A-SIT

Secure Information Technology Center – Austria

Certificate Transparency for Relying Party Certificates

Certificate Transparency for Relying Party Certificates

Author:
Edona Fasllija
Mail:edona.fasllija@tugraz.at

eIDAS 2's European Digital Identity framework shifts control to the user via a secure wallet app. Relying Parties (RPs) willing to access wallet data must register with the authorities and declare their intended data use (purpose) before doing so. During a transaction, an RP Access Certificate (RPAC) authenticates the RP's service instance; an RP Registration Certificate (RPRC) enumerates the specific attributes the RP is authorized to request. Together, these certificates implement privacy by design: the wallet can inform users about which RP is asking for which attributes and refuse requests that go beyond what was officially registered as necessary.

However, the regulation leaves a critical enforcement gap: there is no built-in mechanism for users or auditors to verify what has been registered or whether those limits are respected. This project proposes RP Certificate Transparency (RPCT), a public, cryptographically verifiable log infrastructure inspired by Certificate Transparency and adapted to the RP authentication and authorization model of eIDAS 2. RPCT provides inclusion, consistency, and history proofs suitable for unlinkable wallet verification, supports revocation transparency and per-RP audit trails, and enables regulators and third-party monitors to detect misissuance and over-scoped requests.

1. Introduction & Motivation **1**
2. Background **2**
2.1. RPACs and RPRCs in eIDAS 2 2
2.2. From CT to RPCT 3
3. RPCT Design **4**
3.1 Entities and Roles 4
3.2 RPCT Workflow/ Lifecycle: 6
4 Authenticated Data Structure: Merkle Patricia Trie & Per-RP Hash **7**
5 Discussion **10**
6 Conclusions: **12**

1. Introduction & Motivation

The EU Digital Identity (EUDI) framework under eIDAS 2[1] introduces digital identity wallets that enforce access control based on certificates issued to Relying Parties (RPs). RPs must register with a national authority and declare their data access permissions. Each RP must obtain two certificates from designated Certificate Authorities (CAs): an RP Access Certificate (RPAC), authenticating the RP's service instance, and an RP Registration Certificate (RPRC), where applicable, listing the user attributes the RP is authorized to request. This two-certificate scheme enforces data minimization: The wallet verifies the RP's identity via the RPAC and ensures any requested attributes fall within the RPRC's declared scope. Users are informed which attributes are requested and can be warned if an RP asks for data outside its registered scope.

Without an audit mechanism, a malicious or compromised CA could secretly misissue certificates. For example, a CA might issue an RPRC with excessive permissions not actually approved or registered by the RP, allowing a rogue service to request more user data than allowed. Similarly, a bogus RPAC could be issued to an imposter RP, enabling phishing or identity fraud. Such misissuance would remain undetectable by users or other parties if there's no transparency or public log of issued certificates. Another risk is *silent revocation*: a CA could revoke an RP's certificate without notice, potentially disrupting services or, worse, allowing an attacker to present a revoked certificate if verifiers don't have up-to-date revocation information. These threats undermine user privacy, consent, and the regulation's intent.
The Web PKI addressed TLS certificate misissuance via Certificate Transparency (CT)[2] logs. Standard CT requires CAs to publish every TLS certificate to a public append-only log, enabling anyone to monitor for unauthorized certificates. However, naïvely applying CT to eIDAS RP certificates is insufficient. Classic CT logs *only* certificate issuance, not revocations. There's no built-in way to tell if a certificate was later revoked. Also, CT logs return Signed Certificate Timestamps (SCTs) as proof of logging, but clients must often contact log servers or monitors to validate SCTs. This leaks user browsing history (a privacy issue) and shifts trust onto the log servers. Furthermore, CT relies on external monitors to catch misissuances by scanning the entire log, an approach that is bandwidth and computationally heavy. In summary, the EUDI wallet ecosystem needs a transparency solution that covers both issuance and revocation, preserves user privacy, and scales for potentially millions of RPs.

To address these gaps, this project proposes Relying Party Certificate Transparency (RPCT): a transparency logging architecture tailored to the eIDAS 2 RP certificates context. RPCT's goal is to *deter* and *detect* misissuance of RPACs/RPRCs and catch any authorization scope abuse. Every issued or revoked RP certificate is recorded in a public, tamper-evident log. This gives RPs, wallet providers, regulators, and users the ability to audit who is authorized to request what data. In essence, RPCT acts as a "*guardian of the registry*,"[3] ensuring that the official RP registration info (the "registry") aligns with the certificates used in practice.

2. Background

This section provides background on the two key RP certificate types in eIDAS 2: RPACs (for RP authentication) and RPRCs (for authorization scope). The section describes how they function and why they are required by law. Finally, we highlight the trust problem if these certificates are not publicly logged and how a bad actor can subvert the system.

2.1. RPACs and RPRCs in eIDAS 2

RP Access Certificates (RPACs): Under eIDAS 2, an RP must identify itself to the user's wallet using an RP Access Certificate. An RPAC is essentially an authentication certificate for a service instance (analogous to a web server certificate in TLS) issued by a trusted authority (CA). It binds the RP's identity (and possibly domain or organization info) to a public key, allowing the wallet to verify the RP's identity before any data is shared. This stems from the Regulation Article 5b(8) that mandates RP authentication: "Wallets must not share data with an RP lacking a valid RPAC". In practice, when a user attempts a credential presentation to an RP, the wallet checks that the RP presents a valid, up-to-date RPAC. This prevents rogue or unregistered services from masquerading as legitimate RPs.

RP Registration Certificates (RPRCs): In addition to proving identity, RPs (optionally) obtain an RPRC that encodes the intended use of the data and the specific attributes the RP is allowed to request. Essentially, an RPRC is a machine-readable "permission slip" listing, for example, that a banking app can ask for your name and account status, but not your full identity profile. This certificate is typically issued by an RP CA related to a Registrar authority after the RP registers its data usage purpose with a government or competent authority (a requirement per Article 5b(1–2) of eIDAS). Figure 1 depicts an overview of how these certificates are used in eIDAS 2.

The wallet *enforces* that any attribute an RP requests during a transaction is covered by the attributes listed in the RPRC. If the RP requests more data than listed, the wallet can flag this to the user or even refuse the request, as a result upholding the principle of *lawfulness and user consent* (RPs "shall not request more data than indicated in the register"). RPACs and RPRCs are central to establishing *trust* in the ecosystem. They assure users that (a) the RP is who it claims (RPAC) and (b) the RP is *only* asking for data it legitimately registered (RPRC). However, these assurances depend on the integrity of the CAs/authorities issuing the certificates. If a CA cheats (intentionally or due to compromise) and issues a certificate outside the rules, users and other RPs wouldn't know. For example, a malicious RP could collude with a corrupt CA to get an RPRC that lists excessive attributes (violating what was filed in the official register). Or an attacker could trick a CA into issuing an RPAC for a fake service name like a legitimate one (a phishing attempt). In both cases, the wallet alone cannot detect the anomaly because it will faithfully verify the certificates it's given, unless there's an external audit log.

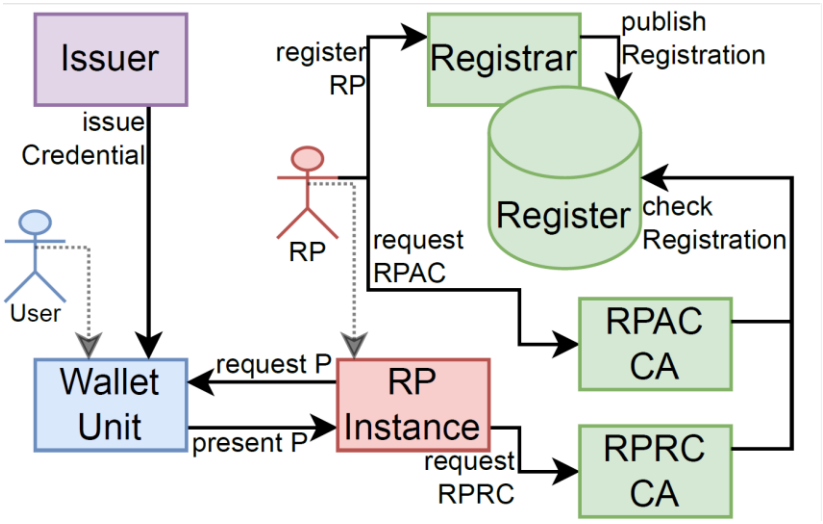

Figure 1 RPAC/RPRC Overview in eIDAS 2

2.2. From CT to RPCT

Certificate Transparency (CT) introduced public logs of certificates to detect misissuance. In CT's model, CAs log each certificate, log servers maintain an append-only Merkle Tree of entries, and independent monitors and auditors oversee the logs. The classic CT ecosystem architecture is as follows: CAs submit certificates to log servers, monitors watch logs for suspicious entries (e.g., a fake certificate for a domain), and auditors ensure the log's integrity. RPCT adopts a similar architecture of logs, monitors, and auditors, but adapts it for the EUDI wallet context with RPs and their special certificates.

CT Basics: In Web PKI CT, every TLS certificate issuance is recorded in a public log as a new entry (leaf) in a *Merkle Hash Tree*. Logs periodically publish a signed tree head (STH) as a global digest of all entries. CAs provide clients with an SCT (Signed Certificate Timestamp) as proof of log submission. Monitors (any interested party) download and inspect log entries to find certificates for domains they care about, flagging any misissued certs (e.g., a bogus certificate for example.com). Auditors (which can be browsers or third parties) check that log updates are append-only and consistent over time. This system makes it possible to catch CA misbehavior in the wild.

Limitations of CT for Our Use Case: CT's design works for domain certificates, but doesn't directly solve our RP certificate problem. We list the main limitations below:

No Revocation Logging: In CT, if a certificate is revoked, that event isn't logged or reflected in the Merkle Tree (the tree only grows with new entries). Thus, CT alone can't prove to a client that a certificate presented is still valid, only that it was once logged.

Privacy Concerns: The act of verifying SCTs can reveal which sites a user is visiting. For example, a browser might query a CT log or ask a monitor about a specific SCT, effectively revealing "user X is visiting site Y." In a wallet scenario, if each time you use a credential, the wallet had to query an RP certificate log, it could reveal your service usage patterns and violate eIDAS's privacy requirements (no tracking of users by third parties).

Efficiency: Classic CT requires scanning the entire logs to find entries of interest. With potentially thousands of RPs per country, having monitors re-download huge log trees continuously to find problematic RP certificates would be very inefficient.

Key Transparency (KT) Influence[4], [5]: A related concept is Key Transparency, which extends the transparency idea to user public keys (as in CONIKS). KT systems use an append-only dictionary (often a Merkle prefix tree) mapping each user (or identifier) to their current public key. Instead of scanning everything, clients can directly lookup a specific user's key with a cryptographic proof of correctness. This inspires part of our solution: we want *per-RP lookup efficiency*. In other words, one should be able to easily retrieve all certificates for "RP X" from the log, without combing through unrelated entries.

3. RPCT Design

RPCT marries CT's public accountability with KT's efficient lookup. It retains the familiar roles: RP CAs (for RPACs and RPRCs) must submit every certificate issuance and revocation to the log. The RPCT Log server accepts these submissions and maintains an append-only data structure (more on this next) that allows both chronological auditing and key-based queries. Wallets (Clients) act somewhat like browsers in CT: they consume proofs from the log to validate an RP's certificates before releasing data. Monitors (which could be run by regulators, watchdog organizations, or even RPs themselves) fetch log data and watch for anomalies: e.g., an RPRC that contains attributes beyond what that RP should request. They might cross-check the log's contents against official RP registries published by Member States (as required by eIDAS Article 5b)[6] to detect inconsistencies. Auditors ensure the log server is honest: that the log's state never forgets an entry or forks into inconsistent versions for different people.

Extending CT Principles: In essence, RPCT extends CT in three crucial ways:
(1) *Revocation Transparency*: not just logging additions, but also when a certificate is revoked or removed, so clients can obtain cryptographic proof of a certificate's non-revocation (or conversely, detect if it was revoked).
(2) *Per-RP Organization*: using a data structure that allows direct lookup of an RP's certificate chain, avoiding full-log scans and making monitoring more targeted and scalable.
(3) *Privacy-Preserving Proofs:* enabling wallets to validate an RP's certs offline, using proof data that can be pre-fetched or provided by the RP, so the wallet doesn't need to query the log in real time (preventing observers from tracking which RP a user is engaging with).
These enhancements aim to meet eIDAS 2's specific requirements for accountability (public logging of RP certificates), consistency with official registrations, and privacy.

3.1 Entities and Roles

This section walks through the system components and how they interact to hold CAs accountable and give regulators the tools to supervise the whole system.

In the RPCT system architecture (illustrated in Figure 2), we have the following key entities:

RP CAs: These are the authorities (either government-designated or accredited private CAs) that issue RPACs and RPRCs. Under RPCT, an RP CA is obligated to submit each newly issued or revoked certificate to the transparency log immediately. One can think of them as "appenders" to the log. If a CA doesn't log a certificate, that certificate will be considered untrustworthy by wallets (since there is no proof of logging).

RPCT Logger (Log Server): A server (or distributed service) that maintains the append-only log of RP certificates. It provides APIs for CAs to add entries and for clients/monitors to query proofs. The log server periodically produces a new global digest (a Merkle Tree root hash) that commits to all entries so far. In RPCT, this log is built on a specialized authenticated data structure (discussed in Section 4). The logger also generates cryptographic proofs: when a certificate is added, it can return an *inclusion proof* showing that the certificate is now in the log's digest. It can similarly provide *proofs of absence* (non-inclusion) or *consistency* between log states over time.

Wallet: The user's EUDI wallet application (on a phone, for instance) acts as a verifying client. When an RP presents its credentials (RPAC and RPRC) during a transaction, it must also provide a log proof (or a set of proofs) to the wallet. The wallet will verify the certificates and their log proofs before releasing any user data. This means the wallet checks that the RPAC/RPRC were indeed logged and not revoked. If the proof is missing or invalid, the wallet should refuse the credential presentation, ensuring unlogged (and thus potentially fraudulent) certificates aren't accepted.

Monitor Services: These can be run by anyone, but more likely by competent authorities (e.g., Data Protection Authorities, national supervisory bodies) or privacy-focused NGOs. A monitor continuously watches the log for suspicious certificates. For example, a DPA might monitor all RPRCs issued under its country code and verify that the attributes requested match what the RP registered. If a rogue entry is detected (say an RP registered only "name and birthdate" but an RPRC in the log for that RP also lists "social security number"), the monitor can alert the affected parties (the RP, regulators, or the public). Monitors could also watch for unauthorized RPACs (e.g., a certificate for an RP that isn't in the official registry). Because RPCT's data structure allows per-RP queries, a monitor can focus on specific segments of the log relevant to their oversight domain (like by country or sector), instead of processing the entire global log.

Auditors: An auditor ensures the log's operation remains honest and append-only. Auditors can be independent services or even integrated with clients. Auditors fetch periodic log digests and use consistency proofs to make sure that the log has not been tampered with (i.e., that a newer digest indeed includes all older entries and hasn't rewritten history). Auditors also check that the log isn't presenting different views to different people (no forked history). In practice, many of these functions can be automated: e.g., browsers in CT play auditor by requiring a consistent view of logs. In eIDAS, perhaps the European Commission or a coalition of Member States might run auditing processes to ensure any official RPCT logs are behaving correctly.

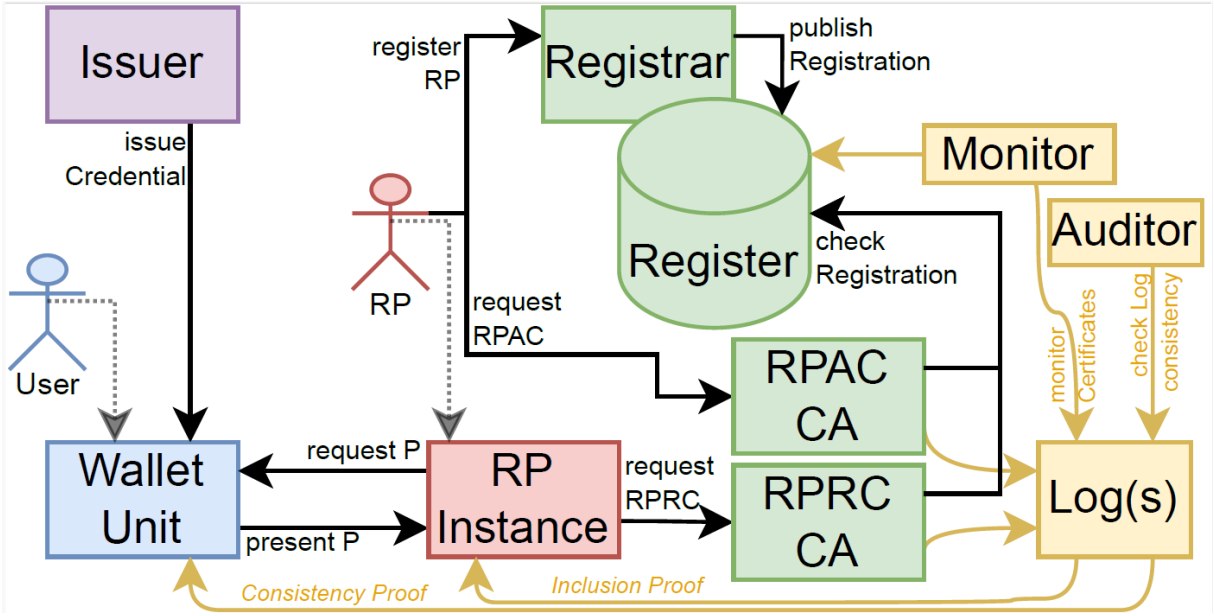

Figure 2 RPCT Overview

3.2 RPCT Workflow/ Lifecycle:

1. RP Registration: An RP first registers with an RP Registrar (per eIDAS Implementing Act), providing its identity and intended data usage. The Registrar (a government body) adds the RP to the official RP Registry (a public list/database of RPs and their approved attribute uses). This step is outside RPCT but critical context: the RPRC will later need to match this info.

2. Certificate Issuance: The RP obtains an RPAC (and optionally an RPRC) from the respective CAs. When the CA issues these certificates, it simultaneously submits them to the RPCT log. The submission could include the RP's identifier as the log key and the certificate data as the value.

3. Logging and Proof: The RPCT Logger appends this new certificate entry. It updates its internal data structure and computes a new root digest. The log returns an inclusion proof to the CA. The CA can forward this proof to the RP along with the certificate. Alternatively, the log could also send proofs directly to monitors subscribed to that RP's ID. At this point, the certificate is publicly visible in the log (auditable by anyone), and there's cryptographic evidence of its inclusion.

4. Credential Presentation: When a user wants to access the RP's service, the RP presents its RPAC and RPRC to the user's wallet (likely via some protocol like OpenID for Verifiable Presentations). Now, with RPCT, the RP also provides the latest inclusion proof for those certificates (and possibly a consistency proof if needed to show the log state progression). The proof convinces the wallet that the certificates were logged in the global digest it trusts. The wallet uses the log's public key to verify the proof against a trusted digest (which it may have cached or can get from a gossip network or periodic update).

5. Wallet Verification: The wallet checks: (a) The RPAC signature (is it issued by a trusted CA and not expired/revoked?), (b) The RPRC contents (does it cover the attributes being requested?), and now (c) RPCT proofs: ensuring the RPAC/RPRC are in the transparency log and have not been revoked. If all checks out, the wallet proceeds to ask the user for consent to release the requested data. If something fails (e.g., the inclusion proof is invalid or the log indicates the RPRC was revoked), the wallet will refuse or warn the user.

6. Monitoring & Alerts: Meanwhile, as log entries accumulate, monitors may detect any policy violations. For instance, if an RP's RPRC has a purpose that doesn't match what's in the official registry (violating the consistency requirement), a monitor could notify the authorities to investigate that RP or CA. If a

certificate is revoked, the log will show its removal in the next update; a monitor might inform relying parties (or wallets via software updates) of mass revocations or suspicious patterns (like many revocations from one CA, indicating that CA might be compromised).

7. Revocation Handling: If an RP's certificate needs to be revoked (for instance, if the RP lost its accreditation or had a security breach), the CA initiates a revocation. In RPCT, the CA submits a "removal" update to the RPCT log for that RP's entry. The log adds a new record for that RP's key, indicating the certificate is no longer active (Section 4 provides details about how this is done without deleting history). Wallets and monitors then use the log to get cryptographic assurance of revocation status (the absence of the certificate in latest state serves as proof of revocation).

Binding RPACs and RPRCs: One important feature of RPCT is that it can enforce that an RP's two certificates are *cryptographically linked* in the log. Since both the RPAC and RPRC for a given RP share the same identifier key in the log (the RP's unique ID), any update to that RP's entry can include both certificates together as a set. This means the log can ensure that a valid RPAC and a corresponding RPRC are seen together. If a rogue tries to log an RPRC for a non-existent RPAC (or vice versa), it would be evident. The wallet can also retrieve the *current set of certificates* for the RP from the log to check consistency (e.g., ensure the RP presenting a certain RPRC also has a matching RPAC logged).

4 Authenticated Data Structure: Merkle Patricia Trie & Per-RP Hash

At the heart of RPCT is an authenticated data structure (Illustrated in Figure 3) that differs from the conventional CT Merkle tree. The design goal was to support chronological append-only logging and efficient per-RP lookups. The solution is to use a *Merkle Patricia Trie (MPT)* [7]keyed by RP identifiers, combined with individual *hash chains per RP*.

Key-Indexed Merkle Trie: Instead of one big timeline tree of all certificates, RPCT organizes the log as a dictionary (map) from `RP ID -> certificates`. Each RP is assigned a unique key (for example, this could be a hashed identifier like an EORI number, a domain name, or a country-scoped ID) that determines its position in the trie. All certificates belonging to that RP (both its RPAC and RPRC, and historical versions) will reside under that key's node. The use of a Patricia Trie (a compressed prefix tree) ensures that this key space is represented efficiently: common key prefixes (e.g., same country code) share branches, and leaves are at varying depths depending on keys. This structure is similar to Key Transparency's approach (mapping user IDs to keys), but here the "user" is an RP.

Per-RP Hash Chain (Chronological history): Each leaf in the trie corresponds to an RP and stores a cryptographic hash chain that records the sequence of that RP's certificate event. Let's break that down: for a given RP, when it first gets a certificate, a hash h_1 is created by hashing together

(i) a timestamp or epoch T_1 ,

(ii) the set of active certificate values v_1 (maybe the RPAC or RPRC or both), and

(iii) an initial previous hash h_0 (which could be a null value).

This h_1 becomes the "head" of the RP's chain. If later the RP updates or gets a new certificate (or one is revoked), a new hash $h_2 = \text{Hash}(T_2 \parallel v_2 \parallel h_1)$ is computed, linking to the prior head. Over time, this builds a chain $h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow \ldots \rightarrow h_t$ where each link immutably records that RP's certificate state at each epoch. The value v_i at each step can be thought of as the current set of certificates for that RP after that update (for example, initially v_1 might contain the first RPAC; if an RPRC is later issued, v_2

contains {RPAC, RPRC}; if the RPAC is renewed, v_3 updates that, etc.). Because each new hash includes the previous hash, it forms an append-only history for that RP.

Global Log as Trie of Heads: Now, every RP's latest chain head h_t is placed in the Merkle Patricia Trie at the position for that RP's key. We then compute the Merkle root of this entire trie. That root is the global log digest (R), which reflects the state of all RP entries at the current epoch. If any RP's entry changes (a new certificate or revocation), its leaf hash will change, and thus the root hash will change. The root serves as a commitment to all RPs' latest certificate states. The log signs this root regularly (like a Signed Tree Head in CT).

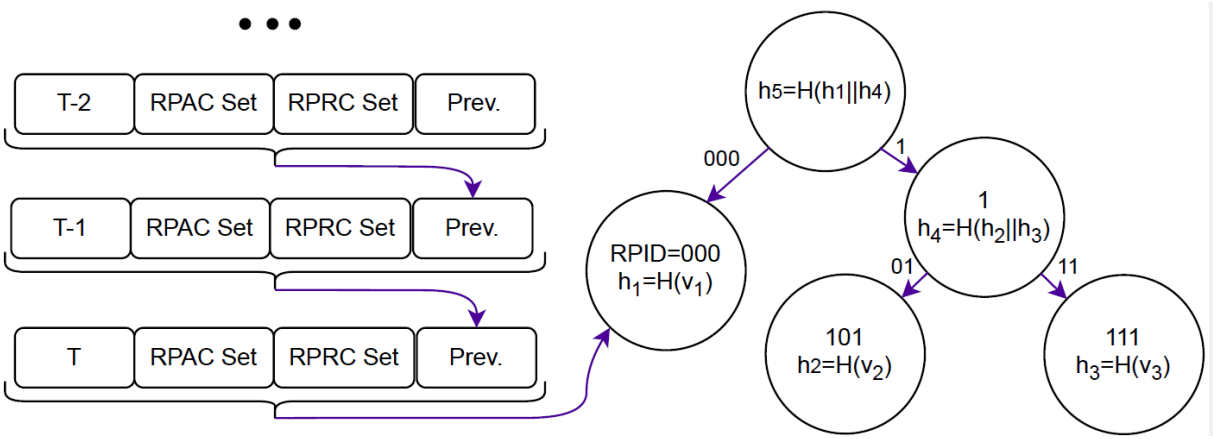

Figure 3 The RPCT Authenticated Data Structure

Example: Suppose RP "Alice's Service" initially registers and gets an RPAC. The log's trie gets a leaf for Alice's RPID with hash chain head $h_1 = \text{Hash}(T_1 \parallel \{RPAC\} \parallel \perp)$. The global root R_1 now covers Alice's entry. Later, Alice obtains an RPRC listing attributes "email, name". The CA logs this as an update: compute $h_2 = \text{Hash}(T_2 \parallel \{RPAC, RPRC\} \parallel h_1)$. The leaf for Alice is updated to h_2. The global root changes to R_2. If someone queries the log for Alice's RPID at this point, they find h_2 and can verify (with a proof) that h_2 is in R_2. If the CA then revokes Alice's RPRC (say Alice changed her data usage and the old RPRC is invalidated), the CA logs a new update: $h_3 = \text{Hash}(T_3 \parallel \{RPAC\} \parallel h_2)$, effectively removing the RPRC from the current set. Now the latest head reflects only the RPAC, and R_3 is the new root. Importantly, the old RPRC hasn't vanished from history: it's just not in the current set. Anyone can inspect the chain $[h_1, h_2, h_3]$ to see that at T_2 the RPRC was added and at T_3 it was removed.

The MPT allows efficient proof generation for lookups. If a wallet needs to verify an RP's certificates, it can ask for a lookup proof for that RP's key. The log will return the current value (the set of active certs or the head hash) and a Merkle proof path up the trie to the root. This proof is logarithmic in the number of keys, not the number of total entries, which in a continental-scale system is crucial. For example, if there are 1 million RPs, the proof might involve traversing, say, a 20-level tree rather than including potentially millions of unrelated entries. The Patricia trie is particularly useful if keys are structurally related (like domain names or country codes). Different encoding strategies for keys (hashing vs. semantic segments) can be employed to balance tree depth and proof size.

Proof Types Provided: The RPCT log supports multiple queries:

Inclusion Proof: Given a specific certificate (or rather a specific RP and a certificate fingerprint), the log can prove it's included in the log at a certain epoch digest. In practice, since we store per-RP sets, an inclusion proof might show that "for RP key k, value v (which includes the certificate) is in the set at leaf's hash h_t" and provide the path in the trie.

Consistency Proof: Given two root digests R_i and R_j ($i<j$), a consistency proof proves that the later state includes all entries from the earlier state (i.e., the log only grew). In the trie-of-chains design, a consistency

proof ensures no RP leaf that existed at R_i was removed by R_j – formally that every leaf hash at state **i** is still present at state **j**, possibly with updated values, and no leaf was deleted. This is analogous to CT's consistency proofs between STHs.

Lookup/Absence Proof: If a wallet queries "*does RP X exist in the log (registered)?*", the log can show a proof of absence using the trie structure. The proof consists of the path where the key would have been, demonstrating a gap where no such key exists, thus proving that RP is not (yet) in the log (which could mean unregistered).

History Proof: One can also request the entire history for an RP. The log can return the sequence of all values $\{v_0, v_1, \ldots, v_t\}$ that were ever associated with that RP, along with a proof that this sequence is complete. Because each chain link hashes the previous, a verifier can check the integrity of the history. This is useful for auditors or monitors to examine how an RP's permissions may have changed over time.

Data Structure Comparison (CT vs. RPCT): In standard CT, each new certificate is a new leaf in a Merkle tree and monitors must track leaves by scanning the whole tree. In RPCT's MPT+chains, the "leaf" is per RP, and updates go into that leaf's chain. This yields smaller proof sizes and localized monitoring: e.g., to monitor a particular RP or a set of RPs, one can query just those keys. Table 1 shows this structure significantly reduces the overhead for both inclusion proofs and consistency checks at EU scale . It also naturally handles revocation: removing a cert just creates a new chain element, rather than having to mark a leaf as revoked or maintain a separate revocation list.

Table 1 CT vs RPCT Proof Complexities

Operation	CT	RPCT
Lookup by key	Not supported natively, scan O(N)	Supported O(logN)
Inclusion Proof	O(logN) hashes	O(logN) hashes + 1 chain link
Consistency Proof	O(logN) hashes	O(logN) per update
History Proof	Not native (multiple inclusion proofs needed)	Supported via chain (O(L) links for L updates)

Revocation Transparency and Consistency

A major improvement of RPCT over vanilla CT is the support for revocation transparency. In other words, anyone can verify not only that a certificate was logged, but also whether it's still valid or has been revoked, using the transparency log.

Logging Revocations: In RPCT, when a CA revokes an RPAC or RPRC, it submits an update to the log just like an issuance event. The update for a revocation essentially means "remove certificate X from RP Y's active set." Technically, the CA computes a new value for that RP's leaf (e.g., if the prior value was **{Cert1, Cert2}**, and **Cert2** is revoked, the new value v_{new} **= {Cert1},** excluding **Cert2**). This new value gets hashed with the previous chain head, producing a new head for that RP's chain, and the global root is updated. Because the previous head remains linked in the chain, the fact that Cert2 *used to be present* is still auditable in the history, but a current lookup will show it gone. There is no need for a special "revoked" marker or a separate revocation list : the log's state simply no longer includes the certificate.

Proving Revocation (or Non-Revocation): For a wallet, proving that a certificate has not been revoked comes down to getting a proof of inclusion in the latest state. If the certificate is still included, it's not revoked; if it's excluded, it has been revoked (or never existed). The log can generate a *proof of absence*

for a certificate in the latest epoch, which combined with a historical inclusion proof can convince a client that "certificate X was logged at time T and as of now it no longer appears, hence it was revoked between T and now." Conversely, a proof of inclusion in the latest state is evidence the cert is still valid. This is analogous to an OCSP "good" status but backed by the transparency log's integrity. Importantly, monitors could also watch revocation patterns: e.g., if a particular RP or CA suddenly revokes many RPRCs, that might indicate a burst of misbehavior that warrants investigation.

Privacy-Preserving Proofs and Offline Verification

One of the standout features of RPCT is that it enables *offline proof verification*, which is crucial for user privacy. In traditional CT, a browser often had to contact a log or a monitor to ensure a certificate's SCT is included in the log. In RPCT, by contrast, the wallet can verify an RP's certificate status locally without needing to query the log at presentation time.

Proof Bundle Delivery: When an RP obtains its certificates and they are logged, the RP (or the CA) can also retrieve the needed proofs from the log. For example, after logging, the CA could provide the RP with an *inclusion proof* for the RPRC and RPAC against the latest root (or a recent root). This proof might include: the Merkle path in the trie for the RP's key, and the latest root hash signed by the log. The CA might also provide a *consistency proof* linking that root to a known earlier root hash that the wallet trusts (if the wallet only has an older reference). Alternatively, the wallet software or operating system could periodically fetch updated log digests (much like browsers update a list of trusted CT log roots or use gossip protocols to learn STHs). The key point is that by the time a user is presenting credentials to an RP, the necessary log digest and proofs can be made available without a live lookup.

Offline Verification: The wallet uses the inclusion proof and the known log digest to verify that the RP's certificate was indeed logged. Because the log digest is globally witnessable (and ideally, the user's device has it or can get it in a privacy-preserving way, e.g., via broadcast or within regulatory bulletins), the wallet doesn't need to contact the log for this one RP. It simply recomputes the hashes from the proof: starting from the RP's leaf (which includes the certificate's hash in the value set), hashing upward through the provided sibling hashes, and checking it matches the trusted root. If it matches, the certificate is confirmed to be present in the log. If the certificate had been revoked, the inclusion proof for the current root would fail (the certificate would no longer appear in the value set at that leaf). In such a case, the RP would not be able to produce a valid proof of inclusion for the latest state.

This is important because eIDAS 2 explicitly demands that wallet use should not allow tracking or profiling of users by external parties. If every time you use your wallet it had to query a government log "Is RP X's cert valid?", an observer of the log traffic or the log itself could infer that you are interacting with RP X. RPCT avoids this by enabling a model where either: The wallet has a cached copy of the entire log's state (not likely, too large), or more realistically, the wallet regularly (or via a trusted channel) receives the latest log root, and the RP provides the specific proof. Since the RP is the one you're connected to anyway, no *new* privacy leak occurs by getting the proof from the RP. Meanwhile, the RP's provided proof can be checked without contacting anyone else.

5 Discussion

5.1. Compliance with eIDAS 2 Requirements

The eIDAS 2 regulation and its draft Implementing Acts lay out specific requirements for Relying Party oversight and security. RPCT was designed with these in mind and achieves compliance. Key regulatory requirements addressed include:

Article 5b(1–2) – RP Registration & Purpose Declaration: RPs must register and declare their intended wallet usage and data needs with authorities.

Compliance: RPCT's use of RPRCs directly supports this. Each RPRC is essentially a signed artifact of that declaration, and by logging them publicly, RPCT makes the registration information transparent (this correlates with a requirement that Member States publish a list or registry of RPs and their permitted data). In fact, RPCT can provide an always up-to-date log of all authorized RPs and their scopes.

Article 5b(8) – RP Authentication (Access Certificates): Users should only send data to RPs that properly identify themselves (hence requiring RPACs).

Compliance: RPCT logs all RPACs, ensuring that there is a public record of which RPs have valid authentication certificates. A rogue service without a logged RPAC would be detectable. Wallets enforce this by only proceeding if an RPAC has a valid log proof, thus meeting the requirement that "wallets must not present data to an RP without a valid access certificate".

Article 5b(3) – Lawfulness of Data Requests: RPs "shall not request data other than that indicated in the register".

Compliance: This is exactly what RPRCs + RPCT enforce. The RPRC is a concrete list of allowed attributes (the "indicated" data), and the wallet is able to compare it against the actual request. Moreover, because the log is public, authorities can verify RPRCs align with registered purposes. If an RP ever requests more data (over-asking), the discrepancy would either be caught by the wallet (if wallets implement that check with help of the log) or by monitors post-fact.

Article 5a(14) & Recital 32 – User Control & Consent: Users must have full control and must give explicit consent for data sharing, and no hidden data transfers should happen.

Compliance: RPCT indirectly supports this by building trust in what the wallet tells the user. When a wallet shows "RP X requests your birth date and email (as registered)", the user can be confident that this request has been vetted through an official process (because of the RPRC transparency). Also, if an RP tried to gather data without proper registration or through a misissued cert, the transparency system is likely to catch it, thereby protecting users from behind-the-scenes abuse.

Article 5a(5)(b) & 5a(16)(a) – Privacy (No tracking or linking): No entity should track users' credential use without permission.

Compliance: As discussed, RPCT's offline proofs ensure that the wallet's act of verification doesn't leak user activity to logs or third parties. Unlike naive CT, where each certificate check could ping a log, RPCT allows the wallet to remain silent on the network during usage.

5.2. Deployment Considerations

This section lists some of the key considerations for implementing and deployinh RP Certificate Transparency at an EU-wide scale in terms of infrastructure, performance, and governance.

Log Infrastructure: Who runs the RPCT log(s)? Options include: a centralized EU-run log service, multiple national logs (each Member State operates one for their RPs. A decentralized model with multiple independent logs is likely advisable (just as CT uses multiple logs). This provides redundancy and prevents single points of failure or trust. If multiple logs are used, standards need to dictate how CAs choose logs and how wallets trust proofs from them (e.g., maintain a list of trusted RPCT logs).

Scalability and Performance: The number of RPs across Europe could be large (considering also medium/small businesses integrate with EUDI wallets). The data structure chosen (MPT) was evaluated for scale: proofs remain logarithmic in the number of RPs, and update operations are efficient. Still,

running a high-throughput log server is a serious task. The log should handle bursts of certificate registrations (e.g., when a new regulation deadline hits and many RPs register at once).

Integration with Wallets and RPs: Wallet software (likely provided by various vendors under the EUDI framework) will need to implement the logic to handle RPCT proofs. This means wallet developers must be given libraries or standards for verifying Merkle proofs, understanding the RPCT data structure, and retrieving the trusted log root. They also need UX design to present warnings if something is wrong (like a cert not logged or an RP over-requesting data). RPs, on the other hand, must obtain and supply proofs. This could be abstracted by middleware: for example, an RP could use an SDK that automatically fetches the latest inclusion proof from a transparency log service. We might envision a "Transparency Proof Token" that the RP includes in its OIDC or presentation request to the wallet. All of this should be standardized to ensure interoperability (likely through the EUDI Architecture Reference Framework).

Trust Model and Governance: RPCT introduces a new critical component: the log operator. While it doesn't need to be trusted for correctness (since misbehavior is detectable via cryptography), availability and timely operation are important. The logs' public keys must be distributed to all wallets so they can verify signatures on roots and proofs.

6 Conclusions:

eIDAS 2 introduces a structured approach to relying party (RP) authentication and authorization through the use of RP Registration Certificates (RPRCs) and RP Access Certificates (RPACs). However, the absence of public verifiability in the certificate lifecycle creates risks of misissuance, silent revocation suppression, and over-asking of personal data.

RP Certificate Transparency (RPCT) provides a cryptographically verifiable infrastructure to address these risks. It logs RPACs and RPRCs in a public, append-only Merkle-based structure that enables efficient generation and verification of inclusion, revocation, and historical consistency proofs. Each RP is mapped in a Merkle Patricia Trie, with per-RP hash chains recording the sequence of certificate events, allowing for scalable audits and offline verifiability by wallet applications.

The system meets key regulatory goals: it ensures that RPs cannot exceed their declared permissions (Article 5b), supports runtime enforcement by wallets, and enables independent monitoring. RPCT provides a foundational trust layer by making RP permissions verifiable and RP over-asking auditable at scale.

This work resulted in a paper accepted at ARES 2025 as part of the International Workshop on Emerging Digital Identities (EdID).

References

[1] "EU Digital Identity Wallet Home - EU Digital Identity Wallet -." Accessed: Nov. 11, 2024. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/display/EUDIGITALIDENTITYWALLET/EU+Digital+Identity+Wallet+Home

[2] "Certificate Transparency : Certificate Transparency." Accessed: Nov. 11, 2024. [Online]. Available: https://certificate.transparency.dev/

[3] E. Fasllija and S. More, "Guardians of the Registry: Certificate Transparency for Relying Party Authorization in eIDAS 2," Aug. 11, 2025, *Springer*. Accessed: Jul. 30, 2025. [Online]. Available:

https://tugraz.elsevierpure.com/en/publications/guardians-of-the-registry-certificate-transparency-for-relying-pa

[4] “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/

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

[6] “Regulation - EU - 2024/1183 - EN - EUR-Lex.” Accessed: Jul. 30, 2025. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2024/1183

[7] “What are Patricia Merkle Tries? | Alchemy Docs.” Accessed: Jul. 30, 2025. [Online]. Available: https://www.alchemy.com/docs/patricia-merkle-tries