# **🔒 Security Design Plan**

**Project Name:** Defeah Bible Study  
 **Version:** 1.0  
 **Date:** July 2025

## **1. 🔐 Data Privacy & Access Control**

### **📌 User Data**

| **Type** | **Collected?** | **Notes** |
| --- | --- | --- |
| Email / Auth ID | ✅ | Stored securely via Firebase/Supabase |
| Guide queries | ✅ | Stored only if saved or in history |
| LLM request content | ⚠️ | Sanitized & anonymized before LLM call |

### **🔐 Access Control**

| **Resource** | **Access Scope** |
| --- | --- |
| Study Guides | Public for generation, private for saved |
| User Data | Strict per-user access (row-level security in Supabase) |

### **✅ Measures**

* Use Supabase Row Level Security (RLS) on tables like saved\_guides
* Enforce user-level isolation: user\_id = auth.uid()
* Store only non-sensitive data (no personal prayers, notes, etc.)
* Ensure users cannot access each other’s saved history or feedback

## **2. 🧾 Secure Authentication & Encryption**

| **Mechanism** | **Technology** | **Notes** |
| --- | --- | --- |
| User Login | Firebase Auth or Supabase Auth | Google, Apple |
| Token Validation | JWT-based | Tokens validated per request |
| Network Encryption | HTTPS (TLS 1.2+) | Required for all frontend/backend communication |
| Device Storage | Encrypted local storage | Use Flutter Secure Storage for tokens/preferences |

## **3. 🧠 Prompt Injection Prevention (LLM Security)**

| **Attack** | **Risk** | **Mitigation** |
| --- | --- | --- |
| Prompt Injection | High | Strict system prompt: “Stay within Biblical context…” |
| Malicious Input | Medium | Sanitize inputs: regex (verse), length limits (topics) |
| Abuse/Spam | Medium | Rate limiting per IP/token, max tokens |
| Data Leakage | Low | Never send personal data to LLM, log only hashed identifiers |

## **4. 🛡️ Threat Model & Mitigation Strategies**

| **Threat** | **Vector** | **Mitigation** |
| --- | --- | --- |
| Unauthorized guide access | API misuse | JWT verification + Supabase RLS |
| LLM abuse | Prompt injection | Structured prompts + input filtering |
| API scraping | Anonymous overuse | Server-side throttling (per IP + per user/token) |
| Data tampering | Insecure client storage | Use Flutter Secure Storage |
| Feedback manipulation | Unauthenticated actions | Require login for guide feedback |

## **5. 📋 Logging & Audit Trail**

* Supabase logs and auth logs enabled
* Firebase Crashlytics or Sentry for frontend crash/error capture
* Backend logs LLM request metadata (hashed input values, timestamps)
* Store request/response metrics for performance alerts (guide delay >3s)

## **✅ Summary of Best Practices**

| **Area** | **Implementation** |
| --- | --- |
| Encryption | HTTPS, JWT, encrypted local storage (Flutter Secure Storage) |
| Authentication | Firebase/Supabase Auth (Google, Apple) |
| Access Control | Supabase RLS + JWT + per-user scoping |
| Input Security | Prompt filters, strict templates, regex sanitation |
| Rate Limits | 3/hour (anon), 30/hour (auth), handled in API gateway |
| LLM Privacy | Anonymize + sanitize input, hash logs, avoid storing personal notes/prayers |

## **🧾 Compliance Notes (for Razorpay / Google Pay Integration)**

* ✅ PCI-DSS Scope: Razorpay handles PCI compliance — ensure only Razorpay-hosted UI used.
* ❌ App should not handle or store card data locally.
* 🔐 Use Razorpay's encrypted checkout for donations (e.g., ₹100 default)
* 📝 Privacy Policy must reference Razorpay/Google Pay as third-party processors.