End-To-End RSA-Encrypted Two-Way Communication

N5Digital, Inc.

I. Introduction

RSA (Rivest-Shamir-Adleman) is one of the most widely used public-key encryption algorithms used for end-to-end encrypted communication. It enables secure digital communication through mathematical operations that are easy to compute in one direction (multiplying large primes) but hard to reverse (factoring the product). RSA provides secure encryption, digital signatures, and safe key exchange between parties who do not need to meet in person.

II. RSA Encryption Process

RSA encryption starts with the creation of two or more large prime numbers:

- In standard RSA, two large prime numbers, p and q, are generated.
- In multi-prime RSA, three or more primes may be used: $p \times q \times r \times ...$
  - This variant improves decryption speed in hardware or constrained environments but is rarely used in standard software-based systems.

The process continues as follows:
- Compute the product: $n = p \times q$ (or $n = p \times q \times r \times ...$) - this product n is the modulus for both keys.
- Compute the totient: For two primes: $\phi(n) = (p - 1)(q - 1)$; for multiple primes: $\phi(n) = (p - 1)(q - 1)(r - 1)...$
- Choose a public exponent e such that e is coprime to $\phi(n)$ and $1 < e < \phi(n)$.
- Compute the private exponent d such that $d \times e \equiv 1 \mod \phi(n)$.

Keys:
- Public Key (PUBK) = (e, n) - distributed openly
- Private Key (PRK) = (d, n) - kept secure on the user's device

Who Generates the Prime Numbers and How?

RSA does not rely on an external authority to provide the primes. Instead:

- The app itself (or the cryptographic library it uses) is responsible for generating the large prime numbers.
- This is done using a cryptographically secure random number generator (CSPRNG) and primality tests, such as:
  - Miller-Rabin
  - Fermat's test (as a quick pre-check)

Examples of common crypto libraries:
- OpenSSL
- BouncyCastle

- Microsoft CAPI/CNG
- libsodium, NaCl

This ensures:
- Each key pair is unique
- The private key remains secret
- No third party knows or controls the primes

III. How Messages Become Numbers

RSA works on integers. Any content-text, image, video-must first be converted into a number.

Text Messages:
1. Encode the message into bytes using UTF-8 or ASCII.
2. Convert byte array into a single integer.

Multimedia Content:
1. Read the file in binary mode.
2. Convert to a single number using int.from_bytes(data, byteorder='big').

Hybrid Encryption for Large Content:
1. Generate a random AES key.
2. Encrypt the content with AES.
3. Encrypt the AES key with RSA.
4. Send the RSA-encrypted AES key and the AES-encrypted data.

IV. RSA-Based Two-Way Communication

Key Exchange:
- Each user generates a Public/Private Key pair.
- Public Keys are shared via the app or server.
- Private Keys are stored securely on users' devices.

Sending a Message:
- Alice retrieves Bob's Public Key (e, n).
- Converts her message to a number m.
- Encrypts it: c = m^e mod n.
- Bob decrypts it with his Private Key (d, n).

Responding:
- Bob encrypts his reply with Alice's Public Key.
- Alice decrypts it with her Private Key.

V. RSA Algorithms Explained in Simple Language

Encryption:
Formula: c = m^e mod n
Layman's terms: Turn the message into a number, raise it to the power of e, divide by n and keep the remainder.

Decryption:
Formula: m = c^d mod n
Layman's terms: Raise the encrypted number to the power of d, divide by n and keep the remainder to retrieve the original message.

VI. Security and Best Practices

- RSA relies on the difficulty of factoring n into primes.
- Recommended key size: 2048 bits or more.
- Private Keys must be kept secure.
- For large messages, use hybrid encryption.
- Always use trusted cryptographic libraries.

VII. Summary

RSA provides a robust, mathematically grounded way to:
- Generate secure key pairs
- Encrypt and decrypt messages
- Enable trusted two-way communication
- Protect content of any size using hybrid encryption

Multi-prime RSA exists for performance tuning but is rare. In nearly all cases, the application or library generates the primes itself, ensuring secure, unique keys without relying on external authorities.

VIII. Private Key Storage and Protection

An RSA private key consists of several values, primarily:
- d (private exponent)
- n (modulus)
Optionally, it may also include the primes p and q, and optimization values such as dP, dQ, and qInv.

Private keys are typically stored in:
- PEM format (Base64-encoded text with headers)
- DER format (binary)

Where and how they are stored depends on the platform:
- iOS: Stored in the Secure Enclave or Keychain, non-exportable
- Android: Stored in the Android Keystore, optionally hardware-backed
- Desktops: Often encrypted with a password or stored in secure modules (e.g., HSMs)

Encrypted key storage is strongly recommended. This means the key file itself is encrypted with a symmetric cipher (like AES), often derived from a user password.

Security Risk:
- If an unencrypted key is stored and the device is compromised, intercepted communications can be decrypted.

- If encrypted or protected by secure enclave, even device theft will not expose the key - the attackers **cannot use the key**, even if they steal the device!

Best Practice Summary

Scenario	**Storage Location**	**Encrypted?**	**Exportable?**
iOS App	Secure Enclave / Keychain	Yes	No (non-exportable)
Android App	Android Keystore (hardware-backed)	Yes	No (non-exportable)
Desktop app (best case)	Encrypted PEM with password	Yes	Yes (with access)
Poorly secured device	Raw key in file	No	Yes