TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Mihkel Jõgeda 232625IVCM

PROPAGATION OF ARTIFACTS BETWEEN IOS DEVICES

Master's thesis

Supervisor: Matthew James Sorell

PhD

Tallinn 2026

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mihkel Jõgeda 232625IVCM

Artefaktide levimine iOS seadmete vahel

Magistritöö

Juhendaja: Matthew James Sorell

PhD

Tallinn 2026

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mihkel Jõgeda

11.05.2026

Abstract

When multiple iOS devices share a single iCloud account, forensic artifacts such as Safari browsing records, call history, SMS messages, and iMessages may propagate between devices through iCloud synchronisation. This creates a forensic attribution problem: an examiner acquiring one device cannot immediately determine whether a record was generated locally or received as a synchronised copy from another linked device. Misinterpreting this distinction can lead to incorrect timelines, duplicate evidentiary interpretation, and attribution of user activity to the wrong device.

This thesis investigates whether native iOS databases contain provenance indicators that allow forensic examiners to distinguish locally generated artifacts from iCloud-synchronised records. Three jailbroken iPhones running iOS 12.5.8, iOS 15.8.3, and iOS 16.7.12 were configured under a single Apple ID and subjected to a controlled test plan covering cellular calls, FaceTime calls, SMS, iMessage, Safari bookmarks, and Safari browsing activity. Full filesystem acquisitions were examined using iLEAPP, DB Browser for SQLite, fqlite, and manual field-level comparison of relevant SQLite databases and plist files.

The results show that iOS stores several useful synchronisation and correlation indicators, but no single universal device-origin field exists across the examined artifact categories. Safari bookmark and CloudTabs records preserve CloudKit metadata such as record identifiers, modification timestamps, and the user-assigned device name of the last modifying device. Call history records can be correlated across devices using identifiers such as ZUNIQUE_ID, although synchronisation coverage was inconsistent, especially on the iOS 12 device. SMS and iMessage records contain stable cross-device identifiers such as GUID and CloudKit record fields, while destination_caller_id and timestamp relationships provide conditional provenance value. However, sms.db does not directly identify the physical device that originated a message record.

The thesis concludes that forensic examiners can often confirm that an artifact participated in iCloud synchronisation and can correlate copies of the same event across devices, but device-origin attribution remains artifact-specific and limited.

This thesis is written in English and is 86 pages long, including 6 chapters, 3 figures and 19 tables.

Lühikokkuvõte

Artefaktide levimine iOS seadmete vahel

Kui mitu iOS-seadet kasutavad sama iCloudi kontot, võivad digitaalse ekspertiisi seisukohalt olulised artefaktid, nagu Safari sirvimisandmed, kõneajalugu, SMS-sõnumid ja iMessage'i sõnumid, iCloudi sünkroonimise kaudu seadmete vahel levida. See tekitab seadme omistamise probleemi: ühe seadme uurimisel ei saa ekspert kindlaks teha, kas konkreetne kirje loodi selles seadmes kohapeal või saabus sünkroonitud koopiana mõnest teisest sama kontoga seotud seadmest. Selle eristuse valesti tõlgendamine võib viia ebatäpsete ajajoonte koostamiseni, tõendite väärarvestamiseni ning kasutaja tegevuse omistamiseni valele seadmele.

Käesolevas magistritöös uuritakse, kas iOS-i andmebaasid sisaldavad päritoluindikaatoreid, mis võimaldavad eksperdil eristada kohapeal loodud artefakte iCloudi kaudu sünkroonitud kirjetest. Uuringus kasutati kolme jailbreak'itud iPhone'i, mille operatsioonisüsteemid olid iOS 12.5.8, iOS 15.8.3 ja iOS 16.7.12. Kõik kolm seadet seadistati ühe Apple ID alla ning nendega viidi läbi kontrollitud testiplaan, mis hõlmas mobiilikõnesid, FaceTime'i kõnesid, SMS-sõnumeid, iMessage'i sõnumeid, Safari järjehoidjate lisamist ja Safari veebisirvimist. Failisüsteemi tõmmiseid analüüsiti tööriistade iLEAPP, DB Browser for SQLite ja fqlite abil ning asjakohaseid SQLite-andmebaase ja plist-faile võrreldi käsitsi väljade tasemel.

Tulemused näitavad, et iOS talletab mitmeid kasulikke sünkroonimise ja kirjete seostamise indikaatoreid, kuid uuritud artefaktikategooriates ei esine ühtset universaalset seadme päritolu määravat välja. Safari järjehoidjate ja CloudTabs'i kirjed säilitavad CloudKiti metaandmeid, näiteks kirjeidentifikaatoreid, muutmise ajatempleid ning viimase muudatuse teinud seadme kasutaja määratud nime. Kõneajaloo kirjeid saab seadmete vahel seostada selliste identifikaatorite abil nagu ZUNIQUE_ID, kuigi sünkroonimise ulatus oli ebaühtlane, eriti iOS 12 seadmes. SMS- ja iMessage'i kirjed sisaldavad stabiilseid seadmeteüleseid identifikaatoreid, nagu GUID ja CloudKiti kirjeväljad, samas kui destination_caller_id ja ajatemplite omavahelised seosed pakuvad

tingimuslikku väärtust artefakti päritolu hindamisel. Samas ei tuvasta sms.db otseselt füüsilist seadet, millest sõnumikirje algselt pärines.

Magistritöö järeldab, et digitaalse kriminalistika eksperdid saavad kinnitada, et artefakt on osalenud iCloudi sünkroonimises, ning seostada sama sündmuse koopiaid eri seadmetes, kuid seadme päritolu määramine sõltub artefakti tüübist ja on piiratud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 86 leheküljel, 6 peatükki, 3 joonist, 19 tabelit.

List of abbreviations and terms

A8 / A9 / A10 / A11	Apple-designed System-on-Chip generations used in iPhone 6, iPhone SE 1G, iPhone 7, and iPhone 8 / 8 Plus respectively
API	Application Programming Interface
ASLR	Address Space Layout Randomisation; randomises memory addresses to defeat injection-based exploits
bplist	Binary property list; Apple's binary serialisation format for plist data
CK / CloudKit	Apple's framework for synchronising application data through iCloud
CKRecord	CloudKit object representing a single synchronised record
CRDT	Conflict-free Replicated Data Type; data structure used by CloudKit to merge concurrent edits
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
ETag	Entity Tag; CloudKit-assigned hex string incremented on each server-side write to a record
GUID	Globally Unique Identifier
iLEAPP	iOS Logs, Events, And Plists Parser; open-source iOS-specialised forensic parser
OS	Operating System
Plist	Property list; Apple configuration file format (XML or binary)
TalTech	Tallinn University of Technology
USB	Universal Serial Bus
UUID	Universally Unique Identifier
WAL	Write-Ahead Log; SQLite journaling mode used by iOS databases

Table of contents

List of figures 14

List of tables 15

1 Introduction 16

1.1 Background and Context 17

1.2 Problem Statement 17

1.3 Research Objectives 18

1.4 Significance of the Study 19

2 Literature Review 21

2.1 iOS as a Forensic Investigation Target: Context and Significance 22

2.2 The iOS Security Architecture and Its Forensic Implications 23

2.3 Forensic Acquisition Methods for iOS Devices 24

2.3.1 The Acquisition Hierarchy 24

2.3.2 Logical Extraction via iTunes/Finder Backup 25

2.3.3 Jailbreak-Enabled File-System Extraction 26

2.3.4 iCloud Backup as an Acquisition Vector 26

2.4 Forensic Artifacts on iOS Devices: Databases and Data Structures 27

2.4.1 The Primacy of SQLite 27

2.4.2 SMS and iMessage: sms.db 28

2.4.3 Call Logs: CallHistory.storedata 29

2.4.4 Safari Data 29

2.4.5 Supplementary Artifacts: AppConduit, Accounts3.sqlite, and Plist Files 30

2.5 Forensic Timeline Construction and Its Limitations 31

2.6 iCloud Synchronization: Architecture, Behavior, and Forensic Consequences 32
2.6.1 The iCloud Platform and Its Forensic History 32
2.6.2 System-Controlled Synchronization: The Non-Deterministic Model 33
2.6.3 CloudKit: The Framework Underlying Synchronization 34
2.6.4 CloudKit Throttling and Its Forensic Implications 35
2.7 Cloud Forensics on iOS: Evidence Recovery and Metadata Integrity 35
2.7.1 iCloud Data Acquisition and the Timestamp Alteration Finding 35
2.7.2 Cloud Storage Residual Artifacts on iOS Devices 36
2.7.3 iOS Cloud App Forensics and the APT Security Taxonomy 37
2.7.4 Forensic Analysis of Medical Device Applications as a Model 38
2.7.5 Network Traffic as a Verification Method 38
2.8 Forensic Tool Capabilities and Their Limitations 39
2.8.1 Commercial Forensic Tools 39
2.8.2 Open-Source Tools: iLEAPP and iOS Artifact Coverage 40
2.9 Multi-Device Artifact Propagation: Mapping the Research Gap 41
2.9.1 What the Literature Collectively Establishes 41
2.9.2 The Specific Empirical Gap 42
2.9.3 Consequences of the Gap for Forensic Practice 43
2.9.4 Methodological Framework for the Proposed Research 43
2.10 Conclusion 44
3 Research Design and Methodology 46
3.1 Research Approach and Justification 46
3.2 Forensic Test Plan 47
3.2.1 Device setup 47
3.2.2 Analysis tools 48
3.2.3 Test Cases Description 48

3.2.4 Test Cases: Calls 49
3.2.5 Test Cases: SMS 49
3.2.6 Test Cases: iMessage 50
3.2.7 Test Cases: Safari Bookmarks 51
3.2.8 Test Cases: Safari Web Browsing 51
3.3 Forensic Acquisition Procedure 52
3.3.1 Acquisition Method Selection 52
3.3.2 Jailbreak and Device Preparation 52
3.3.3 File System Extraction 54
3.3.4 Acquisition Timeline 54
3.4 Artifact Analysis Methodology 56
3.4.1 Initial Parse with iLEAPP 56
3.4.2 Marker-Based Record Identification 56
3.4.3 Manual Schema and Metadata Inspection 57
3.4.4 Classification of Provenance Indicators 57
4 Forensic Artifact Catalog 59
4.1 Establishing Multi-Device Context 59
4.1.1 Find My Friends Daemon Plist 60
4.1.2 iCloud Keychain Peer Trust Database 60
4.2 Safari Artifacts 62
4.2.1 History.db 62
4.2.2 Bookmarks.db 63
4.3 Call Log Artifacts 65
4.3.1 Database Schema 66
4.3.2 Key Forensic Fields 67
4.4 SMS/iMessage Artifacts 69
4.4.1 Key Forensic Fields 70

4.4.2 Binary Data Fields 71
5 Analysis and findings 73
5.1 Multi-Device context 73
5.2 Safari Data 74
5.2.1 History.db 74
Bookmarks.db 78
5.2.2 Cloudtabs.db 79
5.2.3 Summary of Findings 81
5.3 Call Log Data 81
5.3.1 Sync Coverage 82
5.4 SMS Data 82
5.4.1 Summary of Findings 86
5.5 Cross-Artifact Identifier Analysis 87
6 Conclusion 89
6.1 Answers to the research questions 89
6.1.1 Primary Research Question: What identifiable forensic indicators allow an examiner to differentiate between locally generated and iCloud-synchronised Safari browsing data, call logs, and SMS messages on iOS devices? 89
6.1.2 Secondary Research Question 1: Which iOS databases contain metadata identifying other devices associated with the same iCloud account? 91
6.1.3 Secondary Research Question 2: What metadata fields or artifact characteristics within these databases can be used to distinguish locally created records from iCloud-synchronised records? 92
6.2 Examiner workflow 93
6.2.1 Establishing the multi-device context 94
6.2.2 Safari bookmarks 95
6.2.3 Safari iCloud Tabs 96
6.2.4 Call history 96

6.2.5 SMS and iMessage 97
6.2.6 Cross-artifact reconciliation 98
6.3 Limitations 99
6.4 Future Work 100
References 102
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis 106

List of figures

Figure 1. Devices used for testing 48
Figure 2 - Apple web product color codes 73
Figure 3. Examiner workflow..... 94

List of tables

Table 1. List of devices..... 47
Table 2. Analysis tools 48
Table 3. Test Cases: Calls..... 49
Table 4. Test Cases: SMS..... 50
Table 5. Test Cases: iMessage..... 50
Table 6. Test Cases: Safari Bookmarks..... 51
Table 7. Test Cases: Safari Browsing..... 51
Table 8. CallHistory.storedata schema evolution..... 67
Table 9. Example output of modified iLEAPP trustedpeers module 74
Table 10. Expected versus actual sync propagation for browsing test cases. 75
Table 11. Origin field values for TC-W02 76
Table 12. Redirect chain preservation during sync (TC-W02) 76
Table 13. Provenance indicators for Safari History.db 77
Table 14. Provenance indicators for Bookmarks.db records..... 79
Table 15. Provenance indicators for Cloudtabs.db records..... 81
Table 16. Test case to record mapping and sync results 82
Table 17. SMS test case to record mapping and sync results..... 83
Table 18. Carrier message sync behaviour..... 84
Table 19. Provenance indicators for SMS/iMessage records. 86

1 Introduction

Apple's ecosystem operates as a highly interconnected environment in which iCloud services synchronize a wide range of digital artifacts such as Safari browsing history, call logs and messaging records across devices including iPhones, iPads, and macOS systems. This continuous synchronization supports usability and allows users to transition seamlessly between devices.

While this enhances the user experience, it introduces substantial complexity for digital forensic investigations. Because multiple devices share and update the same data, a single artifact can appear in several locations. This replication makes it extremely difficult for forensic analysts to determine where an artifact was originally created, how it evolved, and in what sequence events occurred.

In the context of a criminal investigation, the ability to identify the source device on which an action was performed is often critical. Establishing whether a user was actively using a particular device at a given time may form a key part of the prosecution's case, for example by linking suspect activity to a device known to be in the user's possession. Conversely, accurate device attribution is equally important for testing and substantiating defence claims, such as an alibi based on the asserted use or non-use of a specific device.

These challenges become even more pronounced in scenarios where multiple individuals share the same iCloud account. For instance, family members may use a single Apple ID across multiple devices for convenience or cost reasons, or a suspect may claim that browsing activity or calls attributed to their device were actually performed by another person with access to the shared account. In such cases, synchronized artifacts may appear on all devices linked to that Apple ID, making it nearly impossible to determine which individual actually performed the action without reliable forensic indicators. This ambiguity can be exploited to challenge evidence attribution and raise reasonable doubt in legal proceedings.

Further complications arise from the way iCloud synchronization processes data. Sync delays, background update mechanisms, and device-specific handling of timestamps may

produce subtle variations or inconsistencies between devices. These discrepancies can obscure the true timeline of user activity and impede accurate event reconstruction.

For forensic investigators, these challenges have direct practical consequences. When synchronized artifacts are indistinguishable from locally generated ones, investigators may incorrectly attribute actions to the wrong device or misinterpret the timing and context of events. Such misattribution can undermine the reliability of digital evidence, affect investigative conclusions, and potentially compromise legal proceedings where precise device attribution and timeline accuracy are critical.

Current forensic tools and methodologies lack reliable indicators to distinguish between artifacts created locally on a device and those propagated through iCloud synchronization. This gap leaves investigators without systematic guidance for interpreting multi-device evidence in Apple ecosystems, which are increasingly common in criminal investigations.

Given the widespread use of Apple devices and the increasing dependence on iCloud-based synchronization, forensic investigators require a deeper understanding of artifact propagation, transformation, and metadata behavior within the iOS ecosystem. This motivates a structured investigation into how artifacts move between devices, how synchronization affects their forensic characteristics, and how investigators can reliably determine artifact origin in multi-device environments.

1.1 Background and Context

1.2 Problem Statement

The replication of digital artifacts across multiple iOS devices through iCloud synchronization creates significant challenges for forensic interpretation. When artifacts such as Safari history entries or call logs appear simultaneously on different devices, investigators cannot easily determine:

- where each artifact was originally created,
- how it propagated between devices,
- whether its metadata (timestamps, identifiers) reflects actual user activity or system synchronization behavior.

Current forensic tools treat synchronized artifacts and locally generated artifacts similarly, lacking indicators to distinguish between them. This limitation can lead to incorrect reconstruction of event timelines, misattribution of actions to the wrong device, and inconsistent investigative results.

1.3 Research Objectives

Primary Research Question:

What identifiable forensic indicators allow an examiner to differentiate between locally generated and iCloud-synchronized Safari browsing data, call logs, and SMS messages on iOS devices?

Secondary Research Questions

SRQ1: Which iOS databases contain metadata identifying other devices associated with the same iCloud account?

SRQ2: What metadata fields or artifact characteristics within these databases can be used to distinguish locally created records from iCloud-synchronized records?

Scope

This research focuses specifically on:

- Safari browsing data (history entries, bookmarks if applicable, related SQLite/plist files)
- Call log data (incoming, outgoing, missed calls, associated databases and metadata)
- iOS devices running with iCloud synchronization enabled
- Controlled experimental environments where user activities and synchronization events can be systematically documented and analyzed
- Metadata analysis of artifacts extracted through file-system extraction of jailbroken devices

1.4 Significance of the Study

This research addresses a significant gap in digital forensics by examining the forensic interpretation of Safari browsing data, call logs, and SMS messages in multi-device iOS environments that share a single Apple ID. While prior studies have analysed these artifacts on individual devices or explored general iOS forensic techniques, the implications of iCloud-based synchronization for multi-device evidence interpretation have not been systematically examined.

The novelty of this work lies in its shift from artifact generation to forensic inference. Rather than analysing how artifacts are created or how iCloud synchronization operates internally, this research focuses on what investigators can observe and use within extracted forensic data to determine:

- which devices are associated with the same iCloud account,
- which artifacts are likely to originate from a local device versus another synchronized device
- how synchronization influences timeline interpretation and device attribution.

Compared to existing studies, this research differs by:

- Focusing exclusively on Safari, call logs and sms messages within a multi-device iCloud context.
- Using controlled, experimental analysis across multiple devices rather than single-device snapshots.
- Emphasizing artifact origin determination rather than general extraction or tool performance.

The main contribution of this work is the development of a practical forensic methodology that enables investigators to assess artifact provenance with greater accuracy in environments where multiple iOS devices share the same Apple ID. This methodology is supported by a structured mapping of relevant iOS databases and the identification of specific metadata fields that act as indicators of device association and artifact origin.

The findings further demonstrate how these indicators can be operationalized through proof-of-concept scripts for structured extraction and analysis, supporting the generation of device-based timelines. Together, these outcomes equip forensic investigators with actionable techniques to interpret synchronized iOS data, improve device attribution, and enhance the reliability and accuracy of digital investigations involving multi-device Apple ecosystems.

2 Literature Review

In recent years, the combination of personal computing and cloud infrastructure has fundamentally changed the situation that digital forensic investigators are facing. This change is especially visible in the Apple iOS ecosystem, where one iCloud account regularly synchronizes data across iPhones, iPads, and Mac computers — making copies of contacts, messages, call history, browser records, and many other artifact types across devices that can be in different geographic locations and owned by the same person. From the point of view of device-level forensic analysis, this replication creates a serious interpretive problem: an artifact that is found on an examined device may not have been created on that device. It may instead be a propagated copy of an event that happened on a different device, which was transmitted through Apple's iCloud infrastructure at some time after the original event occurred.

The research problem that this thesis is addressing is the absence of documented and validated forensic indicators that would allow an examiner to make a distinction between a locally generated artifact and an iCloud-synchronized artifact within three categories of data that are most important in criminal and civil investigations: Safari browsing history, call logs, and SMS/iMessage records. Forensic tools that currently exist and published methodologies treat locally generated and synchronized records in the same way, using the same extraction procedures, the same timestamp interpretations, and the same analytical frameworks regardless of how an artifact arrived on the device. This approach creates risks of systematic timeline errors, device misattribution, and investigative inconsistency — consequences which can have serious legal implications.

This literature review surveys scholarship in four intersecting domains for establishing the foundation for the proposed research. It begins with the iOS security architecture and the acquisition methods that are available to forensic investigators, then goes to a detailed examination of the artifact databases that are central to the research, then addresses forensic timeline construction and its limitations, the iCloud synchronization architecture and its forensic consequences, cloud forensics methodology and metadata integrity findings, and existing forensic tool capabilities. It finishes by mapping the specific research gap and positioning the proposed contribution.

2.1 iOS as a Forensic Investigation Target: Context and Significance

Apple's iPhone and iPad have achieved a global market presence of sufficient scale that they are routinely encountered in criminal, civil, and regulatory investigations. Sibe and Alayegun note that as of the third quarter of 2024, Apple shipped approximately 45.5 million smartphones globally per quarter, representing a 17% global market share and a position of considerable dominance in the premium device segment [1]. As of the second quarter of 2024, iOS held a 27.62% share of the global mobile operating system market, with the remainder largely divided among Android variants [1]. Ntshangase et al. survey the forensic tool landscape in response to a corresponding growth in mobile-related crime, noting that both law enforcement and the academic community have struggled to keep pace with the rapid evolution of mobile operating systems and their security mechanisms [2].

The significance of iOS as a forensic target is made larger by several characteristics that are unique to Apple's ecosystem. First, iCloud integration is deeply embedded in the platform: unlike Android, where cloud synchronization is optional and fragmented across many providers, iOS users are actively encouraged to enable iCloud synchronization for Messages, Safari, FaceTime, Phone, Contacts, and many other system applications from the first moment of device setup. By the time a device comes to an investigator, it is frequently operating as one node in a multi-device iCloud network. Second, the closed and vertically integrated nature of iOS means that Apple has much greater control over data storage, synchronization protocols, and encryption than any Android manufacturer, making the knowledge required to interpret iOS forensic artifacts both more specialized and, at the same time, more consistent across devices of the same generation than in the fragmented Android ecosystem.

Third, and most relevant to the present research, the multi-device character of iOS use means that evidence found on one device cannot be assumed to have been created on that device. A call log entry on a victim's iPhone may record a call that was made on their secondary phone. A Safari history entry on a suspect's secondary iPhone may reflect browsing that was done on their primary device some hours earlier. Unless forensic analysts have tools and knowledge to make this distinction, they risk presenting evidence that incorrectly attributes device origin, a significant error in any proceeding where device location or identity is a material question.

2.2 The iOS Security Architecture and Its Forensic Implications

A rigorous understanding of iOS security architecture is a prerequisite for understanding both the challenges and the opportunities that are available to forensic investigators. Liu et al. analyze the iOS security mechanisms from the perspective of exploit research, and they provide a clear exposition of the layered defenses that the operating system employs [3]. The trusted boot chain ensures that each stage of the startup process, from the read-only Bootrom through the Low-Level Bootloader (LLB) and iBoot to the kernel and applications, is cryptographically verified before execution is allowed, so that any unauthorized modification to the boot process will cause the device to fail to start. This chain of trust protects the integrity of the system partition from external modification and is the root reason why iOS is resistant to the kinds of firmware-level evidence manipulation that are routine concerns in other digital forensic contexts.

Above the boot chain, iOS implements Data Execution Prevention (DEP), Address Space Layout Randomisation (ASLR), privilege separation, and application sandboxing [3]. DEP, which was introduced in iOS 2.0, prevents the processor from treating data memory as executable code, and this blocks injection-based exploits. ASLR, which was introduced in iOS 4.3, randomises the memory addresses of the main program, dynamic libraries, heap, and stack at each execution, making it impossible for an attacker or a forensic tool to reliably predict where specific code or data will be in memory. Privilege separation restricts third-party applications to the mobile user's privilege level, which prevents them from modifying system resources. The sandbox mechanism, which is implemented through the TrustedBSD mandatory access control framework, isolates each application in its own container, preventing cross-application data access without explicit system mediation.

D'Orazio and Choo extend this analysis to the specific context of forensic investigation, noting that from iOS 8 onwards, the application data container, which is the place that houses the SQLite databases containing messages, call logs, and browser artifacts, is not accessible to commercial forensic software through standard logical acquisition. [4]. Apple File Conduit (AFC), which is the service that exposes the iOS file system over USB, is restricted in its unmodified state to the Media folder, preventing access to the app

data container without either a jailbreak or a legal compulsion directed at Apple itself. This structural inaccessibility is reinforced by Apple's implementation of full-disk encryption using the Secure Enclave, which ties the encryption key to the device hardware and the user's passcode, making offline decryption without the passcode computationally infeasible.

Wu et al. confirm that the iOS operating system's closed-source nature and prohibition on system partition modification restrict forensic access to the user data partition alone [5]. AlBreiki et al. describe how iOS 17 extended these protections with expanded Secure Enclave coverage and the optional Advanced Data Protection for iCloud, which moves iCloud backup encryption keys entirely to the user's device, making even Apple-assisted access to iCloud data impossible without user cooperation [6]. For forensic investigators, this architecture defines a hierarchy of data accessibility: iCloud-encrypted data is the least accessible, followed by encrypted device backups, then unencrypted logical backups, and finally file-system data that is accessible only through jailbroken devices or specialized hardware. The proposed research operates across this hierarchy, using logical backup, jailbreak-enabled file-system extraction, and iCloud backup download for ensuring that the same artifact categories can be examined through all three acquisition pathways.

2.3 Forensic Acquisition Methods for iOS Devices

2.3.1 The Acquisition Hierarchy

Forensic acquisition of iOS devices spans a spectrum of methods that trade forensic completeness against complexity, risk, and legal admissibility. Koganti and Rao document the hierarchy from manual extraction, which involves direct screen interaction and is limited to what is visible on-screen, through logical extraction, file-system extraction, chip-off, and micro-read [7]. Each ascending tier recovers more data at greater cost and with greater potential for evidence modification. Wu et al. categorize the practically relevant tiers for iOS forensics as manual, logical, and physical, noting that physical extraction via chip-off or JTAG is rarely feasible on modern Apple devices because of their integrated circuit design and hardware encryption [5].

2.3.2 Logical Extraction via iTunes/Finder Backup

The logical backup via iTunes (or Finder in macOS Catalina and later versions) remains the most widely used and legally unambiguous acquisition method for iOS devices. Shimmi et al. describe the backup structure in detail: the backup folder is identified by a 40-character hexadecimal string that represents the device's unique identifier, and the files within it are named using SHA1 hashes of their original file system paths [8]. For iOS 9 through 12, they document the presence of SQLite databases for SMS (sms.db), address book contacts, call log (CallHistory.storedata), and calendar data within these backup structures. Sibe and Alayegun confirm this approach on a more recent device, an iPhone XR running iOS 17.5, using Oxygen Forensic Device Extractor v2.13.1 to process both iTunes and iCloud backups, and they recovered contacts, SMS data, call history, media files, and browser data [1]. Their work demonstrates that logical backup remains viable on current iOS versions, though its completeness is limited by what Apple's backup APIs choose to expose.

A critical limitation of logical backup is that it does not provide access to the raw file system, meaning that artifacts stored in locations not covered by the backup API, or that exist only transiently in the file system, are not recoverable. Deleted records, in particular, are typically excluded because logical acquisition does not retrieve allocated but unindexed data. Neyaz et al. demonstrate an alternative logical-equivalent acquisition using iMazing, which is a third-party iOS management tool [9]. Without jailbreaking, iMazing can extract user-space data including message logs, call logs, and iCloud-synced SSID data in formats including text files, CSV files, and plist files. Notably, iMazing's exposure of iCloud-synced SSID data, which records the Wi-Fi networks to which the device has connected, synchronized across the iCloud account, illustrates a category of artifact that carries explicit evidence of synchronization: the SSID list is per-account, not per-device, and its presence on a device that has never connected to a given network reflects iCloud propagation.

2.3.3 Jailbreak-Enabled File-System Extraction

The practical necessity of jailbreaking to achieve comprehensive forensic access to iOS application data is a recurring theme in the literature. D'Orazio and Choo acknowledge this explicitly, noting that without a jailbreak, a forensic researcher cannot access the app data container on any device running iOS 8 or later when using commercial tools [4]. Liu et al. provide a technical account of how jailbreaking defeats the iOS security chain: by exploiting vulnerabilities in the bootloader, kernel, or system processes, jailbreaks achieve root-level code execution, patch the kernel's access controls, and install alternative package managers that allow unrestricted software installation [3]. After this, forensic tools can connect to the jailbroken device via SSH, mount the file system, and perform a bit-for-bit copy of the user data partition.

Laayu and Kurniawan operationalize the comparison between jailbroken and non-jailbroken acquisition in a controlled experiment using an iPhone 7 Plus running iOS 14.8.1 [10]. Their results demonstrate quantitative differences in artifact recovery: jailbroken acquisition recovers substantially more records from application databases that are either partially or entirely excluded from iTunes backups. Wu et al. reach the same conclusion, confirming that jailbreaking dramatically increases the yield of forensically relevant artifacts, particularly instant messaging data and application-level databases [5]. The trade-off is the forensic soundness concern: a jailbreak modifies the device's software state, which may be challenged as evidence tampering in adversarial legal proceedings. In the proposed research, jailbreak-enabled file-system extraction is used alongside logical backup to maximize artifact visibility, with the forensic soundness issue mitigated through careful documentation and hash verification before and after extraction.

2.3.4 iCloud Backup as an Acquisition Vector

The iCloud backup represents a distinct acquisition pathway that is neither a local device backup nor a live file-system image, but rather a snapshot of the device's data state as it existed at the time of the last backup, stored on Apple's servers. Oestreicher is the foundational study of iCloud as a forensic acquisition target, describing a methodology for downloading iCloud-synchronized files via the native Mac OS X synchronization mechanism and verifying their integrity [11]. Sibe and Alayegun confirm that iCloud

backup acquisition via Oxygen Forensic Device Extractor is practical on current iOS versions and can yield artifacts that are comparable to those from local iTunes backups, with the additional advantage that the iCloud backup may be more recent or may contain data from a device that is not physically available to the investigator [1].

The forensic complications of iCloud backup acquisition are substantial. Apple's implementation of Advanced Data Protection encrypts iCloud backup content end-to-end with keys held only on the user's devices, making the data inaccessible to Apple, and therefore inaccessible via legal process directed at Apple, without the user's cooperation [6]. Where Advanced Data Protection is not enabled, Apple retains the ability to provide backup data in response to lawful process, but the content arrives without the contextual metadata that would identify the individual device origins of synchronized records.

2.4 Forensic Artifacts on iOS Devices: Databases and Data Structures

2.4.1 The Primacy of SQLite

iOS's widespread use of SQLite as its embedded database format makes SQLite expertise foundational for iOS forensic analysis. Shimmi et al. observe that SQLite is used by virtually all system applications and many third-party applications to store structured data, including messages, call records, browser history, contacts, and calendar entries [8]. The SQLite format is a single-file relational database that supports multiple tables, views, and indices: compact enough for embedded use yet powerful enough to store rich relational data. How an investigator encounters these files depends significantly on which acquisition method is used. When the device file system is accessed directly through jailbreak-enabled extraction, the databases appear under their actual names and paths (sms.db, CallHistory.storedata, and so on) exactly as iOS itself references them. The iTunes backup format, by contrast, reorganizes these files according to its own scheme: Shimmi et al. document that in the iOS versions they studied (iOS 9–12), the backup stores files under hashed filenames that are derived from their original paths, which requires forensic tools that operate against iTunes backups to maintain internal mappings between those hashes and known artifact locations [8]. The practical consequence is that investigators and tools working from file system images work with familiar, human-

readable filenames, while those working from iTunes backups must rely on tool-maintained lookup tables. This becomes a problem whenever a tool's mapping is incomplete or out of date following an iOS update.

2.4.2 SMS and iMessage: sms.db

The central repository for SMS and iMessage data on iOS is sms.db, which is located at /private/var/mobile/Library/SMS/. Forensafe provides a detailed enumeration of the fields that are available within this database as exposed by forensic parsing tools [12]. The most forensically significant fields include `Message Time` (the timestamp of the message event), `Message Direction` (incoming or outgoing), `Message Delivered Time` (when the message was delivered to the device), `Chat GUID` (a globally unique identifier for the conversation), `Message GUID` (a unique identifier for the individual message), `User Account GUID` (the iCloud account identifier associated with the message), `Is Sent`, `Is Delivered`, `Is System Message`, and `Partner Uncanonicalized ID` (the raw identifier of the other party). The multiplicity of GUID fields is particularly relevant to the research problem: each one represents a distinct level of identity, namely message, conversation, and account, that may carry information about the originating device or service.

Shimmi et al. analyze the schema of sms.db across iOS 9, 10, 11, and 12, and they reveal that the transition from iOS 10 to iOS 11 produced a growth rate and change rate of 56% — which means that more than half of the database's table structure was reorganized in a single major update [8]. Specifically, the number of tables grew from 9 to 14 between iOS 10 and 11, with corresponding additions and deletions at the column level. This magnitude of schema change has direct implications for forensic tools that rely on hardcoded SQL queries: a query written for iOS 10's sms.db schema may fail or return incorrect results against iOS 11's reorganized tables. More relevant for the present research, metadata fields that are relevant to identifying synchronization provenance, such as device origin flags or iCloud record identifiers, may exist in one iOS version's schema but not another, or may be named differently across versions. Any forensic indicator identified in this research must therefore be validated across the specific iOS versions that are under study.

2.4.3 Call Logs: CallHistory.storedata

The call history database, CallHistory.storedata, resides at /private/var/mobile/Library/CallHistoryDB/. This database records incoming, outgoing, and missed calls across all call types, including cellular calls, FaceTime Audio, and FaceTime Video. Studiawan et al. document this file as a key iOS forensic artifact and note that it contains temporal data that is essential for timeline construction [13]. The schema evolution data from Shimmi et al. reveals that the call log database grew from 5 tables in iOS 9–11 to 7 tables in iOS 12, with elevated growth and change rates (40% growth, 40% change) in the iOS 11 to iOS 12 transition — which indicates substantial structural reorganization [8]. The addition of tables in iOS 12 is particularly noteworthy because new tables often represent new categories of data that Apple chose to begin recording, and these potentially include synchronization-related metadata.

The call log database is relevant to the research problem in a specific and important way. Under iCloud synchronization, call logs are synchronized across devices that share an iCloud account, which means that a call received on one iPhone will appear in the call history of every other associated iOS device. The database may contain fields indicating which device handled the call versus which device received the record through synchronization, but no published research has systematically investigated this distinction. This is a primary empirical gap that the proposed research addresses.

2.4.4 Safari Data

Safari stores browsing artifacts across multiple SQLite databases and plist files under /private/var/mobile/Library/Safari/. Studiawan et al. include Bookmarks.db in their catalog of iOS forensic artifacts and demonstrate that it contains timestamp data that is useful for forensic timeline construction [13]. AlBreiki et al. confirm that Safari history is recoverable from iOS 17.3 images through both automated tools and manual database inspection, but they note that privacy-focused browsing modes leave only partial traces [6].

The Safari artifacts are relevant to the present research because Safari history synchronization is a core iCloud feature, which is enabled by default for users who have signed into iCloud with the Safari option activated. When history entries are synchronized from a second device, they appear in the local Safari history database without any visible distinction — at least at the level of what current forensic tools are reporting. The question of whether the database schema includes fields that record synchronization origin is a central empirical question of the present research.

2.4.5 Supplementary Artifacts: AppConduit, Accounts3.sqlite, and Plist Files

Beyond the primary databases, several supplementary iOS artifacts are relevant to the multi-device synchronization problem. Studiawan et al. document AppConduit.log, which is located at /private/var/mobile/Library/Logs/AppConduit/, as an artifact that records "interaction between iOS devices" [13]. This log file is particularly promising for the present research: if AppConduit logging captures events associated with iCloud synchronization, it may provide independent evidence of when specific data arrived on the device from another source. Studiawan et al. also catalog Accounts3.sqlite at /private/var/mobile/Library/Accounts/, which contains account information including usernames, account types, and descriptions — and this potentially records the iCloud account configuration and associated device identifiers. Both artifacts are candidates for systematic investigation in the proposed experimental design.

Plist files constitute another category of supplementary forensic evidence. Studiawan et al. parse the com.apple.CarPlayApp.plist file for demonstrating the forensic value of plist data for timeline construction [13]. More directly relevant to the present research, Wi-Fi configuration plist files (com.apple.wifi.plist) are documented as recording device-level Wi-Fi connection history, and property list files governing iCloud service configuration may encode device identifiers or synchronization state. Gómez-Miralles and Arnedo-Moreno demonstrate through their analysis of AirPrint forensic traces that even incidental system features generate recoverable plist and network-level artifacts that can document inter-device activity — and this is a methodological precedent for the systematic examination of feature-specific artifacts that is proposed here [22].

2.5 Forensic Timeline Construction and Its Limitations

The construction of a unified forensic timeline from iOS artifacts is a prerequisite for any analysis of how events propagated between devices — and yet the literature consistently documents substantial gaps in the tools that are currently used for this purpose. Studiawan et al. directly address this gap by demonstrating that log2timeline plaso, which is the de facto open-source standard for multi-platform forensic timeline construction, fails to parse multiple iOS artifact types when it is applied to an iOS 13.4.1 forensic image [13]. Their experiments show that the call history database (CallHistory.storedata) is among the artifacts not parsed by the unmodified plaso, which means that call events generate no entries in the forensic timeline at all unless a custom plugin is deployed. Studiawan et al. note that iLEAPP (iOS Logs, Events, And Plists Parser), which is a dedicated iOS artifact parser, achieves substantially more complete coverage of iOS-specific artifacts, including system logs, plists, and application databases, than plaso's generic framework [13]. For research that is focused exclusively on iOS artifacts, iLEAPP's iOS-centric design makes it the more appropriate open-source tool.

The implications of plaso's omission extend directly to the synchronization problem. If call log timestamps are not visible in a generic forensic timeline tool, then any analysis of the temporal relationship between call events on different devices, which is necessary to determine whether a call record on a secondary device preceded or followed the call event on the primary device, becomes impossible without dedicated supplementary analysis. Studiawan et al. propose and implement custom plaso plugins for both SQLite and plist artifacts, demonstrating that the gap is technically bridgeable, but the existence of the gap in production tooling means that the majority of operational forensic timelines built from iOS images are incomplete with respect to call log data unless an iOS-specialized tool such as iLEAPP is deployed [13].

AlBreiki et al. similarly observe that automated forensic tools fail to recover artifacts from privacy-focused browsers (DuckDuckGo, Chrome incognito) that are only discoverable through manual database inspection [6]. Their finding that "incognito Chrome sessions and deleted Burner messages were only detectable via file system viewers and database browsers, not through automated reports" reinforces the general principle that automated forensic reporting systematically underdiscovers edge cases —

including, presumably, the synchronization provenance metadata that the present research seeks to identify.

The timestamp problem adds another dimension of complexity. A record in a call history database carries one or more timestamp fields, but these fields typically reflect the time recorded at the originating device, not the time of synchronization delivery to the secondary device. When a synchronized call record arrives on a secondary device five minutes after the call concluded, the timestamp in the database reflects the call's duration and completion time, not the synchronization event. The result is that a forensic timeline built from a secondary device's call history will correctly show the call occurring at its historical time, but will provide no indication that the record arrived later and via synchronization. Oestreicher documents an analogous problem for iCloud file synchronization, demonstrating that while file content is preserved intact, timestamp metadata is altered — and this is a finding that suggests the synchronization process itself may introduce timestamp artifacts that, if properly identified, could serve as indicators of propagation [11].

2.6 iCloud Synchronization: Architecture, Behavior, and Forensic Consequences

2.6.1 The iCloud Platform and Its Forensic History

Apple introduced iCloud in October 2011 as a successor to MobileMe, offering cloud storage, device synchronization, and backup services to users of iOS, macOS, and web-based clients [11]. Oestreicher notes that by June 2013 iCloud already had 320 million user accounts, hosting over 900 billion iMessages and 125 billion photo uploads, a scale that established iCloud as one of the most significant repositories of forensically relevant data in existence [11]. At the time of launch, iCloud supported synchronization of contacts, calendars, email, notes, Safari bookmarks, Photo Stream, and app data through a developer API. The scope of synchronization has expanded continuously with each iOS release.

From a forensic perspective, iCloud introduces a fundamental challenge that static, device-centric forensic analysis was not designed to address: the same data can exist in multiple states (locally resident, locally resident and synchronized, exclusively cloud-resident) on multiple devices at the same time, and the relationship between the device-level representation and the cloud-level representation may differ in ways that matter for evidence interpretation. Zottmann characterises this as an architectural consequence of Apple's decision to treat synchronization as a system service rather than an application feature, which is a decision that affects every iOS application that uses iCloud [14].

2.6.2 System-Controlled Synchronization: The Non-Deterministic Model

Zottmann provides the most comprehensive technical account of how iOS governs iCloud Drive synchronization, drawing directly on Apple's developer documentation and WWDC session materials [14]. The foundational architectural principle is that the iOS operating system — not applications and not users — decides when synchronization occurs. Apple's documentation explicitly states that "there is no API for app developers to force an iCloud Drive synchronization," and the system schedules synchronization opportunistically based on a complex set of factors including network state (with strong preference for Wi-Fi over cellular), battery level (sync pauses entirely when Low Power Mode activates at 20% battery), thermal state (sync pauses when the device overheats), iCloud storage availability, device storage availability, and machine-learned user behavior patterns.

The non-deterministic timing model has profound forensic consequences. When an artifact appears on a secondary device, the time elapsed between the originating event and the artifact's arrival on the secondary device is unknown and potentially variable from seconds to hours. Zottmann documents that while foreground app sync can complete "within seconds," background sync can take "minutes or even up to an hour" [14]. Moreover, the iOS system can learn user behavior patterns over time, with Apple's documentation noting that it can take "seven days for Apple's on-device ML to pick up that the user really wants to use the app", and this is a learning period during which synchronization frequency may be substantially lower than steady-state behavior. These variable delays mean that a secondary device's artifact database cannot be assumed to be

a current reflection of the primary device's state; rather, it reflects a snapshot whose currency is uncertain.

2.6.3 CloudKit: The Framework Underlying Synchronization

The majority of iOS system data services that synchronize across devices including Contacts, Calendar, Notes, Reminders, Safari bookmarks, iMessage, and potentially call logs use CloudKit as their underlying synchronization framework. Rosén, in a detailed technical analysis based on security research into CloudKit's implementation, describes the CloudKit data model: each iCloud account has access to containers (identified by reversed domain strings, e.g., iCloud.com.apple.notes), within which data is organized into environments (Development and Production), scopes (Private, Shared, and Public), zones, and records [15]. Records belong to record types and contain typed fields (INT, BOOLEAN, TEXT, BINARY, etc.). Access control in CloudKit is per-scope, with Private scope accessible only to the authenticated user and their own devices, Shared scope enabling data sharing between iCloud accounts, and Public scope providing broader access.

Young provides a developer-oriented account of the CloudKit API, describing how client applications use CKRecord objects to read and write synchronized data, and how the changeToken mechanism allows a client to retrieve only the records that have changed since its last synchronization operation [16]. The changeToken is a server-issued opaque value that the client caches locally; on the next synchronization, the client presents its cached token and receives a delta of changes since that token was issued. From a forensic perspective, the changeToken stored locally on an iOS device represents a documented artifact of synchronization state: if it can be located and decoded, it may indicate the timestamp of the device's last successful synchronization with the CloudKit server, providing a maximum bound on how stale the device's data is relative to the cloud state.

Matlock documents the practical realities of CloudKit synchronization from a developer perspective, describing a common failure mode where "data loss, mystery conflicts, and silent sync failures" occur without generating any log output [17]. Matlock's account of building a custom sync engine for a production iOS application, including an upload-first strategy, conflict resolution, and retry logic, illustrates the complexity that is underlying

what appears to users as seamless synchronization. For forensic investigators, the implication is that synchronization failures may leave a device's artifact database in a partially updated state where some records from a secondary device have arrived and others have not, creating a forensic picture that is neither entirely local nor entirely representative of the cloud state.

2.6.4 CloudKit Throttling and Its Forensic Implications

Zottmann details CloudKit's throttling behavior, noting that the system implements "30-second minimum intervals between rapid operations" as documented in Apple's Technical Note TN3162 [14]. Rosén confirms this behavior from the security research perspective, observing it as a constraint during his systematic analysis of CloudKit's access control mechanisms [15]. The throttling is protective of server infrastructure, but its forensic implication is that rapid successive events on a source device may be batched into a single synchronization transaction rather than being delivered individually to the receiving device. If a user makes three phone calls in rapid succession, those calls may arrive on the secondary device as a batch synchronized at the end of the throttling period, rather than individually as each call completes. A forensic timeline built from the secondary device's call history may show the calls at their correct historical times, but the synchronization batch delivery would be invisible — unless the investigator specifically examines synchronization event logs.

2.7 Cloud Forensics on iOS: Evidence Recovery and Metadata Integrity

2.7.1 iCloud Data Acquisition and the Timestamp Alteration Finding

Oestreicher remains the most directly relevant study to the metadata integrity question that is central to this research [11]. By conducting controlled experiments in which files of known content and metadata were uploaded to iCloud and then downloaded to an examination machine via Mac OS X's native synchronization mechanism, Oestreicher systematically compared MD5 hash values and timestamp metadata between originals and synchronized copies. His findings established that while MD5 hash values, which

reflect file content, were preserved intact through the synchronization process, timestamp metadata was altered. This finding is forensically significant in two respects: it confirms that the synchronization process is not metadata-neutral, and it suggests that synchronized copies carry detectable differences from originals that could, in principle, identify them as synchronized rather than locally generated.

Oestreicher situates this finding within a broader landscape of cloud storage research, noting that Quick and Choo, who examined Dropbox, Google Drive, and Microsoft SkyDrive (now OneDrive), reached the same conclusion: content fidelity does not guarantee metadata fidelity [11]. The pattern across multiple cloud providers is that timestamps are unreliable through the synchronization process, while content (and therefore content-based hash values) survives intact. The forensic consequence is that timestamp-based evidence of creation time, modification time, or access time may not survive synchronization accurately — and this is a problem that becomes even bigger when the investigator is attempting to use these very timestamps to reconstruct timelines or establish alibis.

2.7.2 Cloud Storage Residual Artifacts on iOS Devices

Huang et al. investigate what types of cloud storage artifacts can be recovered from iOS devices after users have interacted with third-party cloud services (Google Drive, Dropbox, OneDrive) using a controlled experimental design [18]. Their methodology — designing specific user scenarios, performing controlled interactions, and examining the resulting artifact landscape — is methodologically analogous to the experimental design that is required for the present research. Key findings include that viewing a file (as opposed to merely having it synchronized) dramatically increases the artifact recovery rate because file viewing triggers the creation of thumbnails and cache entries that viewing-only access does not create. The cello.db database for Google Drive, located in the application's data container, contains a metadata table with fields including file creation time, browsing time, modification time, and sync time — and this explicitly records the synchronization timestamp as a distinct field from the creation and modification timestamps.

The implication for the present research is significant. If third-party cloud applications on iOS explicitly record a sync_time field in their metadata databases, distinct from creation and modification times, then Apple's own native applications — which have even deeper integration with the iCloud synchronization framework — may similarly record synchronization event metadata in their own databases. The question is whether such fields exist in sms.db, CallHistory.storedata, or the Safari history database, and whether forensic tools currently surface them.

Huang et al. also demonstrate that the artifact landscape varies by device state: clearing the cache eliminates many artifacts that would otherwise persist, and there are no significant differences between artifacts recovered from a powered-on versus a powered-off device, since the relevant data is stored in non-volatile memory [18]. This finding is relevant for the present research in confirming that the targeted databases — which are persistent SQLite files rather than volatile cache — will be consistently recoverable regardless of the device power state at the time of seizure.

2.7.3 iOS Cloud App Forensics and the APT Security Taxonomy

D'Orazio and Choo present a study of 18 popular iOS cloud applications, demonstrating techniques to circumvent their security mechanisms — including certificate pinning and SSL/TLS validation — to enable forensic acquisition of network traffic in addition to local artifacts [4]. Their adversary model, which formalizes the capabilities available to a forensic investigator on a jailbroken iOS device, includes the ability to intercept, decrypt, and analyze the data exchanged between a cloud application and its server. This network-level analysis captures synchronization events in real time, providing a ground-truth record of what data was transmitted between the device and the cloud at what time — and this is an ideal reference dataset for validating inferences drawn from on-device database artifacts.

For the present research, D'Orazio and Choo's approach suggests a complementary methodology: by simultaneously monitoring network traffic during controlled synchronization experiments, it may be possible to correlate network-level synchronization events with the appearance of new records in the on-device databases, thereby establishing the precise timing and content of synchronization deliveries [4]. The

difference in timestamps between the network-captured synchronization event and the timestamps recorded in the resulting database records would directly measure the degree of timestamp alteration that is introduced by the synchronization process.

2.7.4 Forensic Analysis of Medical Device Applications as a Model

Grispos et al. present a methodologically relevant study of forensic artifacts generated by smartphone applications that interface with connected medical devices on Android and iOS [19]. While the specific artifact domain is distinct from the present research, their approach — systematically documenting the types and locations of forensic artifacts generated by specific application categories through controlled experiments — provides a model for the kind of ground-truth artifact cataloguing that is needed to identify synchronization provenance indicators. Their finding that smartphone applications can serve as "an alternative source of digital evidence" when the primary device is not directly accessible is analogous to the present research's goal of finding in-device indicators of multi-device activity.

2.7.5 Network Traffic as a Verification Method

Malik et al. demonstrate that iOS devices exhibit characteristic and measurable network behavior patterns in their ICMP and TCP/IP traffic that allow them to be identified and distinguished from Android and Windows devices using machine learning classifiers [20]. While their primary contribution is OS fingerprinting, the underlying observation — that iOS devices' power management and network throttling behavior produces distinctive inter-packet spacing signatures — illustrates the general principle that synchronization behavior may leave measurable network traces. For the present research, this suggests that network-level observation during controlled synchronization experiments could complement on-device database analysis, providing an independent channel of evidence about when synchronization events occurred.

2.8 Forensic Tool Capabilities and Their Limitations

2.8.1 Commercial Forensic Tools

The commercial forensic tool landscape for iOS devices is dominated by a small number of vendors whose products are validated for evidentiary use and widely deployed in law enforcement. AlBreiki et al. evaluate four tools — Cellebrite UFED, MOBILedit Forensic Express, FTK Imager, and DB Browser for SQLite — against iOS 17.3 using a publicly available forensic image [6]. Their findings reveal a consistent pattern: commercial automated tools (Cellebrite and MOBILedit) produce structured, human-readable reports of SMS, call logs, contacts, and browser history, but their coverage is partial and tool-specific. Artifacts that are recoverable by one tool may not appear in another tool's report. Manual examination using FTK Imager and DB Browser for SQLite consistently reveals artifacts that were missed by automated tools, including incognito browser records and application metadata not surfaced in automated reports.

Sibe and Alayegun use Oxygen Forensic Device Extractor on an iPhone XR running iOS 17.5 and find it capable of processing both iTunes backups and iCloud backup data, producing artifact reports covering contacts, SMS data, call history, media files, database content, and browser data [1]. Their work confirms that current commercial tools can acquire and report the primary artifact categories that are relevant to the present research, but it does not investigate whether these tools distinguish locally generated from synchronized records — a distinction that is absent from the reports of every tool documented in the literature.

Ntshangase et al. survey both commercial and open-source forensic tools for Android and iOS, cataloguing their strengths and limitations in a comparative framework [2]. Their survey underscores the diversity of the tooling landscape and the absence of standardized testing that would allow investigators to know with confidence which tool is most appropriate for a given artifact type on a given iOS version.

Neyaz et al. contribute a forensic evaluation of iMazing, which is a third-party iOS management tool that was not designed specifically for forensics [9]. Their analysis on an iPhone and iPad reveals that iMazing can expose artifacts including iCloud-synced SSID data — a category of artifact that carries inherent evidence of synchronization because SSID lists are maintained per-iCloud-account rather than per-device. This

finding suggests that non-forensic tools may expose synchronization-relevant data that dedicated forensic tools have not been designed to surface, and that a comprehensive investigation of synchronization indicators should cast a wide net across available acquisition methods.

2.8.2 Open-Source Tools: iLEAPP and iOS Artifact Coverage

The iOS-specialized open-source tool landscape is anchored by iLEAPP (iOS Logs, Events, And Plists Parser), which is a purpose-built parser for iOS artifact recovery that achieves substantially broader iOS coverage than general-purpose forensic timeline tools. Studiawan et al. document the comparative limitations of log2timeline plaso — demonstrating that it fails to parse CallHistory.storedata, CarPlay history, and several other iOS-specific artifact types from a public iOS 13.4.1 forensic image — and they identify iLEAPP as a more complete alternative that explicitly targets Apple-specific metadata and database formats [13]. Unlike plaso, which is architecture-agnostic and requires community-contributed plugins to support individual iOS artifact types, iLEAPP was designed from the beginning around iOS's artifact structure, with parsers for system logs, plist configuration files, application databases, and device metadata that generic timeline tools lack [13]. For research focused on iOS artifact provenance, iLEAPP's iOS-centric architecture makes it the appropriate open-source tool: it maximizes artifact visibility across the full range of iOS-specific data stores, including the supplementary logs (AppConduit, system plists) most likely to contain synchronization evidence.

While Studiawan et al. demonstrate that plaso's gaps can be partially bridged through custom plugin development, this requires ongoing community maintenance for each iOS version — and this is a fragility that iLEAPP's dedicated design avoids [13]. The proposed research therefore adopts iLEAPP as its primary open-source artifact parsing tool, deploying it alongside commercial tools (Cellebrite UFED, Oxygen Forensic Detective) and manual database inspection to ensure comprehensive artifact recovery.

The general pattern emerging from both commercial and open-source tool evaluation is that coverage of primary artifact content (message text, call numbers, URLs) is reasonably complete, while coverage of synchronization-related metadata — including fields that might identify artifact origin or synchronization timestamp — is systematically

absent from tool reports [6]. This absence is not necessarily because the data does not exist in the databases; it may reflect that no forensic tool has been specifically designed to look for and report it. iLEAPP's parser modules are open-source and extensible, making it feasible to add targeted parsers for newly identified synchronization provenance fields as part of the proposed research.

2.9 Multi-Device Artifact Propagation: Mapping the Research Gap

2.9.1 What the Literature Collectively Establishes

The literature reviewed in the preceding sections establishes a coherent and multi-layered foundation for the proposed research. Five bodies of established knowledge are directly relevant.

First, the primary artifact databases on iOS devices namely sms.db, CallHistory.storedata, and the Safari history database are well-documented in terms of location, general structure, and content [8], [13], [12]. Second, iCloud synchronization is an inherently non-deterministic, system-controlled process that introduces variable delays between event creation on a source device and artifact propagation to secondary devices, and that demonstrably alters at least some metadata during transmission [14], [11]. Third, CloudKit, which is the framework underlying most iOS data synchronization, uses a structured data model with typed records, access-controlled scopes, and a changeToken mechanism that potentially leaves recoverable traces of synchronization state on each device [15], [16]. Fourth, existing forensic tools both commercial and open-source do not systematically distinguish between locally generated and synchronized artifacts in their reporting, and their coverage of synchronization-related metadata is absent [6], [13], [2]. Fifth, there is methodological precedent for the kind of controlled experimental investigation that is needed to identify synchronization indicators, drawn from [11] and [18].

2.9.2 The Specific Empirical Gap

Despite this substantial foundation, no study reviewed has directly investigated whether the native iOS databases for Safari, call logs, or SMS messages contain metadata fields that encode the artifact's device origin or its synchronization pathway. Oestreicher documents timestamp alteration during iCloud file synchronization but does not examine the native application databases for these artifact categories [11]. Huang et al. identify synchronization metadata in third-party cloud storage application databases but not in Apple's own system databases [18]. Studiawan et al. parse CallHistory.storedata for timeline entries but do not investigate whether any field within the database distinguishes locally recorded calls from synchronized ones [13]. Forensafe enumerates sms.db fields including Message GUID, Chat GUID, and User Account GUID, identifiers that may carry device-level provenance information, but does not interpret their behavior under multi-device synchronization [12].

The AppConduit logs that are documented by Studiawan et al. as recording "interaction between iOS devices" represent an underexplored artifact category that may directly address the synchronization provenance question [13]. If AppConduit logging captures synchronization delivery events, noting which records arrived from external sources and when, it could serve as a synchronization event register against which database-level artifact appearance can be correlated. Similarly, Accounts3.sqlite, which records iCloud account configuration and potentially device identifiers, may contain metadata that connects individual devices to shared iCloud accounts in ways that could support multi-device artifact analysis.

The iMazing study by Neyaz et al. offers a tantalizing indirect indicator: the discovery of iCloud-synced SSID data as a distinct artifact category suggests that Apple's iCloud synchronization infrastructure propagates certain data categories in ways that carry implicit device-of-origin information [9]. If SSID data that a device has never directly encountered appears in its iCloud-synced dataset, that is a detectable marker of synchronization, and an analogous marker may exist within the call log, message, or browsing databases.

2.9.3 Consequences of the Gap for Forensic Practice

The practical consequences of this gap are significant. When an examiner finds a call record on a suspect's tablet showing a call to a victim's number at a critical time, and the same record appears on the suspect's phone, the examiner cannot currently determine which device placed the call and which device merely received the record through synchronization. The inability to make this determination can lead to three categories of error: timeline errors (if the time of synchronization delivery differs from the time of the original event), device misattribution (if the wrong device is identified as the one on which the relevant activity occurred), and evidentiary duplication (if the same event is counted twice as independent evidence because it appears on two devices).

Alblooshi et al. note in their comparative smartphone forensics study that "smartphone applications, built on complex architectures (APIs, servers, storage, and cloud, etc.), typically require the use of specialized tools to gather, segregate, assess, analyze, and report on data, making acquisition and analysis more complex and time-consuming" [21]. The synchronization provenance problem is precisely the kind of cloud-architectural complexity that requires specialized knowledge that is not yet codified in forensic practice.

2.9.4 Methodological Framework for the Proposed Research

The literature provides clear methodological guidance for addressing the gap. The proposed experimental design creates controlled, documented synchronization events across iOS devices that share an iCloud account, then examines the resulting artifact landscape across three acquisition methods (logical backup, file-system extraction, and iCloud backup download), directly adapting the approaches of Oestreicher [11] and Huang et al. [18]. The analytical method involves systematic comparison of database metadata fields before and after synchronization, with special attention to fields that differ between the originating device and the receiving device, following the SQLite-level examination approach of AlBreiki et al. [6] and Shimmi et al. [8]. The network traffic monitoring approach of D'Orazio and Choo offers a complementary validation pathway that can establish ground truth about synchronization timing [4].

The multi-tool verification framework recommended by AlBreiki et al., which combines automated tool extraction with manual database inspection, is adopted as the baseline analytical methodology, ensuring that no metadata field is overlooked because it falls outside the reporting scope of any single tool [6]. iLEAPP is designated as the primary open-source parser in this framework; its iOS-native architecture and extensible module system allow targeted parsing of supplementary artifacts (system plists, iCloud account databases) alongside the primary artifact databases, providing coverage that generic timeline tools cannot match [13]. The version-specific validation requirement identified by Shimmi et al., that schema evolution across iOS versions means identified indicators may behave differently across OS generations, is addressed through version-controlled experimental devices and explicit documentation of iOS version for every finding [8].

2.10 Conclusion

The literature reviewed in this survey charts the substantial maturation of iOS digital forensics as a discipline. From foundational studies of iTunes backup structure and iCloud acquisition methods through detailed investigations of SQLite schema evolution, forensic timeline construction, CloudKit synchronization architecture, and multi-tool artifact comparison, the field has developed both a comprehensive catalog of iOS artifact types and a sophisticated understanding of the forensic challenges that are introduced by iOS's security architecture and cloud integration.

What the literature has not yet addressed is the specific forensic problem of distinguishing locally generated artifacts from iCloud-synchronized ones within the native iOS system databases for Safari browsing history, call logs, and SMS messages. This gap is not peripheral to forensic practice: it lies at the intersection of the most evidentiary-rich artifact categories (messages, calls, and browsing) and the most pervasive features of modern iOS use (multi-device iCloud synchronization). The risk of misattributing device origin, reconstructing erroneous timelines, or double-counting events is present in any investigation involving an iOS device that participates in an iCloud account — which, in practice, is the majority of investigated iOS devices.

The proposed research responds to this gap through controlled experimental investigation that is methodologically grounded in the approaches established by Oestreicher [11], Huang et al. [18], Shimmi et al. [8], and AlBreiki et al. [6]. iLEAPP is adopted as the

primary open-source iOS artifact parser, chosen over general-purpose timeline tools because of its iOS-native architecture and broader coverage of the supplementary artifact categories — system logs, plist files, and iCloud account databases — that are most likely to encode synchronization provenance evidence [13]. By systematically examining the artifact metadata produced during documented synchronization events, across multiple acquisition methods and iOS versions, the research aims to identify validated forensic indicators — observable in the databases that are accessible to practitioners — that allow the distinction between locally generated and iCloud-synchronized records. The resulting indicators will address a practical need with direct implications for the reliability and accuracy of iOS-based forensic evidence.

3 Research Design and Methodology

3.1 Research Approach and Justification

This study adopts an experimental research strategy. Rather than observing artifacts from real-world criminal investigations, this research generates artifacts under controlled conditions where all relevant variables, namely the device on which an action was taken, the time of the action, and the synchronization state of each device, are predetermined and documented. This controlled approach allows a verified ground truth to be established, meaning the researcher knows with certainty which device produced each artifact before examining what the artifacts themselves reveal. Without this baseline, observed differences between artifact copies on multiple devices could not be reliably attributed to synchronization rather than to other factors.

An alternative to controlled experimentation would have been an empirical approach drawing on artifacts from live forensic casework, which would offer the advantage of ecological validity. However, this approach presents several fundamental limitations for the purposes of this study.

The most significant limitation is the absence of ground truth. Artifacts encountered in live investigations carry no verified record of which device originally generated a given entry. Without knowing the true origin of each artifact, it is impossible to determine whether any observed difference between records on two devices reflects synchronization behavior or some other cause entirely. The entire analytical framework of this research depends on the ability to compare artifact properties against a known and documented baseline, a condition that only a controlled experimental design can satisfy.

Beyond this, access to real forensic casework raises ethical and institutional barriers that fall outside the scope of a master's thesis. The sensitivity of criminal investigation data and the chain-of-custody obligations governing its use make empirical case data impractical for academic research. Finally, the research questions of this study are exploratory: the goal is to discover whether provenance indicators exist within artifact

metadata, and to characterize them precisely. This discovery-oriented objective is better served by a design that permits complete control over experimental conditions than by one that relies on opportunistic access to unpredictable real-world data.

For these reasons, the experimental approach is not merely a pragmatic compromise but the most scientifically appropriate method for answering the research questions of this study.

3.2 Forensic Test Plan

3.2.1 Device setup

The devices were selected to represent different major iOS generations for which a public jailbreak was available. This selection was intentional because the research required filesystem-level access to native iOS databases and metadata that may not consistently be available through ordinary logical acquisition. Using devices across iOS 12, iOS 15, and iOS 16 also allowed the study to observe whether provenance-related fields and synchronisation behaviour remained consistent across different database schema versions. At the same time, this design introduces a limitation: differences observed between devices may reflect iOS version differences as well as synchronisation behaviour.

Label	Model	Role	iOS Version	Notes
Device A	iPhone 8 Plus	iCloud-linked	iOS 16.7.12	Primary iCloud device
Device B	iPhone SE	iCloud-linked	iOS 15.8.3	Secondary iCloud device
Device C	iPhone 6	iCloud-linked	iOS 12.5.8	Third iCloud device; oldest schema version
External 1	iPhone XR	External	iOS 18.7.1	Not on shared iCloud account
External 2	iPhone SE 2020	External	iOS 26.2	Not on shared iCloud account

Table 1. List of devices

Device A, B, and C share a single Apple ID with iCloud sync enabled. External 1 and External 2 operate under separate Apple IDs.

Figure 1. Devices used for testing

3.2.2 Analysis tools

Tool	Purpose
iLEAPP	Initial artifact parse of full acquisition set
DB Browser for SQLite	Schema inspection and direct SQL queries
fqlite	SQLite write ahead log files inspection
Python / sqlite3	Bulk field comparison and CSV export where needed

Table 2. Analysis tools

3.2.3 Test Cases Description

Before running test cases perform baseline acquisition of Device A, B, and C

Test cases coverage:

- Calls: outgoing answered, outgoing missed, incoming answered, incoming missed, FaceTime Audio (answered), FaceTime Video (missed), same-account, external;
- SMS: inbound from each external device to each iCloud device, outbound from each iCloud device to external, same-account in both directions;
- iMessage: inbound from each external to each iCloud device, outbound from each iCloud device to each external, same-account forming a full triangle (A→B, B→C, C→A), read event on non-recipient synced device;

- Safari bookmark: each iCloud device adds a bookmark; two devices later delete their own bookmark to observe how deletion propagates via sync;
- Safari web browsing: browse 3-5 different web pages on each of the iCloud device

3.2.4 Test Cases: Calls

ID	Description	Action	Initiating Device	Target	Expected Sync To
TC-C01	Outgoing answered call — to external	Place cellular call. Answer on External 1. Duration ≥ 30 s.	Device A (8 Plus)	External 1 (XR)	Device B, Device C
TC-C02	Outgoing answered call — to external	Place cellular call. Answer on External 2. Duration ≥ 30 s.	Device B (SE)	External 2 (SE2020)	Device A, Device C
TC-C03	Outgoing missed call — to external	Place cellular call. Do not answer. End after 5 rings.	Device C (6)	External 1 (XR)	Device A, Device B
TC-C04	Incoming answered call — from external	Call from External 2. Answer on Device A. Duration ≥ 30 s.	External 2 (SE2020)	Device A (8 Plus)	Device B, Device C
TC-C05	Incoming missed call — from external	Call from External 1 to Device B. Do not answer.	External 1 (XR)	Device B (SE)	Device A, Device C
TC-C06	FaceTime Audio — same-account, answered	Place FaceTime Audio call. Answer on Device C. Duration ≥ 30 s.	Device A (8 Plus)	Device C (6)	Device B
TC-C07	FaceTime Video — same-account, missed	Place FaceTime Video call. Do not answer on Device A.	Device B (SE)	Device A (8 Plus)	

Table 3. Test Cases: Calls

3.2.5 Test Cases: SMS

ID	Description	Message Text	Initiating Device	Target	Expected Sync To
TC-S01	Inbound SMS — from external to Device A	`SMS_EXT1_TO_A`	External 1 (XR)	Device A (8 Plus)	Device B, Device C
TC-S02	Inbound SMS — from external to Device B	`SMS_EXT2_TO_B`	External 2 (SE2020)	Device B (SE)	Device A, Device C
TC-S03	Inbound SMS — from external to Device C	`SMS_EXT1_TO_C`	External 1 (XR)	Device C (6)	Device A, Device B
TC-S04	Outbound SMS — from Device A to external	`SMS_A_TO_EXT2`	Device A (8 Plus)	External 2 (SE2020)	Device B, Device C

TC-S05	Outbound SMS — from Device B to external	`SMS_B_TO_EXT1`	Device B (SE)	External 1 (XR)	Device A, Device C
TC-S06	Outbound SMS — from Device C to external	`SMS_C_TO_EXT2`	Device C (6)	External 2 (SE2020)	Device A, Device B
TC-S07	SMS between same-account devices — A to B	`SMS_A_TO_B_SELF`	Device A (8 Plus)	Device B (SE)	Device C
TC-S08	SMS between same-account devices — B to C	`SMS_B_TO_C_SELF`	Device B (SE)	Device C (6)	Device A

Table 4. Test Cases: SMS

3.2.6 Test Cases: iMessage

ID	Description	Message Text	Initiating Device	Target	Expected Sync To
TC-I01	Inbound iMessage — from external to Device A	`IMSG_EXT1_TO_A`	External 1 (XR)	Device A (8 Plus)	Device B, Device C
TC-I02	Inbound iMessage — from external to Device B	`IMSG_EXT2_TO_B`	External 2 (SE2020)	Device B (SE)	Device A, Device C
TC-I03	Inbound iMessage — from external to Device C	`IMSG_EXT1_TO_C`	External 1 (XR)	Device C (6)	Device A, Device B
TC-I04	Outbound iMessage — from Device A to external	`IMSG_A_TO_EXT2`	Device A (8 Plus)	External 2 (SE2020)	Device B, Device C
TC-I05	Outbound iMessage — from Device B to external	`IMSG_B_TO_EXT1`	Device B (SE)	External 1 (XR)	Device A, Device C
TC-I06	Outbound iMessage — from Device C to external	`IMSG_C_TO_EXT2`	Device C (6)	External 2 (SE2020)	Device A, Device B
TC-I07	iMessage same-account — Device A to Device B	`IMSG_A_TO_B_SELF`	Device A (8 Plus)	Device B (SE)	Device C
TC-I08	iMessage same-account — Device B to Device C	`IMSG_B_TO_C_SELF`	Device B (SE)	Device C (6)	Device A
TC-I09	iMessage same-account — Device C to Device A	`IMSG_C_TO_A_SELF`	Device C (6)	Device A (8 Plus)	Device B
TC-I10	Read synced iMessage on non-recipient device	read TC-I01 on Device B	Device B (SE)		Device A, Device C

Table 5. Test Cases: iMessage

3.2.7 Test Cases: Safari Bookmarks

ID	Description	Action	Initiating Device	Expected Sync To
TC-B01	Add bookmark — Device A	Bookmark `https://www.wikipedia.org`	Device A (8 Plus)	Device B, Device C
TC-B02	Add bookmark — Device B	Bookmark `https://www.reddit.com`	Device B (SE)	Device A, Device C
TC-B03	Add bookmark — Device C	Bookmark `https://www.github.com`	Device C (6)	Device A, Device B
TC-B04	Delete bookmark — Device A	Delete the "Wikipedia" bookmark added in TC-B01	Device A (8 Plus)	Device B, Device C
TC-B05	Delete bookmark — Device B	Delete the "Reddit" bookmark added in TC-B02	Device B (SE)	Device A, Device C

Table 6. Test Cases: Safari Bookmarks

3.2.8 Test Cases: Safari Web Browsing

ID	Description	URLs to Visit (in order)	Initiating Device	Expected Sync To
TC-W01	Browse 5 pages — Device A	1. bbc.com 2. wikipedia.org 3. github.com 4. stackoverflow.com 5. err.ee	Device A (8 Plus)	Device B, Device C
TC-W02	Browse 5 pages — Device B	1. reuters.com 2. reddit.com 3. medium.com 4. theguardian.com 5. linkedin.com	Device B (SE)	Device A, Device C
TC-W03	Browse 5 pages — Device C (3 overlap with Device A)	1. bbc.com 2. wikipedia.org 3. github.com 4. nytimes.com 5. youtube.com	Device C (6)	Device A, Device B

Table 7. Test Cases: Safari Browsing

Note on TC-W03: The first three URLs intentionally repeat Device A's TC-W01 visits. After sync, both Device A and Device C will hold history records for these URLs. The research question is whether any metadata field (visit count, device-of-origin, or sync timestamp) differs between the two devices' records for the same URL.

3.3 Forensic Acquisition Procedure

3.3.1 Acquisition Method Selection

Three acquisition pathways are available for iOS devices: logical backup via iTunes or Finder, jailbreak-enabled file-system extraction, and iCloud backup download. This research uses file-system extraction exclusively, for reasons specific to the research objective.

Logical backup is excluded because it does not provide access to the raw database files. As D'Orazio and Choo establish, the application data container is inaccessible through Apple's backup API on any device running iOS 8 or later [4], meaning a logical backup exposes only the fields that Apple elects to include. Since the objective of this research is to identify provenance indicators at the field level, including fields that standard tools do not currently surface, relying on logical backup would make any indicator located in an unexposed column invisible by design. File-system extraction is a strict superset of what logical backup can provide, so using it makes the logical acquisition method redundant for the purposes of this study.

iCloud backup is excluded on different grounds. Even where iCloud backup data is available to Apple and can be obtained through lawful process, access in practice depends on whether the account holder has enabled Advanced Data Protection, which encrypts backup content end-to-end with keys held only on the user's devices, making the data inaccessible to Apple and therefore inaccessible through legal compulsion directed at Apple [6]. In many investigations, the iCloud backup is simply not available. Beyond the access problem, iCloud backup snapshots are taken at intervals determined by Apple's infrastructure and do not correspond to the controlled acquisition points required by this research design. For both reasons, iCloud backup acquisition is outside the scope of this study.

3.3.2 Jailbreak and Device Preparation

File-system level access to iOS application data is a prerequisite for this research. As D'Orazio and Choo establish, the application data container, which houses the SQLite databases targeted by this study, is inaccessible to any forensic tool operating through standard interfaces on any device running iOS 8 or later [4]. Obtaining the full database

schemas, including columns not exposed through Apple's backup API, therefore requires root-level access to the device file system.

This access is obtained through a checkm8-based jailbreak. checkm8 is a bootrom exploit discovered by the independent security researcher axi0mX and released publicly on 27 September 2019 [23]. The exploit targets a use-after-free vulnerability in the USB Device Firmware Upgrade (DFU) mode's core implementation, allowing arbitrary shellcode to be uploaded and executed with full permissions within the BootROM [24]. The vulnerability affects Apple SoCs from A5 through A11, making it applicable to all three iCloud-linked devices used in this research: Device A (iPhone 8 Plus, A11 Bionic), Device B (iPhone SE first generation, A9), and Device C (iPhone 6, A8). For Device A and Device B, the jailbreak is applied using palera1n; for Device C, checkra1n is used. Both tools exploit the same underlying bootrom vulnerability and produce equivalent root access to the file system.

The forensic significance of checkm8 stems directly from its execution model. Because the exploit operates exclusively within the BootROM and executes shellcode in the device's SRAM, it does not write to the NAND flash at any point during the jailbreak process [23], [24]. Apple's boot architecture includes a trampoline stage that resets the device to a known state by clearing all memory from the previous boot stage before transferring control to iBoot [24]; the effect of checkm8 is therefore transient and is erased on each subsequent normal reboot. This distinguishes checkm8 categorically from software-based jailbreaks, which achieve persistent root access by patching the operating system image on disk. A checkm8-based jailbreak leaves the on-disk state of the device unchanged, preserving the integrity of the artifact databases that are the subject of this research. The exploit is also hardware-level and unpatchable through software updates, meaning the jailbreak capability is stable and reproducible across the full experimental timeline [23].

Before each acquisition session, the jailbreak is applied fresh to each device following a standardised procedure. The device is booted into DFU mode, the jailbreak tool is run from a dedicated acquisition machine, and the device is confirmed to have booted into jailbroken state by verifying SSH connectivity. No additional packages or modifications are installed on the device beyond those required to establish the SSH connection and permit file-system access.

3.3.3 File System Extraction

Once SSH access is confirmed, the contents of the user data partition are extracted using tar. The extraction command archives the entire /private/var/ directory, which contains all user data including the application data containers for the target databases. The archive is transferred to the acquisition machine over the local network using scp. A dedicated Wi-Fi network is used for all transfers to isolate acquisition traffic from any background network activity that could trigger iCloud synchronisation during the extraction window.

The target databases are additionally extracted individually after the full archive transfer. This direct extraction allows the databases to be made available for iLEAPP and DB Browser analysis without requiring the full archive to be unpacked first, and provides a secondary copy whose integrity can be verified independently. The individually extracted files and the copies extracted from the full archive are compared by hash to confirm they are identical.

Each extraction is assigned a unique identifier following the format [DeviceLabel]_[AcquisitionPoint]_[Date]_[Time], for example DevA_Baseline_20260220_0912. All extracted files are stored in a structured directory hierarchy on the acquisition machine and are not modified after extraction.

3.3.4 Acquisition Timeline

Three acquisition points are defined for each device, producing a minimum of nine total acquisitions across Device A, B, and C.

The baseline acquisition is performed before any test case activity begins. Both iCloud-linked and external devices are in their cleared state. The baseline acquisition documents this empty state and establishes the reference hash values for all target database files. Any file that is present in a post-test acquisition but absent or different from the baseline can be attributed to the test procedure.

The post-test acquisition is performed immediately after all test cases have been executed on the devices. At this point, Device A, B, and C each contain locally generated artifacts from whichever test cases they initiated, but iCloud synchronisation may not yet have delivered all records to all devices. The post-test acquisition therefore captures a snapshot of each device's state before full synchronisation has occurred. Devices are placed in

aircraft mode immediately after the final test case action and before the post-test acquisition begins, to prevent any further synchronisation from altering the databases during the extraction window.

The post-sync acquisition is performed after a waiting period of 4 hours following the post-test extraction. Devices are returned to normal network connectivity, iCloud synchronisation is allowed to complete. This can be confirmed by monitoring network activity until traffic on the dedicated WiFi network ceases. Aircraft mode is re-engaged before the extraction begins. This acquisition captures the fully synchronised state and is the primary source for the field-level comparison analysis.

SHA-256 hash values are computed for every file in each acquisition immediately after extraction, using a custom written hashing script that produces a manifest file listing the path and hash of every file in the archive. This manifest is the primary instrument for identifying which files were modified between acquisition points.

By comparing the baseline manifest against the post-test manifest, every file that was created, modified, or deleted during the test procedure can be identified precisely. The comparison is performed using a diff of the two manifests sorted by file path. Files present in the post-test manifest but absent from the baseline are newly created; files present in both but with differing hashes were modified; files present in the baseline but absent from the post-test manifest were deleted. This approach focuses the subsequent manual analysis on only the files that actually changed, rather than requiring the researcher to inspect the entire file system, and provides a documented audit trail of every modification produced by the test cases.

The same comparison is then performed between the post-test manifest and the post-sync manifest to identify which additional files were created or modified by iCloud synchronisation between the two acquisition points. The intersection of the changed-file list with the known locations of the target databases (sms.db, CallHistory.storedata, History.db, Bookmarks.db) produces the precise set of database files that require detailed analysis.

3.4 Artifact Analysis Methodology

3.4.1 Initial Parse with iLEAPP

Each post-test and post-sync acquisition is processed with iLEAPP as the first analytical step. iLEAPP is run against the full extracted archive, producing a structured HTML report covering all artifact categories it supports. As Studiawan et al. establish, iLEAPP achieves substantially broader coverage of iOS-specific artifact types than general-purpose forensic timeline tools, with dedicated parsers for sms.db, CallHistory.storedata, Safari databases, system logs, and supplementary plist files [13]. The iLEAPP report serves two purposes in this research: it provides an initial confirmation that the expected test case artifacts are present and recoverable, and it establishes which metadata fields iLEAPP currently surfaces to a forensic examiner forming the baseline against which any newly identified provenance fields can be compared.

The iLEAPP outputs for Device A, B, and C are reviewed in parallel, with the report for each device examined for the presence of records corresponding to each test case. Correspondence is confirmed using the message text identifiers embedded in every test case.

3.4.2 Marker-Based Record Identification

Every test case in the plan was designed with a unique, machine-searchable text string for example, SMS_EXT1_TO_A for the inbound SMS from External 1 to Device A, or IMSG_A_TO_B_SELF for the same-account iMessage from Device A to Device B. These strings serve as markers that allow any individual test case record to be located unambiguously within a database, regardless of which device is being examined or which acquisition point is under analysis.

Record identification proceeds by executing targeted SQL queries against each target database using DB Browser for SQLite. Once a record is located on the originating device, the equivalent record is located on every other device in the acquisition set using the same marker. This produces a matched set of database rows which forms the unit of comparison for the field-level analysis.

3.4.3 Manual Schema and Metadata Inspection

The iLEAPP report provides a parsed view of each database's content, but it does not expose the full database schema. The second analytical layer therefore uses DB Browser for SQLite or fqlite to inspect each target database directly, examining every table and every column present in the schema, including columns that iLEAPP does not include in its parsed output. Where column values are stored as binary-encoded property list (bplist) data, a format that DB Browser renders as a raw byte sequence, custom Python scripts are used to deserialise the values, making the encoded fields available for inspection and comparison. The same approach is applied to standalone plist files in the acquisition where their content is relevant to provenance analysis.

For each matched record pair, every column value in the originating device's record is recorded alongside the corresponding column value in the receiving device's record. This produces a per-field comparison matrix for each test case. Fields that are absent from the schema of a lower iOS version are noted as version-specific, with the implication that any provenance indicator found in that field cannot be applied universally across all iOS versions.

SQLite's Write-Ahead Logging (WAL) mode commits new writes to a separate -wal companion file before merging them into the main database at a checkpoint event. Until that checkpoint occurs, recently written records exist in the WAL but not in the main file. Standard SQLite tools present the database as it exists after the most recent checkpoint, meaning records written to the WAL after that point are accessible through normal queries but earlier uncommitted frames are not. In addition to querying the checkpointed database state, fqlite is applied to relevant -wal file in the acquisition set. fqlite reads WAL files frame by frame outside of SQLite's standard I/O layer, making it possible to recover records from frames not yet committed to the main database file. Any records recovered through this method that are not present in the checkpointed database state are documented separately, as they could be invisible to analysis relying solely on the standard query path.

3.4.4 Classification of Provenance Indicators

Because each artifact database has its own distinct schema with no shared fields, classification is necessarily performed within each artifact category independently. No

assumption is made that an indicator found in one database will have an equivalent in another.

Within each artifact category, candidate indicators are classified along two dimensions: reliability across test cases, and scope across iOS versions.

A reliable indicator is a field that consistently differs between locally generated and synchronised records across all test cases within its artifact category, regardless of which device is the originating device and regardless of the specific content of the message, call, or browsing event. Reliability is assessed by examining whether the direction and magnitude of the difference holds across every test case in the category. A reliable indicator is the most operationally useful finding within its artifact type, as its interpretation does not depend on the specific circumstances of a given test.

A version-specific indicator is a field that carries provenance information on devices running one iOS major version but is absent from the schema, or carries a different meaning, on devices running another. As Shimmi et al. demonstrate, major iOS releases can substantially reorganise database schemas [8]; because Device A, B, and C run iOS 16.7.12, iOS 15.8.3, and iOS 12.5.8 respectively, version-specific indicators are anticipated. Such indicators must be reported with explicit iOS version qualifiers and cannot be generalised to acquisitions from devices outside the tested version range.

These two dimensions are not mutually exclusive: an indicator may be both reliable across all test cases within its category and version-specific in its iOS scope. Each finding is therefore described along both dimensions rather than assigned to a single class.

For each identified indicator, the following is documented: the database and table in which the field was found; the field name and SQLite data type; the value observed in the originating device's record; the value observed in the receiving device's record; whether the difference was consistent across all test cases in the category; the iOS versions on which the finding was confirmed; and whether the field is currently surfaced by iLEAPP or is visible only through direct database inspection. This last point is particularly relevant for the practical contribution of the research: an indicator that iLEAPP already reports provides no new operational value, whereas an indicator only accessible through manual schema inspection or a custom query highlights a gap in current tooling that the research directly addresses.

4 Forensic Artifact Catalog

A full filesystem dump of an iOS device contains data that may have originated on that device, on a linked device, or on both. When an investigator receives a single image, there is no immediate indication that other devices exist or that any of the captured data arrived via iCloud synchronisation rather than local user activity. Two questions must therefore be answered. First: does the device belong to an iCloud account with other enrolled devices? Second: for each artifact of interest, was the data generated on the dumped device or synchronised from a linked one?

Firstly the writer establishes whether other devices are present, using two files whose contents record the iCloud account's full device membership. These files were identified through a dedicated test on Device A. Because the jailbreak used is untethered, the device loses the jailbroken state on reboot and must be re-jailbroken before a filesystem dump can be acquired. Device A was rebooted, re-jailbroken in aircraft mode, and a first filesystem dump was taken. Aircraft mode was then disabled, the Settings application was opened, and the iCloud account's linked devices list was viewed. This action causes iOS to query iCloud for the current device roster. A second filesystem dump was taken immediately afterwards. Comparing the file modification timestamps and file hashes between the two dumps isolated the files that iOS wrote in response to the linked devices query.

The remainder of the chapter addresses the second question by cataloguing the four target artifact types documenting each database's structure, schema, and field-level properties. Chapter 5 then applies those findings comparatively across all three devices to determine which fields carry stable cross-device values, which are rewritten locally on receipt, and what combinations of field values allow an examiner to distinguish a locally generated record from a synchronised one.

4.1 Establishing Multi-Device Context

4.1.1 Find My Friends Daemon Plist

com.apple.icloud.fmfd.notbackedup.plist is written by the Find My Friends daemon (fmfd) to private/var/mobile/Library/Preferences/ and caches the set of iOS devices registered to the iCloud account. The notbackedup component of the filename reflects an explicit exclusion from iCloud Backup. The exclusion means that the file is rebuilt from the account's server state on demand and is not preserved across a restore. All three devices carry this file across all acquisitions.

The file uses two levels of encoding. The outer bplist is a single-key dictionary (kFMFDStoredDataKey) whose value is a raw byte sequence. Decoding that byte sequence reveals an NSKeyedArchiver-formatted dictionary, itself containing the structured account data. The decoded dictionary carries six keys. Two are relevant to device identity: kFMFDMeDeviceKey holds an FMFDevice object representing the device that wrote the file. kFMFDDevicesListKey holds an NSMutableSet of FMFDevice objects, one per registered account member. Each FMFDevice carries four fields relevant to forensic analysis:

- deviceId which appears to the Find My network device id;
- idsDeviceId, an uppercase UUID from Apple's Identity Service (IDS). IDS is also the same service that underlies iMessage and FaceTime routing;
- deviceName, the user-assigned device name;
- isThisDevice, a boolean identifying which entry belongs to the device holding the file.

All three devices carry six FMFDevice entries in kFMFDDevicesListKey. The device IDs and names are identical across all three databases. Each database's isThisDevice flag points to the correct entry for that device

4.1.2 iCloud Keychain Peer Trust Database

com.apple.security.keychain-defaultContext.TrustedPeersHelper.db located in /private/var/Keychains is the local store for iOS's iCloud Keychain peer trust infrastructure. The file is written by the com.apple.security.cuttlefish XPC service, which manages the cryptographic peer relationships that allow devices sharing an Apple ID to synchronise end-to-end encrypted data, including iCloud Keychain credentials, Health

data, and HomeKit configuration, without those secrets being accessible to Apple’s servers. Each participating device maintains its own copy of the trust database, recording the cryptographic identities of every other currently enrolled device [25].

Device A and Device B carry this database, device C does not. The database is present only on devices running iOS 13 or later. Device C runs iOS 12.5.8 and uses the earlier iCloud Keychain synchronisation mechanism, which does not produce this file. The database is using Apple's standard Z prefix naming convention. The relevant tables are ZPEER, ZMACHINE, ZESCROWCLIENTMETADATA, ZESCROWMETADATA.

ZPEER holds one row per enrolled peer device, each carrying a SHA-256-based peer ID in field ZPEERID and a ZISEGOPEER flag field marking which row belongs to the device holding the database.

Device model inventory can be done via ZESCROWCLIENTMETADATA. ZMACHINE holds the machine IDs of devices enrolled under this account in Apple's Identity Management Service. Cross-referencing with ZESCROWCLIENTMETADATA.ZDEVICEMID resolves each ID to a hardware model. Both Device A and Device B list five distinct machine IDs: an iPhone 6 (iPhone7,2), an iPhone SE 1st generation (iPhone8,4), an iPhone 7 (iPhone9,3), a MacBook Pro 2017 (MacBookPro14,2), and an iPad mini 2 (iPad4,5). The iPhone 6 is Device C and the iPhone SE 1st generation is Device B. Device C's hardware model appears in acquisitions even though Device C cannot hold this database itself.

ZPEERINFO field in ZESCROWMETADATA table is a DER-encoded binary blob. Decoding it reveals keys OSVersion, SerialNumber, ModelName, and ComputerName. After deduplicsting the list five devices remain and the result is identical across both databases. Device C (iPhone 6, iPhone7,2) appears with serial number FFNWN32AHXR5 and OS build 16H81 (iOS 12.5.8), matching the version confirmed at acquisition.

4.2 Safari Artifacts

4.2.1 History.db

Safari's browsing history is stored in a SQLite database named History.db. On iOS 13 and later (Devices A and B), the database is located at /var/mobile/Library/Safari/History.db. On iOS 12 (Device C), it resides within the Safari application container at /var/mobile/Containers/Data/Application/<UUID>/Library/Safari/History.db. The database participates in iCloud Safari history synchronisation through CloudKit.

The History.db database contains eight tables on Devices A and B (iOS 16 and 15): history_items, history_visits, history_tombstones, history_client_versions, history_event_listeners, history_events, history_tags, history_items_to_tags, metadata, and sqlite_sequence. Device C (iOS 12) contains the same set minus history_tags and history_items_to_tags, which support Safari's tab group and tagging features introduced in later iOS versions.

The history_items table stores unique URLs with aggregate visit statistics. Devices A and B contain 10 columns including status_code (storing HTTP response codes), while Device C contains 9 columns without this field. All other columns are shared: id, url, domain_expansion, visit_count, daily_visit_counts, weekly_visit_counts, autocomplete_triggers, should_recompute_derived_visit_counts, and visit_count_score.

The history_visits table stores individual visit events and is identical across all three iOS versions, containing 13 columns: id, history_item, visit_time, title, load_successful, http_non_get, synthesized, redirect_source, redirect_destination, origin, generation, attributes, and score.

The following fields are of particular relevance in a multi-device forensic examination:

origin. An integer field in history_visits that indicates whether a visit was performed locally or received via iCloud sync. A value of 0 indicates a visit originating from direct browsing on the device holding the record. A value of 1 indicates a visit received via iCloud Safari history synchronisation from another device. This field serves as the primary provenance indicator for Safari history records, analogous in function to the ZLOCALPARTICIPANTUUID field in call records.

visit_time. A Mac Absolute Time timestamp (seconds since 2001-01-01 00:00:00 UTC) recording when the page visit occurred. For synced records, the original visit timestamp is preserved; the forensic significance of timestamp behaviour during sync is examined in Section 5.2.1.

redirect_source and redirect_destination. Integer fields that reference other visit id values to form redirect chains. When a user navigates to an HTTP URL that redirects to HTTPS, Safari records both visits: the HTTP visit carries a redirect_destination value pointing to the HTTPS visit's id, and the HTTPS visit carries a redirect_source value pointing back to the HTTP visit. These fields allow reconstruction of the full navigation path, including intermediate redirects. The presence of an HTTP-to-HTTPS redirect pair provides forensic evidence that the user typed the URL without a scheme prefix, triggering the initial HTTP request.

generation. An integer tracking sync generation cycles. Each device maintains its own generation counter, which increments when new records are created or received. The counter is per-device and is not preserved during sync; a record received from another device is assigned the receiving device's current generation number, not the source device's.

attributes. An integer encoding visit properties. Observed values include: 0 (direct navigation), 2 (sub-resource or redirect load), 3, and 4 (back/forward navigation). The meaning of value 3 is not independently determinable from the test data. This field is not perfectly preserved during sync; the forensic implications of this are discussed in Section 5.2.1.

history_item. A foreign key referencing the id column in history_items, linking each visit to its corresponding URL record. Since history_items.id is assigned locally by each device's SQLite autoincrement, the same URL may have different history_item values on different devices.

4.2.2 Bookmarks.db

Bookmarks.db stores Safari bookmarks, reading list items, and the favorites bar. All three devices carry the database at private/var/mobile/Library/Safari/Bookmarks.db. Large WAL files are present at extraction3: 1.4 MB on Device A, 1.1 MB on Device B, and 338

KB on Device C. The main database file is 4 KB on Devices A and B and 116 KB on Device C, indicating that most of Device A's and Device B's current bookmark state resides in the WAL and had not been checkpointed at acquisition time.

The database has six tables on all three devices: bookmarks, bookmark_title_words, folder_ancestors, generations, sqlite_sequence, and sync_properties. Device A additionally carries participant_presence, an empty table supporting the Shared Tab Groups feature introduced in iOS 16. The bookmarks table holds all records: bookmarks, reading list items, and folder nodes. Row counts at extraction3 are 18 on Device A, 17 on Device B, and 19 on Device C. The columns differ by iOS version. Device C (iOS 12) has 31 columns. Device B (iOS 15) adds date_closed and last_selected_child, bringing the count to 32. Device A (iOS 16) further adds subtype, giving 33 columns.

The columns most relevant to provenance analysis are the following: id is the SQLite ROWID, it differs for the same bookmark across devices meaning it is assigned locally. type distinguishes bookmark records (0) from folder nodes (1). syncable is 0 for local-only records and 1 for records eligible for iCloud sync. server_id is the sync server identifier assigned by iCloud. The server_id is stable across devices and can be used as the cross-device bookmark identifier. external_uuid is a per-device UUID assigned locally on creation; it differs for the same bookmark on different devices meaning it is not a cross-device identifier. locally_added is 0 for all records in the test dataset, indicating that the sync infrastructure treats every record as server-originated rather than locally created on that device. The added and last_modified columns carry Mac Absolute Time values; both are zero (2001-01-01 00:00:00 UTC) across all records on all three devices

The sync_data column in bookmarks table carries a binary blob for every synced record (syncable = 1). The blob is a two-layer NSKeyedArchiver encoding. Decoding the outer layer yields a root dictionary whose EncodedCKRecordSystemFields key holds a second NSKeyedArchiver-encoded blob. Decoding that inner blob reveals fields that might become forensically relevant. ModifiedByDevice is a plain string holding the user-assigned name of the device that last wrote the record's content to iCloud. RecordCtime and RecordMtime are NSDate values (seconds since 2001-01-01) recording the CloudKit record creation and most-recent-modification times. Unlike the added and last_modified columns in bookmarks table, these values are non-zero and represent real events. ETag is

a hex string incremented server-side on every write, providing a monotonic version sequence for the record. RecordType is the string BookmarkList for all records in the test dataset. KnownToServer is a boolean, True for every synced record. RecordID is the CloudKit record identifier, corresponding to the server_id column.

4.3 Call Log Artifacts

The call history on iOS devices is stored in a CoreData-backed SQLite database located at /var/mobile/Library/CallHistoryDB/CallHistory.storedata. CoreData is Apple's object-graph persistence framework. Rather than writing SQL directly, iOS applications define data models as object classes and CoreData manages the underlying SQLite storage automatically. A consequence of this is that the table and column names in the database are not chosen by the application developer but generated by CoreData using its own naming convention: tables and columns are prefixed with Z (e.g., ZCALLRECORD, ZADDRESS), and internal bookkeeping columns such as Z_PK, Z_ENT, and Z_OPT are added automatically. [26] The database is accompanied by Write-Ahead Logging (WAL) journal files (`.storedata-wal` and `.storedata-shm`) which may contain uncommitted transactions of forensic relevance.
Call records are synchronised across devices through Apple's CloudKit framework. The sync process uses the ZUNIQUE_ID field as the record identifier in the CloudKit container. When a call occurs on any device associated with the iCloud account, a CloudKit record is created and pushed to all other devices in the sync circle. Each receiving device inserts the record into its local CallHistory.storedata database. The metadata table in each device's database tracks the sync state. The cached_sync_circle_size value was observed as 3 on all three devices, confirming that the iCloud account recognised three devices in its sync circle during the test period.

Analysis of the records revealed that the sync direction was not uniform. Of the 27 unique call records observed across all devices, 8 appeared on all three devices, 11 appeared on exactly two devices (all on the A+B pair), and 8 were unique to a single device (2 on Device A only, 6 on Device C only). The concentration of two-device records on the A+B pair and the larger number of unique records on Device C suggest that Device C's older

iOS version (12.5.8) participated in sync less completely than the newer devices.

4.3.1 Database Schema

The CallHistory.storedata database contains seven tables across all three iOS versions examined. These are: ZCALLDBPROPERTIES, ZCALLRECORD, ZHANDLE, Z_2REMOTEPARTICIPANTHANDLES, Z_METADATA, Z_MODELCACHE, and Z_PRIMARYKEY. The primary table of forensic interest is ZCALLRECORD, which stores individual call records along with their metadata.

The number of columns in ZCALLRECORD varies by iOS version. Device A (iOS 16.7.12) contains 27 columns, Device B (iOS 15.8.3) contains 26 columns, and Device C (iOS 12.5.8) contains 23 columns. The following table summarises the schema differences.

Column	iOS 16 (A)	iOS 15 (B)	iOS 12 (C)
Z_PK	Y	Y	Y
Z_ENT	Y	Y	Y
Z_OPT	Y	Y	Y
ZANSWERED	Y	Y	Y
ZCALL_CATEGORY	Y	Y	Y
ZCALLTYPE	Y	Y	Y
ZDISCONNECTED_CAUSE	Y	Y	Y
ZFACE_TIME_DATA	Y	Y	Y
ZHANDLE_TYPE	Y	Y	Y
ZNUMBER_AVAILABILITY	Y	Y	Y
ZORIGINATED	Y	Y	Y
ZREAD	Y	Y	Y
ZDATE	Y	Y	Y
ZDURATION	Y	Y	Y
ZADDRESS	Y	Y	Y
ZISO_COUNTRY_CODE	Y	Y	Y
ZLOCATION	Y	Y	Y
ZNAME	Y	Y	Y
ZSERVICE_PROVIDER	Y	Y	Y
ZUNIQUE_ID	Y	Y	Y
ZLOCALPARTICIPANTUUID	Y	Y	Y
ZOUTGOINGLOCALPARTICIPANTUUID	Y	Y	Y
ZFILTERED_OUT_REASON	Y	Y	N
ZJUNKCONFIDENCE	Y	Y	N
ZVERIFICATIONSTATUS	Y	N	N

ZIMAGEURL	Y	N	N
ZPARTICIPANTGROUPUUID	Y	N	N
ZLOCAL_ADDRESS	N	N	Y

Table 8. CallHistory.storedata schema evolution

The schema evolution reflects Apple's incremental additions to the call record structure. ZFILTERED_OUT_REASON and ZJUNKCONFIDENCE were introduced in iOS 13 as part of Apple's call-blocking and spam-identification framework. ZVERIFICATIONSTATUS supports the STIR/SHAKEN caller ID verification standard, which Apple first implemented in iOS 13 [27] the column is present on Devices A and B but absent from Device C (iOS 12). ZIMAGEURL and ZPARTICIPANTGROUPUUID, also iOS 16 additions, relate to group FaceTime functionality and contact image handling. Notably, Device C (iOS 12) retains a ZLOCAL_ADDRESS column that was removed in later versions; this column stores the local device's phone number for the call, providing an explicit provenance marker absent from newer schema versions.

4.3.2 Key Forensic Fields

The following fields are of particular relevance in a multi-device forensic examination:

ZUNIQUE_ID. A UUID string that serves as the cross-device identifier for a call record. When a call is synced via iCloud (CloudKit), the same ZUNIQUE_ID appears on all devices that receive the synced record. This field is the primary key for correlating call records across devices and establishing whether a record originated locally or was propagated through iCloud sync.

ZLOCALPARTICIPANTUUID. A 16-byte binary blob storing a UUID that identifies the local SIM or phone line that participated in the call. The field is stored as raw bytes; converting to hexadecimal yields a 32-character string that can be formatted as a standard UUID (e.g., bytes 3f 0e 0c 07 d2 25 49 e8 9b bb db 40 2b ee 97 92 become 3f0e0c07-d225-49e8-9bbb-db402bee9792). The mapping between UUID values and phone lines is established through the test cases, where the initiating device is known by design:

TC-C01 (outgoing call from Device A to External 1, record 2C22C6B3): on Device A, ZLOCALPARTICIPANTUUID = 5e7ef9e570ad44feaf45e94c4ccf7f70. Since TC-C01 was executed on Device A (+37253952660), this UUID maps to Device A's phone line.

TC-C02 (outgoing call from Device B to External 2, record 3550220A): on Device B, ZLOCALPARTICIPANTUUID = c763999632b0499191d26f2f07c8db92. Since TC-C02 was executed on Device B (+37254370088), this UUID maps to Device B's phone line.

TC-C03 (outgoing call from Device C to External 1, records 2D0E7860, 27219F60, B2A56A28): on Device C, all carry ZLOCALPARTICIPANTUUID = 3f0e0c07d22549e89bbbdb402bee9792. Since TC-C03 was executed on Device C (+37253507826), this UUID maps to Device C's phone line.

Once established, these mappings can be used to determine provenance of any record in the dataset. On Device A, records synced from Device C carry Device C's UUID (3f0e0c07d22549e89bbbdb402bee9792), preserving the originating phone line across sync. On Device B (iOS 15), the field is null for all synced records and populated only for locally originated calls, providing a binary local-versus-synced classifier. On Device C local records carry its own UUID (3f0e0c07d22549e89bbbdb402bee9792) while some synced records carry Device A's UUID (5e7ef9e570ad44feaf45e94c4ccf7f70) records synced from Device B show null.

ZDISCONNECTED_CAUSE. An integer indicating why the call ended. On locally originated records, this field carries meaningful values (e.g., 6 for normal hangup, 15 for no answer, 0 for unanswered incoming). On Device B, synced records consistently show null for this field, while local records show an integer value. This null pattern constitutes a second provenance indicator on Device B.

ZFACE_TIME_DATA. On Device B, this field is 0 for synced records and null for locally originated telephony calls. This inverted null/zero pattern (relative to ZDISCONNECTED_CAUSE) provides a third provenance indicator that, in combination with the other two, allows robust classification of synced versus local records.

ZADDRESS. The phone number or contact address for the call. On Device A, this field contains plaintext phone numbers for locally originated calls (e.g., 53368505, +37253368505). On Devices B and C, synced records store the address as hex-encoded byte strings (e.g., 2b3337323533333638353035 decodes to +37253368505). This encoding difference arises from how CoreData serialises the string during CloudKit sync and provides an additional forensic indicator of record provenance.

ZLOCATION. A text field for geographic location of the caller. This field is populated exclusively on Device C (iOS 12) with values such as "Estonia" and "Lithuania" for incoming calls. Devices A and B show null for this field across all records. The presence of location data on Device C and its absence elsewhere suggests this field was populated by the local telephony stack on iOS 12 and was not included in the sync payload.

ZDATE. A Mac Absolute Time timestamp (seconds since 2001-01-01 00:00:00 UTC). For synced records, the timestamp values are nearly identical across devices but exhibit minor variations of up to approximately 5 seconds, which likely reflect differences in when each device processed the incoming sync notification rather than the actual call time. The present test plan does not include a test case that would isolate whether the timestamp on a receiving device reflects the original call time or the sync receipt time (e.g., by placing a call while one device is offline and observing the timestamp upon reconnection). Such a test case would be a valuable addition in future work.

4.4 SMS/iMessage Artifacts

sms.db located at private/var/mobile/Library/SMS/ on all three devices contains twelve tables on Device A and Device B and ten on Device C. The core tables for provenance analysis are message (one row per message event), handle (phone numbers and email addresses), chat (conversation threads), and three join tables: chat_handle_join, chat_message_join, and message_attachment_join. Device A additionally carries chat_recoverable_message_join and recoverable_message_part, which support the iOS 16 message recall and editing features. All three devices contain deleted_messages, sync_deleted_messages, sync_deleted_chats, sync_deleted_attachments, message_processing_task, and kvtable. Device C (iOS 12.5.8) carries two companion databases absent from the other devices: chatRenderMetaData.db and replayDB.db in the same directory. Both contain only housekeeping tables, no content relevant to provenance analysis was found in either.

Sixty-five columns appear in the message table on all three devices. The differences fall into two groups: iOS 16 additions on Device A, and legacy CloudKit columns on Device C from the iOS 12 synchronisation architecture.

Device A (iOS 16.7.12) carries six columns absent from the other devices: date_retracted and date_edited for message recall and in-place editing timestamps; part_count for message part count; was_detonated for expirable messages; and is_stewie and is_kt_verified, whose functions were not publicly documented at the time of examination. The chat table on Device A also carries is_recovered.

Device C (iOS 12.5.8) carries three columns absent from Devices A and B: sr_ck_sync_state, sr_ck_record_id, and sr_ck_record_change_tag in message, and sr_ck_sync_state, sr_cloudkit_record_id, and sr_server_change_token in chat. The sr_ prefix likely marks a secondary CloudKit sync record from the iOS 12 architecture. All sr_ fields were empty or zero across all records examined; they were not active in the test environment.

Ten columns appear on Devices A and B but not on Device C, reflecting features added after iOS 12: sort_id, reply_to_guid, is_spam, has_unseen_mention, thread_originator_guid, thread_originator_part, syndication_ranges, synced_syndication_ranges, was_delivered_quietly, and did_notify_recipient. Any provenance indicator in these fields cannot be applied to iOS 12 acquisitions.

4.4.1 Key Forensic Fields

guid. A globally unique identifier string for each message record. In the test dataset, all GUIDs are standard UUID strings (e.g., F2E85E64-F1AC-BA7E-ACC8-988A538B44A4). This field serves as the cross-device correlation key, analogous to ZUNIQUE_ID in call records. When a message is synced via CloudKit, the same guid appears on all devices that receive the synced record.

destination_caller_id. Records which phone number or Apple ID email address was used as the local identity for a given message. This field stores the phone line or Apple ID that the originating device used to send or receive the message (e.g., +37253952660 or ray.dio@icloud.com).

ck_sync_state. An integer indicating the CloudKit synchronisation status of a message record. Observed values are 0 and 1.

ck_record_id. A hex string identifying the CloudKit record corresponding to a message. When populated, this value is shared across devices that received the same message from

CloudKit, providing a secondary cross-device correlation key independent of the guid field.

date. A nanosecond-precision Mac Absolute Time timestamp (nanoseconds since 2001-01-01 00:00:00 UTC) recording when the message was sent or received.

is_from_me. A boolean indicating whether the message was sent (1) or received (0) by the device's user. This field reflects the account-level perspective (sent by the shared Apple ID) rather than device-level provenance: a message sent from Device B still shows is_from_me=1 on Device A if it was sent by the shared account.

service. A string indicating the delivery mechanism: iMessage for Apple's encrypted messaging service or SMS for carrier-delivered text messages. This field determines which sync path the message follows and affects whether independent delivery records may exist alongside synced copies.

4.4.2 Binary Data Fields

Four sms.db columns carry binary data that standard SQLite inspection cannot decode.

attributedBody. Every message record carries attributedBody encoded in Apple's NSTypedStream format, a binary serialisation of NSAttributedString that predates the bplist format. The field encodes message text with inline styling and Apple Data Detector annotations: one-time codes as __kIMOneTimeCodeAttributeName, URLs as __kIMLinkAttributeName, monetary values as __kIMMoneyAttributeName, and calendar dates as __kIMCalendarEventAttributeName. The field is not synced. Byte content is identical for most GUIDs across devices, but diverges where iOS versions differ in data detection capability. A promotional that was sent by carrier to every prepaid sim in the test phones carrying a monetary amount and URL produced 1154 bytes on Devices A and B and 520 bytes on Device C. iOS 12's data detector annotated fewer spans. attributedBody records device-specific post-processing rather than a synced value and cannot serve as a cross-device comparator.

message_summary_info. Where present, this bplist dictionary contains ust (boolean), amc (integer, value 0), and in one Device B record hbr. The field is sparsely populated: 17 of 44 records on Device A, 16 of 37 on Device B, 1 of 44 on Device C. No cross-device

variation was observed for the same GUID; the field carries no device-specific provenance information.

payload_data. All message records carry a NULL payload_data value. The column holds structured metadata for attachments, tapbacks, and other non-text content. The test dataset contained only text messages.

chat.properties. The properties column in chat is a bplist dictionary with two relevant fields. CKChatWatermarkMessageID holds the local ROWID of the most recently CloudKit-tracked message in that thread. ROWIDs are device-local, so this value differs for the same thread across devices. CKChatPreviousAccountsDictionaryKey maps service names to the account_guid previously associated with that thread's CloudKit sync on this device, linking a chat thread to a specific device's account identity.

5 Analysis and findings

5.1 Multi-Device context

During the research conducted for this thesis, two parser modules were written and submitted to the iLEAPP repository via pull requests to address the multi-device detection capability gap identified in Chapter 4.

Extension of Trusted Peers module. iLEAPP already included a parser for com.apple.security.keychain-defaultContext.TrustedPeersHelper.db. The original module produced six columns: escrow timestamp, hardware model, model identifier, device name, serial number, and passcode length. While investigating the data written to database it was found that ZESCROWMETADATA table holds a field blob field ZPEERINFO. When trying to decode the blob it became clear that it includes information not visible in the database in clear text format. Notably serial numbers of all linked devices and their last known software versions. This information was added to module output. Additionally the ZESCROWCLIENTMETADATA table holds ZDEVICECOLOR and ZDEVICEENCLOSURECOLOR, the hexadecimal bezel and back cover colour codes captured at escrow time. These values correspond to the CSS color codes used on Apple's public product pages to represent device color variants.

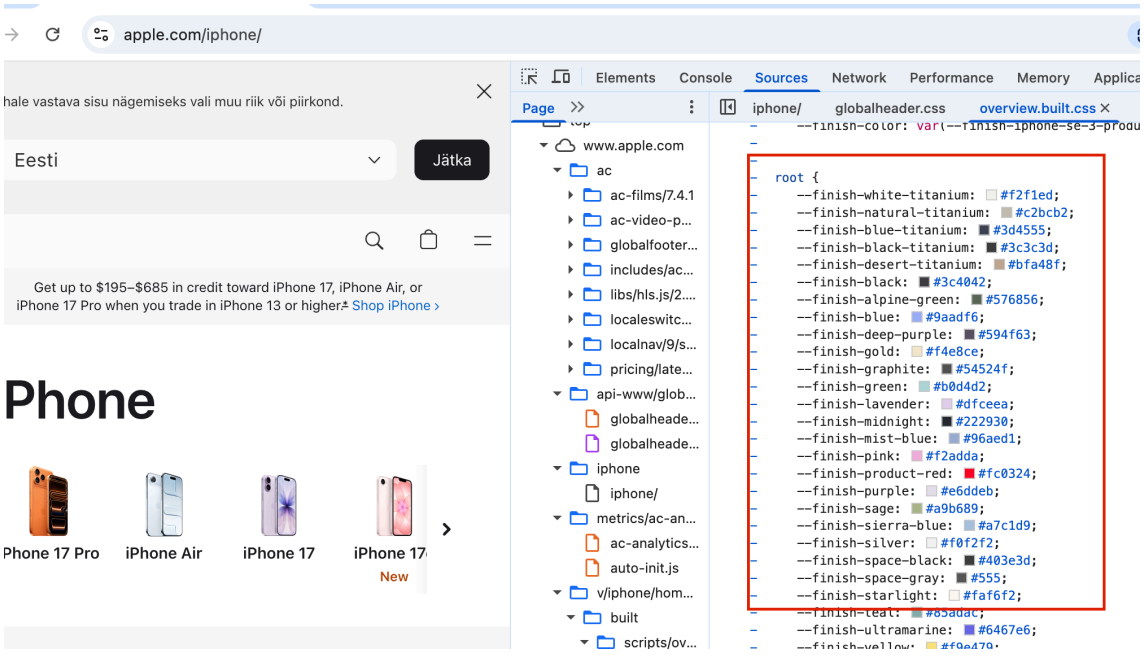

Figure 2 - Apple web product color codes

Therefore this data can help an examiner visually confirm they have the correct physical device. The updated module produces ten columns: escrow timestamp, hardware model, model identifier, device name, serial number, passcode length, bezel color, back color, os version and serial number via decoded peerinfo.

Timestamp	Model	Model Version	Device Name	Serial Number	Passcode Length	Bezel Color	Back Color	OS Version	Serial (PeerInfo)
2025-12-15 17:36:55	iPhone 7	iPhone9,3	iPhone	F71Z3355HG7F	6	1	1	19H365	F71Z3355HG7F
2025-12-15 17:39:11	iPhone 6	iPhone7,2	iPhone		6	#3b3b3c	#b4b5b9	16H81	FFNWN32AHXR5
2025-12-15 19:45:40	iPhone SE (1st generation)	iPhone8,4	iPhone	FR8SD0HXH2XN	6	#121211	#aeb1b8	19H386	FR8SD0HXH2XN
2025-12-16 16:54:37	MacBookPro14,2	MacBookPro14,2	Ray's MacBook Pro	C02WJ4FSHV2R	0			22H730	C02WJ4FSHV2R
2026-01-12 15:57:14	iPad mini 2	iPad4,5	iPad		4	#3b3b3c	#999989b	16H81	F9FS8BGSFLML

Table 9. Example output of modified iLEAPP trustedpeers module

5.2 Safari Data

5.2.1 History.db

This section presents the analysis of Safari browsing history records corresponding to the three web browsing test cases (TC-W01 through TC-W03) defined in Section 3.2.8. Records that were present in the database prior to the test execution are excluded from the analysis, as no ground truth about their origin or expected sync behaviour is available. Each test case specifies the browsing device and the URLs to be visited, providing a controlled basis for evaluating provenance indicators.

Safari's browsing history is stored in the History.db SQLite database located at /var/mobile/Library/Safari/History.db on iOS 13 and later. On iOS 12 (Device C), the

database resides within the Safari application container at /var/mobile/Containers/Data/Application/<UUID>/Library/Safari/History.db. The database participates in iCloud Safari history synchronisation through CloudKit.

Each test case was verified against the corresponding device's database by matching the visited URLs and timestamps. All test case URLs were confirmed present on their respective originating devices. TC-W03 included three URLs that overlap with TC-W01 (bbc.com, wikipedia.org, github.com). Since both devices visited these URLs independently, the resulting records have different timestamps and different local id values, allowing unambiguous attribution despite the shared URL.

Comparing expected and actual sync destinations reveals asymmetric propagation. The following table summarises the results.

Test Case	Originating Device	Expected Sync To	Actual Sync To
TC-W01	A	B, C	Neither
TC-W02	B	A, C	A
TC-W03	C	A, B	Neither

Table 10. Expected versus actual sync propagation for browsing test cases.

Only TC-W02 records propagated successfully, and only to Device A. All five TC-W02 URLs (reddit.com, reuters.com, medium.com, theguardian.com, linkedin.com) appeared on Device A with origin=1, confirming receipt via iCloud sync. TC-W01 records did not appear on either Device B or Device C despite the expectation that they would. TC-W03 records similarly failed to propagate.

Device C (iOS 12) neither received synced records from the other devices nor successfully propagated its own records outward during the test period.

The history_visits table contains an origin field that serves as a direct provenance indicator. A value of 0 indicates a visit originating from the local device's direct browsing, while 1 indicates a visit received via iCloud sync. However the data does not reveal the sync source.

URL	Device B	Device A
http://reddit.com/	origin=0	origin=1
http://reuters.com/	origin=0	origin=1
http://medium.com/	origin=0	origin=1
http://theguardian.com/	origin=0	origin=1
https://www.theguardian.com/europe	origin=0	origin=1
http://linkedin.com/	origin=0	origin=1

Table 11. Origin field values for TC-W02

All locally browsed records on Device B carry origin=0, and their synced copies on Device A carry origin=1. No exceptions were observed among the test case records. On Device C, all test case visits show origin=0, consistent with direct local browsing and the absence of inbound sync.

For TC-W02 records present on both Devices A and B, the visit_time values are identical to floating-point precision. The identical timestamps confirm that synced records preserve the original visit time rather than recording the sync receipt time.

Safari's History.db preserves HTTP redirect chains through the redirect_source and redirect_destination fields in history_visits. For TC-W02 records that synced from Device B to Device A, the redirect chain structure was faithfully reproduced. While the local id values differ between devices (as expected, since each device assigns its own sequential primary keys), the redirect relationships are structurally preserved: the HTTP visit record's redirect_destination points to the corresponding HTTPS visit, and the HTTPS record's redirect_source points back. The redirect chains on Device C for locally browsed TC-W03 URLs (http://github.com/ → https://github.com/, http://nytimes.com/ → https://www.nytimes.com/, http://youtube.com/ → https://m.youtube.com/) similarly demonstrate that HTTP-to-HTTPS redirects were recorded, providing forensic evidence that URLs were typed without the scheme prefix.

Redirect Chain	Device B	Device A
http://reddit.com/ → https://www.reddit.com/	visit 20 → 21	visit 62 → 63
http://reuters.com/ → https://www.reuters.com/	visit 23 → 24	visit 77 → 78
http://medium.com/ → https://medium.com/	visit 28 → 29	visit 82 → 83
http://theguardian.com/ → https://www.theguardian.com/europe	visit 31 → 32	visit 85 → 86
http://linkedin.com/ → https://www.linkedin.com/	visit 33 → 34	visit 87 → 88

Table 12. Redirect chain preservation during sync (TC-W02)

The history_visits table includes a generation column, and the metadata table stores current_generation and last_synced_generation values:

Device A: current_generation=30, last_synced_generation=29

Device B: current_generation=5, last_synced_generation=4

Device C: current_generation=11, last_synced_generation=10

In all three cases, current_generation is exactly one ahead of last_synced_generation. Generation numbers are assigned per-device and are not preserved during sync. For example, TC-W02 records that are at generation 1–2 on Device B appear at generations 27–28 on Device A. The generation on the receiving device reflects when the record was inserted into the local database, not the source device's generation counter.

Following the classification methodology defined in Section 3.4.4, each indicator is assessed along two dimensions: reliability across test cases and iOS version scope.

Indicator	Local Record	Synced Record	Reliability	iOS Version Scope
origin	0	1	Reliable across all test cases; no exceptions observed	iOS 12, 15, 16
visit_time	Original timestamp	Identical to source	Not a provenance indicator; preserved during sync	iOS 12, 15, 16
redirect chains	Present	Preserved (new local IDs)	Not a provenance indicator; structure preserved during sync	iOS 12, 15, 16
attributes	Local value	May differ (3 becomes 2)	Not reliable as provenance indicator; value modified inconsistently during sync	iOS 12, 15, 16
generation	Local counter	Receiving device's counter	Not a provenance indicator; reassigned on receipt	iOS 12, 15, 16

Table 13. Provenance indicators for Safari History.db

Bookmarks.db

The added and last_modified columns carry zero values across every record on all three devices and are uninformative for timeline analysis. The _dav_generation counter is 24 on Device A, 26 on Device B, and 55 on Device C; Device C's higher count reflects a longer history of WebDAV sync round-trips over its lifetime, not any difference in current bookmark state.

The sync_data blob in each synced record encodes a two-layer NSKeyedArchiver structure whose inner EncodedCKRecordSystemFields holds the CKRecord system metadata. Four fields within this metadata are forensically significant. ModifiedByDevice is a plain string naming the device that most recently wrote the record's content to iCloud; for all 15 synced records, this value is byte-for-byte identical across all three test devices, confirming the field is distributed verbatim by CloudKit and is never rewritten locally on receipt. RecordCtime and RecordMtime are NSDate timestamps recording when CloudKit first received the record and when it was last modified on the server; unlike added and last_modified, these values are non-zero and provide a reliable creation timeline independent of any local device clock. ETag is a server-assigned hex string incremented on each write; its values across the 15 records form a monotonic sequence from 6 through l, representing the order in which records were written to iCloud.

Indicator	Local Record	Synced Record	Reliability	iOS Version Scope
syncable	0	1	Reliable; unambiguous structural separation between local and synced records	iOS 12, 15, 16
server_id	Absent	Stable cross-device identifier	Reliable; identical for the same bookmark across all devices	iOS 12, 15, 16
sync_data.ModifiedByDevice	Absent	Device name that last wrote the record	Reliable; byte-for-byte identical across all three devices for all 15	iOS 12, 15, 16

			synced records	
sync_data.RecordCtime / RecordMtime	Absent	CloudKit creation and modification timestamps	Reliable; non-zero real timestamps, independent of local device clock	iOS 12, 15, 16
sync_data.ETag	Absent	Monotonic server-version sequence	Reliable; provides write-order but not device-of-origin	iOS 12, 15, 16
external_uuid	Per-device UUID	Differs across devices for the same bookmark	Not a cross-device correlator; assigned locally	iOS 12, 15, 16
added / last_modified	Zero	Zero	Not informative; zero across all records on all devices	iOS 12, 15, 16
_dav_generation	Device-local counter	Device-local counter	Not a provenance indicator; reflects device's sync history length	iOS 12, 15, 16

Table 14. Provenance indicators for Bookmarks.db records

5.2.2 Cloudtabs.db

The cloud_tabs.device_uuid column is a foreign key into cloud_tab_devices.device_uuid, providing a stable, UUID-based link between each open tab and the device that owns it. Joining the two tables on device_uuid yields the owning device's name and CloudKit enrollment metadata in a single query. The system_fields.ModifiedByDevice string in cloud_tabs provides independent confirmation of the owner name, and in all 16 tabs in the test dataset the two attribution paths agree.

Timestamps in this artifact are meaningful. RecordCtime and RecordMtime in the decoded system_fields carry real, non-zero values throughout, in contrast to the zero-valued added and last_modified columns in Bookmarks.db. For cloud_tab_devices records, RecordCtime records the moment the device first registered itself in the CloudTabs sync group: Ray's MacBook Pro enrolled on 2026-03-02 at 19:05:56 UTC, Device A on 2026-03-05 at 13:59:09, and Device B at 13:59:43, a 34-second gap

consistent with sequential manual test steps. All five of Device C's tabs carry a RecordCtime of 2026-03-05 14:04:03 UTC, indicating a batch upload at a single point rather than tabs accumulated over separate sessions.

The most significant finding in this artifact is that Devices A and B share a common CloudTabs zone while Device C maintains an entirely separate one. The cloud_tab_devices table on Devices A and B lists three enrollees (MacBook Pro, Device A, Device B); Device C lists only itself. No tab from Device C appears in Devices A or B, and no tab from the others appears in Device C. The ETag strings for Device C's device record ('1h') fall in an entirely different server-version series from those on Devices A and B ('mm9jukwy', 'mmdj7lr6', 'mmdj8bws'), confirming that Device C's CloudTabs zone has not received updates from the same server push sequence as the others. An examiner who has access only to Device C will observe only Device C's own five tabs and will find no evidence that the account's other devices had any open tabs at the time of acquisition. The full picture, comprising eleven additional tabs across a MacBook Pro and two other iPhones, is visible only from a device that shares the same sync zone. Whether this partition resulted from Device C running iOS 12 or from Device C being offline or unconfigured for iCloud tab sharing during the relevant window cannot be determined from the data alone.

Indicator	Local Record	Synced Record	Reliability	iOS Version Scope
device_uuid	Device's own UUID	Originating device's UUID (preserved)	Reliable; all 16 tabs correctly attributed via UUID join	iOS 12, 15, 16
system_fields.ModifiedByDevice	Device's own name	Originating device's name (preserved)	Reliable; agrees with device_uuid attribution for all 16 tabs	iOS 12, 15, 16
RecordCtime / RecordMtime	CloudKit creation and modification timestamps	CloudKit creation and modification timestamps	Reliable; non-zero real timestamps throughout	iOS 12, 15, 16
Zone partition	—	—	Device C in separate zone from A and B; cause not determinable	iOS 12 vs. iOS 15, 16

			from data alone	

Table 15. Provenance indicators for Cloudtabs.db records

5.2.3 Summary of Findings

Across the three Safari artifact types, CloudKit-distributed records consistently carry stable, device-independent identifiers while locally assigned identifiers diverge. In Bookmarks.db, server_id is the stable cross-device identifier and the sync_data blob provides the richest per-record provenance, encoding ModifiedByDevice, RecordCtime, ETag, and the CRDT deviceIdentifier, all distributed verbatim by CloudKit and never rewritten locally. The syncable flag provides an unambiguous structural separation between locally created and server-synchronised records. In CloudTabs.db, device_uuid links each tab directly to its owning device without inference, and CloudKit timestamps record enrollment and upload events with sub-second precision. The zone partition between Device C and Devices A and B demonstrates that iOS version differences or connectivity gaps can produce entirely disjoint views of the same account's tab state, a finding with direct implications for investigations involving devices running different iOS generations.

5.3 Call Log Data

This section presents the analysis of call records corresponding to the seven call test cases (TC-C01 through TC-C07) defined in Section 3.2.4. Each test case specifies the initiating device, the target, and the expected sync destinations, providing a controlled basis for evaluating provenance indicators.

Each test case was matched to its corresponding database record using the ZADDRESS, ZORIGINATED, ZDURATION, and ZSERVICE_PROVIDER fields in conjunction with the test protocol. External 1 (iPhone XR) was identified as +37253368505 and External 2 (iPhone SE 2020) as +37254350791 based on the address values observed in the matched records.

Test Case	Description	ZUNIQUE_ID	Expected Sync	Actual Presence
TC-C01	A → Ext 1, answered, 35.5 s	2C22C6B3	B, C	A, B

TC-C02	B → Ext 2, answered, 40.6 s	3550220A	A, C	A, B
TC-C03	C → Ext 1, missed	B2A56A28	A, B	C
TC-C04	Ext 2 → A, answered, 32.7 s	BC185F6F	B, C	A, B, C
TC-C05	Ext 1 → B, missed	F0447E05	A, C	A, B
TC-C06	FT Audio A → C, answered, 34.4 s	9309730A	B	A, B, C
TC-C07	FT Video B → A, missed	9410C75C / 0D6040C4F105	C	A, B

Table 16. Test case to record mapping and sync results

TC-C07 produced two distinct records with different ZUNIQUE_ID values: 9410C75C represents Device A's perspective (incoming FaceTime from Device B, not answered), while 0D6040C4F105 represents Device B's perspective (outgoing FaceTime to Device A). Both records were synced between Devices A and B. A third record (BE08AFDA) appeared on Device C only, representing Device C's independent observation of the incoming FaceTime call from Device B.

5.3.1 Sync Coverage

Comparing expected and actual sync destinations reveals that Device C (iOS 12) consistently failed to receive records that were expected to sync to it. TC-C01, TC-C02, and TC-C05 were all expected to propagate to Device C but did not. Conversely, TC-C04 and TC-C06 did reach Device C, suggesting that sync to Device C was intermittent.

TC-C03 (originated on Device C) did not sync to Devices A or B despite the expectation that it would. This is consistent with incomplete outbound sync from iOS 12.

TC-C06 propagated to Device C even though the test plan only expected sync to Device B. This indicates that FaceTime calls between same-account devices create records on all devices in the sync circle, not only on the two participants.

5.4 SMS Data

This section presents the analysis of SMS and iMessage records corresponding to the test cases in Section 3.2.5 and 3.2.6, as well as carrier-originated messages whose provenance

is known by nature (a carrier SMS addressed to a specific SIM can only have been received on the device holding that SIM).

Test Case	Description	GUID (prefix)	Expected Sync	Actual Presence
TC-S01	Ext 1 → A, SMS	F2E85E64	B, C	A, B, C
TC-S02	Ext 2 → B, SMS	CC707E51	A, C	A, B, C
TC-S03	Ext 1 → C, SMS	24C3E3E4	A, B	C
TC-S04	A → Ext 2, SMS	174BF45B	B, C	A, B, C
TC-S05	B → Ext 1, SMS	C5E9A377	A, C	A, B, C
TC-S06	C → Ext 2, SMS	02B80FA4	A, B	A, B, C
TC-S07	A → B, same-acct	8BD03F95	C	A, B, C
TC-S08	B → C, same-acct	ABDB64F1	A	A, B, C
TC-I01	Ext 1 → A, iMessage	6C9BBF68	B, C	A, B, C
TC-I03	Ext 1 → C, iMessage	B2EB7A4F	A, B	C
TC-I05	B → Ext 1, iMessage	A48541A0	A, C	A, B, C
TC-I07	A → B, same-acct iMessage	B4FFBA47	C	A, B, C
TC-I08	B → C, same-acct iMessage	C6FFBFCD	A	A, B, C
TC-I09	C → A, same-acct iMessage	D3C6EA88	B	A, B, C

Table 17. SMS test case to record mapping and sync results

Of the 15 executed test cases, 12 propagated to all three devices. Two test cases — TC-S03 (SMS from External 1 to Device C) and TC-I03 (iMessage from External 1 to Device C) — appeared only on Device C and failed to sync to Devices A and B. Both of these were messages received directly by Device C's phone number (+37253507826) and not propagated outward. This is consistent with the pattern observed across all artifact categories: Device C (iOS 12) consistently failed to propagate locally received records to the other devices,

Message	GUID (prefix)	destination_caller_id	Device A	Device B	Device C

"Juhhei, sinu Super on laetud!" top-up confirmation	8AB94320	+37253952660 (A)	Present	Not present	Not present
"Ae-ae!" expiry warning	FFEB072C	+37253952660 (A)	Present	Present	Present
"Hei-hei!" expired notification	A352F7EA	+37253952660 (A)	Present	Present	Present
Telia SSO code 505082	6E5A2C42	+37253952660 (A)	Present	Not present	Not present
"Aeg laadida!" balance alert	D2B84A6B	+37253507826 (C)	Not present	Present	Present
"Supersuur tänu..." welcome 1	998543AC	+37254370088 (B)	Present	Present	Present

Table 18. Carrier message sync behaviour

Carrier messages addressed to Device A's SIM (+37253952660) showed mixed sync: two of four propagated to all three devices, while the top-up confirmation and the Telia SSO verification code remained on Device A only. Carrier messages addressed to Device B's SIM (+37254370088) propagated to all three devices. The single carrier message addressed to Device C's SIM (+37253507826) reached Device B but not Device A.

The destination_caller_id field in carrier messages reliably identifies which SIM received the message, and this value is preserved on devices that receive the synced copy.

SMS and iMessage timestamps use nanosecond-precision Mac Absolute Time. For the test case records present on all three devices, two timestamp patterns emerge depending on message type:

- iMessage timestamps are identical or near-identical across devices. TC-I01 (imsg_ext1_to_a) carries date=794411538885107456 on all three devices, a difference of zero. TC-I07 and TC-I08 show sub-microsecond variations (less than 0.5 milliseconds);
- SMS timestamps show larger variations. TC-S04 (sms_a_to_ext2) shows a spread of 281 milliseconds across devices, TC-S05 shows 495 milliseconds, and TC-S06 shows 1.26 seconds. The cause of these sub-second variations is not determinable

from the test data, but the pattern consistently distinguishes SMS from iMessage records.

For inbound SMS from external parties (TC-S01, TC-S02), the timestamps are identical across all three devices, indicating that the original receipt timestamp was preserved during sync.

Four indicators in sms.db and its companion files link all three devices to the same account environment. First, the chat table records the iCloud account or phone number associated with each conversation thread. On Device A, threads are associated with P:+37253952660 and, in one case, E:ray.dio@icloud.com. On Device B, they are associated with P:+37254370088 and E:ray.dio@icloud.com. On Device C, they are associated with P:+37253507826 and E:ray.dio@icloud.com. The shared iCloud address links all three databases to the same account, while the P: entries identify each device's own phone number: +37253952660 for Device A, +37254370088 for Device B, and +37253507826 for Device C.

Second, each device contains handle records for the other two devices' phone numbers in the handle and chat tables. This confirms the presence of active message threads addressed to the other enrolled devices.

Third, each database also contains a thread addressed to its own phone number or iCloud email, created during the test cases. These self-addressed threads provide an internal confirmation of each device's identity.

A fourth linkage source is the com.apple.messages.geometrycache plist stored in the sms.db directory. This file caches rendered message bubble dimensions using message GUIDs as keys. Its version suffix varies by iOS version: _v7 on Device A, _v6 on Device B, and _v3 on Device C. All 44 GUIDs found in Device A's cache and all 37 GUIDs in Device B's cache also appear in Device C's cache. In addition, 32 GUIDs are present in all three caches, and every cached GUID corresponds to a GUID in that device's message table. This overlap provides source-independent confirmation that the same logical message events are represented across all three databases.

5.4.1 Summary of Findings

Following the classification methodology defined in Section 3.4.4, each indicator is assessed along two dimensions: reliability across test cases and iOS version scope.

Indicator	Local Record	Synced Record	Reliability	iOS Version Scope
destination_caller_id	Device's own phone line	Originating device's phone line (preserved)	Reliable for external messages;	iOS 12, 15, 16
ck_sync_state	0 (Device A) or 1 (Devices B, C)	1 (Devices B, C)	Not a local/synced classifier; uniform per device regardless of record origin	iOS 12, 15, 16
ck_record_id	null (Device A) or populated (B, C)	Populated, identical across receiving devices	Not a local/synced classifier; confirms CloudKit participation	iOS 12, 15, 16
guid	Stable cross-device correlator	Same GUID on all devices	Not a provenance indicator; identifies same event across devices	iOS 12, 15, 16
date	Original timestamp	Identical for iMessage; sub-second variations for SMS	Not a provenance indicator; preserved during sync	iOS 12, 15, 16
date_delivered	Near date value	Gap of minutes/hours/days from date	Conditional; requires multiple device acquisitions to establish direction	iOS 12, 15, 16

Table 19. Provenance indicators for SMS/iMessage records.

The combined evidence from the ck_sync_state, ck_record_id, GUID, and timestamp fields establishes a consistent picture of iCloud synchronisation as the propagation mechanism for cross-device message records. The GUID provides a stable, device-independent record identity, while ck_sync_state and ck_record_id together confirm CloudKit registration and permit direct cross-device record linkage. The date field is preserved verbatim through CloudKit for iMessage records, making the timestamp on a

synchronised record forensically equivalent to the timestamp on the originating record for that field, whereas date_read is explicitly local and carries no cross-device meaning. The date_delivered field provides a timing indicator for synchronisation. A gap of minutes, hours, or days between date and date_delivered on a device where Messages in the Cloud is active indicates that the record arrived through iCloud sync rather than direct delivery. Using this indicator requires more than one filesystem dump: without a baseline acquisition from the originating device, the gap is observable but its direction cannot be established.
The destination_caller_id field is the most informative provenance indicator for SMS/iMessage data, as it directly identifies which phone line handled the message.

No field in sms.db identifies which device originated a record. GUID and ck_record_id confirm that records across devices are copies of the same event but carry no device-of-origin value. The date field reflects the time of the original event and does not distinguish the device that sent or received the message from the devices that received the record as a synchronised copy. date_read records reading behaviour on the local device rather than creation origin. An examiner working from sms.db can confirm that a message record propagated across devices through iCloud synchronisation, but the schema provides no field from which to determine on which device the message originated.

5.5 Cross-Artifact Identifier Analysis

Reviewing the identifier fields documented across all artifact categories reveals a structural property that applies to every data source examined in this chapter: each iCloud service assigns its own identifier to represent the same physical device, and none of these identifiers appear in any other service's database or plist.

Three values cross artifact boundaries. The user-assigned device name appears as deviceName in the fmfd plist, as ModifiedByDevice in the sync_data blobs of Bookmarks.db and CloudTabs.db, and as ComputerName in the DER-decoded ZPEERINFO blob in TrustedPeers.db. It is the only cross-artifact device reference in the dataset, but it is user-configurable: a renamed device, or two devices sharing the same default name, breaks any attribution built on it. The serial number appears only in the decoded ZPEERINFO blob in TrustedPeers.db, requires DER decoding to access, and is

absent from every other artifact database examined. The phone number and Apple ID email appear in chat.account_login in sms.db and nowhere else in the artifact catalog.

An examiner working across artifact types from a single filesystem dump has no field that can be queried in one database and matched in another to confirm a common device origin. Each artifact database holds internally coherent CloudKit-distributed identifiers across devices, but is opaque to every other database in the set. Cross-artifact device attribution routes through device name, serial number from TrustedPeers, or phone number and Apple ID email from sms.db, of which serial number is the only value that is both stable and verifiable against physical hardware.

6 Conclusion

This thesis investigated whether the native iOS databases for Safari browsing, call history, and SMS/iMessage carry metadata that allows a forensic examiner to distinguish records generated locally on the examined device from records propagated to the device through iCloud synchronisation. Three iCloud-linked iPhones running iOS 16.7.12, iOS 15.8.3, and iOS 12.5.8 were configured under a single Apple ID and subjected to a controlled test plan covering calls, SMS, iMessage, Safari bookmarks, and Safari browsing. This chapter consolidates the findings by answering the research questions, stating the contributions, discussing practical implications, acknowledging limitations, and outlining future work.

6.1 Answers to the research questions

6.1.1 Primary Research Question: What identifiable forensic indicators allow an examiner to differentiate between locally generated and iCloud-synchronised Safari browsing data, call logs, and SMS messages on iOS devices?

The research found that identifiable forensic indicators do exist, but their strength varies significantly between artifact categories. No universal device-origin field was identified across Safari, call history, SMS, and iMessage artifacts. Instead, the examiner must evaluate each artifact type according to its own database structure, synchronisation behaviour, and available metadata.

For Safari artifacts, the strongest indicators were found in Bookmarks.db and CloudTabs.db. Bookmarks.db preserved CloudKit-related metadata, including stable server-side identifiers and synchronisation data that could distinguish records that had participated in iCloud synchronisation from records that were only locally present. The syncable flag was particularly useful for separating local-only records from synchronised records. The sync_data blob also contained useful provenance-related metadata, including modified-by-device information, record creation time, ETag values, and CRDT-related device identifiers. These fields do not always prove physical device origin

in isolation, but they provide strong evidence of synchronisation state and modification history.

CloudTabs.db provided even stronger device-association evidence because records were linked to device-specific tab state. Fields such as device_uuid and CloudKit timestamps allowed tabs to be associated with the device that owned or published the tab state. However, the analysis also showed that iOS version differences and synchronisation inconsistencies can result in different devices having incomplete or disjoint views of the same iCloud account's tab state. Therefore, CloudTabs.db can support device attribution more strongly than many other artifacts, but its absence on a device should not be interpreted as proof that no relevant activity occurred.

Safari History.db provided more limited provenance value. Browsing records could be correlated and their propagation could be observed, but the available fields did not provide a reliable direct device-origin indicator. In this artifact category, the examiner can often identify that a history entry exists across devices and may infer synchronisation behaviour from record presence, timestamps, and related identifiers, but should treat physical origin attribution with caution.

For call logs, CallHistory.storedata supported event correlation across devices through fields such as ZUNIQUE_ID, address fields, duration, call direction, and service provider values. These fields made it possible to match records belonging to the same call event across devices. However, the database did not consistently provide a direct field identifying the physical device that initiated or received the call. In addition, synchronisation coverage was inconsistent. Some records propagated to expected devices, while others did not, and the iOS 12 device showed particularly irregular synchronisation behaviour. This means that call log artifacts can support correlation and partial synchronisation analysis, but they should not be treated as reliable standalone proof of device origin.

For SMS and iMessage artifacts, sms.db contained stable identifiers such as message GUIDs and conversation GUIDs that were useful for linking copies of the same message across devices. CloudKit-related fields also provided evidence that some records were synchronised. However, sms.db did not directly identify the physical device from which a message originated. Fields such as destination_caller_id, message direction, service

type, and timestamp relationships can provide conditional provenance value, especially when interpreted against known account configuration and test context, but they are not sufficient on their own to make an unconditional device-origin claim.

Therefore, the answer to the primary research question is that forensic indicators exist at three levels: synchronisation indicators, correlation indicators, and conditional provenance indicators. Safari Bookmarks.db and CloudTabs.db provide the strongest provenance-related evidence. CallHistory.storedata and sms.db provide useful event-correlation evidence but weaker direct attribution evidence. Across all examined categories, examiners should distinguish between proving that an artifact was synchronised, proving that two records represent the same event, and proving which physical device originally generated the event.

6.1.2 Secondary Research Question 1: Which iOS databases contain metadata identifying other devices associated with the same iCloud account?

The research identified several iOS data sources that can assist in establishing a multi-device iCloud context. The most relevant sources were the iCloud Keychain trusted peer database and related device-context artifacts, which contained information about devices associated with the same iCloud trust environment. These artifacts are important because provenance analysis depends first on establishing that the examined devices were part of the same iCloud ecosystem.

CloudTabs.db also contained device-level information relevant to the same iCloud account. In this database, device_uuid and related device records associated tab state with specific devices. This made CloudTabs.db particularly useful not only for Safari analysis but also for demonstrating that multiple devices were participating in a shared iCloud synchronisation environment.

Safari Bookmarks.db also contained synchronisation metadata distributed through CloudKit. While it did not always identify all associated devices in a general account-wide sense, its per-record metadata could identify the device or device-related identifier involved in creating or modifying a bookmark record. This makes it valuable for per-artifact provenance analysis rather than general device inventory.

sms.db and CallHistory.storedata were less useful for identifying the set of devices associated with the iCloud account. These databases contained identifiers that supported cross-device event correlation, but they did not function as reliable inventories of trusted or linked devices. Their value lies mainly in comparing event records between devices after the multi-device context has already been established through other artifacts.

In summary, the databases and artifacts most useful for identifying other devices associated with the same iCloud account were the trusted peer/iCloud Keychain data sources and CloudTabs.db. Bookmarks.db provided useful device-related metadata at the record level. CallHistory.storedata and sms.db supported event correlation but did not directly identify the broader set of associated iCloud devices.

6.1.3 Secondary Research Question 2: What metadata fields or artifact characteristics within these databases can be used to distinguish locally created records from iCloud-synchronised records?

The fields and characteristics that support this distinction vary by artifact type.

In Safari Bookmarks.db, the most useful indicators were server_id, syncable, and the contents of the sync_data blob. The server_id field functioned as a stable cross-device identifier for synchronised bookmark records. The syncable field provided a structural indication of whether a record was intended to participate in synchronisation. The sync_data blob contained the richest provenance-related information, including fields such as ModifiedByDevice, RecordCtime, ETag, and CRDT deviceIdentifier. Together, these fields allow examiners to distinguish between records that remained local and records that were distributed through CloudKit.

In CloudTabs.db, the most important fields were device_uuid and CloudKit timestamp values. The device_uuid field linked tab records to a specific device's tab state, making it a strong indicator for associating a record with a device in the iCloud account. CloudKit timestamps provided additional information about synchronisation and upload timing. However, the research also showed that CloudTabs synchronisation may be incomplete across iOS versions, so examiners should consider both the presence and absence of records carefully.

In Safari History.db, provenance indicators were weaker. Record presence across multiple devices, timestamp relationships, origin-related values, and redirect-chain preservation could support synchronisation analysis, but the database did not provide a reliable direct device-origin field. History records should therefore be interpreted as supporting correlation and synchronisation inference rather than direct device attribution.

In CallHistory.storedata, ZUNIQUE_ID was the most useful cross-device correlation field. When the same ZUNIQUE_ID appeared on multiple devices, it allowed the examiner to identify records representing the same call event. Other fields, including ZADDRESS, ZORIGINATED, ZDURATION, and ZSERVICE_PROVIDER, supported matching and interpretation. However, these fields did not consistently identify which physical device originated the call record. The inconsistent propagation observed during testing also means that the absence of a call record on one device cannot be treated as proof that the call did not synchronise or did not occur.

In sms.db, message GUIDs and chat GUIDs were the strongest event-correlation indicators. CloudKit-related fields provided evidence of synchronisation participation. Additional fields such as destination_caller_id, service type, message direction, and timestamp relationships provided conditional provenance value, especially when combined with known test markers and account configuration. However, sms.db did not contain a direct physical-origin field for message records.

Overall, the most reliable indicators for distinguishing local from synchronised records were those that explicitly reflected CloudKit participation or server-side synchronisation state. Fields that merely match events across devices are useful for correlation but should not be treated as proof of origin. The key interpretive distinction is therefore between synchronisation evidence, event-correlation evidence, and device-origin evidence.

6.2 Examiner workflow

The findings of this thesis can be organised as a procedure that an examiner can apply when presented with a single iOS filesystem dump suspected to belong to a multi-device iCloud account. The procedure proceeds in two stages. The first stage establishes whether other devices exist on the same iCloud account and identifies them. The second stage examines each artifact category in turn, applying the indicators identified in Chapter 5 to

classify records as locally generated, synchronised, or correlated copies of the same event. The stages are sequential because attribution of any record to "another device" is meaningful only once the set of other devices is known.

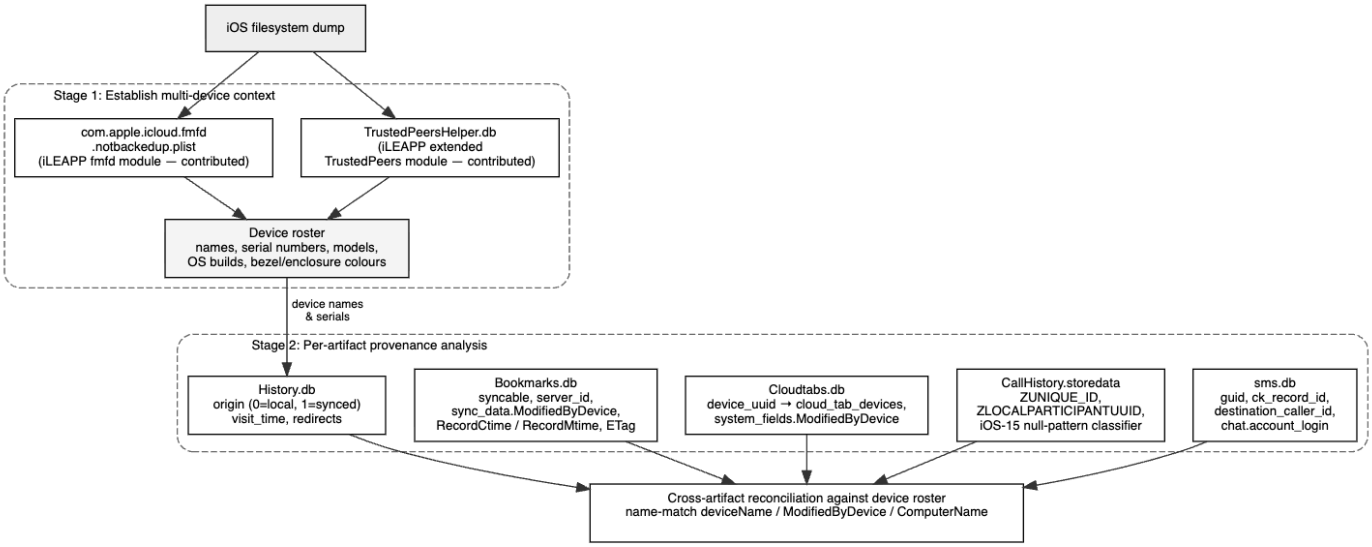

Figure 3. Examiner workflow

6.2.1 Establishing the multi-device context

The first step in any examination is to determine whether the dumped device belongs to an iCloud account with other enrolled devices, and if so, to enumerate those devices.

com.apple.icloud.fmfd.notbackedup.plist in /private/var/mobile/Library/Preferences/ is present on every iOS version examined and is the simplest starting point. The outer bplist is decoded, the kFMFDStoredDataKey value is treated as an NSKeyedArchiver archive, and the resulting dictionary's kFMFDDevicesListKey set is enumerated. Each FMFDevice entry yields a deviceName, a Find My device id, and an IDS UUID; the entry whose isThisDevice flag is true identifies the dumped device. The roster is recovered from the file even though the file itself is not preserved across iCloud restore, because iOS rebuilds it from the account state at runtime.

com.apple.security.keychain-defaultContext.TrustedPeersHelper.db in /private/var/Keychains/ corroborates and extends this roster on iOS 13 and later. The

ZPEER table lists each enrolled peer with a SHA-256 peer id; ZESCROWCLIENTMETADATA resolves each peer's machine id to a hardware model, a device color, and an enclosure color; and the ZPEERINFO blob in ZESCROWMETADATA decodes (DER) to a structure containing each device's serial number, OS build, model name, and computer name. The serial numbers recovered here are the only stable, hardware-bound identifiers exposed in any of the artifact databases examined and should be the canonical reference when correlating against physical hardware. The database is absent on iOS 12, so an examiner working on a pre-iOS-13 device must rely on the fmfd plist alone.

An examiner working with a current build of iLEAPP can produce the device roster directly from its report output without manually decoding the underlying files, since the parser modules for both artifacts were contributed open-source as part of this thesis

The output of stage one is a list of devices on the account, with names, models, and (where available) serial numbers and OS versions. Each subsequent finding is interpreted relative to this list.

6.2.2 Safari bookmarks

For each row in the bookmarks table, the examiner should first check the syncable flag: a value of 0 indicates a local-only record, a value of 1 indicates a record that participated in CloudKit synchronisation. For the synced rows, the server_id column provides the cross-device identifier that allows the same bookmark to be located on any other device in the account, and the sync_data blob, after two-layer NSKeyedArchiver decoding, exposes the CloudKit system fields that carry the most useful provenance metadata.

Within those fields, ModifiedByDevice is a plain string holding the user-assigned name of the device that most recently wrote the record's content to iCloud. Cross-referenced against the device roster from §6.2.1, this names the source device for the most recent edit — provided no two devices on the account share the same user-assigned name. RecordCtime and RecordMtime are CloudKit-side timestamps that are independent of the local device clock and therefore unaffected by clock drift on the dumped device, and ETag gives a monotonic server-version sequence that places the record in the order it was written to iCloud relative to other records.

The added and last_modified columns of the bookmarks table itself are zero across every record examined and should not be used for timeline construction. The external_uuid is per-device and is not a cross-device correlator. The output of this step is, for each synced bookmark, a triple of (server-side identifier, last-modifying device name, server-side modification time) that can be used to attribute the most recent edit to a named device on the account.

6.2.3 Safari iCloud Tabs

Cloudtabs.db is the strongest device-attribution artifact identified in the thesis. Each row in the cloud_tabs table carries a device_uuid foreign key into cloud_tab_devices, which lists the devices that have published tab state into the account's CloudTabs zone with their user-assigned names and CloudKit enrollment metadata. Joining the two tables on device_uuid therefore attributes every open tab to a named device without inference. The system_fields.ModifiedByDevice string in cloud_tabs agrees with this attribution in every test record and provides an independent confirmation.

RecordCtime on rows of cloud_tab_devices records the moment each device first registered itself in the CloudTabs zone, which is interpretable as the first time that device's user opened Safari with iCloud Tabs enabled after enrollment. RecordCtime on cloud_tabs rows records when each individual tab was published.

A material caveat applies. The CloudTabs zone is not necessarily uniform across the account: the test data showed that the iOS 12 device participated in a separate zone from the iOS 15 and iOS 16 devices, with the result that an examiner inspecting the iOS 12 device would have seen only that device's own tabs and would have had no on-device evidence that other devices on the account had any open tabs at all. The output of this step is therefore a list of (tab, owning device, publish time) tuples for every device that participates in the same zone as the dumped device, and not necessarily for the full account roster.

6.2.4 Call history

For each row in ZCALLRECORD, the examiner should treat ZUNIQUE_ID as the cross-device correlator: where the same ZUNIQUE_ID appears on multiple devices on the account, those rows are CloudKit-distributed copies of the same call event. The presence

or absence of a row with a given ZUNIQUE_ID on the dumped device, set against the call's expected sync destinations from §6.2.1, allows the examiner to characterise the call's propagation footprint.

ZLOCALPARTICIPANTUUID is the field that turns correlation into device-of-origin attribution. It identifies the local SIM or phone line that participated in the call, stored as a 16-byte binary blob whose hex representation is a UUID. To use this field operationally, the examiner first establishes a UUID-to-phone-line mapping from records of known origin: for any locally originated outgoing call (where direction and the dumped device's own SIM are known), the ZLOCALPARTICIPANTUUID value is the UUID for that SIM. Once each device's SIM UUID is mapped, any synced record on any device can be checked: if its ZLOCALPARTICIPANTUUID matches the UUID of a known SIM, the record was originated by the device holding that SIM. On iOS 15 (Device B in the test set), ZLOCALPARTICIPANTUUID exhibited an additional pattern: the field was null for every synced record and populated for every locally originated record, which serves as a binary local-versus-synced classifier without requiring a UUID mapping. ZDISCONNECTED_CAUSE and ZFACE_TIME_DATA on the same iOS version showed complementary null patterns that, in combination, allow synced records to be classified with three independent indicators. The examiner should not assume these patterns generalise to other iOS versions: on iOS 16, ZLOCALPARTICIPANTUUID was populated on synced records as well, and the binary classifier did not apply.

The output of this step, for each call event of interest, is the set of devices on which the event has a record, the SIM that handled the call (via the ZLOCALPARTICIPANTUUID mapping), and a confidence statement about whether each row is local or synced based on the iOS-version-specific patterns observed on the dumped device.

6.2.5 SMS and iMessage

For each message of interest, the examiner should first locate it across devices using the guid field, which is the cross-device correlator for sms.db. The service field distinguishes iMessage from carrier-delivered SMS and determines which subsequent fields are meaningful: iMessage records are CloudKit-distributed and carry ck_record_id populated with a hex string that is identical across receiving devices, while SMS records are carrier-delivered and may not carry the same CloudKit footprint.

destination_caller_id is the most operationally useful provenance field for SMS and iMessage records. It records which phone number or Apple ID email address was used as the local identity for the message. For an inbound message, the value identifies the phone line or Apple ID that originally received the message, and the device whose SIM holds that phone line is by inference the originating device for the local copy. For an outbound message, the value identifies the phone line or Apple ID from which the message was sent, again pointing to a specific physical device on the account. Cross-referenced against the device roster from §6.2.1 — specifically, the per-device phone numbers visible in the chat table's account_login column on each device — destination_caller_id allows the examiner to attribute message records to physical devices in cases where the account uses multiple SIMs or where iMessage is configured against a shared Apple ID with per-device phone-line aliases.

The combination of ck_sync_state and ck_record_id provides a CloudKit-participation indicator but not a local-versus-synced classifier on the dumped device: in the test data both fields took device-uniform values regardless of whether a given record was originally generated locally or arrived via sync, reflecting the device's CloudKit role rather than each record's origin. They are useful for confirming that a record participated in synchronisation; they are not useful for determining whether this device originated this record.

The date field is preserved verbatim through CloudKit synchronisation for iMessage records and is forensically equivalent across devices for the same guid. SMS records show timestamp variations across devices, the cause of which is not determinable from the test data. For timeline analysis at second precision, this is immaterial, but for sub-second forensic claims the variation should be acknowledged.

6.2.6 Cross-artifact reconciliation

A finding from one artifact category should be reconciled against findings from the others where possible. The user-assigned device name appears as deviceName in the fmfd plist, as ModifiedByDevice in Bookmarks.db and Cloudtabs.db's sync_data blobs, and as ComputerName in the decoded ZPEERINFO blob in TrustedPeersHelper.db. An examiner should confirm that the same names appear consistently across all sources before relying on any one of them. The phone number appears in chat.account_login in

sms.db and as the resolved value of ZLOCALPARTICIPANTUUID mappings in CallHistory.storedata; agreement between the two confirms that the same physical SIM is being attributed in both artifact categories.

The serial number from the decoded ZPEERINFO blob, is the only field that ties the dumped image to specific physical hardware in a way that the user cannot rename. Where a chain-of-custody question turns on identifying the physical device beyond doubt, this field should be the anchor.

No identifier in the artifact categories examined cross-references to any other category through a structural foreign-key relationship. Cross-artifact attribution is therefore a matter of name-matching against the device roster of §6.2.1, and is only as strong as the uniqueness of the device names on the account. For accounts where two devices share the same default name, or where a device has been renamed during the relevant period, the examiner should report the ambiguity explicitly rather than attempt to resolve it from the artifact data alone.

6.3 Limitations

Several limitations constrain the generalisability of the findings.

The study used three devices running specific iOS versions (12.5.8, 15.8.3, 16.7.12). While these span a range of generations, intermediate versions (iOS 13, iOS 14, iOS 17, iOS 18, iOS 26) were not tested. The provenance indicators identified may behave differently on untested versions, particularly given that Apple modifies CloudKit sync behaviour between releases.

The test plan generated a limited number of records per category. Fifteen SMS/iMessage test cases, seven call test cases, and three browsing test cases provided sufficient data to identify patterns, but edge cases such as messages with attachments, group conversations, FaceTime group calls, or concurrent high-volume sync events were not exercised. The indicators' reliability under these conditions remains untested.

All three devices were connected to Wi-Fi and had sync enabled throughout the test. Scenarios involving intermittent connectivity, disabled sync features, or partial iCloud configuration were not examined. The sync coverage results (particularly Device C's incomplete participation) may differ under other network conditions, though the consistency of the pattern across all artifact categories suggests an iOS-version rather than connectivity cause.

The user-assigned device name is treated throughout §6.2 as the principal cross-artifact attribution key. iOS allows the user to change a device's name at any time via Settings → General → About → Name, and there is nothing in Apple's interface that prevents two devices on the same account from carrying the same name. Neither scenario was exercised in this research: every test device retained its initial name, and no two devices shared a name during the test period. The behaviour of the ModifiedByDevice field on a record written before a rename, the question of whether the field is updated to the new name on subsequent writes, and the consequences of two devices sharing a default name (e.g. two new iPhones each carrying "iPhone") are therefore not addressed by the findings reported here.

6.4 Future Work

Several directions for future research follow from this thesis.

The methodology of this thesis deliberately scoped itself to file-system extraction via a jailbreak, on the grounds that file-system access is a strict superset of what logical backup exposes and is necessary to reach fields not surfaced by Apple's backup API. The findings are therefore stated against the most permissive acquisition pathway available, and the question of which findings remain visible under more constrained pathways is left open. A targeted follow-up study could repeat the test plan defined in Chapter 3 with iCloud-linked devices but capture, for each acquisition point, both an unencrypted iTunes/Finder backup and an iCloud backup. Each backup would then be checked field by field against the indicators identified in this thesis.

Future studies could repeat the experiment across a larger set of devices and iOS versions, including current iOS releases and different combinations of iPhones, iPads, and macOS systems. This would help determine whether the indicators identified in this thesis remain stable across Apple's broader ecosystem.

Further work should also test the effect of synchronisation conditions. Experiments could vary Wi-Fi and cellular connectivity, Low Power Mode, device lock state, charging state, iCloud storage availability, and elapsed time between artifact creation and acquisition. This would help quantify how environmental conditions affect propagation reliability and timestamp interpretation.

Finally, future research should examine whether similar provenance patterns exist in other iCloud-synchronised artifact categories, including Notes, Photos, Contacts, Calendar, Reminders, Health data, and Wi-Fi-related artifacts. Expanding the scope beyond the artifacts studied here would support a broader forensic framework for interpreting synchronised evidence in Apple ecosystems

References

[1] R. T. Sibe and A. Alayegun, "Digital forensic investigation of an iOS mobile phone using iTunes and iCloud backup," J. Cybersecur. Inf. Manage., vol. 17, no. 2, pp. 167–177, 2026, doi: 10.54216/JCIM.170212.

[2] S. Ntshangase, N. Nelufule, D. Mulihase, M. Mtshali, C. Mokoena, and P. Moloi, "A survey of digital forensic tools for Android and iOS smart phones," in Proc. IEEE Int. Conf. Cyber Secur. Resilience (CSR), 2024, doi: 10.1109/CSR61664.2024.10679486.

[3] F. Liu, K. Liu, C. Chang, and Y. Wang, "Research on the technology of iOS jailbreak," in Proc. 6th Int. Conf. Instrumentation Meas. Comput. Commun. Control, 2016.

[4] C. J. D'Orazio and K.-K. R. Choo, "Circumventing iOS security mechanisms for APT forensic investigations: A security taxonomy for cloud apps," Future Gener. Comput. Syst., vol. 79, pp. 247–261, 2018, doi: 10.1016/j.future.2016.11.010.

[5] M.-H. Wu, T.-C. Chang, and Y. Li-Min, "Digital forensics security analysis on iOS devices," J. Web Eng., vol. 20, no. 3, pp. 775–794, 2021, doi: 10.13052/jwe1540-9589.20310.

[6] M. AlBreiki, F. Alazemi, A. Wani, F. Iqbal, and N. Seghid, "Forensic examination of iOS platform artifacts: A comparative multi-tool study using publicly available data," in Proc. 2025 6th Int. Workshop Cybercrime Investig. Digital Forensics (NTMS), 2025, doi: 10.1109/NTMS65597.2025.11077003.

[7] H. Koganti and G. S. N. Rao, "Forensic acquisition of IOS devices," Int. J. Recent Technol. Eng., vol. 8, no. 4, 2019, doi: 10.35940/ijrte. D4374.118419.

[8] S. S. Shimmi, G. Dorai, U. Karabiyik, and S. Aggarwal, "Analysis of iOS SQLite schema evolution for updating forensic data extraction tools," in Proc. IEEE Int. Conf. Informatics, IoT, Enabling Technol. (ICIoT), 2020, doi: 10.1109/ICIoT48696.2020.9089513.

[9] A. Neyaz, A. Kumar, and B. Liu, "Digital forensics analysis of iMazing phone explorer tool to aid mobile forensics investigation," in Proc. 2025 13th Int. Symp. Digital Forensics Secur. (ISDFS), 2025, doi: 10.1109/ISDFS65363.2025.11012021.

[10] M. R. Laayu and A. Kurniawan, "Comparison of acquisition results on iPhone 7 Plus (iOS 14.8.1) between jailbreaking vs non-jailbreaking device," in Proc. 2022 10th Int. Conf. Inf. Commun. Technol. (ICoICT), 2022, doi: 10.1109/ICOICT55009.2022.9914862.

[11] K. Oestreicher, "A forensically robust method for acquisition of iCloud data," Digital Investig. , vol. 11, suppl., pp. S106–S113, 2014, doi: 10.1016/j.diin.2014.05.006.

[12] Forensafe, "Investigating iOS SMS," forensafe.com, Mar. 1, 2024. [Online]. Available: https://forensafe.com/blogs/iOSSMS.html

[13] H. Studiawan, T. Ahmad, B. J. Santoso, and B. A. Pratomo, "Forensic timeline analysis of iOS devices," in Proc. 2022 Int. Conf. Eng. Emerging Technol. (ICEET), 2022, doi: 10.1109/ICEET56468.2022.10007150.

[14] C. Zottmann, "iOS iCloud Drive synchronization deep dive," zottmann.org, Sep. 8, 2025. [Online]. Available: https://zottmann.org/2025/09/08/ios-icloud-drive-synchronization-deep.html

[15] F. Rosén, "Hacking CloudKit – How I accidentally deleted your Apple Shortcuts," Detectify Labs, Sep. 13, 2021. [Online]. Available: https://labs.detectify.com/writeups/hacking-cloudkit-how-i-accidentally-deleted-your-apple-shortcuts/

[16] P. Young, "A guide to CloudKit: How to sync user data across iOS devices," Toptal, Jan. 16, 2026. [Online]. Available: https://www.toptal.com/developers/ios/sync-data-across-devices-with-cloudkit

[17] W. Matlock, "72 syncs and counting: How PushTo100 survived CloudKit," Medium, Jun. 23, 2025. [Online]. Available: https://medium.com/@wesleymatlock/72-syncs-and-counting-how-pushto100-survived-cloudkit-66ed3069ae92

[18] C.-T. Huang, H.-J. Ko, Z.-W. Zhuang, P.-C. Shih, and S.-J. Wang, "Mobile forensics for cloud storage service on iOS systems," in Proc. Int. Symp. Inf. Theory Appl. (ISITA), 2018, pp. 178–182.

[19] G. Grispos, K.-K. R. Choo, and W. B. Glisson, "Sickly apps: A forensic analysis of medical device smartphone applications on Android and iOS devices," Mobile Netw. Appl., vol. 28, pp. 1282–1292, 2023, doi: 10.1007/s11036-022-02049-8.

[20] N. Malik, J. Chandramouli, P. Suresh, K. Fairbanks, L. Watkins, and W. H. Robinson, "Using network traffic to verify mobile device forensic artifacts," in Proc. 2017 14th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC), 2017, pp. 114–119, doi: 10.1109/CCNC.2017.7983088.

[21] A. Alblooshi, N. Aljneibi, F. Iqbal, R. Ikuesan, M. Badra, and Z. Khalid, "Smartphone forensics: A comparative study of common mobile phone models," in Proc. 2024 12th Int. Symp. Digital Forensics Secur. (ISDFS), 2024, doi: 10.1109/ISDFS60797.2024.10527262.

[22] L. Gómez-Miralles and J. Arnedo-Moreno, "Analysis of the forensic traces left by AirPrint in Apple iOS devices," in Proc. 2013 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA), 2013, doi: 10.1109/WAINA.2013.40.

[23] axi0mX, "ipwndfu: Public jailbreak tool for many iOS devices," GitHub, Sep. 27, 2019. [Online]. Available: https://github.com/axi0mX/ipwndfu

[24] G. Haas, S. Potluri, and A. Aysu, "iTimed: Cache Attacks on the Apple A10 Fusion SoC," in Proc. 2021 IEEE Int. Symp. Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 2021, pp. 80–90, doi: 10.1109/HOST49136.2021.9702290.

[25] Apple Inc., "Secure keychain syncing," Apple Platform Security, [Online]. Available: https://support.apple.com/guide/security/secure-keychain-syncing-sec0a319b35f/web.

[26] X. Yang, "How Core Data saves data in SQLite," fatbobman.com, Oct. 2, 2022. [Online]. Available: https://fatbobman.com/en/posts/tables_and_fields_of_coredata/

[27] C. Gartenberg, "Apple's STIR/SHAKEN implementation in iOS 13 is currently useless," The Verge, Sep. 20, 2019. [Online]. Available: https://www.theverge.com/2019/9/20/20875500/stir-shaken-robocalls-ios-13-apple-call-verified-currently-useless-iphone

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I, Mihkel Jõgeda grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Propagation of Artifacts Between iOS Devices", supervised by Matthew James Sorell

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2 I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3 I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

[1] The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.