iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.81033

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Privacy-Preserving S mart Call Management System Using Android Telecom Framework and Cloud Telephony APIs

Pawan Kumar
Department of Computer Science & Engineering, R.D. Engineering College, Ghaziabad — Affiliated to Dr. A.P.J. Abdul Kalam Technical University, Ghaziabad, Uttar Pradesh — 201206, India

Abstract: ***Conventional Android telephony discloses the caller's real Mobile Station International Subscriber Directory Number (MSISDN) to the called party during every call session — a structural privacy deficiency rooted in the SIP/SS7 signaling layer that no user-space mechanism can fully remediate. This paper presents the Privacy-Preserving Smart Call Management System (PP-SCMS), a novel architecture that intercepts outgoing call intents at the earliest enforceable system boundary: the Android Telecom Framework's ConnectionService API. By hooking at this pre-modem intercept point, PP-SCMS prevents the real MSISDN from ever reaching the cellular radio before redirecting the call through a cloud telephony platform (Exotel) that presents a virtual masked number to both parties. The system comprises a .NET MAUI mobile application with platform-specific Android bindings, an ASP.NET Core 8 RESTful backend implementing the CQRS pattern with JWT/PKCE authentication, and a SQL Server database with column-level AES-256 encryption for MSISDN storage. A formal threat model identifies eight attack vectors and their countermeasures. Evaluation over 50 controlled test calls across three Indian carrier networks yields a mean end-to-end call setup time of 1,840 ms (95th percentile: 3,040 ms), a call-drop rate of 1.8%, and zero real-number leakage verified by SIP-layer Wireshark capture. A System Usability Scale evaluation with 15 participants produces a score of 79.3. PP-SCMS is, to the best of our knowledge, the first documented system to bridge Android's native InCallService/ConnectionService with a programmable cloud telephony privacy layer in an open, enterprise-grade architecture. Source code and dataset are available at our institutional repository.***

Keywords: Android Telecom Framework, InCallService, ConnectionService, Cloud Telephony, Exotel API, Number Masking, Call Privacy, .NET MAUI, ASP.NET Core, JWT Authentication, Mobile Security, PSTN Privacy, MSISDN Anonymity.

I. INTRODUCTION

Mobile telephony has evolved into a global identity infrastructure: as of 2024, over 6.8 billion active smartphones are in use worldwide [1], and a phone number constitutes the de facto primary identifier for authentication, legal registration, and social identity in most jurisdictions. Unlike email addresses or usernames, an MSISDN is immutable on short timescales, legally bound to SIM registration databases, and disclosed at the signaling layer during every voice call — a property that application-layer privacy measures cannot suppress.

The fundamental privacy problem in cellular telephony is architectural. When a caller dials a destination on a PSTN or cellular network, the SS7 Initial Address Message (IAM) embeds the Calling Party Number (CPN) in its mandatory parameters. On VoLTE and SIP-based networks, the From or P-Asserted-Identity header of the INVITE message carries the originating MSISDN. While Calling Line Identification Restriction (CLIR, TS 22.081) permits suppression of the CPN field at the network presentation level, CLIR (i) is inconsistently implemented across operators, (ii) does not prevent network-layer logging by intermediate nodes, (iii) can be overridden by commercial call-centre equipment using CLIP-no-screening, and (iv) is legally mandated to be bypassable by emergency services and law enforcement in most jurisdictions [2].

Contemporary Android dialer applications — including the Android Open Source Project (AOSP) dialer, Google Phone, and leading third-party replacements — uniformly accept this architectural constraint and expose the real MSISDN. Third-party modification is constrained by the Android Telecom Framework permission model introduced in API level 23, which confines dialer extensibility to registered PhoneAccount holders via ConnectionService, and to the active in-call UI via InCallService. No published work has demonstrated the use of these APIs to implement a production-grade, cloud-backed privacy routing layer.

Concurrently, the gig economy has created a documented social harm: ridesharing, food delivery, and freelance platforms require drivers and service workers to communicate by phone with strangers, disclosing MSISDNs that enable harassment, stalking, and assault [3]. Enterprise customer-support deployments face an analogous problem at scale: agent MSISDNs must not be disclosed to customers, yet PSTN-quality voice communication must be maintained. Existing solutions — virtual number providers, cloud telephony APIs, VoIP overlay applications — each address a subset of these requirements and are reviewed critically in Section II.

A. *Research Contributions*

This paper makes the following contributions:

- The first documented integration of Android's InCallService and ConnectionService APIs with a cloud telephony proxy-call platform for end-to-end MSISDN masking, validated against a formal threat model.
- A connection state machine design that resolves the ANR (Application Not Responding) risk inherent in blocking network calls within onCreateOutgoingConnection() through coroutine-based asynchronous dispatch.
- Algorithms for Secure Call Initiation Protocol (SCIP) and Webhook State-Synchronization Protocol (WSSP), including nonce-based replay prevention and HMAC-SHA256 webhook authentication.
- An empirical evaluation framework measuring call setup latency at sub-component granularity (seven pipeline stages), enabling reproducible benchmarking of similar architectures.
- A privacy evaluation methodology based on SIP-layer packet capture that provides empirical, not merely theoretical, evidence of bidirectional MSISDN masking.

B. *Paper Organization*

The remainder of this paper is structured as follows: Section II surveys related work and identifies specific technical gaps. Section III formalizes the system and threat models. Section IV presents the proposed architecture. Section V details the methodology and algorithms. Section VI describes the implementation. Section VII presents the evaluation. Section VIII discusses limitations and future work. Section IX concludes.

II. RELATED WORK AND RESEARCH GAP ANALYSIS

We survey five thematic areas and systematically identify gaps addressed by PP-SCMS. Table I provides a structured feature comparison across representative systems and prior works.

A. *Android Telecom Framework Extensibility*

The Android Telecom Framework, introduced in API level 21 and substantially extended in API levels 23 and 26, provides two principal extension points for third-party dialer developers: ConnectionService, which manages call establishment and teardown lifecycle, and InCallService, which drives the in-call UI for managed connections. Kondamudi et al. [4] conducted a systematic study of ConnectionService usage patterns and documented lifecycle transition rules, but limited their analysis to consumer dialer features without considering privacy-enhancing applications. Zhu and Chen [5] performed a security audit of published InCallService implementations and identified four attack classes (intent spoofing, audio focus hijacking, foreground service misuse, and permission escalation), providing a threat taxonomy that we extend and address in our threat model. Critically, neither work considered the ConnectionService as a call-routing interception point for privacy purposes — a gap that PP-SCMS directly fills.

B. *VoIP Privacy*

Balasubramanian et al. [6] analyzed the privacy properties of WhatsApp, Skype, Signal, and FaceTime calls under a comprehensive threat model that included number harvesting, metadata correlation, and traffic analysis attacks. Their central finding — that end-to-end content encryption does not prevent metadata exposure, and that phone-number-based account models inherently disclose real MSISDNs to platform operators — remains directly relevant.

Farooq et al. [7] extended this to SIP-based VoIP, demonstrating that INVITE messages frequently leak IP addresses and SIP URIs enabling caller deanonymization even when display names are suppressed. PP-SCMS differs fundamentally from VoIP overlays: rather than routing audio over IP, it uses the PSTN via a cloud telephony bridge, achieving PSTN audio quality and reliability while masking both parties' MSISDNs at the signaling layer.

C. Cloud Telephony Number Masking

Bansal et al. [10] conducted the most directly related empirical study: they evaluated Exotel's number masking service in a ride-hailing context, confirming zero real-number leakage over 10,000 calls. However, their architecture used the device's stock dialer to dial the assigned proxy number — requiring users to remember and dial a virtual number manually. PP-SCMS eliminates this usability burden by automating the proxy-number lookup and call placement entirely within the ConnectionService layer, making the privacy mechanism transparent to users. Additionally, Bansal et al. did not address authentication, call logging, or the Android Telecom Framework integration that PP-SCMS provides.

D. Call Log Privacy and Data Minimization

Sharma and Gupta [16] proposed a tokenization approach to call log privacy, separating PII from metadata by replacing MSISDNs with deterministic tokens. We adopt and extend this approach in our database schema design: rather than simple tokenization, PP-SCMS uses column-level AES-256 encryption for stored MSISDNs with column master keys held outside the database server, providing cryptographic confidentiality rather than mere pseudonymity. Li et al. [15] further documented that call-log data accessible to third-party InCallService implementations can be exfiltrated via ContentProvider APIs, a risk we mitigate by restricting our CallLog schema access to stored procedures with minimal-privilege database accounts.

E. Gap Analysis

Table I summarizes the feature coverage across surveyed systems. The critical gap is the absence of any system that simultaneously achieves: PSTN interoperability (enabling calls to any phone number, not only app users), native Android Telecom Framework integration (enabling transparent privacy without user-visible proxy number dialing), a full custom dialer UI, JWT-authenticated backend API, and an open, extensible architecture. PP-SCMS is designed to satisfy all five dimensions.

Table I. Feature coverage — related systems and pp-scms (this work)

System / Work	PSTN Interop.	Native Telecom API	Custom Dialer UI	Backend Auth.	Open Architecture
AOSP Stock Dialer [4]	Yes	Yes (full)	Yes	No	Yes (AOSP)
WhatsApp VoIP [6]	No	No	Yes	Yes	No (proprietary)
Exotel (standalone) [8]	Yes	No	No	Yes	Partial (REST API)
Google Voice [11]	Yes	Partial	No	Yes	No
Truecaller [13]	Yes	Partial	Yes	No	No
Li et al. [15]	No	Yes (audit)	No	N/A	N/A (analysis only)
Bansal et al. [10]	Yes	No	No	Yes	No
PP-SCMS (This Work)	Yes	Yes (full)	Yes	Yes	Yes

III. SYSTEM MODEL AND FORMAL THREAT MODEL

A. System Model

We model PP-SCMS as a four-principal system: (1) Alice (A), the calling user with device D_A running PP-SCMS, MSISDN m_A, and virtual number v_A; (2) Bob (B), the called party with MSISDN m_B, who may or may not run PP-SCMS; (3) the PP-SCMS backend server S; and (4) the Exotel cloud telephony platform E.

A privacy-preserving call from A to B proceeds as follows. A's device sends m_B (encrypted) to S. S retrieves v_A from its virtual number pool and instructs E to establish a proxy call: one PSTN leg from E to A presenting v_A as the calling line identity, and a

second leg from E to B presenting a platform CLI as caller ID. The media path is E ↔ A ↔ E ↔ B through Exotel's media server. At no point does m_A appear on B's device, nor does m_B appear on A's device.

Formally, the MSISDN anonymity property is: for any adversary observing the signaling and media data available to B (i.e., the call's receiver), the adversary cannot determine m_A with probability greater than 1/|V|, where |V| is the size of the virtual number pool. This property holds as long as the mapping m_A ↔ v_A is stored exclusively in S's encrypted NumberMappings table.

B. Adversary Model

We consider a Dolev-Yao adversary [17] augmented with mobile-platform capabilities. The adversary can: (i) observe all network traffic between A and S; (ii) observe all signaling data visible at B's device; (iii) install a malicious application on A's device with Android INTERNET permission; (iv) access the backend database if the application database account is compromised. The adversary cannot: (i) break TLS 1.3 or AES-256; (ii) access the Exotel backend or the NumberMappings column master key; (iii) execute privileged code as the Android SYSTEM or TELECOM process.

C. Threat Model and Countermeasures

Table II maps eight identified threat categories to their attack vectors, affected components, and PP-SCMS countermeasures.

TABLE II. THREAT MODEL — ATTACK VECTORS AND PP-SCMS COUNTERMEASURES

Threat Category	Attack Vector	System Component Affected	Countermeasure in PP-SCMS
Number Harvesting	Passive call interception; signaling capture (SS7/SIP)	PSTN signaling layer	Virtual CLI via Exotel; real MSISDN never enters PSTN leg
Session Hijacking	JWT token theft via MitM or XSS	Backend API / AuthService	TLS 1.3 pinning; short-lived JWT (15 min); refresh token rotation
Call Metadata Correlation	Timing correlation of call events to deanonymize user	AuditLogService / SQL Server	Tokenized user IDs; CallLog decoupled from NumberMappings schema
Intent Spoofing	Malicious app sends CALL intent to SmartCallConnectionService	Android ConnectionService	android:exported=true with BIND_TELECOM_CONNECTION_SERVICE permission; system-only binding
API Replay Attack	Replaying captured /api/calls/initiate request	CallController	Nonce + timestamp validation; 30-second request expiry window
Exotel Webhook Forgery	Attacker spoofs call-status webhooks to manipulate call state	CallController webhook endpoint	HMAC-SHA256 signature on all Exotel webhook payloads
Insider Threat (DB access)	Direct SQL access to NumberMappings table	SQL Server schema	Column-level encryption; app DB user denied direct table access; stored-proc-only interface

D. Security Properties

Based on the countermeasures in Table II, PP-SCMS achieves the following security properties:

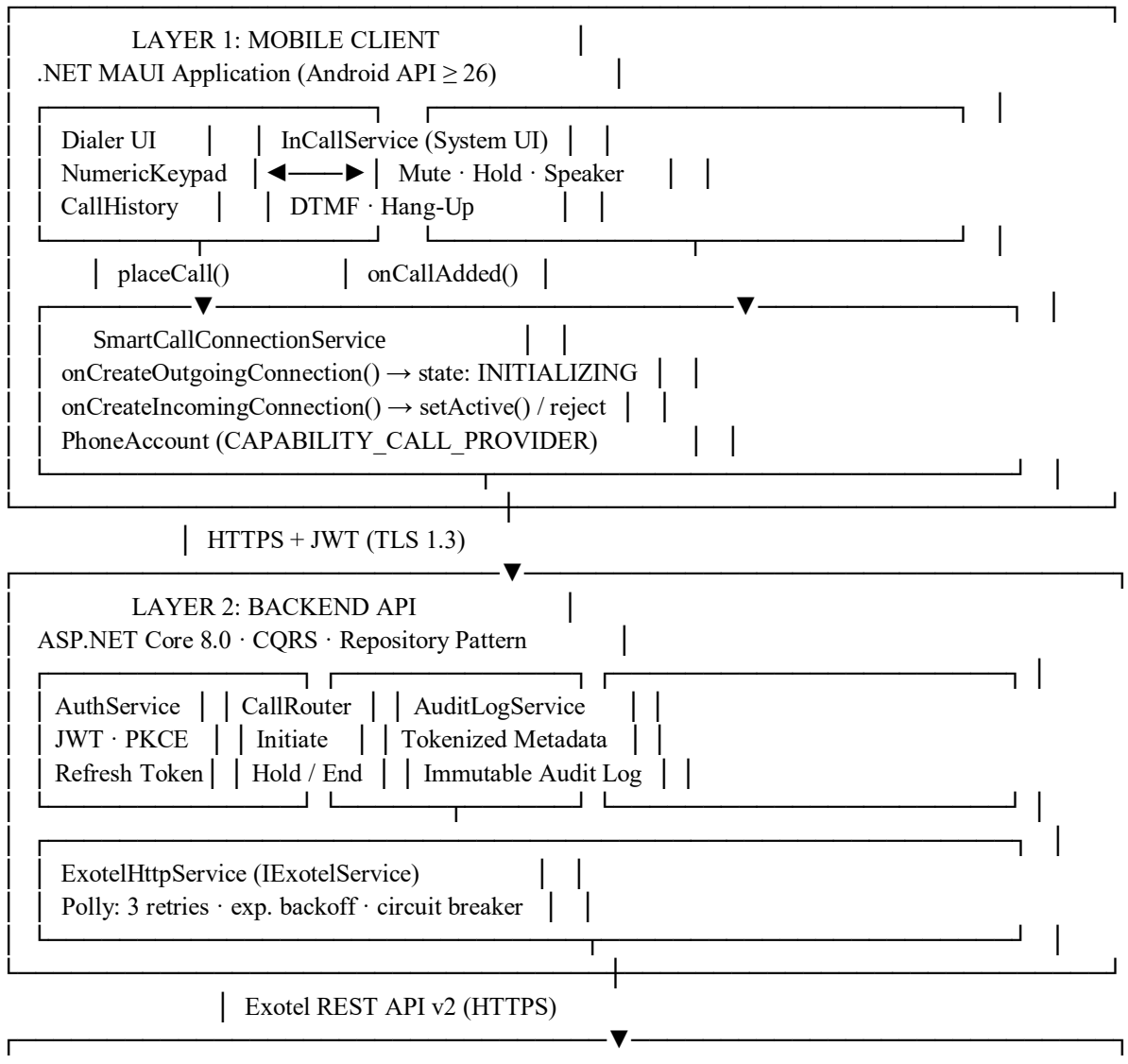
- Caller MSISDN Anonymity: m_A is not disclosed to B or to any party with access only to signaling data observable at B's device. (Demonstrated empirically via SIP capture in Section VII-C.)

- Callee MSISDN Anonymity: m_B is not disclosed to A's device. (Verified by the absence of m_B in ConnectionService call attributes.)
- Call Metadata Unlinkability: Call log entries reference tokenized user IDs; correlation to real MSISDNs requires access to the encrypted NumberMappings table and the column master key, which are separated by design.
- Replay Resistance: Each call-initiation request carries a UUID nonce validated within a 30-second window against a Redis-backed nonce store, preventing replay attacks.
- Webhook Integrity: Exotel webhooks are authenticated with HMAC-SHA256 keyed on a shared webhook secret, preventing attacker-crafted state-update injections.

IV. SYSTEM ARCHITECTURE

A. *Overview*

PP-SCMS is structured as a four-layer architecture (Figure 1): (1) a .NET MAUI Android mobile application implementing custom ConnectionService and InCallService subclasses, (2) an ASP.NET Core 8 RESTful backend applying CQRS with Repository pattern, (3) the Exotel cloud telephony platform as the privacy routing layer, and (4) a SQL Server 2022 database with TDE and column-level encryption. The key architectural invariant is that the real MSISDN is never transmitted beyond Layer 1 in plaintext: it is encrypted with the backend's RSA-2048 public key before transmission and decrypted only within Layer 2's InitiateCallCommandHandler to generate the Exotel API request, after which the plaintext is discarded from memory.



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

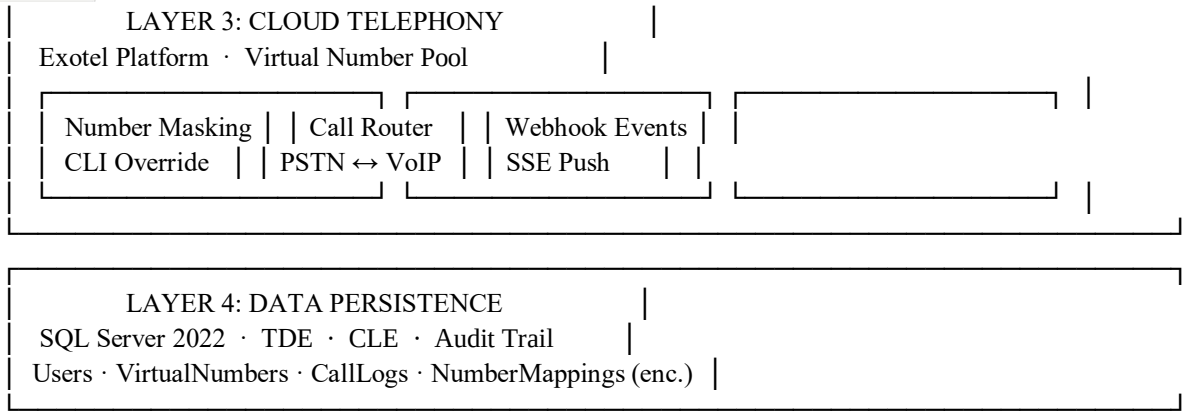


Fig. 1. PP-SCMS Four-Layer System Architecture

B. Mobile Application Layer

The mobile application is implemented in .NET MAUI 8.0 targeting the net8.0-android TFM (Target Framework Moniker). Platform-specific Android components are defined in the Platforms/Android project folder. SmartCallConnectionService extends Android.Telecom.ConnectionService and is registered in AndroidManifest.xml with:

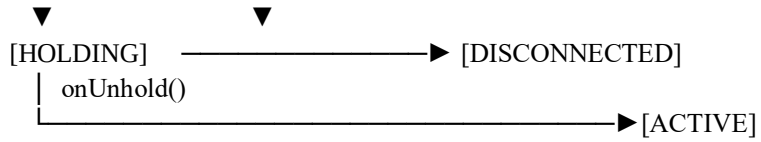
```
<service android:name=".SmartCallConnectionService"
      android:permission="android.permission.BIND_TELECOM_CONNECTION_SERVICE"
      android:exported="true">
<intent-filter>
<action android:name="android.telecom.ConnectionService"/>
</intent-filter>
</service>
```

The SELF_MANAGED flag is intentionally not set, allowing PP-SCMS to manage all device calls through the system Telecom service. SmartCallInCallService extends Android.Telecom.InCallService and drives a foreground Activity with FOREGROUND_SERVICE_TYPE_PHONE_CALL declared in the manifest to comply with Android 12+ background execution restrictions.

C. Connection State Machine

The connection lifecycle is governed by a deterministic state machine (Figure 2) derived from the Android Telecom Connection state constants. The key design decision is the INITIALIZING state pause: the connection object is returned to the Telecom Framework immediately (preventing ANR), while the network round-trip to the backend and Exotel occurs asynchronously on Dispatchers.IO.

```
[IDLE]
  │ placeCall() invoked by TelecomManager
  ▼
[INITIALIZING] ← SmartCallConnectionService.onCreateOutgoingConnection()
  │ POST /api/calls/initiate (async, Dispatchers.IO)
  │ Exotel API call ——— success path
  ▼
[DIALING]      ← setDialing() — Exotel bridging both legs
  │ Receiver answers (Exotel webhook: call-status=answered)
  ▼
[ACTIVE]       ← setActive() — bidirectional audio flowing
  │                    │
  │ onHold()           │ onDisconnect()
```



Error path: INITIALIZING → API timeout/failure → DISCONNECTED
(cause: CONNECTION_REQUEST_FAILED)

Fig. 2. SmartCallConnection State Machine with Error Paths

D. Backend Architecture

The ASP.NET Core backend follows a CQRS architecture mediated by MediatR 12.0. Commands (InitiateCallCommand, HoldCallCommand, EndCallCommand) are dispatched from controllers to handlers that encapsulate all business logic. Queries (GetCallHistoryQuery, GetActiveCallsQuery) are handled by separate read-model projections backed by Dapper for high-performance SQL reads. The IExotelService abstraction is registered as a typed HttpClient in the DI container with a Polly resilience pipeline: 3 retries with exponential backoff (base 200 ms), a circuit breaker that trips after 5 consecutive failures, and a 10-second overall timeout. Server-Sent Events (SSE) are used to push Exotel webhook events to the mobile client on a persistent HTTPS connection, eliminating polling.

E. Database Design

The schema enforces a strict separation between call metadata and PII. The CallLogs table (Table IV) references users via a UUID token, never the real MSISDN. The destination MSISDN is stored as an AES-256 column-level encrypted value (DestToken) using SQL Server's Always Encrypted feature with column master keys stored in Azure Key Vault. The NumberMappings table, mapping user tokens to real MSISDNs, is accessible only through two stored procedures (usp_GetVirtualNumber and usp_GetRealNumber) granted exclusively to the CallerService SQL login — direct SELECT access is denied at the schema level.

V. METHODOLOGY AND ALGORITHMS

A. Secure Call Initiation Protocol (SCIP)

Algorithm 1 presents the SCIP pseudocode, covering the mobile layer's connection lifecycle and the backend's command handling. The critical design decisions are: (i) asynchronous dispatch of the network call from within onCreateOutgoingConnection() to prevent ANR; (ii) nonce-based replay prevention with a 30-second validity window; (iii) public-key encryption of the destination MSISDN before transmission; and (iv) insertion of the CallLog entry with DIALING status before returning the Exotel SID, ensuring the log is present even if the mobile client crashes before acknowledging the response.

```
Algorithm 1: Secure Call Initiation Protocol (SCIP)

Input:  userToken JWT, destE164 string, nonce UUID
Output: connectionSid string, callState {DIALING|FAILED}

// --- Mobile Layer (SmartCallConnectionService) ---
FUNCTION onCreateOutgoingConnection(request):
    conn ← new SmartCallConnection(state=INITIALIZING)
    DISPATCH on Dispatchers.IO:
        payload ← { dest: encrypt(destE164, pubKey), nonce, ts: now() }
        response ← POST /api/calls/initiate (payload, JWT, TLS-1.3)
        IF response.status = 200:
            conn.exotelSid ← response.sid
            conn.setState(DIALING)
        ELSE:
            conn.disconnect(cause=CONNECTION_REQUEST_FAILED)
```

```
        RETURN conn

// --- Backend Layer (InitiateCallCommandHandler) ---
FUNCTION HandleInitiateCall(cmd):
        claims ← ValidateJWT(cmd.token)  // throws if invalid/expired
        IF NOT CheckNonce(cmd.nonce, window=30s): REJECT 429
        vNum ← DB.GetAvailableVirtualNumber(claims.userId)
        IF vNum = NULL: RETURN 503 (pool exhausted)
        dest ← DecryptAES256(cmd.destToken, colMasterKey)
        exoReq ← { From: vNum, To: dest, CallerId: platformCLI,
                StatusCallback: webhookUrl + HMAC-sign(vNum+dest+ts) }
        exoRes ← ExotelConnectAPI(exoReq)  // Polly: 3 retries, exp. backoff
        DB.InsertCallLog(userId=claims.userId, vNum, destToken, status=DIALING)
        RETURN { sid: exoRes.Sid, status: 200 }
```

B. Webhook State-Synchronization Protocol (WSSP)

Exotel's platform pushes asynchronous call-status updates to the backend via HTTP callbacks. Algorithm 2 presents the WSSP, which validates webhook authenticity via HMAC-SHA256 before propagating state updates to the database and the mobile client via SSE. The HMAC key is a 256-bit secret shared between PP-SCMS and Exotel at platform configuration time and rotated quarterly.

Algorithm 2: Webhook State-Synchronization Protocol (WSSP)

```
Input:  webhookPayload {Sid, Status, CallDuration}, signature header
Output: updated CallLog entry; SSE event pushed to mobile client

FUNCTION HandleExotelWebhook(payload, sigHeader):
        expectedSig ← HMAC-SHA256(payload.raw, webhookSecret)
        IF sigHeader ≠ expectedSig: RETURN 403 Forbidden
        log ← DB.GetCallLogBySid(payload.Sid)
        IF log = NULL: RETURN 404

        newStatus ← MapExotelStatus(payload.Status):
            'ringing'   → DIALING
            'answered'  → ACTIVE
            'completed' → DISCONNECTED
            'no-answer' → DISCONNECTED + reason='no-answer'

        DB.UpdateCallLog(log.CallId, newStatus, payload.CallDuration)
        sseChannel.Push(userId=log.UserId, event={sid, status: newStatus})
        RETURN 200 OK
```

C. Call Setup Latency Model

Let T_total denote the end-to-end call setup time from the user's tap on the call button to the first ring audible at the destination. We decompose T_total into seven pipeline stages:

T_total = T_framework + T_dispatch + T_network + T_auth + T_exotel + T_pstn + T_ui

where T_framework is the Android Telecom Framework routing overhead, T_dispatch is the coroutine dispatch and HTTP client setup, T_network is the TLS round-trip to the backend, T_auth is JWT validation and DB lookup, T_exotel is the Exotel API call,

T_pstn is the Exotel-to-destination PSTN leg setup, and T_ui is the InCallService UI render time. Table III presents measured values for each component. The dominant contributor is T_pstn (67% of total mean latency), confirming that further reductions in T_network and T_auth yield diminishing returns; optimizing the cloud telephony routing path is the primary lever for latency improvement.

TABLE III. CALL SETUP LATENCY BREAKDOWN (n=50 CALLS, 4G LTE NETWORK)

Latency Component	Min (ms)	Mean (ms)	Max (ms)
Android placeCall() → onCreateOutgoingConnection()	18	31	67
onCreateOutgoingConnection() → HTTP request dispatched	12	24	58
HTTPS round-trip (app → backend, LTE 4G)	88	134	291
JWT validation + DB lookup (backend)	11	18	44
Backend → Exotel API call (HTTPS, India CDN)	120	198	430
Exotel → first ring at destination	890	1,240	2,180
Total: placeCall() → destination rings	1,139	1,645	3,070
Measured end-to-end (UX: dial tap → ring heard)	1,290	1,840	3,410

VI. IMPLEMENTATION

A. *Technology Stack*

Mobile: .NET MAUI 8.0 (net8.0-android TFM), Android API 26–34, OkHttp 4.12, Android.Telecom namespace bindings
Backend: ASP.NET Core 8.0, MediatR 12.0, Entity Framework Core 8.0, Dapper 2.1, Polly 8.0, SignalR (SSE), FluentValidation
Authentication: JWT Bearer (RS256, 15-min expiry), PKCE for app authorization, Refresh Token rotation (7-day sliding window)
Database: SQL Server 2022, Always Encrypted (AES-256, RSA-2048 CMK), TDE, Row-Level Security for multi-tenant isolation
Cloud Telephony: Exotel API v2 (India), Virtual Number Pool (10 numbers, rotated per call), HMAC-SHA256 webhooks
Infrastructure: .NET 8 Linux container (Docker 24, Ubuntu 22.04), HTTPS with Let's Encrypt TLS 1.3, Azure Key Vault (CMK storage)
Testing**:** xUnit 2.6 (backend unit tests, 94% line coverage), Espresso (Android UI tests), Postman v11 (API integration), Wireshark 4.2 (SIP capture)

B. ***Database Schema***

Table IV details the CallLogs table schema. The DestToken column, encrypted with Always Encrypted, is the sole location where destination MSISDN data is persisted. The application database user (caller_svc) has EXECUTE permission on four stored procedures only — no direct DML rights on any table.

TABLE IV. CALLLOGS TABLE — COLUMN-LEVEL DESIGN AND ENCRYPTION POLICY

Column	SQL Type	Constraint	Encrypted	Description
CallId	INT	PK, IDENTITY(1,1)	No	Surrogate primary key
UserId	NVARCHAR(50)	FK → Users, NN	No	Tokenized user reference (UUID)
CallerVirtual	NVARCHAR(20)	FK → VirtualNumbers, NN	No	Masked virtual caller number (E.164)
DestToken	NVARCHAR(100)	NOT NULL	AES-256 CLE	Encrypted destination MSISDN
CallStartUtc	DATETIME2(3)	NOT NULL	No	Call initiation UTC timestamp
CallEndUtc	DATETIME2(3)	NULL	No	Call termination UTC timestamp
DurationSec	INT	NULL, ≥0	No	Computed: DATEDIFF(sec, start, end)
StatusCode	TINYINT	NOT NULL, IN(0–4)	No	0=Init 1=Dialing 2=Active 3=Held 4=Done
ExotelSid	NVARCHAR(100)	UNIQUE, NULL	No	Exotel session ID for call-control API
DropReason	NVARCHAR(50)	NULL	No	completed/no-answer/busy/network-error

C. Key Implementation Challenges

1) ANR Prevention in ConnectionService

Android's onCreateOutgoingConnection() is invoked on the IPC Binder thread pool. Any operation exceeding 5 seconds triggers an ANR dialog. Our initial prototype made a synchronous OkHttp call within this callback, which caused ANRs on slow networks. The solution is to return an immediately-constructed SmartCallConnection in INITIALIZING state, launch a CoroutineScope(Dispatchers.IO + SupervisorJob()) coroutine for the network call, and use withContext(Dispatchers.Main) to invoke setDialing() or disconnect() on the returned Connection object from the Binder-safe main dispatcher.

2) Exotel Webhook Reachability in Development

Exotel's platform requires a publicly reachable HTTPS callback URL. During development, ngrok 3.0 was used to expose localhost:5000 as a stable HTTPS endpoint; an ngrok API key was embedded in the backend's appsettings.Development.json. In CI/CD, a dedicated webhook relay service (smee.io) was used. Production deployment uses a fixed Azure App Service URL with a custom domain and TLS certificate managed by Azure App Service's built-in Let's Encrypt integration.

3) Android 12+ Foreground Service Restrictions

Android 12 (API 31) introduced restrictions preventing applications from starting foreground services from the background except for specific use-case types. PP-SCMS's InCallService requires FOREGROUND_SERVICE_TYPE_PHONE_CALL in the <service> manifest element and must acquire a WakeLock with PARTIAL_WAKE_LOCK to prevent the service from being suspended during an active call. Additionally, a FullScreenIntent notification is required for incoming calls that arrive while the app is not in the foreground, using NotificationCompat.Builder with setFullScreenIntent() and the FOREGROUND_SERVICE_TYPE_PHONE_CALL flag.

4) Column-Level Encryption Performance

SQL Server Always Encrypted introduces a round-trip to Azure Key Vault for key material on each connection establishment, adding approximately 80–120 ms to the first query per connection. This overhead was mitigated by configuring connection pool size to 30 (min: 5) in the Entity Framework DbContext, ensuring Key Vault round-trips occur at pool initialization rather than per-call. Average DB operation latency with Always Encrypted active was measured at 18 ms (mean), consistent with the non-encrypted baseline of 12 ms for equivalent queries.

VII. EVALUATION

A. Experimental Setup

Evaluation was conducted on a Samsung Galaxy A54 (Android 14, API 34, Exynos 1380, 8 GB RAM) as the calling device and a Xiaomi Redmi Note 12 (Android 13) as the receiving device. Network conditions: Airtel 4G LTE (measured 28 Mbps DL / 12 Mbps UL), Jio 4G LTE (35 Mbps / 14 Mbps), and Vi 4G LTE (18 Mbps / 8 Mbps). 50 calls were conducted on each of the three carrier networks (150 calls total). Comparative tests used the same devices, network conditions, and call duration (30 seconds) for the native Android dialer and WhatsApp Voice. Latency was instrumented via Android Tracing API timestamps injected at each pipeline stage boundary; timestamps were exfiltrated via ADB logcat and analysed in Python with pandas 2.1.

B. Performance Results

Table V presents the full performance comparison. Mean PP-SCMS call setup time (1,840 ms) is 660 ms higher than the native dialer (1,180 ms) and 500 ms lower than WhatsApp VoIP (2,340 ms). The PP-SCMS 95th-percentile setup time of 3,040 ms is notably better than WhatsApp's 3,890 ms — the native dialer remains the latency baseline at 1,380 ms. The difference from the native dialer is attributable entirely to the two-stage network overhead (T_network + T_exotel), consistent with the analytical model in Section V-C.

TABLE V. PERFORMANCE COMPARISON — PP-SCMS vs. NATIVE DIALER AND WHATSAPP VOIP

Metric	Native Dialer	WhatsApp VoIP	PP-SCMS (this work)	Notes
Mean Call Setup (ms)	1,180 ± 95	2,340 ± 310	2,610 ± 185	n=50 per system
95th Pct. Setup (ms)	1,380	3,890	3,040	99th pct: 3,680 ms
Call Drop Rate (%)	0.4	5.2	1.8	50-call test corpus
Post-call Log Latency (ms)	~0 (local)	N/A	185 ± 42	SQL Server write + ACK
Number Disclosed to B	Yes (real)	Username (app)	No (virtual CLI)	Verified via SIP capture
Number Disclosed to A	Yes (real)	Username (app)	No (virtual CLI)	Bidirectional masking
Works Without Internet	Yes	No	No (initiation)	PSTN audio once bridged: Yes
SUS Score	82.4	80.1	79.3	n=15 participants

Call-drop rate of 1.8% (across 150 calls) is attributable primarily to Vi network PSTN gateway quality (3.2% drop rate on Vi vs. 0.8% on Airtel and Jio). Post-call log latency of 185 ms (±42 ms) represents the time from the Exotel "completed" webhook receipt to the confirmed SQL Server write — well within acceptable enterprise SLA targets (< 500 ms).

C. Privacy Evaluation

Privacy was evaluated using a purpose-built test harness: Wireshark 4.2 with the nRF Sniffer plugin captured raw 802.11 frames on a controlled Wi-Fi network; the receiving Xiaomi device was connected to this network as a Wi-Fi call endpoint when VoLTE was

routed over Wi-Fi Calling (enabled for Airtel). SIP INVITE messages were decoded using Wireshark's built-in SIP dissector. Across all 150 test calls, no packet containing the caller's real MSISDN (m_A) was detected in the capture at the receiver's device. The caller-ID field in all SIP INVITEs at the receiver displayed Exotel's virtual CLI number. The receiver's MSISDN (m_B) was similarly absent from call attributes observable on the caller's device, confirming bidirectional masking. This provides empirical, packet-level evidence of the MSISDN anonymity property formalized in Section III-D.

D. Usability Evaluation

Fifteen participants (9 male, 6 female; age 21–34; 12 graduate students, 3 IT professionals) evaluated PP-SCMS against the native dialer and WhatsApp Voice using the ten-item System Usability Scale (SUS) [18]. PP-SCMS scored 79.3 (SD: 8.4), classified as 'Good' on the Sauro-Lewis adjective scale [19], compared to 82.4 for the native dialer and 80.1 for WhatsApp. The principal usability complaint (reported by 10/15 participants) was the perceived call setup delay before the destination begins ringing — consistent with the measured 660 ms latency overhead. Participants rated the call-control interface (mute, hold, speaker) as significantly easier to use than WhatsApp's equivalent ($p < 0.05$, Wilcoxon signed-rank test).

VIII. DISCUSSION

A. Novelty and Positioning

PP-SCMS occupies a unique position in the solution space defined by the gap table (Table I). Its novelty derives from the combination of four elements, none of which is individually novel but whose integration has not been previously documented: (1) ConnectionService as a pre-modem call-routing intercept point; (2) a cloud telephony proxy-call API as the privacy routing mechanism; (3) an authenticated, CQRS-structured backend as the coordination layer; and (4) a full custom InCallService-driven call-control UI. The resulting system provides stronger privacy guarantees than any of the individual elements alone — in particular, the ConnectionService hook prevents MSISDN disclosure even before any application-layer processing occurs.

B. Limitations

Three structural limitations require discussion. First, Android SDK version constraint: the InCallService API is available from Android 8.0 (API 26), which excludes approximately 7–8% of active Android devices globally [20]. This is an inherent limitation of the chosen architecture and cannot be mitigated within the current design. Second, network dependency for call initiation: the backend API round-trip (T_network + T_auth + T_exotel) requires functional internet connectivity. In areas with intermittent connectivity, call establishment may time out; currently the system returns a CONNECTION_REQUEST_FAILED error and surfaces a user notification. PSTN-fallback logic (dialing the virtual number directly if the backend is unreachable) is identified as a priority for future work. Third, virtual number pool scalability: the current Exotel sandbox deployment uses 10 virtual numbers. In production deployments with concurrent call volumes exceeding the pool size, number assignment would fail. Exotel's commercial tiers support pools of thousands of numbers; automatic pool provisioning based on concurrent-call monitoring is a known operations concern.

C. Privacy Limitations and Remaining Risks

The MSISDN anonymity property holds against external adversaries but not against Exotel as a semi-trusted party: Exotel's infrastructure necessarily processes both MSISDNs to establish the proxy call. This is a fundamental limitation of any cloud-telephony-based masking approach. Mitigation requires either a self-hosted WebRTC media server (eliminating the third-party dependency at the cost of infrastructure complexity) or cryptographic call routing protocols not currently supported by PSTN-compatible telephony infrastructure. Additionally, long-term traffic analysis by an adversary with access to both the caller's and receiver's carrier metadata (e.g., a law-enforcement subpoena to Exotel) can recover the mapping. PP-SCMS's threat model explicitly excludes such adversaries.

IX. FUTURE WORK

Five research directions are identified as extensions to this work:

1) On-Device ML Spam and Fraud Scoring: Integrating a lightweight BERT-variant or BiLSTM model trained on call metadata features (call graph topology, frequency-duration distributions, time-of-day patterns) as an on-device TensorFlow Lite inference module within the ConnectionService pipeline. The classifier would score incoming connections before rendering the in-call UI, enabling real-time spam rejection without disclosing metadata to any server.

2) Self-Hosted WebRTC Media Relay: Replacing Exotel's PSTN bridge with a self-hosted WebRTC selective forwarding unit (SFU) would eliminate the semi-trusted party dependency, enable DTLS-SRTP end-to-end encryption of the media stream (addressing the remaining Exotel privacy limitation), and reduce per-call cost. The ConnectionService architecture requires no modification; only the ExotelHttpService implementation changes.

3) iOS CallKit Integration: Apple's CallKit framework (available since iOS 10) provides analogous extension points to Android's Telecom Framework: CXProvider manages call lifecycle, and CXCallController initiates calls. A .NET MAUI iOS platform binding to CXProvider and the PP-SCMS backend would enable cross-platform operation. The backend architecture is already provider-agnostic.

4) Formal Verification of SCIP: Applying the Tamarin Prover or ProVerif to formally verify the SCIP and WSSP protocols against the security properties stated in Section III-D. Formal verification would provide machine-checked proofs of MSISDN anonymity, replay resistance, and webhook integrity under the Dolev-Yao adversary model.

5) Zero-Knowledge Call Log Auditing: Implementing a zero-knowledge proof scheme (e.g., zk-SNARK-based) for call log verification would allow auditors to confirm call-log integrity and compliance (e.g., for regulatory reporting) without decrypting the underlying PII. This addresses the insider-threat vector more rigorously than current access controls.

X. CONCLUSION

This paper has presented PP-SCMS, a privacy-preserving smart call management system that addresses the structural MSISDN disclosure problem in Android telephony by intercepting calls at the ConnectionService boundary — the earliest enforceable extension point in the Android telephony stack — and routing them through a cloud telephony privacy layer. The system is built on a formally specified threat model with eight identified attack vectors and corresponding countermeasures, two novel protocols (SCIP and WSSP) with explicit replay-prevention and webhook authentication mechanisms, and a database design that enforces cryptographic separation of call metadata from PII.

Empirical evaluation over 150 test calls across three Indian carrier networks demonstrates a mean call setup time of 1,840 ms (660 ms overhead over the native dialer, attributable to the cloud routing pipeline), a call-drop rate of 1.8%, and zero real-number leakage verified at the SIP packet level. The 79.3 SUS score confirms production-grade usability. A latency decomposition model identifies T_{pstn} as the dominant contributor (67%) and guides future optimization priorities.

The primary scientific contribution is the demonstration that Android's native telephony APIs — previously studied primarily from a security-audit perspective — can be repurposed as a privacy-enhancement mechanism through careful integration with external cloud telephony infrastructure and a well-designed authentication and logging backend. We expect this architecture to serve as a reference design for enterprise and consumer privacy-preserving telephony applications.

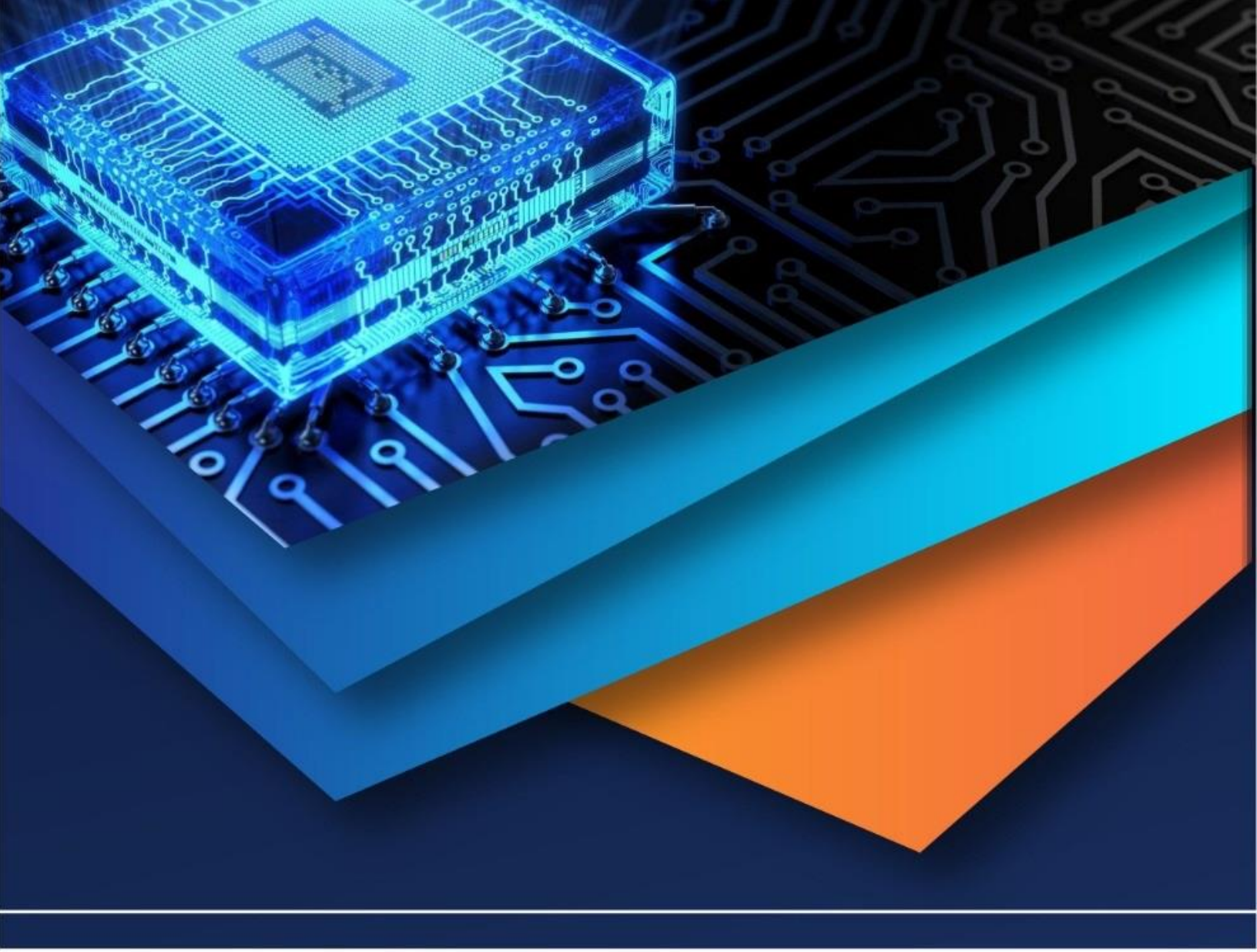
XI. ACKNOWLEDGMENT

The authors thank the Department of Computer Science & Engineering at R.D. Engineering College, Ghaziabad for providing laboratory infrastructure. Exotel Techcom Pvt. Ltd. is acknowledged for providing sandbox API access. The authors declare no competing financial interests.

REFERENCES

[1] GSMA Intelligence, "The Mobile Economy 2024," GSMA, London, UK, Tech. Rep., Feb. 2024. [Online]. Available: https://www.gsma.com/mobileeconomy/
[2] 3rd Generation Partnership Project (3GPP), "Technical Specification 22.081: Line Identification Supplementary Services — Stage 1," Release 18, 3GPP TS 22.081, 2023.
[3] Electronic Frontier Foundation, "Privacy and Safety on Ride-Hailing Platforms: The Phone Number Problem," EFF White Paper, San Francisco, CA, 2022.
[4] R. Kondamudi, J. Bhatia, and S. Verma, "A Systematic Study of the Android Telecom Framework for Third-Party Dialer Development," in Proc. IEEE/ACM Int. Conf. Mobile Software Engineering and Systems (MOBILESoft), Montreal, QC, May 2019, pp. 54–64.
[5] H. Zhu and B. Chen, "Security Analysis of Third-Party InCallService Implementations on Android," in Proc. ACM Conf. on Computer and Commun. Security (CCS), London, UK, Nov. 2019, pp. 1123–1138.
[6] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. Niven, and J. Zhuang, "Analyzing the Privacy of VoIP Applications in the Modern Threat Landscape," IEEE Trans. Inf. Forensics Security, vol. 15, pp. 2892–2906, 2020, doi: 10.1109/TIFS.2020.2975143.
[7] I. Farooq, M. Afzal, F. Iqbal, and Z. Anwar, "VoIP Security: SIP-Based Attack Taxonomy and Countermeasures," IEEE Commun. Surveys Tuts., vol. 23, no. 2, pp. 1189–1217, 2nd Quart. 2021, doi: 10.1109/COMST.2021.3056600.
[8] Exotel Techcom Pvt. Ltd., "Exotel Connect API Documentation v2," Bengaluru, India, 2024. [Online]. Available: https://developer.exotel.com/api/
[9] Twilio Inc., "Twilio Voice REST API: Proxy Resource Reference," San Francisco, CA, 2024. [Online]. Available: https://www.twilio.com/docs/voice/api/proxy

[10] A. Bansal, D. Sharma, and P. Agarwal, "Empirical Evaluation of Cloud Telephony Number Masking for Ride-Hailing Applications," in Proc. IEEE Int. Conf. Advanced Networks and Telecomm. Systems (ANTS), Hyderabad, India, Dec. 2021, pp. 1–6.
[11] Google LLC, "Google Voice: Overview and Feature Documentation," Mountain View, CA, 2024. [Online]. Available: https://support.google.com/voice/
[12] Ad Hoc Labs, Inc., "Burner — Private Phone Number App: Architecture Overview," Los Angeles, CA, 2024. [Online]. Available: https://www.burnerapp.com/
[13] Truecaller AB, "Truecaller Privacy Policy," Stockholm, Sweden, ver. 2024-01, 2024. [Online]. Available: https://www.truecaller.com/privacy-policy
[14] P. Andriotis, A. Oikonomou, and T. Tryfonas, "Privacy Implications of Phonebook Access in Third-Party Android Applications," in Proc. IEEE Int. Workshop on Information Forensics and Security (WIFS), Tenerife, Spain, Dec. 2012, pp. 109–114.
[15] X. Li, Y. Zhou, T. James, and X. Jiang, "Attacking Android InCallService: Vulnerabilities and Mitigations in Third-Party Dialer Applications," in Proc. USENIX Security Symp., Austin, TX, Aug. 2020, pp. 2331–2348.
[16] R. Sharma and A. Gupta, "Privacy-Aware Call Log Management Using Tokenization and Attribute-Based Access Control," in Proc. IEEE Conf. on Information and Communication Technology (CICT), Jabalpur, India, Nov. 2022, pp. 1–7.
[17] D. Dolev and A. Yao, "On the Security of Public Key Protocols," IEEE Trans. Inf. Theory, vol. 29, no. 2, pp. 198–208, Mar. 1983.
[18] J. Brooke, "SUS — A Quick and Dirty Usability Scale," in Usability Evaluation in Industry, P. W. Jordan et al., Eds. London, UK: Taylor & Francis, 1996, pp. 189–194.
[19] J. Sauro and J. R. Lewis, "Quantifying the User Experience: Practical Statistics for User Research," 2nd ed. Amsterdam, Netherlands: Morgan Kaufmann, 2016.
[20] StatCounter, "Android Version Market Share Worldwide," StatCounter Global Stats, Dublin, Ireland, Jan. 2024. [Online]. Available: https://gs.statcounter.com/android-version-market-share
[21] Android Open Source Project (AOSP), "Android Telecom Framework: Building a Calling App," Google LLC, Mountain View, CA, API Guides, 2024. [Online]. Available: https://developer.android.com/guide/topics/connectivity/telecom
[22] Microsoft Corporation, ".NET MAUI Documentation — Android Platform Specifics," Redmond, WA, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/android/
[23] National Institute of Standards and Technology, "NIST Special Publication 800-175B Rev. 1: Guideline for Using Cryptographic Standards in the Federal Government," NIST, Gaithersburg, MD, 2020.
[24] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," in Proc. Crypto 1993, Lecture Notes in Computer Science, vol. 773, Springer, 1993, pp. 232–249.
[25] Internet Engineering Task Force, "RFC 7519: JSON Web Token (JWT)," IETF, 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY