Strategy for Protecting RSA Encryption System

Pegasus Case

N5Digital, Inc.

This is a sequel to our earlier exploration of RSA end-to-end encryption technology.

SECTION 1: Breaking RSA System by Pegasus

Pegasus, developed by NSO Group, is a highly advanced **zero-click spyware**—and yes, **even an RSA-secured system** can be compromised **not by breaking RSA**, but by bypassing its protections entirely.

I. Can RSA be broken by Pegasus directly?

No. Pegasus does not break RSA math (e.g., factoring n, cracking d, etc.).
RSA, when implemented properly with strong key lengths (2048 bits or more), is considered mathematically secure.
However, Pegasus operates by:

- Infecting the device itself
- Reading decrypted messages and keys from memory
- Intercepting user activity at the app level

So the encryption can be perfect—but it's irrelevant if the attacker has access to the decrypted output and keys.

🔍 II. How Pegasus Bypasses RSA Security

☑ What Pegasus can access:

Target	Is it vulnerable to Pegasus?	How/Why?
Private Keys	☑ Yes	If stored in memory or file system unencrypted
Decrypted Messages	☑ Yes	Intercepted after RSA decryption by app
Encrypted Messages	🔒 No (but irrelevant)	Pegasus waits for the message to be decrypted
User Input (Keyboard)	☑ Yes	Keylogging, screen capture
Secure Messaging Apps	☑ Yes	Reads messages from the app screen or memory

🧠 Think of it this way: RSA protects the tunnel, but Pegasus spies from inside your house.

🔐 III. What About Devices with Secure Enclave / TPM?

Secure Enclaves (Apple) or TPMs (Trusted Platform Modules) protect private keys by:

- Preventing keys from being exported
- Restricting use to signed, approved operations

BUT: Pegasus may gain privileged access (root/kernel-level) on the device, allowing it to:

- Monitor RAM contents, where decrypted messages or temporary private key data may be loaded
- Spoof input/output
- Bypass biometric/PIN prompts

So even secure hardware can be subverted if the OS is fully compromised.

🧩 IV. Summary of Pegasus Threat to RSA-Based Communications

RSA Component	Vulnerability to Pegasus	Comment
Private Key (d, n)	☑ High if unencrypted or in memory	
Decryption Process	☑ High (can spy on decrypted data)	
Public Key	❌ Not sensitive	Public anyway
Stored Messages (Encrypted)	❌ Safe unless decrypted	
Stored Messages (Decrypted)	☑ Vulnerable if app stores them	
Encrypted Transmission	❌ Safe over the wire	Pegasus doesn't need to intercept it

---

☑ How to Minimize the Risk

1. Use secure messaging apps (e.g., Signal) with disappearing messages.
2. Keep your OS and apps updated to close known exploits.
3. Never store unencrypted private keys.
4. Favor apps using hardware-backed key storage (Secure Enclave, Android Keystore).
5. Limit sensitive communications on personal phones.
6. Use air-gapped devices for highly sensitive keys or message storage.

SECTION 2: PATH FORWARD TO SAFETY

QUESTION: If Secure App Architecture is possible, why Isn't It used everywhere?.
FOLLOW UP: An example of a secure app architecture.

1. ANSWER

Here are the main reasons why not:

Reason	Explanation
Cost & Complexity	Designing and maintaining truly secure apps with hardware integration, sandboxing, and isolated encryption requires advanced expertise, longer dev cycles, and often special hardware access.
User Experience Trade-offs	Secure design often limits convenience: fewer features, slower performance, or inability to use third-party integrations (e.g., cloud backups, auto-login).
Commercial Pressure	Most companies prioritize speed-to-market, features, and monetization over end-to-end security, especially when users don’t demand it.
Low User Awareness	Users trust apps blindly. If they aren't aware of threats like Pegasus, there's no market pressure on developers to invest in advanced protections.
Limited Device Capabilities	Some secure practices (e.g., hardware-backed storage, secure enclaves) are only available on modern premium devices (iPhone, Pixel, etc.). Apps must remain compatible with less capable devices.
Security ≠ Profit	Truly private apps often can't monetize user data — which goes against the business model of many companies.

---

2. FOLLOW UP: Secure App Architecture for RSA-Based Messaging

Let’s now design a Pegasus-resilient messaging app that uses RSA for asymmetric encryption and a hybrid model for efficient secure messaging.

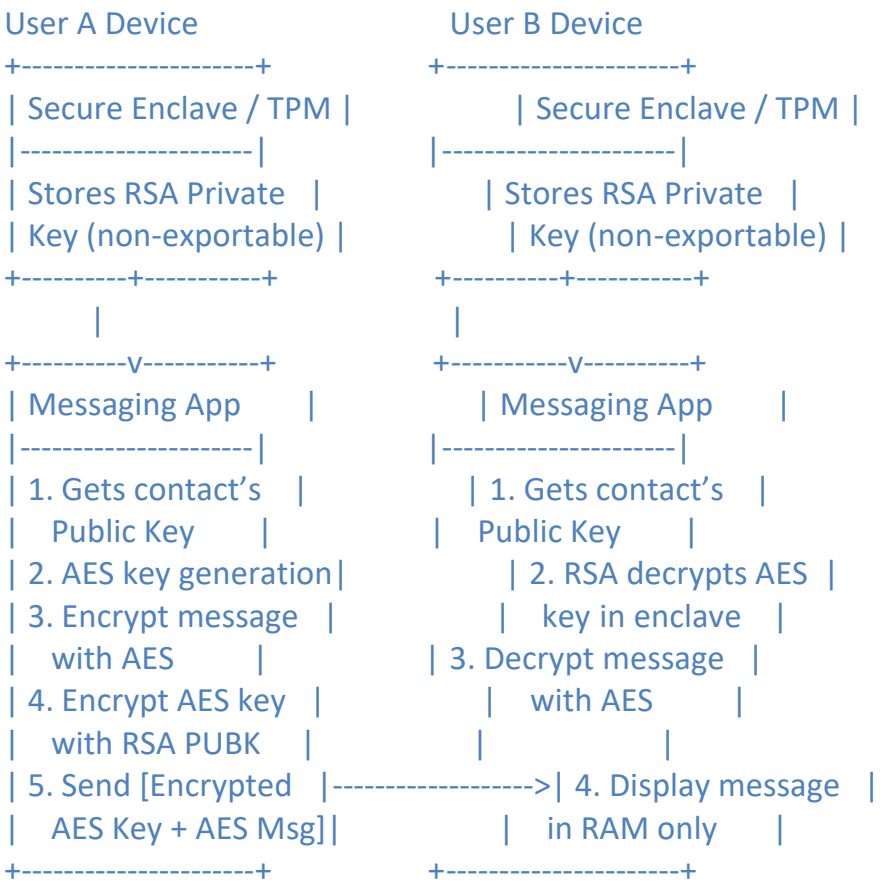
☑ A. Security Goals

- Private key is never exposed to the OS or app memory.
- Messages are encrypted end-to-end with strong algorithms.
- App should not store decrypted messages.
- Minimize user friction without compromising security.

🏗 B. Architecture Overview

pgsql

CopyEdit



C. Technical Design Components

Component	Design Features
Private Key Storage	Stored in Secure Enclave (iOS) or Android Keystore (hardware-backed). Never leaves enclave.
RSA Operations	Only done inside secure enclave. App cannot access the raw key or decrypted output directly.
Message Encryption	Messages encrypted with 256-bit AES session key (generated per message).
AES Key Encryption	AES key encrypted using recipient's RSA Public Key.
Message Storage	Messages only exist in memory; decrypted content never written to disk.
User Authentication	Biometric or PIN unlock required per session or per message view.
No Backups	Messages and keys are not backed up to

Component	Design Features
	cloud storage (even encrypted).
Tamper Detection	App uses device integrity APIs (e.g., SafetyNet on Android, Secure Boot on iOS) to detect jailbreaking/rooting.
Optional: Secure element-based signature for metadata (message origin proof without compromising content).	

D. Extra Layers (Optional for Ultra-Security)

Feature	Description
Hardware Token	Use external YubiKey-like device to decrypt messages or approve sending.
Air-gapped Key Pair Setup	Generate RSA keys on a disconnected device; public key copied to app; private key used only offline.
Disappearing Messages	Auto-delete messages after viewing; zero retention.
Encrypted Local Storage	If message must be stored (e.g., for async delivery), use encrypted local storage with per-message ephemeral keys.
Forward Secrecy	Add Diffie-Hellman component for each session to prevent message replay and future compromise of past messages.

E. Security Audit Checklist

Category	Questions
Code	Is the app audited by third-party infosec experts?
Keys	Are private keys encrypted, non-exportable, and hardware-bound?
Messages	Are messages ever stored decrypted? If yes, are they wiped promptly?
System	Does the app detect rooted/jailbroken devices?
UX	Does it balance convenience and security effectively?

F. Example Stack

Area	Tech
Encryption	RSA 4096, AES-256 GCM
Storage Security	Android Keystore / iOS Secure Enclave
Messaging Protocol	Custom hybrid encryption over TLS

Area	Tech
User Interface	Flutter, React Native (restricted bridge access)
Audit Tools	OWASP MASVS checklist, Mobile Security Framework (MobSF)