITE V2 bezüglich des Ziels Privatsphäre eine Verbesserung dar.

4.1.3 Vollständigkeit

In der ersten Variante namens Scanning Window führt der Client selbst die Verifizierung der Block Header, Prüfsummen und damit der Transaktionen durch. Da der Client die Block Header aus dem P2P-Netzwerk direkt übernehmen kann und die Enklave alle Transaktionen aus Sicht ihrer Blockchain zurückgibt, kann der Client selbst auf Vollständigkeit verifizieren [Mat19].

In der zweiten Variante namens Oblivious Database verifiziert die Enklave alle Blöcke und übergibt dem Client die Prüfsumme über den letzten Block. Der Client kann die Prüfsummen mit den Prüfsummen von anderen P2P-Teilnehmern vergleichen und damit auf Vollständigkeit verifizieren [Mat19].

Ein Angreifer kann also in beiden Varianten die Full-Node nicht hinreichend manipulieren, da der Client die Prüfsumme immer mit anderen P2P-Teilnehmern abgleichen kann [Mat19]. Aktuelle Bitcoin-Ansätze implementieren diese Technik bereits, daher stellt BITE V1 und V2 bezüglich des Ziels Vollständigkeit keine Verbesserung und oder Verschlechterung dar.

4.1.4 Umgehen von Intel SGX Restriktionen

Das beschriebene Angreifer-Modell geht davon aus, dass ein Seitenkanalangriff funktioniert, aber der Angreifer die Intel SGX Restriktionen nicht vollständig umgehen kann, das heißt er

kann den Kontrollfluss nicht (vollständig) mitlesen oder beeinflussen [Mat19]. Dennoch stellt sich die Frage, was bei einem erfolgreichen Angriff auf die Enklaven möglich wäre:

In der ersten Variante namens Scanning Window verliert der Client seine Privatsphäre, da die Identität zum Beispiel über die Analyse des Kontrollflusses bekanntgegeben wird. Da diese Variante als Erweiterung zu SPV gesehen werden kann, gewährt BITE in diesem Fall immer noch die gleiche Sicherheit von aktuellen Lightweight-SPV-Clients, d.h. der Angreifer kann nicht mit der Client-Identität Transaktionen auslösen oder falsche Zahlungen akzeptieren [Mat19].

In der zweiten Variante namens Oblivious Database könnte eine kompromittierte Enklave den Client dazu bringen, falsche Zahlungen zu akzeptieren, da die Enklave zum Beispiel dem Client ein durch Konsens validiertes UTXO-Set mit falschen Geldbeträgen senden könnte. Das stellt jedoch keine realistische Bedrohung dar, da der Client der Transaktion mit dem gefälschten Betrag aktiv zustimmen müsste und es komplex ist, die gleiche Prüfsumme wie mit einem nicht gefälschten Betrag über eine Transaktion zu berechnen. Die Manipulation könnte sehr leicht über die Auswertung der Blockchain nachvollzogen werden, was den Full-Node-Betreiber als nicht vertrauenswürdig gelten ließe [Mat19].

Zusammenfassend kann gesagt werden, dass BITE gleichwertigen Schutz bezüglich des Ziels Privatsphäre gegenüber einer Lightweight-Node mit SPV bietet, falls die Restriktionen von Intel SGX vollständig umgangen werden.

4.2 Performance Analyse

Dieser Teil der Evaluation bezieht sich auf das Ziel: **Performance**, vergleiche Abschnitt 3.2, und wird auf drei Oberpunkte aufgeteilt: Verarbeitungszeit, Kommunikations-Overhead und Speicherbedarf.

Hinweis: TLS-Handshake-Zeiten werden in der Evaluation vernachlässigt, da diese nicht ins Gewicht fallen (ungefähr 100 ms für einen neuen TLS-Handshake und <10 ms für TLS Session Resumption). Die Implementierung stützt sich auf die Python-Bibliothek *bitcoinlib*[4]. Der Referenzcomputer rechnet mit einem Intel i7-8700k Prozessor und speichert auf einer Samsung 960 SSD [Mat19].

4.2.1 Verarbeitungszeit

Die Verarbeitungszeit umfasst sowohl die Anfragebearbeitung von Client zur Enklave, als auch die Zeit, die die Enklave benötigt, um das UTXO-Set für neue Blöcke zu aktualisieren [Mat19].

Abbildung 14 zeigt das Verhältnis zwischen der Anzahl der gescannten Blöcke und der Zeit [Mat19]:

- Für die erste Variante (Scanning Window) **ohne zusätzlichen Schutz** gegen *Leakage through side channels* entsteht ein Overhead von 81% gegenüber SPV mit Bloomfiltern mit einer FPR von 0,5 bzw. 0,1 Prozent, vergleiche Abschnitt 2.4.

4 `https://github.com/1200wd/bitcoinlib`

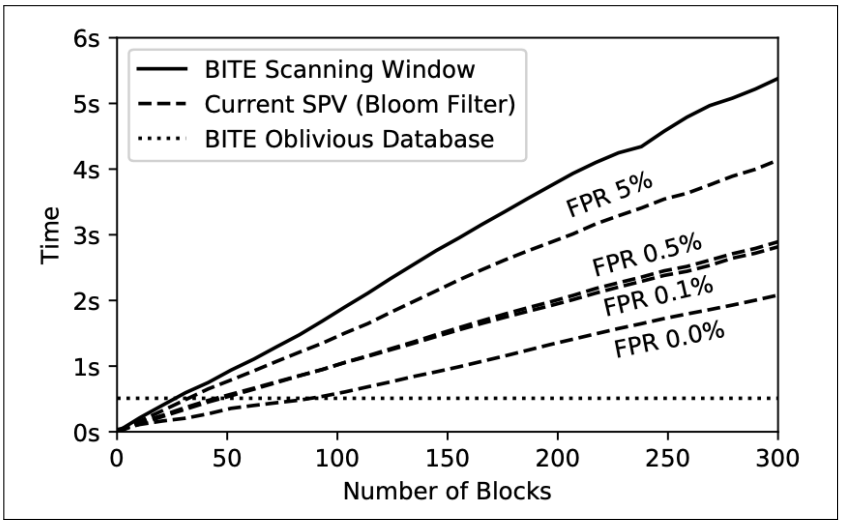

Abbildung 14: Verarbeitungszeit (bzgl. Client-Anfragen) [Mat19]

- Für die erste Variante (Scanning Window) **mit zusätzlichem Schutz** gegen *Leakage through side channels* entsteht ein deutlicher Overhead von einem Faktor 40x. So dauert die Abarbeitung von 100 Blöcken etwa 73 Sekunden. Das ist in der Grafik zu Gunsten der Übersichtlichkeit nicht enthalten.
- Für die zweite Variante (Oblivious-Database) wird auf das UTXO-Set innerhalb der Enklave zugegriffen. Die einzelnen Blöcke in der eigentlichen Blockchain werden nicht gescannt. Die Geschwindigkeit hängt also nur von den ORAM-Zugriffszeiten ab, nicht von der Anzahl der gescannten Blöcke.

Zusammenfassend lässt sich also sagen, dass die erste Variante (Scanning Window) mit und ohne Schutz gegen *Leakage through side channels* immer von Overhead geprägt ist. Die zweite Variante (Oblivious-Database) stellt hier eine Verbesserung gegenüber aktuellen Ansätzen dar.

4.2.2 Kommunikations-Overhead

Der Kommunikations-Overhead wird durch die Antwortgrößen der Enklaven bewertet und wirkt sich auf die erforderliche Bandbreite des Clients aus [Mat19].

Abbildung 15 zeigt das Verhältnis zwischen der Anzahl der Blöcke und der Antwortgröße auf: Beide Varianten zeichnen sich durch kleine Antwortgrößen gegenüber SPV mit Bloomfilter aus, da die Antworten aus der Enklave mit weniger unpassenden Transaktionen aufgefüllt werden. Die zweite Variante (Oblivious Database) reduziert die Antwortgröße noch einmal deutlich, da nur das Ergebnis der Verifikation zurückgegeben wird [Mat19].

4.2.3 Speicherplatzbedarf

Dieser Abschnitt bezieht sich auf den erforderlichen Speicher auf einer Full-Node mit BITE. Bereits 2009 umfasste die gesamte Bitcoin-Blockchain ungefähr 200 GB und das dazugehörige UTXO-Set 2,8 GB [Mat19]. Eine gewöhnliche Full-Node muss die gesamte Blockchain speichern, auch den dazugehörigen UTXO-Satz.

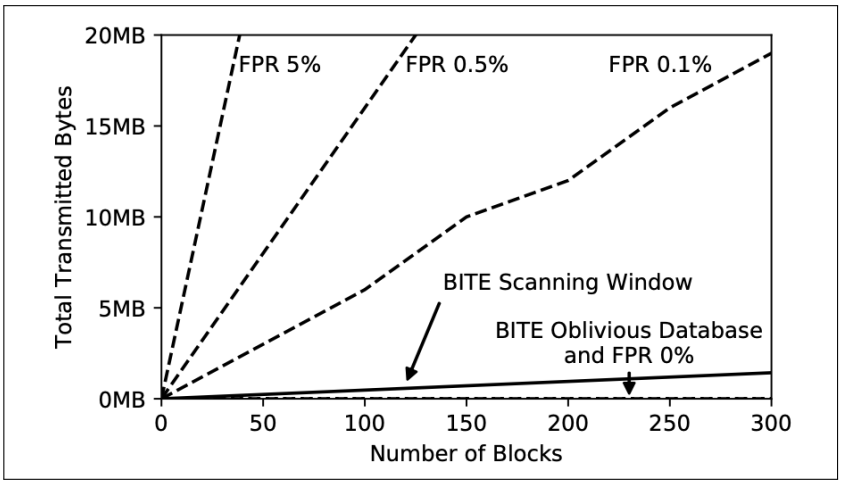

Abbildung 15: Kommunikations-Overhead [Mat19]

Bei der ersten Variante (Scanning Window) speichert die Full-Node die gesamte Blockchain und nicht den dazugehörigen UTXO-Satz. Dadurch entsteht eine geringe Verbesserung [Mat19].

Die zweite Variante (Oblivious Database) benötigt die gesamte Blockchain, außer bei der Initialisierung ihres UTXO-Sets, nicht. Das es sich um ein spezielles UTXO-Set handelt, benötigt dieses mehr Speicher, da nicht alle Festplatten-Teile sinnvoll benutzt werden (vergleiche 3.5.1). Trotz des größeren Speicherplatzbedarfs des UTXO-Sets benötigt diese Variante bedeutend weniger Speicher, da die Blockchain nicht mit gespeichert wird [Mat19].

4.3 Fazit des Autors

Es bieten sich drei sinnvolle Möglichkeiten an, BITE als Erweiterung zu einer bestehenden Full-Node laufen zu lassen:

- Erste Version (Scanning Window) ohne zusätzlichen Schutz gegen *Leakage through side channels*. Im folgenden als **Var1a** bezeichnet.
- Erste Version (Scanning Window) mit zusätzlichem Schutz gegen *Leakage through side channels*. Im folgenden als **Var1b** bezeichnet.
- Zweite Version (Oblivious Database) mit zusätzlichem Schutz gegen *Leakage through side channels*. Im folgenden als **Var2** bezeichnet.

Die genannten Anforderungen und Folgerungen an den Full-Node-Betreiber werden in der Abbildung 16 noch einmal zusammengefasst.

Die Sicherheit von BITE hängt vor allem von der verwendeten TEE, in diesem Fall Intel SGX und dem eingesetzten Verfahren ab. Da selbst bei einer kompromittierten TEE das System den gleichen Schutz bietet wie bisherige SPV-Lightweight-Lösungen (selbst mit Bloomfilter), stellt BITE hier immer eine Verbesserung dar. Da Intel SGX eine Intel-Architektur

	Processing		Communication	Storage		
	Request	UTXO Update	Response	Blockchain	UTXO	Leakage Protection
Scanning Window[1]	1.9s	-	500kB	200GB	0	✓/✗[4]
Oblivious SW[1]	73s	-	500kB	200GB	0	✓
Oblivious Database[3]	0.5s	78.5s	12kB	50MB[5]	6GB	✓
Stan. SPV FPR 0.5%[1]	1.1s	≈2s	17MB	200GB	2.8GB	✗
Stan. SPV FPR 0.0%[1,2]	0.6s	≈2s	14kB	200GB	2.8GB	✗

[1] For 100 blocks. [2] SPV with no privacy protection. [3] For 10 addresses.
[4] Protects against external leakage but not side-channels.
[5] Only the block headers need to be stored.

Abbildung 16: Vergleich der Varianten [Mat19]

voraussetzt, könnte darüber nachgedacht werden, ob auch andere (Open-Source) TEEs wie Open-Source Keystone[5] verwendet werden könnten.

Zusammenfassend würde der Autor der Arbeit sich wünschen, dass Full-Node-Betreiber diese Techniken in ihre Full-Nodes integrieren, da diese eine Verbesserung der Privatsphäre darstellen. Zwar steigert das System die Verarbeitungszeit und damit auch die Anforderungen an die Hardware, der Full-Node-Betreiber könnte aber über gesteigerte Transaktionsgebühren entlohnt werden. Dabei könnte **Var1a** von einem vertrauenswürdigen Unternehmen betrieben werden, da *Leakage through side channels* immer noch möglich wären. **Var1b** und **Var2** könnten von jedem Full-Node-Betreiber betrieben werden, da diese einen integrierten Schutz gegen *Leakage through side channels* und *Leakage through external accesses* besitzen.

5 `https://opensource.google/projects/keystone`

Literatur

[Ant15] A. Antonopoulos. In *Mastering Bitcoin (ISBN: 978-1-4493-7404-4)*. O'Reilly Media, 2015.

[Bra17] F. Brasser. Software Grand Exposure: SGX Cache Attacks Are Practical. https://www.usenix.org/system/files/conference/woot17/woot17-paper-brasser.pdf, 2017. Abgerufen am 12.06.2020.

[Fil20] H.-G. Fill. In *Blockchain kompakt (978-3-658-27461-0)*. Springer, 2020.

[Int13] Intel. Intel SGX for Dummies (Intel SGX Design Objectives). https://software.intel.com/content/www/us/en/develop/blogs/protecting-application-secrets-with-intel-sgx.html, 2013. Abgerufen am 14.06.2020.

[Int16] Intel. Intel 64 and IA-32 Architectures Software Developers Manual. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3d-part-4-manual.pdf, 2016. Abgerufen am 14.06.2020.

[Liu] S. Liu. Size of the Bitcoin blockchain from 2010 to 2020, by quarter. https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/. Abgerufen am 14.06.2020.

[Man20] M. Mantel. Neue CPU-Sicherheitslücken in Intel-Prozessoren umgehen Kern-Grenzen. https://www.heise.de/news/Neue-CPU-Sicherheitsluecken-in-Intel-Prozessoren-umgehen-Kern-Grenzen-4780215.html, 2020. Abgerufen am 15.06.2020.

[Mat19] S. Matetic. Bite: Bitcoin Lightweight Client Privacy using Trusted Execution. https://www.usenix.org/system/files/sec19-matetic.pdf, 2019. Abgerufen am 12.06.2020.

[Nak08] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. Abgerufen am 12.06.2020.

[PDF] New P6 OpCodes: CMOV. http://www.rcollins.org/p6/opcodes/CMOV.html. Abgerufen am 15.06.2020.

[Ran15] A. Rane. Raccoon: Closing Digital Side-Channels through Obfuscated Execution. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-rane.pdf, 2015. Abgerufen am 12.06.2020.

[Ste14] E. Stefanov. Path ORAM: An Extremely Simple Oblivious RAM Protocol. https://arxiv.org/abs/1202.5150v3, 2014. Abgerufen am 14.06.2020.

[Yan20] M. Yano. In *Blockchain and Crypto Currency (978-981-153-376-1)*. Springer, 2020.