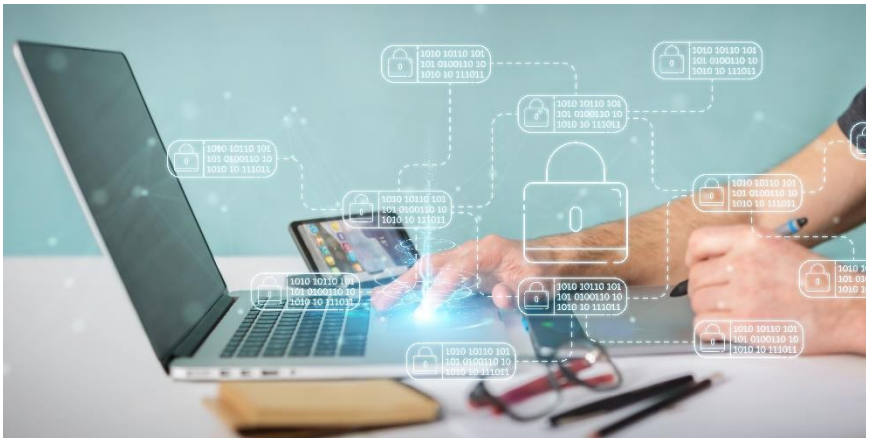
A-SIT

Secure Information Technology Center – Austria

Split-View Attack Protection for Transparency Systems



Split-View Attack Protection for Transparency Systems

Autor:
Edona Fasllija
Mail:edona.fasllija@iaik.tugraz.at

Transparency Systems, such as Certificate Transparency or Key Transparency, are designed to provide accountability and verifiability of data. However, they are vulnerable to split-view attacks, where adversaries manipulate different versions of data to deceive participants by providing inconsistent views of the system. This project focuses on analyzing tools and mechanisms to detect, mitigate, and prevent split-view attacks in transparency systems.

As part of this project, we explore split-view detection mechanisms, such as gossiping and distributed monitoring frameworks (witnessing), that aim to ensure all participants view the same data. Additionally, we evaluate the effectiveness of gossip protocols and witnessing to maintain global consistency across the system.

1. Introduction

Transparency logs are public, append-only records that provide cryptographic proof verifying both their content and history. They play a critical role in systems requiring public accountability—such as Certificate Transparency (CT)[1] and Key Transparency (KT)[2]by ensuring that all log data is immutable and uniformly visible.

Nevertheless, transparency logs are vulnerable to log equivocation, or split-view attacks, where a malicious log server presents different versions of the log to different clients. Incidents of such attacks have been documented in CT logs, as noted in [3]and [4]. Preventing equivocation fundamentally requires that everyone views a single, global log—so that any discrepancies are eventually detected.

Two primary mechanisms have emerged to address this risk: witnessing and gossiping. Witnessing is a proactive method in which trusted third parties validate a log's consistency before clients accept a proof. In contrast, gossiping is a reactive approach whereby multiple entities periodically compare their views of the log to expose any inconsistencies. While both approaches can effectively detect misbehavior, they come with challenges. Witnessing relies on the availability and trustworthiness of external auditors, whereas gossiping demands broad participation. Balancing these requirements is a fundamental challenge in developing robust equivocation detection mechanisms.

2. Goals and Challenges

Certificate Transparency was initially envisioned to remove the need for trusted third parties [5] by publicly recording certificate issuance events. In theory, this is a public ledger that allows anyone to independently verify the legitimacy and integrity of certificates without relying on intermediary authorities. However, new challenges emerge if the log itself behaves maliciously—by omitting entries, altering data, or presenting different views to different users (a split-view attack).

A truly secure and reliable transparency system must do more than simply maintain an immutable record. Although systems like Certificate Transparency (CT), Key Transparency (KT), and Binary Transparency (BT) [6]provide cryptographic guarantees that logged data cannot be tampered with after the fact, they do not automatically ensure that inconsistencies or attacks are detected in a timely manner. For these systems to be effective, they need to be truly *end-to-end transparent*. This means integrating additional mechanisms that cover the entire verification lifecycle. This section elaborates on the key goals and challenges related to designing split-view protection mechanisms.

Goals:

Consistency: The foremost goal is to guarantee that every client eventually receives the same state of the log. This "one global view" underpins the system's integrity, ensuring that even if discrepancies arise, they will be exposed through comparisons between independently obtained log snapshots.

Accountability: A successful split-view protection mechanism must provide *cryptographic proofs* that can be audited independently. If a log server equivocates, the resulting evidence (*proof of misbehavior*) should be unforgeable, enabling third parties to hold the misbehaving entity accountable.

Timely Detection with Low Overhead: The protection mechanism should detect inconsistencies timely while imposing minimal computational and communication overhead on clients. This ensures that even mobile or low-power devices can participate in the auditing process.

Resilience: The design of the protection mechanism should tolerate some degree of misbehavior from participants (for example, a fraction of witnesses being faulty or compromised) without undermining the overall security guarantees.

Minimal Trust Assumptions: Protecting against split views may force reliance on additional entities (e.g., witnesses or auditors). A key challenge is to limit these trust assumptions such that the system remains secure even if some participants are compromised or behave maliciously.

In short, while the immutable nature of transparency logs is a critical foundation, a secure transparency system must combine that with real-time monitoring, regular auditing, and strong equivocation defenses to fully protect against sophisticated attacks.

3. Background

In this section, we describe the building blocks relevant to this study.

3.1. Preliminaries

Append-Only Data Structure: At the core, an append-only data structure is essential to ensure that once data is recorded, it cannot be removed or altered without detection. For example, a Merkle tree is often used to efficiently store and verify large sets of data, where each log entry is represented by a leaf node, and the root hash acts as a compact commitment to the entire log. More advanced structures, such as Merkle Patricia Trees and Sparse Merkle Trees, provide additional benefits like authenticated key-value storage and efficient proofs of non-inclusion, while vector commitments can be used to bind data to a commitment that supports indexed proofs of both inclusion and consistency.

Cryptographic Proofs: Inclusion proofs demonstrate that a specific entry exists in the log at a particular state, typically using Merkle proofs. Consistency proofs, on the other hand, verify that a new log state properly extends a previous state, thereby ensuring that no entries have been modified or removed. Moreover, non-inclusion proofs show that a particular entry is absent from the log.

Checkpoints: Periodically, the log generates a signed snapshot[7] of its state—a Signed Tree Head (STH)—which serves as a commitment to the log's current Merkle root. Some systems enhance this mechanism by involving independent witnesses who sign off on these checkpoints, ensuring that the commitment is corroborated by multiple trusted parties. In certain designs, these commitments are even anchored on a blockchain, providing an immutable and tamper-resistant record that bolsters trust in the log's history.

3.2. Auditing and Monitoring

Auditing and monitoring are two key mechanisms in transparency systems, each serving a distinct role in ensuring integrity and consistency. While both aim to detect misbehavior, they differ in scope, timing, and how they interact with users and the system as a whole.

Auditing is a point-in-time, global, and asynchronous process that evaluates the entire system rather than individual user interactions [8]. Auditors periodically review the transparency logs, checking for inconsistencies, violations of the append-only property, or attempts at log equivocation. This process is typically performed by third parties, such as independent researchers, regulatory bodies, or security organizations, rather than by individual users. Because auditing is asynchronous, it does not occur in real-time but rather at scheduled intervals or when triggered by a verification request. For example, in Certificate Transparency (CT), auditors may periodically fetch all Signed Tree Heads (STHs) and inclusion proofs to verify that no certificates have been removed or altered. Similarly, in Key Transparency (KT), auditors might check that users are not being served conflicting key histories.

Monitoring, in contrast, is synchronous, per-user, and real-time. Rather than analyzing the entire system at once, monitoring focuses on verifying updates or changes that affect an individual user. A user or service continuously monitors the transparency log to detect unauthorized modifications to their own certificates, public keys, or other security-critical records. Unlike auditing, which operates on a global scale, monitoring is localized to the user's specific interest—such as a website ensuring its TLS certificate has not been fraudulently issued or a messaging app confirming that a user's public key has not changed unexpectedly. Monitoring is synchronous because it often happens immediately when an update is made. In Key Transparency, for example, a messaging app could automatically check the transparency log every time a user looks up a contact's encryption key, ensuring that it has not changed

unexpectedly. In CT, a domain owner might set up an automated system that continuously watches for any unauthorized certificates issued for their domain, allowing for immediate detection and response.

Both mechanisms are essential for maintaining trust in transparency systems—auditing provides long-term accountability while monitoring offers proactive security at the user level.

3.3. Protection against rogue loggers/ Misbehaving Logs

A transparency log ensures that data is append-only and publicly verifiable, but it does not automatically protect against *log equivocation*—a scenario where a log operator serves different views of the log to different users (Figure 1). A malicious or compromised log operator could deceive individual users while appearing honest to others if a system merely stores data in a log without mechanisms to enforce global consistency. For example, a certificate authority (CA) could issue a fraudulent certificate and include it in the transparency log but only serve proof of its existence to certain parties, preventing the victim from detecting it. Similarly, in a KT system, an attacker could serve different encryption keys to different users, facilitating man-in-the-middle (MITM) attacks.

Misbehaving CT Log

A CT log may misbehave in several ways. For instance, it might fail to incorporate a certificate linked to a Signed Certificate Timestamp (SCT) into the Merkle Tree within the designated Maximum Merge Delay (MMD). It may also present conflicting views of the Merkle Tree at different times or to different parties, issue Signed Tree Heads (STHs) too frequently or alter the signature of a logged certificate.

To detect violations of the MMD contract, log clients can request inclusion proof for each observed SCT. These checks, which need to be performed only once per certificate and can occur asynchronously, help confirm that each update has appropriately been appended to the tree, although this process might raise privacy concerns. In contrast, breaches of the append-only property or the STH issuance rate limit can be identified when multiple clients compare their respective STHs—a.k.a. *gossip*. Evidence of misbehavior might be a rapid succession of STHs, indicating an excessive issuance rate, or an STH whose root hash does not match the one computed from a locally stored copy of the log, signaling a violation of the append-only guarantee.

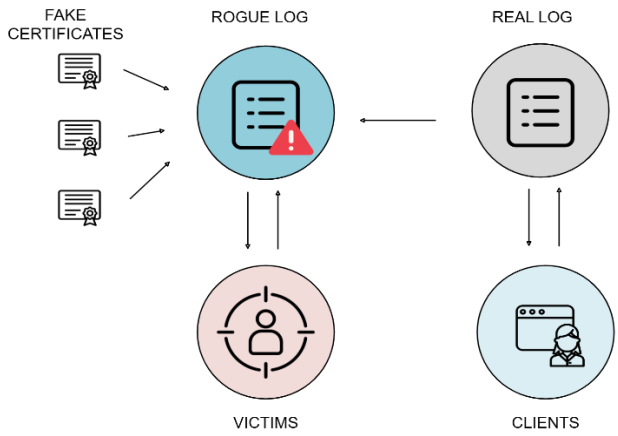


Figure 1 Split View Attack

Misbehaving KT Log

Similarly, in the context of Key Transparency, a log can equivocate by providing different views of the key directory to different clients. Although the system is designed around an append-only Merkle tree that records every key update, a malicious or compromised log server might deliberately present divergent states—showing one version of the key bindings to one user while presenting a conflicting version to another. This split-view attack can occur if the log omits, reorders, or alters certain updates before producing a new Signed Tree Head (STH).

4. Approaches

There are two primary approaches to detecting and mitigating log misbehavior: gossiping and witnessing. Each serves a distinct role in ensuring transparency and security in log operations.

Witnessing is a *proactive* mechanism that verifies log integrity before accepting a proof. It involves third-party entities, known as witnesses, that independently monitor logs and validate the consistency of the Signed Tree Heads (STHs) and Merkle inclusion proofs before they are trusted by clients. By requiring logs to demonstrate consistency and adherence to append-only properties before proof acceptance, witnessing helps prevent attacks where a log retroactively alters its state. This approach ensures that any tampering is detected at the point of verification rather than after a proof has already been accepted.

Conversely, *gossiping* is a reactive protocol that detects inconsistencies after proofs have been accepted. It works by having multiple clients or entities exchange and compare log data—such as STHs, inclusion proofs, or consistency proofs. If different parties observe conflicting views of a log, they can detect misbehavior, such as equivocation (where a log presents different versions of its state to different clients). Gossiping is particularly effective in catching violations of the append-only property and ensuring that all participants see a consistent history of the log.

While both mechanisms help detect log equivocation attacks, they come with their own challenges. Witnessing requires participation from independent third parties who must be trusted to verify and store log information correctly. This can introduce operational complexity and overhead. Gossiping, on the other hand, relies on broad participation and network-wide cooperation. Moreover, since gossiping is a reactive mechanism, some systems may be designed in a way that accepting proof from the transparency log prevents gossiping. In such cases, clients would be unable to detect misbehavior through gossiping, making them more vulnerable to logs that present conflicting views selectively.

5. Protection Mechanisms

This section analyzes and compares the main protection mechanism that has been proposed for gossiping and witnessing. It also elaborates on anonymous auditing as a potential solution to help detect equivocation attacks.

5.1. Out of Band Gossip

Gossip can be categorized into in-band and out-of-band methods, depending on how the information is exchanged. *Out-of-Band Gossip* occurs through separate, dedicated communication channels, independent of the system's primary protocol. This method can involve direct peer-to-peer exchanges, third-party monitoring services, or distributed networks. A common example in CT is a standalone gossip network where independent monitors, auditors, or clients periodically share and compare the STHs they have observed. Out-of-band gossip offers more flexibility and can be implemented without modifying existing protocols. It also enables the involvement of dedicated monitors and witnesses, improving the

detection of misbehavior even if regular clients do not participate in gossiping. However, this approach requires additional infrastructure, such as external gossip networks, and may have higher communication and computational costs. Since data exchanges do not happen as frequently as in in-band gossip, detection may also be slower.

5.2. In band Gossip (Piggybacked Gossip)

In-band gossip occurs within the normal communication channels of the system, meaning that log-related information (such as Signed Tree Heads (STHs) and inclusion proofs) is exchanged as part of routine protocol interactions.
For example, in CT, Web browsers and TLS clients could include STHs in their HTTPS requests, allowing servers to compare and detect inconsistencies [9]. Similarly, TLS servers could include SCTs and STHs in their responses, facilitating client verification.

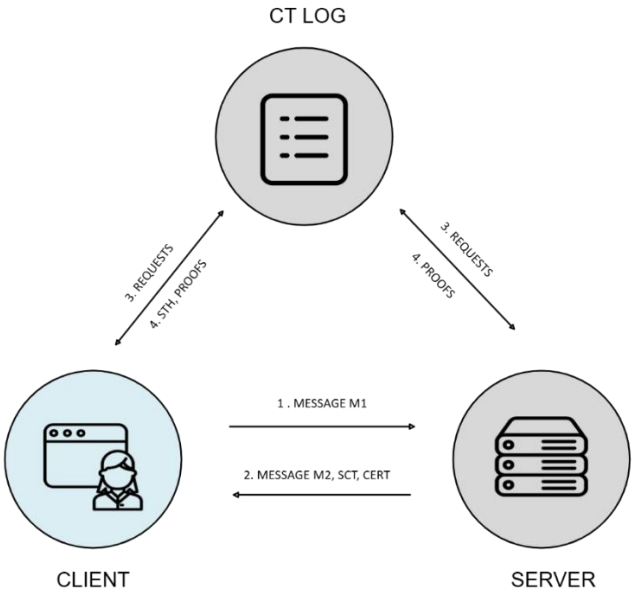


Figure 2 In-Band Gossip Mechanism for CT

A messaging app (e.g., Signal, WhatsApp) using Key Transparency could include gossip messages in encrypted messages. When Alice sends a message to Bob, the app could attach gossip messages, including STHs verifying that Bob's public key hasn't changed unexpectedly.
Furthermore, an email provider could embed key transparency proofs in email headers to ensure that recipients receive the expected public key for encrypted communication.

The advantage of in-band gossip is that verification happens automatically during normal use, reducing overhead and ensuring users do not need separate verification steps. Since clients constantly check certificates or key inclusion proofs, equivocation can be detected quickly. However, this approach requires modifications to existing protocols and relies on widespread adoption by clients and servers.

The best approach depends on the threat model and deployment environment:

In-band gossip is more practical for wide-scale adoption since it integrates with normal protocol operations, but it depends on protocol modifications and client participation. Out-of-band gossip is more suitable for security-critical applications where independent verification is needed, even if regular clients do not actively participate in gossiping.

Gossip Type	Communication Channel	Example in CT	Example in KT	Pros	Cons
In-Band	Integrated within the normal protocol	Piggybacking gossip to the HTTPS protocol	Piggybacking gossip messages in messages	Efficient, low overhead, automatic	Requires protocol modifications
Out-of-Band	Separate communication channels, independent audits	Clients, third-party monitors, logs comparing log states through separate channel	Clients, third-party monitors, logs comparing log states through a separate channel	Works across platforms, detects misbehavior even if in-band is blocked	Requires extra infrastructure, detection may be slower

5.3. Witnessing

Witnessing involves multiple independent parties who observe the log and ensure that all participants see the same version of the log. Witnesses verify that logs are *append-only* and *globally consistent*, preventing unauthorized changes and split-view attacks [10]. They check update proofs and cosign log checkpoints. After each update, the log server notifies witnesses with a signed message containing the previous and updated root hashes, epoch number, and an updated proof [11]. Witnesses validate the proof and confirm the epoch's progression before signing the update.

Witnessing mechanisms are resilient against adversarial settings; For example, Parakeet[11] employs *3f + 1* witnesses, tolerating up to f faulty witnesses. At least 2f + 1 witnesses must sign the same update notification to certify an update. Even in the worst-case scenario, where faulty witnesses attempt to split signatures across conflicting updates, only one can reach the required 2f + 1 signatures, preventing equivocation.

Clients only accept updates accompanied by a valid certificate, which they verify using witness signatures. They can also query witnesses for past root hashes and certificates, ensuring a consistent tree history without requiring a hash chain.
To prevent a malicious service provider from ignoring key update requests, clients can escalate their requests to witnesses. If the provider fails to include an update, witnesses refuse to sign future updates, making it impossible to issue a valid certificate, thus forcing compliance.

5.4. Anonymous Communication

Instead of relying on direct gossip between clients to detect inconsistencies in the Transparency Log, a more structured approach involves querying the log through two independent channels:

Authenticated Channel: Each client queries the Transparency Log over an authenticated channel. The server knows the client's identity, and may tailor its response accordingly.

Anonymous Channel: The client separately queries for the same STH over an anonymous channel, which conceals the client's identity from the KT server. This could be done through a proxy, mix network, or anonymity-preserving service such as Tor. Since the request is anonymous, the server cannot tailor its response based on the client's identity.

By comparing the STH received over both channels, the client can check for discrepancies. If the KT server is behaving honestly, both responses should match. However, if the server is equivocating—presenting different STHs to different clients—then the difference will be exposed when the authenticated response differs from the anonymously retrieved one.

This approach strengthens transparency and detection of equivocation without requiring direct client-to-client communication, making it more scalable and resistant to attacks where a malicious server selectively forks the log for different users

5.5. Blockchain-based Solutions

Another potential solution for the instantiation of this public bulletin board of Signed Tree Heads is Blockchains. In these proposals [12], the log's state updates are recorded on a public blockchain. Because blockchain entries are immutable and globally visible, they serve as external commitments to the log's history. While Blockchains inherently provide a single, append-only ledger where all participants can verify the history of log updates, they can suffer from high latency and limited throughout, and requiring clients to interact with a Blockchain may hinder the adoption of these solutions.

5.6. Practical Deployments

Apple's Contact Key Verification: In contrast to CT deployments and WhatsApp's KT system—which depend on external parties to validate the consistency of their entire data structures—the Messages app performs on-device checks to ensure that the critical append-only logs in Apple's KT system remain consistent. Moreover, to uncover any split-view attacks that might occur if an attacker compromises the KT service, the app embeds log hashes into the encrypted portion of a small subset of messages. These hashes are then shared with other iMessage clients, which independently verify that the log hashes they receive match, thereby detecting any inconsistency in real time [13].

WhatsApp Key Transparency: WhatsApp's Key Transparency solution is based on Parakeet, and as such, it envisions a network of distributed witnesses that need to be highly available to audit the update proofs (cryptographical evidence that each update only adds leaves to the tree without modifying or deleting existing ones). Furthermore, these witnesses play a central role in the consistency protocol by verifying that the service provider presents an identical history to all users, thereby preventing equivocation. Practically, their approach uses an AWS S3 bucket with a WORM (write-once-read-many) model with a 5-year retention period with a retention period of 5 years. [2]

6. Conclusions:

Split-view attacks remain a significant threat to transparency log systems. While various protection mechanisms exist, each comes with its set of tradeoffs. Whether conducted out-of-band or in-band (piggybacked), gossip protocols rely on clients exchanging log snapshots (such as Signed Tree Heads or inclusion proofs) to verify that everyone sees the same log state. Out-of-band gossip uses dedicated channels separate from regular application traffic, while in-band gossip embeds the log data into normal communication. Both methods have the advantage of decentralization, but they require widespread participation and dependable communication channels to be effective.

Witnessing, by contrast, involves a set of independent and highly available third-party entities (witnesses or auditors) that proactively validate log updates before clients accept them. This approach can yield immediate, non-repudiable evidence of misbehavior, though it introduces additional trust assumptions regarding the witnesses' honesty and requires extra operational coordination.

Blockchain-based solutions leverage blockchains' immutability and global consensus to anchor the log's state. By publishing log commitments (such as root hashes) to a blockchain, these approaches ensure that the log's history is difficult to alter once recorded. However, blockchains can impose significant latency and higher computational or financial costs and may raise privacy concerns due to their public nature.

Anonymous querying adds another dimension to the detection toolbox. In this approach, a client submits two queries for the same log data—one over an authenticated channel and one over an anonymous channel (e.g., via Tor or a VPN). By comparing the responses, the client can detect if the log server is providing tailored or inconsistent views based on the client's identity. This method requires the log to support both types of channels and depends on the availability of robust anonymous networks. It does not require additional infrastructure beyond what clients already use for anonymity and complements other techniques by directly isolating identity-dependent misbehavior.

References

[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-key-transparency/

[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/