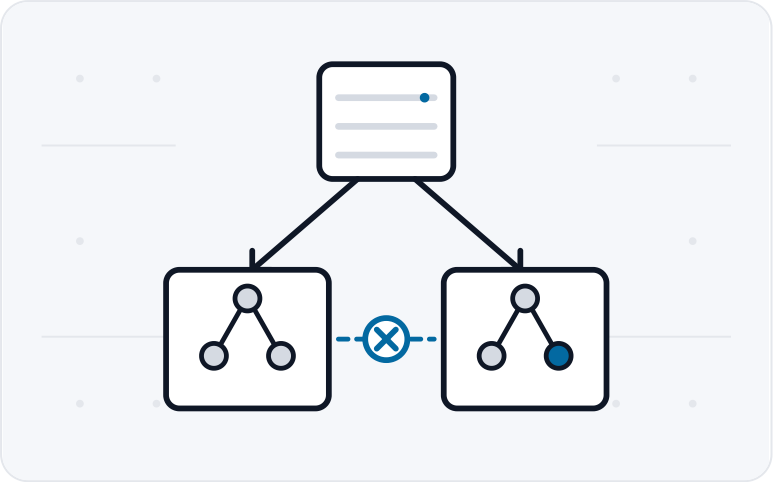
A=SIT Technology
E-Government Innovationszentrum

27. Februar 2025 Überblick · Transparenz · 18 Min. Lesezeit

Split-View-Angriffsschutz für Transparenzsysteme

Edona Fasllija



Autorinnen und Autoren	Edona Fasllija
Quelle	https://technology2.a-sit.at/veroeffentlichungen/schutz-vor-split-view-angriffen-fuer-transparenzsysteme/
Hinweis	Maschinell übersetzt.

Kernaussagen

- Transparency Logs sind append-only, schützen aber nicht vor Log-Equivocation: Ein bösartiger Log-Server kann Clients unterschiedliche Sichten präsentieren.
- Witnessing ist proaktiv: Witnesses prüfen Log-Aktualisierungen, bevor Clients sie akzeptieren. Dafür braucht es unabhängige, hochverfügbare Dritte.
- Gossiping ist reaktiv: Clients vergleichen Log-Daten wie STHs. Das erfordert breite Beteiligung; In-Band-Gossip braucht Protokolländerungen.
- Parakeet toleriert bis zu f fehlerhafte von 3f + 1 Witnesses; 2f + 1 Signaturen beglaubigen eine Aktualisierung, sodass Equivocation verhindert wird.
- Anonyme Abfragen, Blockchain-Verankerung und die Praxislösungen von Apple und WhatsApp ergänzen die Erkennung, jeweils mit eigenen Zielkonflikten.

Transparenzsysteme wie Certificate Transparency oder Key Transparency sollen die Nachvollziehbarkeit und Überprüfbarkeit von Daten gewährleisten. Sie sind jedoch anfällig für Split-View-Angriffe, bei denen Angreifer verschiedene Versionen von Daten manipulieren, um die Teilnehmer zu täuschen, indem sie widersprüchliche Ansichten des Systems liefern. Dieses Projekt konzentriert sich auf die Analyse von Werkzeugen und Mechanismen zur Erkennung, Abschwächung und Verhinderung von Split-View-Angriffen in Transparenzsystemen.

Im Rahmen dieses Projekts untersuchen wir Mechanismen zur Erkennung von Split-Views, wie z. B. Gossiping und verteilte Überwachungssysteme (Witnessing), die sicherstellen sollen, dass alle Teilnehmer dieselben Daten sehen. Außerdem bewerten wir die Wirksamkeit von Gossip-Protokollen und Witnessing zur Aufrechterhaltung der globalen Konsistenz im gesamten System.

1. Einleitung

Transparenzprotokolle (Transparency Logs) sind öffentliche, nur erweiterbare Aufzeichnungen (append-only), die kryptografische Beweise für ihren Inhalt und ihre Historie liefern. Sie spielen eine zentrale Rolle in Systemen, die öffentliche Rechenschaftspflicht erfordern, etwa Certificate Transparency (CT)[1] und Key Transparency (KT)[2], indem sie sicherstellen, dass alle Protokolldaten unveränderlich und für alle gleichermaßen sichtbar sind.

Dennoch sind Transparenzprotokolle anfällig für Log-Equivocation, auch Split-View-Angriffe genannt, bei denen ein bösartiger Log-Server verschiedenen Clients unterschiedliche Versionen des Protokolls präsentiert. Vorfälle solcher Angriffe wurden bei CT-Logs dokumentiert, wie in [3] und [4] festgehalten. Die Verhinderung von Equivocation setzt grundsätzlich voraus, dass alle dasselbe, globale Protokoll sehen, sodass Abweichungen früher oder später entdeckt werden.

Zur Bewältigung dieses Risikos haben sich zwei Hauptmechanismen herausgebildet: Witnessing und Gossiping. Witnessing ist ein proaktives Verfahren, bei dem vertrauenswürdige Dritte die Konsistenz eines Protokolls prüfen, bevor Clients einen Beweis akzeptieren. Gossiping dagegen ist ein reaktiver Ansatz, bei dem mehrere Instanzen regelmäßig ihre Sichten auf das Protokoll vergleichen, um Inkonsistenzen aufzudecken. Beide Ansätze können Fehlverhalten wirksam erkennen, bringen aber Herausforderungen mit sich. Witnessing hängt von der Verfügbarkeit und Vertrauenswürdigkeit externer Prüfinstanzen ab, während Gossiping eine breite Beteiligung erfordert. Diese Anforderungen auszubalancieren ist eine grundlegende Herausforderung bei der Entwicklung robuster Mechanismen zur Erkennung von Equivocation.

2. Ziele und Herausforderungen

Certificate Transparency war ursprünglich dazu gedacht, durch die öffentliche Aufzeichnung der Ausstellung von Zertifikaten die Notwendigkeit vertrauenswürdiger Dritter zu beseitigen [5]. Theoretisch entsteht so ein öffentliches Hauptbuch (Ledger), mit dem jede Person die Rechtmäßigkeit und Integrität von Zertifikaten unabhängig überprüfen kann, ohne auf zwischengeschaltete Stellen angewiesen zu sein. Neue Herausforderungen entstehen jedoch, wenn sich das Protokoll selbst bösartig verhält, indem es Einträge auslässt, Daten verändert oder verschiedenen Nutzenden unterschiedliche Sichten präsentiert (ein Split-View-Angriff).

Ein wirklich sicheres und verlässliches Transparenzsystem muss mehr leisten, als nur eine unveränderliche Aufzeichnung zu führen. Systeme wie Certificate Transparency (CT), Key Transparency (KT) und Binary Transparency (BT) [6] bieten zwar kryptografische Garantien, dass protokollierte Daten nachträglich nicht manipuliert werden können, sie stellen aber nicht automatisch sicher, dass Inkonsistenzen oder Angriffe rechtzeitig erkannt werden. Damit diese Systeme wirksam sind, müssen sie durchgängig (Ende-zu-Ende) transparent sein. Das bedeutet, zusätzliche Mechanismen zu integrieren, die den gesamten Verifikationszyklus abdecken. Dieser Abschnitt erläutert die wichtigsten Ziele und Herausforderungen beim Entwurf von Schutzmechanismen gegen Split-Views.

Ziele: Konsistenz: Das wichtigste Ziel ist die Garantie, dass jeder Client früher oder später denselben Zustand des Protokolls erhält. Diese „eine globale Sicht" bildet die Grundlage der Integrität des Systems. Sie stellt sicher, dass Abweichungen, sollten sie auftreten, durch den Vergleich unabhängig bezogener Protokoll-Snapshots aufgedeckt werden.

Rechenschaftspflicht (Accountability): Ein erfolgreicher Schutzmechanismus gegen Split-Views muss kryptografische Beweise liefern, die unabhängig geprüft werden können. Wenn ein Log-Server Equivocation betreibt, sollte das daraus entstehende Beweismaterial (Beweis des Fehlverhaltens) nicht fälschbar sein, sodass Dritte die sich fehlverhaltende Instanz zur Verantwortung ziehen können.

Rechtzeitige Erkennung bei geringem Overhead: Der Schutzmechanismus sollte Inkonsistenzen rechtzeitig erkennen und dabei den Rechen- und Kommunikationsaufwand für Clients minimal halten. So können auch mobile oder leistungsschwache Geräte am Auditing teilnehmen.

Resilienz: Der Entwurf des Schutzmechanismus sollte ein gewisses Maß an Fehlverhalten von Teilnehmenden tolerieren (zum Beispiel, wenn ein Teil der Witnesses fehlerhaft oder kompromittiert ist), ohne die Sicherheitsgarantien insgesamt zu untergraben.

Minimale Vertrauensannahmen: Der Schutz vor Split-Views kann dazu zwingen, sich auf zusätzliche Instanzen (z. B. Witnesses oder Auditoren) zu verlassen. Eine zentrale Herausforderung besteht darin, diese Vertrauensannahmen so zu begrenzen, dass das System auch dann sicher bleibt, wenn einige Teilnehmende kompromittiert sind oder sich bösartig verhalten.

Kurz gesagt: Die Unveränderlichkeit von Transparenzprotokollen ist zwar eine entscheidende Grundlage, ein sicheres Transparenzsystem muss sie aber mit Echtzeit-Überwachung, regelmäßigem Auditing und starken Abwehrmaßnahmen gegen Equivocation verbinden, um vor raffinierten Angriffen umfassend zu schützen.

3. Hintergrund

In diesem Abschnitt beschreiben wir die für diese Studie relevanten Bausteine.

3.1. Grundlagen

Nur erweiterbare Datenstruktur (Append-Only Data Structure): Im Kern ist eine nur erweiterbare Datenstruktur erforderlich, damit einmal aufgezeichnete Daten nicht ohne Entdeckung entfernt oder verändert werden können. Häufig dient ein Merkle-Baum dazu, große Datenmengen effizient zu speichern und zu verifizieren: Jeder Protokolleintrag wird durch einen Blattknoten repräsentiert, und der Wurzel-Hash fungiert als kompakte Festlegung (Commitment) auf das gesamte Protokoll. Weiterentwickelte Strukturen

wie Merkle-Patricia-Bäume und Sparse-Merkle-Bäume bieten zusätzliche Vorteile, etwa authentifizierte Schlüssel-Wert-Speicherung und effiziente Beweise der Nicht-Aufnahme, während Vektor-Commitments Daten an eine Festlegung binden können, die indizierte Beweise sowohl der Aufnahme als auch der Konsistenz unterstützt.

Kryptografische Beweise: Inclusion Proofs (Aufnahmebeweise) zeigen, dass ein bestimmter Eintrag in einem bestimmten Zustand im Protokoll enthalten ist, in der Regel mithilfe von Merkle-Beweisen. Consistency Proofs (Konsistenzbeweise) wiederum prüfen, dass ein neuer Protokollzustand einen früheren Zustand korrekt fortschreibt, und stellen so sicher, dass keine Einträge verändert oder entfernt wurden. Außerdem zeigen Non-Inclusion Proofs (Beweise der Nicht-Aufnahme), dass ein bestimmter Eintrag im Protokoll nicht vorhanden ist.

Checkpoints: In regelmäßigen Abständen erzeugt das Protokoll einen signierten Snapshot[7] seines Zustands, den Signed Tree Head (STH), der als Festlegung auf die aktuelle Merkle-Wurzel des Protokolls dient. Manche Systeme erweitern diesen Mechanismus um unabhängige Witnesses, die diese Checkpoints gegenzeichnen, sodass die Festlegung von mehreren vertrauenswürdigen Parteien bestätigt wird. In bestimmten Entwürfen werden diese Festlegungen sogar auf einer Blockchain verankert. Das liefert eine unveränderliche und manipulationssichere Aufzeichnung, die das Vertrauen in die Historie des Protokolls stärkt.

3.2. Auditing und Monitoring

Auditing und Monitoring sind zwei Schlüsselmechanismen in Transparenzsystemen, die jeweils eine eigene Rolle bei der Sicherstellung von Integrität und Konsistenz spielen. Beide zielen darauf ab, Fehlverhalten zu erkennen, unterscheiden sich aber in Umfang, Zeitpunkt und der Art, wie sie mit Nutzenden und dem Gesamtsystem interagieren.

Auditing ist ein punktueller, globaler und asynchroner Prozess, der das gesamte System bewertet und nicht einzelne Interaktionen von Nutzenden [8]. Auditoren prüfen die Transparenzprotokolle regelmäßig auf Inkonsistenzen, Verletzungen der Append-Only-Eigenschaft oder Versuche der Log-Equivocation. Dieser Prozess wird in der Regel von Dritten durchgeführt, etwa von unabhängigen Forschenden, Regulierungsbehörden oder Sicherheitsorganisationen, und nicht von einzelnen Nutzenden. Da Auditing asynchron ist, findet es nicht in Echtzeit statt, sondern in festgelegten Intervallen oder wenn es durch eine Verifikationsanfrage ausgelöst wird. In Certificate Transparency (CT) können Auditoren beispielsweise regelmäßig alle Signed Tree Heads (STHs) und Inclusion Proofs abrufen, um zu prüfen, dass keine Zertifikate entfernt oder verändert wurden. Ebenso könnten Auditoren in Key Transparency (KT) prüfen, dass Nutzenden keine widersprüchlichen Schlüsselhistorien ausgeliefert werden.

Monitoring ist dagegen synchron, nutzerbezogen und in Echtzeit. Statt das gesamte System auf einmal zu analysieren, konzentriert sich Monitoring auf die Prüfung von Aktualisierungen oder Änderungen, die eine einzelne nutzende Person betreffen. Eine nutzende Person oder ein Dienst überwacht das Transparenzprotokoll fortlaufend, um unbefugte Änderungen an den eigenen Zertifikaten, öffentlichen Schlüsseln oder anderen sicherheitskritischen Einträgen zu erkennen. Anders als Auditing, das im globalen Maßstab arbeitet, ist Monitoring auf das spezifische Interesse der nutzenden Person ausgerichtet, etwa eine Website, die sicherstellt, dass ihr TLS-Zertifikat nicht betrügerisch ausgestellt wurde, oder eine Messaging-App, die bestätigt, dass sich der öffentliche Schlüssel einer nutzenden Person nicht unerwartet geändert hat. Monitoring ist synchron, weil es häufig unmittelbar bei einer Aktualisierung stattfindet. In Key

Transparency könnte eine Messaging-App beispielsweise jedes Mal automatisch das Transparenzprotokoll prüfen, wenn eine nutzende Person den Verschlüsselungsschlüssel eines Kontakts abfragt, um sicherzustellen, dass er sich nicht unerwartet geändert hat. In CT könnte ein Domaininhaber ein automatisiertes System einrichten, das fortlaufend nach unbefugt für die eigene Domain ausgestellten Zertifikaten sucht und so sofortige Erkennung und Reaktion ermöglicht.

Beide Mechanismen sind unverzichtbar, um das Vertrauen in Transparenzsysteme zu erhalten: Auditing sorgt für langfristige Rechenschaftspflicht, Monitoring bietet proaktive Sicherheit auf Ebene der Nutzenden.

3.3. Schutz vor bösartigen Loggern / sich fehlverhaltenden Logs

Ein Transparenzprotokoll stellt sicher, dass Daten nur angehängt werden und öffentlich überprüfbar sind, schützt aber nicht automatisch vor Log-Equivocation, also einem Szenario, in dem ein Log-Betreiber verschiedenen Nutzenden unterschiedliche Sichten auf das Protokoll liefert (Abbildung 1). Ein bösartiger oder kompromittierter Log-Betreiber könnte einzelne Nutzende täuschen und gegenüber anderen ehrlich erscheinen, wenn ein System Daten lediglich in einem Protokoll ablegt, ohne Mechanismen zur Durchsetzung globaler Konsistenz. So könnte etwa eine Zertifizierungsstelle (CA) ein betrügerisches Zertifikat ausstellen und im Transparenzprotokoll erfassen, den Nachweis seiner Existenz aber nur bestimmten Parteien liefern, sodass das Opfer es nicht entdecken kann. Ebenso könnte in einem KT-System ein Angreifer verschiedenen Nutzenden unterschiedliche Verschlüsselungsschlüssel liefern und so Man-in-the-Middle-Angriffe (MITM) ermöglichen.

Sich fehlverhaltendes CT-Log Ein CT-Log kann sich auf verschiedene Weise fehlverhalten. Es könnte beispielsweise ein Zertifikat, das mit einem Signed Certificate Timestamp (SCT) verknüpft ist, nicht innerhalb der festgelegten Maximum Merge Delay (MMD) in den Merkle-Baum aufnehmen. Es kann auch zu verschiedenen Zeitpunkten oder gegenüber verschiedenen Parteien widersprüchliche Sichten auf den Merkle-Baum präsentieren, Signed Tree Heads (STHs) zu häufig ausgeben oder die Signatur eines protokollierten Zertifikats verändern.

Um Verstöße gegen die MMD-Vorgabe zu erkennen, können Log-Clients für jeden beobachteten SCT einen Inclusion Proof anfordern. Diese Prüfungen müssen nur einmal pro Zertifikat durchgeführt werden und können asynchron erfolgen. Sie helfen zu bestätigen, dass jede Aktualisierung ordnungsgemäß an den Baum angehängt wurde, auch wenn dieses Verfahren Datenschutzbedenken aufwerfen kann. Verstöße gegen die Append-Only-Eigenschaft oder gegen die Begrenzung der STH-Ausgaberate lassen sich dagegen erkennen, wenn mehrere Clients ihre jeweiligen STHs vergleichen, auch Gossip genannt. Hinweise auf Fehlverhalten können eine rasche Abfolge von STHs sein, die auf eine zu hohe Ausgaberate hindeutet, oder ein STH, dessen Wurzel-Hash nicht mit dem Hash übereinstimmt, der aus einer lokal gespeicherten Kopie des Protokolls berechnet wurde, was eine Verletzung der Append-Only-Garantie signalisiert.

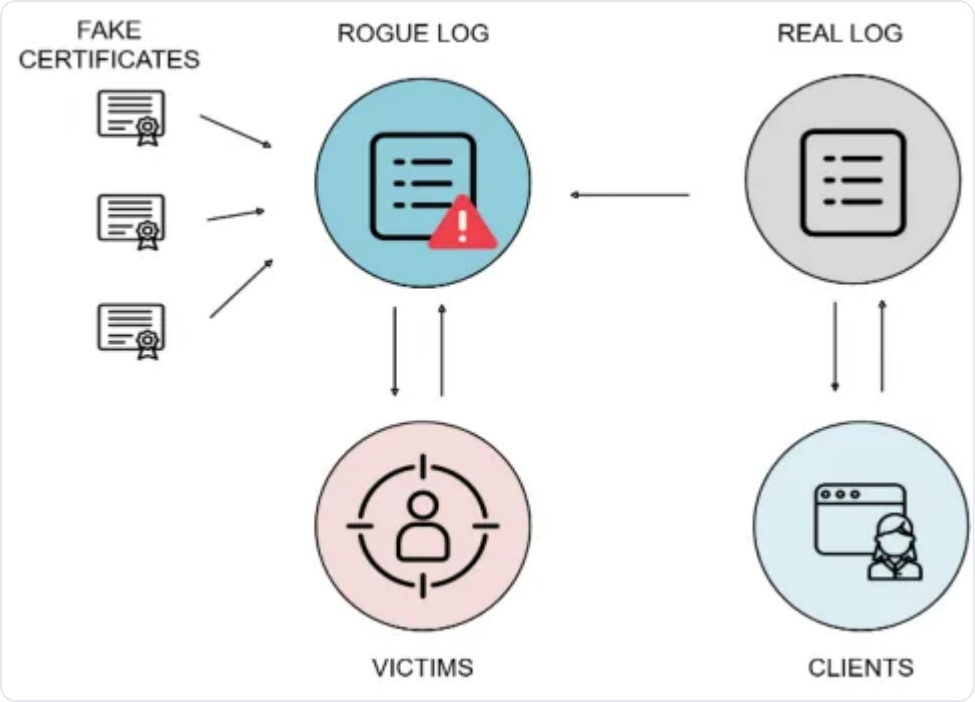


Abbildung 1: Split-View-Angriff bei sich fehlverhaltendem KT-Log

Ähnlich kann ein Log im Kontext von Key Transparency Equivocation betreiben, indem es verschiedenen Clients unterschiedliche Sichten auf das Schlüsselverzeichnis liefert. Obwohl das System auf einem nur erweiterbaren Merkle-Baum beruht, der jede Schlüsselaktualisierung aufzeichnet, könnte ein bösartiger oder kompromittierter Log-Server absichtlich voneinander abweichende Zustände präsentieren und einer nutzenden Person eine Version der Schlüsselzuordnungen zeigen, einer anderen dagegen eine widersprüchliche Version. Dieser Split-View-Angriff kann auftreten, wenn das Log bestimmte Aktualisierungen auslässt, umordnet oder verändert, bevor es einen neuen Signed Tree Head (STH) erzeugt.

4. Ansätze

Es gibt zwei Hauptansätze, um Fehlverhalten von Logs zu erkennen und abzuschwächen: Gossiping und Witnessing. Beide erfüllen eine eigene Rolle für Transparenz und Sicherheit im Log-Betrieb.

Witnessing ist ein proaktiver Mechanismus, der die Integrität des Logs prüft, bevor ein Beweis akzeptiert wird. Er bezieht Drittinstanzen ein, sogenannte Witnesses, die Logs unabhängig überwachen und die Konsistenz der Signed Tree Heads (STHs) und der Merkle-Inclusion-Proofs validieren, bevor Clients ihnen vertrauen. Indem Logs ihre Konsistenz und die Einhaltung der Append-Only-Eigenschaften nachweisen müssen, bevor ein Beweis akzeptiert wird, hilft Witnessing, Angriffe zu verhindern, bei denen ein Log seinen Zustand nachträglich verändert. Dieser Ansatz stellt sicher, dass jede Manipulation am Punkt der Verifikation entdeckt wird und nicht erst, nachdem ein Beweis bereits akzeptiert wurde.

Gossiping dagegen ist ein reaktives Protokoll, das Inkonsistenzen erkennt, nachdem Beweise akzeptiert wurden. Dabei tauschen mehrere Clients oder Instanzen Log-Daten wie STHs, Inclusion Proofs oder Consistency Proofs aus und vergleichen sie. Beobachten verschiedene Parteien widersprüchliche Sichten auf ein Log, können sie Fehlverhalten wie Equivocation erkennen (wenn ein Log verschiedenen Clients

unterschiedliche Versionen seines Zustands präsentiert). Gossiping ist besonders wirksam, um Verletzungen der Append-Only-Eigenschaft aufzudecken und sicherzustellen, dass alle Beteiligten eine konsistente Historie des Logs sehen.

Beide Mechanismen helfen, Angriffe durch Log-Equivocation zu erkennen, bringen aber jeweils eigene Herausforderungen mit sich. Witnessing erfordert die Beteiligung unabhängiger Dritter, denen vertraut werden muss, dass sie Log-Informationen korrekt prüfen und speichern. Das kann betriebliche Komplexität und Overhead verursachen. Gossiping hingegen beruht auf breiter Beteiligung und netzwerkweiter Zusammenarbeit. Da Gossiping zudem ein reaktiver Mechanismus ist, könnten manche Systeme so gestaltet sein, dass die Annahme eines Beweises aus dem Transparenzprotokoll das Gossiping verhindert. In solchen Fällen könnten Clients Fehlverhalten durch Gossiping nicht erkennen und wären anfälliger für Logs, die selektiv widersprüchliche Sichten präsentieren.

5. Schutzmechanismen

Dieser Abschnitt analysiert und vergleicht die wichtigsten vorgeschlagenen Schutzmechanismen für Gossiping und Witnessing. Außerdem wird anonymes Auditing als mögliche Lösung zur Erkennung von Equivocation-Angriffen behandelt.

5.1. Out-of-Band-Gossip

Gossip lässt sich je nach Art des Informationsaustauschs in In-Band- und Out-of-Band-Verfahren einteilen. Out-of-Band-Gossip läuft über separate, dedizierte Kommunikationskanäle, unabhängig vom primären Protokoll des Systems. Dieses Verfahren kann direkten Peer-to-Peer-Austausch, Überwachungsdienste Dritter oder verteilte Netzwerke umfassen. Ein gängiges Beispiel in CT ist ein eigenständiges Gossip-Netzwerk, in dem unabhängige Monitore, Auditoren oder Clients regelmäßig die von ihnen beobachteten STHs austauschen und vergleichen. Out-of-Band-Gossip bietet mehr Flexibilität und lässt sich ohne Änderung bestehender Protokolle umsetzen. Es ermöglicht außerdem die Einbindung dedizierter Monitore und Witnesses und verbessert so die Erkennung von Fehlverhalten, selbst wenn reguläre Clients nicht am Gossiping teilnehmen. Dieser Ansatz erfordert jedoch zusätzliche Infrastruktur, etwa externe Gossip-Netzwerke, und kann höhere Kommunikations- und Rechenkosten verursachen. Da der Datenaustausch weniger häufig stattfindet als beim In-Band-Gossip, kann die Erkennung zudem langsamer sein.

5.2. In-Band-Gossip (Piggybacked Gossip)

In-Band-Gossip findet innerhalb der normalen Kommunikationskanäle des Systems statt. Das bedeutet, dass logbezogene Informationen (wie Signed Tree Heads (STHs) und Inclusion Proofs) im Rahmen routinemäßiger Protokollinteraktionen ausgetauscht werden. In CT könnten beispielsweise Webbrowser und TLS-Clients STHs in ihre HTTPS-Anfragen aufnehmen, sodass Server sie vergleichen und Inkonsistenzen erkennen können [9]. Ebenso könnten TLS-Server SCTs und STHs in ihre Antworten aufnehmen und so die Verifikation durch Clients erleichtern.

Eine Messaging-App (z. B. Signal, WhatsApp) mit Key Transparency könnte Gossip-Nachrichten in verschlüsselten Nachrichten mitsenden. Wenn Alice eine Nachricht an Bob sendet, könnte die App Gossip-Nachrichten anhängen, darunter STHs, die verifizieren, dass sich Bobs öffentlicher Schlüssel nicht

unerwartet geändert hat. Darüber hinaus könnte ein E-Mail-Anbieter Key-Transparency-Beweise in E-Mail-Headern einbetten, damit Empfangende für die verschlüsselte Kommunikation den erwarteten öffentlichen Schlüssel erhalten.

Der Vorteil von In-Band-Gossip besteht darin, dass die Verifikation bei der normalen Nutzung automatisch erfolgt. Das verringert den Overhead und stellt sicher, dass Nutzende keine separaten Verifikationsschritte benötigen. Da Clients ständig Zertifikate oder Schlüssel-Inclusion-Proofs prüfen, lässt sich Equivocation schnell erkennen. Dieser Ansatz erfordert jedoch Änderungen an bestehenden Protokollen und setzt eine breite Übernahme durch Clients und Server voraus.

Der beste Ansatz hängt vom Bedrohungsmodell und der Einsatzumgebung ab:

In-Band-Gossip ist für eine breite Einführung praktikabler, da es sich in den normalen Protokollbetrieb einfügt, hängt aber von Protokolländerungen und der Beteiligung der Clients ab. Out-of-Band-Gossip eignet sich besser für sicherheitskritische Anwendungen, in denen eine unabhängige Verifikation nötig ist, selbst wenn reguläre Clients nicht aktiv am Gossiping teilnehmen.

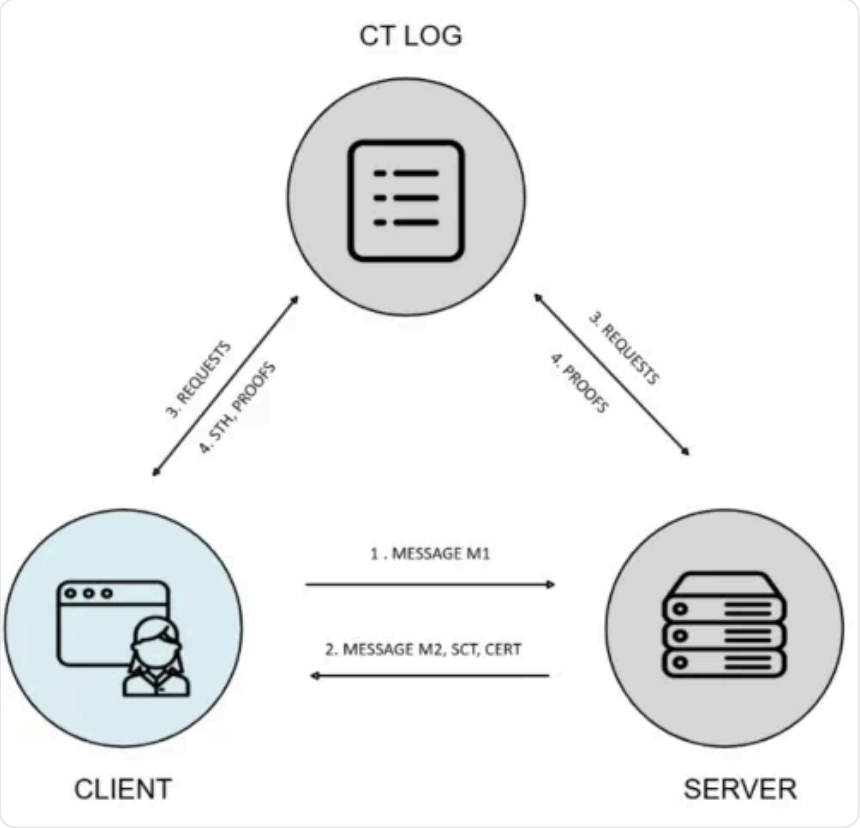


Abbildung 2 In-Band-Gossip-Mechanismus für CT

Gossip-Art	Kommunikations-kanal	Beispiel in CT	Beispiel in KT	Vorteile	Nachteile
In-Band	Im normalen Protokoll integriert	Gossip im HTTPS-Protokoll mitsenden	Gossip-Nachrichten in Nachrichten mitsenden	Effizient, geringer Overhead, automatisch	Erfordert Protokoll-änderungen
Out-of-Band	Separate Kommunikations-kanäle, unabhängige Audits	Clients, externe Monitore, Logs vergleichen Log-Zustände über einen separaten Kanal	Clients, externe Monitore, Logs vergleichen Log-Zustände über einen separaten Kanal	Funktioniert plattformübergreifend, erkennt Fehlverhalten auch bei blockiertem In-Band-Gossip	Erfordert zusätzliche Infrastruktur, Erkennung kann langsamer sein

5.3. Witnessing

Witnessing bezieht mehrere unabhängige Parteien ein, die das Log beobachten und sicherstellen, dass alle Beteiligten dieselbe Version des Logs sehen. Witnesses prüfen, dass Logs nur erweitert werden und global konsistent sind, und verhindern so unbefugte Änderungen und Split-View-Angriffe [10]. Sie prüfen Update-Beweise und zeichnen Log-Checkpoints gegen. Nach jeder Aktualisierung benachrichtigt der Log-Server die Witnesses mit einer signierten Nachricht, die den vorherigen und den aktualisierten Wurzel-Hash, die Epochennummer und einen aktualisierten Beweis enthält [11]. Die Witnesses validieren den Beweis und bestätigen das Fortschreiten der Epoche, bevor sie die Aktualisierung signieren.

Witnessing-Mechanismen sind widerstandsfähig in feindlichen Umgebungen; Parakeet[11] setzt beispielsweise 3f + 1 Witnesses ein und toleriert bis zu f fehlerhafte Witnesses. Mindestens 2f + 1 Witnesses müssen dieselbe Update-Benachrichtigung signieren, um eine Aktualisierung zu beglaubigen. Selbst im schlimmsten Fall, in dem fehlerhafte Witnesses versuchen, Signaturen auf widersprüchliche Aktualisierungen aufzuteilen, kann nur eine davon die erforderlichen 2f + 1 Signaturen erreichen, wodurch Equivocation verhindert wird.

Clients akzeptieren nur Aktualisierungen, die von einem gültigen Zertifikat begleitet werden, das sie anhand der Witness-Signaturen verifizieren. Sie können außerdem Witnesses nach früheren Wurzel-Hashes und Zertifikaten fragen und so eine konsistente Baumhistorie sicherstellen, ohne eine Hash-Kette zu benötigen. Um zu verhindern, dass ein bösartiger Dienstanbieter Anfragen zur Schlüsselaktualisierung ignoriert, können Clients ihre Anfragen an Witnesses eskalieren. Nimmt der Anbieter eine Aktualisierung nicht auf, verweigern die Witnesses die Signatur künftiger Aktualisierungen. Damit wird es unmöglich, ein gültiges Zertifikat auszustellen, und der Anbieter wird zur Einhaltung gezwungen.

5.4. Anonyme Kommunikation

Statt sich auf direktes Gossip zwischen Clients zu verlassen, um Inkonsistenzen im Transparenzprotokoll zu erkennen, besteht ein strukturierterer Ansatz darin, das Protokoll über zwei unabhängige Kanäle abzufragen:

Authentifizierter Kanal: Jeder Client fragt das Transparenzprotokoll über einen authentifizierten Kanal ab. Der Server kennt die Identität des Clients und kann seine Antwort entsprechend anpassen.

Anonymer Kanal: Der Client fragt denselben STH separat über einen anonymen Kanal ab, der die Identität des Clients vor dem KT-Server verbirgt. Das könnte über einen Proxy, ein Mix-Netzwerk oder einen Anonymisierungsdienst wie Tor geschehen. Da die Anfrage anonym ist, kann der Server seine Antwort nicht anhand der Identität des Clients anpassen.

Durch den Vergleich der über beide Kanäle erhaltenen STHs kann der Client Abweichungen feststellen. Verhält sich der KT-Server ehrlich, sollten beide Antworten übereinstimmen. Betreibt der Server jedoch Equivocation und präsentiert verschiedenen Clients unterschiedliche STHs, wird der Unterschied sichtbar, sobald die authentifizierte Antwort von der anonym abgerufenen abweicht.

Dieser Ansatz stärkt Transparenz und die Erkennung von Equivocation, ohne direkte Kommunikation zwischen Clients zu erfordern. Dadurch ist er besser skalierbar und widerstandsfähiger gegen Angriffe, bei denen ein bösartiger Server das Log selektiv für verschiedene Nutzende aufspaltet.

5.5. Blockchain-basierte Lösungen

Eine weitere mögliche Lösung für die Umsetzung dieses öffentlichen schwarzen Bretts für Signed Tree Heads sind Blockchains. In diesen Vorschlägen [12] werden die Zustandsaktualisierungen des Logs auf einer öffentlichen Blockchain aufgezeichnet. Da Blockchain-Einträge unveränderlich und global sichtbar sind, dienen sie als externe Festlegungen auf die Historie des Logs. Blockchains bieten von Haus aus ein einziges, nur erweiterbares Hauptbuch, in dem alle Beteiligten die Historie der Log-Aktualisierungen überprüfen können. Sie können jedoch unter hoher Latenz und begrenztem Durchsatz leiden, und die Notwendigkeit, dass Clients mit einer Blockchain interagieren, kann die Einführung dieser Lösungen erschweren.

5.6. Praktische Einsätze

Apples Contact Key Verification: Im Gegensatz zu CT-Einsätzen und dem KT-System von WhatsApp, die auf externe Parteien angewiesen sind, um die Konsistenz ihrer gesamten Datenstrukturen zu validieren, führt die Nachrichten-App Prüfungen auf dem Gerät durch, um sicherzustellen, dass die kritischen nur erweiterbaren Logs im KT-System von Apple konsistent bleiben. Um außerdem Split-View-Angriffe aufzudecken, die auftreten könnten, wenn ein Angreifer den KT-Dienst kompromittiert, bettet die App Log-Hashes in den verschlüsselten Teil einer kleinen Teilmenge von Nachrichten ein. Diese Hashes werden dann mit anderen iMessage-Clients geteilt, die unabhängig prüfen, ob die empfangenen Log-Hashes übereinstimmen, und so jede Inkonsistenz in Echtzeit erkennen [13].

WhatsApp Key Transparency: Die Key-Transparency-Lösung von WhatsApp basiert auf Parakeet und sieht daher ein Netzwerk verteilter Witnesses vor, die hochverfügbar sein müssen, um die Update-Beweise zu auditieren (kryptografische Belege, dass jede Aktualisierung dem Baum nur Blätter hinzufügt, ohne bestehende zu verändern oder zu löschen). Außerdem spielen diese Witnesses eine zentrale Rolle im Konsistenzprotokoll, indem sie verifizieren, dass der Dienstanbieter allen Nutzenden eine identische Historie präsentiert, und so Equivocation verhindern. In der Praxis nutzt ihr Ansatz einen AWS-S3-Bucket mit einem WORM-Modell (Write Once, Read Many) mit einer Aufbewahrungsfrist von 5 Jahren. [2] 6. Schlussfolgerungen:

Split-View-Angriffe bleiben eine erhebliche Bedrohung für Transparenzprotokoll-Systeme. Es gibt zwar verschiedene Schutzmechanismen, doch jeder bringt seine eigenen Zielkonflikte mit sich. Ob Out-of-Band oder In-Band (Piggybacked): Gossip-Protokolle beruhen darauf, dass Clients Log-Snapshots (wie Signed Tree Heads oder Inclusion Proofs) austauschen, um zu verifizieren, dass alle denselben Log-Zustand sehen. Out-of-Band-Gossip nutzt dedizierte Kanäle, getrennt vom regulären Anwendungsverkehr, während In-Band-Gossip die Log-Daten in die normale Kommunikation einbettet. Beide Verfahren haben den Vorteil der Dezentralisierung, erfordern aber eine breite Beteiligung und verlässliche Kommunikationskanäle, um wirksam zu sein.

Witnessing hingegen bezieht eine Gruppe unabhängiger, hochverfügbarer Drittinstanzen (Witnesses oder Auditoren) ein, die Log-Aktualisierungen proaktiv validieren, bevor Clients sie akzeptieren. Dieser Ansatz kann unmittelbare, unabstreitbare Belege für Fehlverhalten liefern, führt aber zusätzliche Vertrauensannahmen hinsichtlich der Ehrlichkeit der Witnesses ein und erfordert zusätzliche betriebliche Koordination.

Blockchain-basierte Lösungen nutzen die Unveränderlichkeit und den globalen Konsens von Blockchains, um den Zustand des Logs zu verankern. Durch die Veröffentlichung von Log-Festlegungen (wie Wurzel-Hashes) auf einer Blockchain stellen diese Ansätze sicher, dass die Historie des Logs nach der Aufzeichnung nur schwer verändert werden kann. Blockchains können jedoch erhebliche Latenz und höhere Rechen- oder Finanzkosten verursachen und wegen ihres öffentlichen Charakters Datenschutzbedenken aufwerfen.

Anonyme Abfragen erweitern den Werkzeugkasten zur Erkennung um eine weitere Dimension. Bei diesem Ansatz stellt ein Client zwei Abfragen für dieselben Log-Daten, eine über einen authentifizierten und eine über einen anonymen Kanal (z. B. über Tor oder ein VPN). Durch den Vergleich der Antworten kann der Client erkennen, ob der Log-Server auf die Identität des Clients zugeschnittene oder inkonsistente Sichten liefert. Diese Methode setzt voraus, dass das Log beide Kanalarten unterstützt, und hängt von der Verfügbarkeit robuster anonymer Netzwerke ab. Sie erfordert keine zusätzliche Infrastruktur über das hinaus, was Clients bereits für Anonymität nutzen, und ergänzt andere Techniken, indem sie identitätsabhängiges Fehlverhalten direkt isoliert.

Literatur

[1] “Certificate Transparency : Certificate Transparency.” Accessed: Nov. 11, 2024. [Online]. Available: https://certificate.transparency.dev/

[2] “Deploying key transparency at WhatsApp - Engineering at Meta.” Accessed: Nov. 11, 2024. [Online]. Available: https://engineering.fb.com/2023/04/13/security/whatsapp-keytransparency/

[3] “Upcoming Log Removal: Venafi CT Log Server.” Accessed: Feb. 06, 2025. [Online]. Available: https://groups.google.com/a/chromium.org/g/ct-policy/c/KMAcNT3asTQ

[4] “Upcoming CT Log Removal: Izenpe.” Accessed: Feb. 06, 2025. [Online]. Available: https://groups.google.com/a/chromium.org/g/ct-policy/c/qOorKuhL1vA

[5] H. Leibowitz, E. Syta, and A. Herzberg, “CTng: Secure Certificate and Revocation Transparency In God we Trust; Loggers we Monitor”.

[6] “Binary Transparency.” Accessed: Feb. 11, 2025. [Online]. Available: https://binary.transparency.dev/

[7] “C2SP/tlog-checkpoint.md at main · C2SP/C2SP.” Accessed: Feb. 11, 2025. [Online]. Available: https://github.com/C2SP/C2SP/blob/main/tlog-checkpoint.md

[8] “From Witnessing to Transparent Ecosystems | Trillian.” Accessed: Feb. 11, 2025. [Online]. Available: https://transparency.dev/summit2024/witness.html

[9] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri, “Efficient gossip protocols for verifying the consistency of Certificate logs,” *2015 IEEE Conference on Communications and NetworkSecurity, CNS 2015*, pp. 415–423, Dec. 2015, doi: 10.1109/CNS.2015.7346853.

[10] S. Meiklejohn *et al.*, “Think Global, Act Local: Gossip and Client Audits in Verifiable Data Structures,” Nov. 2020, Accessed: Feb. 11, 2025. [Online]. Available: https://arxiv.org/abs/2011.04551v1

[11] H. Malvai *et al.*, “Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging,” *Cryptology ePrint Archive*, 2023, doi: 10.14722/NDSS.2023.24545.

[12] M. Al-Bassam and S. Meiklejohn, “Contour: A Practical System for Binary Transparency,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11025 LNCS, pp. 94–110, 2018, doi: 10.1007/978-3-030-00305-0_8/FIGURES/1.

[13] “Blog - Advancing iMessage security: iMessage Contact Key Verification - Apple Security Research.” Accessed: Feb. 10, 2025. [Online]. Available: https://security.apple.com/blog/imessage-contact-key-verification/