Security Architecture & Development Lifecycle

This technical whitepaper provides a detailed description of Tekle Holographics' security architecture and the processes we follow in developing and maintaining our products securely. It is aimed at enterprise security architects, IT professionals, and auditors who require an in-depth understanding of how Tekle's systems are structured to protect data and how our development practices minimize vulnerabilities. We will cover the architecture of both our cloud and device components, the security features and controls at each layer, and the lifecycle activities (from design to deployment to maintenance) that ensure our solutions remain resilient against threats.

System Architecture Overview

Tekle's product suite includes both **hardware devices** (like the Holo-Table and Holo-Wall series of holographic displays) and **software components** (RealityBridge applications, cloud services, SDKs). The architecture can be viewed in two main parts: the **device-side architecture** and the **server/cloud-side architecture**, which communicate over secure channels.

Device-Side (Edge) Architecture: Each Tekle holographic device is essentially a specialized computing system integrated with projection and tracking hardware. Taking the Holo-Table as an example: it consists of a high-performance Windows-based workstation embedded within the table, connected to projectors and motion-tracking cameras. On that workstation runs Tekle's device software stack, which includes:

- The **Rendering Engine** – responsible for generating the holographic images using the 3D models and tracking data.
- The **Device Controller** – which manages device hardware (projector alignment, tracking calibration, etc.) and provides a local API for control.
- The **RealityBridge Client** – which handles data synchronization with either a local server or the cloud. This client is what sends or receives model data, updates, user annotations, etc.
- The **Local Database/Cache** – where the device can store recently used models, settings, and logs, typically on an encrypted disk partition.

Security features on the device side include a hardened OS environment: we configure Windows with Group Policies to disable unnecessary services, enforce strong account passwords (and often set the device to auto-logon with a limited user for the Tekle software, while the Administrator account is protected). We enable Windows Firewall to only allow needed inbound connections (perhaps from the local server for updates or remote admin from a whitelisted IP). The RealityBridge Client uses only outbound connections (if cloud-connected) initiated from the device, which means the device does not open any listening ports to the wider network, reducing its attack surface. Data on the device's local drive (including the model cache) is encrypted using BitLocker (with TPM if available, plus recovery keys escrowed to the admin). If the device is in an offline deployment, it may still connect with a local server – that communication is typically over a secure WebSockets or gRPC channel on the LAN, using mutual authentication (the device and server exchange certificates). Each device has a unique device certificate issued by Tekle's internal PKI at manufacturing time; this is used to authenticate it to servers (proving it's a legitimate device) and to enable encrypted comms at the device level.

Server/Cloud-Side Architecture: On the server side, the architecture can either be our multi-tenant cloud or a dedicated on-prem server. We'll describe the general architecture applicable to both, with notes on differences:

- The **Application Server** – this is the core of Tekle's collaboration and data management. It runs our server software which includes the collaboration engine (manages multi-user sessions, merges updates from different users), the API backend (REST/GraphQL API that clients call for various operations like fetching model lists, saving changes, authentication, etc.), and integration modules (like connectors to external data sources if needed).

- The **Database** – we use a relational database (PostgreSQL in our cloud) to store metadata: user accounts, permissions, project listings, and references to model files. The database is configured with TLS encryption for connections and row-level security to enforce tenant isolation (each query is automatically filtered by customer ID when in multi-tenant mode).

- The **Storage** – large files (model files, point clouds, etc.) are stored in an object storage service (like Azure Blob Storage or AWS S3 in cloud deployments, or a file system on local disk in on-prem). All stored files are encrypted at rest. Access to these files is always through the application server (in cloud mode, direct pre-signed URLs are used with short lifetimes, so even if someone got a URL it expires quickly).

- **Web Frontend/Portal** – There is a web-based portal for administrators and sometimes end-users, to manage content and settings. This runs as a web app served by the server. It includes dashboards and controls (for example, an admin can upload a new model file via the portal, or assign user permissions). The portal uses the same backend APIs under the hood, so it's subject to the same authentication and authorization checks.

In cloud, the above components run in Tekle's managed environment on Azure. We have them containerized (Docker/Kubernetes), which provides isolation between services and makes scaling easier. Each microservice (if broken down) has its own role, and network policies restrict cross-communication only to what's necessary. We integrate Azure Key Vault for managing secrets/credentials that the application uses (so database passwords, API keys for integration, etc., aren't stored in code or config in plaintext).

Multitenancy and Isolation: In the cloud setup, multiple customer organizations share the infrastructure but not data. We designed the software to enforce tenant isolation at several layers:

- Authentication tokens include the tenant context, and every request is checked against that. For example, if Customer A's user tries to access a resource ID belonging to Customer B by guessing an ID, the system would see a tenant mismatch and deny access.

- The database schemas can be separate per tenant if needed (for high-security clients we even offer an option where each tenant gets their own schema or database instance), but by default we use a single database with tenant ID columns and row-level security policies to automatically segregate.

- File storage paths are segregated by tenant ID as well, and application-level checks ensure one tenant cannot generate a URL for another tenant's file.

- We run automated tests specifically for multitenant security (trying cross-tenant access) in our QA cycle to verify isolation holds.

In on-prem deployments, typically its single-tenant (one organization's data), but even then we might have user-group separation inside (which our app supports via project-level permissions).

Integration Points: Integration can happen at the server side too. For example, if RealityBridge is pulling data from Revit through an API, the server has a connector that authenticates to Revit's API (with stored OAuth tokens or credentials provided by the user admin). These are stored securely (encrypted in DB or in Key Vault). We design integration modules to run with least privilege; e.g., if we integrate with a GIS database, our connector will use read-only credentials unless write-back is needed. We also sandbox integration processes—if a conversion or external call must happen, it's done in a restricted environment to prevent any malicious external data from affecting the rest of the system (think of converting a potentially complex CAD file: we run that conversion in an isolated container to mitigate crashes or exploits in the parser).

Network Architecture: Whether cloud or on-prem, the networking follows a secure model:

- The server listens on specific ports for device or client connections (for instance, port 443 for HTTPS API, maybe another port for a real-time collaboration socket).
- In cloud, an Azure Application Gateway or front-door is used as the entry point, providing TLS termination (with our certificates) and a Web Application Firewall (WAF).
- In on-prem, the server might be directly on the LAN, so we ensure it's behind your firewall; or if remote access is needed, you would expose it via your DMZ with similar protections (we provide guidance on setting up a reverse proxy with a WAF module if needed).
- Devices and web clients connect to the server's endpoints using HTTPS/WSS only. For local LAN, if some clients don't support certs (unlikely, but say a scenario of VR headsets connecting – we ensure they do support encryption), we could support an offline mode with plain HTTP but with network isolation. However, by default, even on LAN we encourage using certificates (we can have the server generate a self-signed CA for the local environment and have devices trust it so that LAN traffic is still TLS encrypted without requiring internet-based CA verification).

Authentication & Authorization: Tekle supports various auth methods. In our cloud, we manage user identities in our system by default (with username/password stored hashed with bcrypt, plus optional MFA via an authenticator app). We also support Single Sign-On (SSO) – a customer can connect their Azure AD, for instance, and users will authenticate via SAML or OAuth2 to their directory; Tekle's system will accept a SAML assertion or JWT and map to a local user account. In on-prem deployments, we integrate with Active Directory or LDAP for user sync and authentication if desired, meaning you can manage users centrally. Permissions are structured by roles and project membership as mentioned.

From an architecture perspective, an **Auth service** issues JWT tokens to clients upon successful login. These tokens are short-lived (e.g., 1 hour) and refresh tokens are used for longer sessions with inactivity timeouts. The tokens carry scopes/claims about what the user can do, and the backend checks these on every request.

We implement strong session management: for example, if an admin revokes a user's access or changes their role, the system can immediately invalidate that user's tokens (we maintain a token blacklist or adjust a token version in user record that we check). For especially security-conscious deployments, we can even enforce per-session tokens with no refresh to require re-login frequently, though usually that's not user-friendly so we balance it.

Secure Development Practices

(Note: There is some overlap with what was presented in the Trust Center and Data Protection overview, but here we focus on concrete processes and tools, potentially going deeper into technical specifics.)

Development Environment Security: All development work for Tekle's software is done in secured environments. Source code is managed in a private Git repository (hosted on an enterprise Git platform with 2FA enforced for all committers). We have branch protection rules so that code must be reviewed via pull requests before merging. Our build servers are isolated and only pull code from the repository read-only; they produce build artifacts in a controlled pipeline (Azure DevOps or Jenkins locked down to our network). The build environment has restricted internet access to avoid supply chain attacks (e.g., when pulling dependencies, it only pulls from trusted package repositories and verifies checksums). We sign our builds with a code signing certificate; the private keys for signing are stored in an HSM or secure vault accessible only by the build pipeline. This means the executables delivered to customers (device software installer, etc.) are digitally signed by "Tekle Holographics" – users can verify they haven't been tampered with in transit.

Design Phase: Before writing code, significant features go through a **Design Review** which includes a section on security considerations. We maintain an internal threat library (common threats to AR/VR systems like unauthorized access, eavesdropping on data streams, hardware tampering, etc.) and check new designs against these. For example, when designing the multi-user architecture, we enumerated potential threats: one user impersonating another, someone injecting false data into a session, etc., and then built controls (session tokens tied to identity, sequence numbers and sanity checks on collaborative actions, etc.) accordingly.

Coding Standards & Tools: We use multiple programming languages in our stack (C++ for performance-critical rendering code on devices, C#/.NET for some server components, Python for scripting tasks, and web languages for the portal). Each has secure coding guidelines. For C++, we heavily use smart pointers and bounds-checking containers (like std::vector) to avoid memory corruption issues, and our code review focuses on catching any risky manual memory management or unchecked buffers. We compile with stack protections and address sanitizer in debug builds to catch issues early. In C# and higher-level languages, we leverage their managed nature to avoid many memory issues but stay vigilant about things like SQL injection (we use parameterized queries or ORMs exclusively) and cross-site scripting (we output-encode any user-generated content in the portal, though currently most data displayed is models or preset fields, not free-form text). Our web developers follow OWASP Top 10 prevention techniques (we run scans using OWASP ZAP or Burp Suite on the web portal during testing).

We incorporate **Static Analysis** tools: e.g., Microsoft's Visual Studio Code Analysis for C++ (which can find some buffer overruns, misuse of APIs), SonarQube or similar for .NET and JavaScript (flagging everything from potential null reference exceptions to injection flaws). We treat warnings from these tools seriously – our build pipeline fails if any new high-severity static analysis warnings appear.

Dependency Management: Our product relies on some third-party libraries (like OpenGL/DirectX, Unity plugins for SDK, maybe some math or physics libraries). We catalog these and monitor advisories. For instance, if we use OpenSSL in our networking stack, we watch for OpenSSL CVEs and update promptly when patches release. We use package managers (NuGet, npm, pip, etc.) that have built-in audit commands; part of our CI pipeline runs "npm audit" for web components, "pip safety check" for Python, etc., to catch known vulnerable versions. If any are flagged, we evaluate and upgrade or apply patches.

Penetration Testing & Review: Internally, beyond automated scans, our security engineer periodically performs code-assisted pen-testing. They might take a recent build of our server and run custom test scripts trying things like privilege escalation (attempting to call admin-only APIs with a normal user token by modifying a request), testing the strength of encryption (ensuring our TLS config has no weak ciphers), or fuzzing input fields. We documented earlier how we also contract external experts for an objective assessment annually. We ensure that issues found (if any) are not only fixed but result in improvements to our process (like updating our coding guidelines to prevent similar bugs, adding new tests to catch that category, etc.).

Secure Configuration Management: Many security issues come from misconfiguration, so Tekle tries to provide sensible, secure defaults and guides for configuration. For example, our application server by default will generate a strong admin password and require it to be changed on first login, rather than using a hardcoded default. For cloud customers, we handle configuration, so we make sure to follow best practices (like Azure security center recommendations). For on-prem deployments, we provide a hardening guide to the client's IT team: steps like "Place the Tekle server on a segmented network, only allow TCP 443 from client subnets to server, change default port if desired, etc." and we often help review their setup if requested.

One specific measure: We know that human error in configuration can cause data leaks (e.g., leaving an S3 bucket public). To counter that, in our cloud, we use policies that *prevent* misconfiguration – our S3 equivalent buckets are configured with deny rules on public access and our code that generates presigned URLs ensures they're scoped to specific files and expire quickly. On the device side, there's not much user config open to them, which is deliberate – we don't expose settings that could lower security unless absolutely needed for debugging (and those are behind "enable developer mode" which is not meant for production use).

Continuous Integration & Deployment (CI/CD): Tekle's CI/CD ensures that every code change triggers an automated pipeline of build -> test -> security scan -> package -> (for cloud) deploy to staging. Only after passing all tests and reviews does a release get deployed to production (for cloud) or packaged for customers. We utilize a **blue-green deployment** strategy in our cloud to deploy updates with minimal risk: new code is deployed to a parallel environment, health-checked, and then traffic is switched from old to new gradually. If any anomaly is detected, we can quickly rollback. This reduces downtime and also ensures if a bug slipped through, it has limited blast radius.

Hardening and Benchmarks: Tekle references standards like CIS Benchmarks for OS hardening. We use the Windows CIS benchmark as a basis for device configuration (with adjustments to not interfere with our applications). For Linux servers (if any component uses Linux, say our database on Azure), we similarly ensure unnecessary services are off and kernel parameters are tuned securely. We keep our systems updated – auto-updates are enabled for security patches on supporting systems or we manually apply them in a timely fashion as part of maintenance windows for more sensitive systems.

Development Lifecycle in Operations: After release, maintenance is key. Tekle uses an **Agile methodology** with sprints typically 2-3 weeks long, which allows us to iterate quickly. Importantly, security and bug fixes are prioritized into every sprint (we don't backlog them indefinitely). We classify bug severity, and security bugs are top priority. Even outside normal sprints, if a critical vulnerability is found, we will do an out-of-band patch release immediately. For instance, if tomorrow a vulnerability in our authentication mechanism was discovered, we would aim to patch it and deploy cloud-side fix within hours and provide a patch to on-prem users perhaps same day with high urgency.

Secure Disposal: In development and testing, we might use sample data. We ensure any real customer data used for testing (which is rare, usually we use dummy data) is scrubbed after use. When retiring old hardware (say a development laptop or a server disk), we follow data destruction guidelines (e.g., using secure erase or physically destroying disks).

Incident Response Integration: Our development process ties with our incident response. If an incident reveals a software flaw, that flows back as a high-priority item to development. We treat security incidents as triggers for immediate bugfix releases. Our IR procedure includes steps for development to create hotfix patches if needed and deploy them fast.

Compliance in Development: As we possibly move towards compliance frameworks, we ensure our development processes produce the documentation and evidence needed. For example, we keep records of code reviews, threat models, and test results, which would help in an ISO 27001 audit or customer security assessment to show we systematically follow our processes.

Security Features and Hardening Options

This section highlights specific security features in Tekle's products that customers can leverage and any optional settings that can further harden the deployment:

- **Audit Logs & Reporting:** Tekle's system provides administrators with access to audit logs, as mentioned earlier. Admins can view logs of user logins, data exports, and more via the portal or by exporting logs. We also have an "Audit Report" feature which can summarize key activities over a period (useful for periodic compliance review).

- **Configurable Password/MFA Policies:** For the built-in user management, we offer settings like minimum password length, complexity requirements, session timeout length, and mandatory MFA. These can be adjusted to align with corporate policies. If integrated with SSO, the corporate IdP policies apply.

- **IP Allowlisting:** On request, in our cloud we can enable IP allowlists for admin functions – for instance, only permit access to the admin portal from your office's IP ranges or via your VPN. For on-prem, that's naturally under your network control.

- **Read-Only Modes:** If a deployment is meant to be strictly one-way (data comes in but never modified from Tekle's side), we can set roles such that users only have viewing rights and cannot alter source data through our system. This can be a safeguard in some workflows.

- **Data Segmentation:** You can create separate "projects" or workspaces within Tekle's system to segregate data (like projects for different departments) and control which users see which projects. This internal segmentation helps ensure that even internally, users only access what they should.

- **Backup Encryption and Key Escrow:** For on-prem customers concerned about backup confidentiality, Tekle's backup tools can encrypt backup files with a client-provided key so that even if those backups are stored off-site, only the client can decrypt them. We can integrate with hardware security modules if the client uses them for key storage.

- **Vulnerability Scanning:** We encourage clients to run their own scans (and many do). Tekle will assist in resolving any findings from client-run vulnerability scans of our deployed systems. We design our system to have a good security baseline, so typical Nessus or Qualys scans come out clean or with minor informational warnings, which we then address if possible.

- **Optional Enhanced Security Mode:** For extremely sensitive environments, Tekle can provide an "Enhanced Security Mode" configuration. This is essentially a set of toggles to maximize security: it forces MFA for all users, shortens token lifetimes, enables additional logging verbosity, and can disable certain features that may carry risk (for example, disabling the ability to use the THSDK to run custom code on the device, ensuring only Tekle-signed code runs – useful if you don't plan to allow custom apps for security reasons). It might also incorporate things like requiring dual-admin approval for critical actions (we can implement a feature where two admin accounts are needed to, say, delete a big dataset, if a client wants that kind of safeguard).

Network and Data Flow Diagram

(textual description)

To further clarify how our security architecture comes together, consider a typical data flow in a Tekle Cloud scenario, from data input to visualization:

1. An engineer at a client site has a design model in Autodesk Revit. Using the Tekle RealityBridge plugin within Revit, they click "Upload to Tekle" for a specific model. The plugin, already authenticated to Tekle Cloud via the user's credentials, packages the model data and sends it over HTTPS to Tekle's Upload API endpoint. This request is authenticated with the user's token, and the data is encrypted in transit.

2. Tekle Cloud receives the model data. The Application Server verifies the user's authorization to add models in their project. It then stores the model file in encrypted storage and records an entry in the database (e.g., model name, metadata, owner).

3. A colleague in another location wants to view this model on a Holo-Table. They power on the Holo-Table device and log in (either with their Tekle account or via SSO which the device supports through a web-based login flow on an attached tablet or screen). Upon login, the device's RealityBridge client obtains a scoped session token.

4. Using the device interface, the user selects the project and the model to load. The RealityBridge client requests the model from Tekle Cloud – the request includes the device's identity and user's token. The server authorizes it and responds with either the data or a secure download URL. The device then downloads the model data securely.

5. The model loads on the device and is rendered holographically. As the user interacts (rotating the model, adding an annotation pin, etc.), those interactions can be synced back: e.g., if multi-user, it sends the change (like "annotation added at coordinate X by User Y") up to the server's collaboration service, which then broadcasts to any other users/devices in the session.

6. All through this, the communications are encrypted. If someone were to intercept the network traffic, they'd see only TLS-encrypted data. The use of secure tokens means even if one message was intercepted, it couldn't be reused after token expiration or outside that session.

7. Every significant action is logged: the upload by the engineer is logged with their ID and timestamp, the device login is logged, the model download is logged, and the annotation addition is logged (with content, user, time). These logs are available to the client's admin later.

8. When the model is updated or removed, similar flows occur with checks to ensure only authorized changes happen (maybe only project admins can delete models, for instance).

This narrative underscores how different components interact and how security is enforced at each step. Even in an offline scenario, similar flows occur but within a closed network: the difference being the plugin might directly connect to a local server and the device connects within LAN, but they'd still use secure protocols, just not leaving the facility.

Operational Security & Maintenance

Beyond the design and development, Tekle's operations team ensures the security architecture remains effective during actual use:

- **Patching and Updates:** Tekle operations monitors for any newly disclosed vulnerabilities affecting our stack (OS, DB, web server, etc.). We apply OS patches monthly at minimum (so our cloud servers get the latest security updates; on-prem recommendations are given to clients). Urgent patches (like for 0-day Windows issues) we apply as soon as possible, coordinating brief maintenance windows if needed.

- **Monitoring:** We use a SIEM to aggregate logs from devices (in cloud) and servers, and possibly from certain on-prem deployments if clients send us logs. The SIEM has correlation rules to detect anomalies (like multiple failed logins followed by a success could trigger an alert of a brute-force attempt succeeded). We also leverage cloud-native security services (Azure Security Center) to get alerts about unusual VM behavior or user account usage.

- **Incident Drills:** We periodically simulate incidents to test our response. For example, we might do a drill where we simulate a compromised user account to see if our monitoring catches it and if our team properly rotates keys and locks it down per the runbook. These drills keep the team sharp and often suggest improvements.

- **Customer Access for Auditing:** We provide ways for customers to verify the architecture's security. E.g., a client's security team might want to run a penetration test against their instance of Tekle Cloud. We support this under agreed conditions (like not targeting other tenants, scheduling so it doesn't impact service). Typically, such tests come back showing robust security, reinforcing trust.

Roadmap and Future Enhancements

Security is never "finished". Tekle is continually investing in our security architecture. Some planned or in-progress enhancements include:

- **Zero Trust Architecture:** We are evolving our internal IT and product network to more of a zero-trust model – meaning even within our environment, every service call is authenticated, and minimal privileges are given. This will further mitigate insider risks or lateral movement if one component is compromised.

- **Hardware Security Modules (HSMs):** For cloud key management, we are integrating HSM-backed keys for critical secrets (like the master encryption keys for databases). This adds a layer of physical security assurance (keys cannot be extracted in software).

- **Enhanced Device Security:** We are prototyping secure boot on our next-gen hardware, so that from firmware up to OS, everything is cryptographically verified. Additionally, exploring options like self-encrypting drives for data at rest on devices for even stronger protection.

- **Third-Party Certifications:** We aim to undergo ISO 27001 certification audit next year, which will formally validate our security management program. We are also aligning with IEC 62443 (an industrial control systems security standard) for our hardware, which might be relevant to clients in critical infrastructure.

- **Privacy Enhancements:** Though not directly architecture, we plan to introduce more fine-grained data retention controls (allowing a client to set how long logs are kept, for example, and automating deletion) to ease compliance burdens.

- **AI/ML Security Monitoring:** We are evaluating machine learning tools to analyze patterns in holographic usage that might indicate security issues (like an account that suddenly downloads an unusual amount of data might be an insider exfiltration – an ML model could flag that). This is future-looking, but on our radar.

Tekle Holographics' security architecture is the result of careful planning and continuous refinement. We have built layers of defense into our devices and cloud services to protect against modern threats, and we maintain a disciplined development lifecycle to keep our software trustworthy. This document has walked through the technical specifics of how we achieve security – from encryption and access control to network design and coding practices. We hope it provides clarity and assurance to your security review process.

For any further technical questions or if you require an architecture diagram or a live discussion with our security architects, please reach out. Tekle is committed to openness in our security design, because an informed customer is an empowered customer. By understanding our architecture, you can better integrate our solution into your security environment and policies. We view our relationship as a partnership in which security is a shared priority and responsibility, and we are eager to work with your teams to ensure a successful and secure deployment.