perplexity

When we implement character-by-character encryption, how to provide a way to decrypt the text (could be a parameter in an API call) in a secure way while keeping functionality as simple as possible to consume?

The straightforward approach is to expose a dedicated decryption API endpoint that expects the aggregated encrypted text along with a secure, non-secret parameter (such as a token or key identifier) that instructs the server on how to retrieve the proper decryption key. This method keeps the interface simple for consumers while ensuring keys are never sent or exposed directly.

Secure API Decryption Design

- Ensure that all API calls use TLS (HTTPS) so that data in transit remains encrypted and protected from interception[1].
- Require strong authentication and authorization so that only legitimate users or services can call the decryption endpoint[1].
- Instead of transmitting encryption keys directly (which would compromise security), the client sends a key identifier or an ephemeral decryption token that the server uses in conjunction with its secure key management system (for example, an HSM or cloud-based key vault) to locate the correct key[2].
- Include a message authentication code (MAC) or digital signature along with the encrypted payload to verify both the integrity of the data and the authenticity of the requester[1].

Practical Implementation Strategy

When performing character-by-character encryption (for instance, encrypting each input character as it is entered), the app can accumulate the final encrypted output and then attach an additional parameter (e.g., “keyId” or “decryptionToken”) when making the API call. This token is generated as part of a secure session exchange and never reveals the underlying decryption material directly.

A simplified pseudocode example for a decryption API might be:

```
// Swift example for a decryption API call
func sendEncryptedData(encryptedText: String, decryptionToken: String) {
    // Create URL request for the secure decryption endpoint
    var request = URLRequest(url: URL(string: "https://api.yourserver.com/decrypt")!)
    request.httpMethod = "POST"
    // Attach encrypted text and token as JSON parameters
    let jsonPayload: [String: Any] = [
        "encryptedData": encryptedText,
        "token": decryptionToken
    ]
    request.httpBody = try? JSONSerialization.data(withJSONObject: jsonPayload)
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")

    // Ensure the request is made over HTTPS and proper authentication headers are set (e
    URLSession.shared.dataTask(with: request) { data, response, error in
        // Handle the response, which includes the decrypted text (if authentication and
    }.resume()
}
```

In an Android context using Jetpack Compose, a similar approach is applied when calling the API; the decryption token is simply another parameter sent securely in an HTTPS POST request. By keeping the decryption key exclusively on the server side and using the identifier or token only to retrieve the key from a secure store, both platforms minimize their exposure to cryptographic secrets and simplify client-side integration.

Summary

By encapsulating decryption inside a dedicated, authenticated API endpoint and using a parameter (such as a key identifier or secure token) to specify which key to use for decryption, you create a secure yet simple interface for consumers. This approach adheres to security best practices by ensuring keys are never transferred or stored insecurely and that communication is protected by strong encryption and proper authentication mechanisms[2] [1].

⁂

1. https://www.fleksy.com/blog/how-to-implement-secure-text-input-in-banking-apps/
2. https://stackoverflow.com/questions/14246307/best-practices-for-passing-encrypted-data-between-different-programming-language