

Developer Cryptography Best Practices



Design Considerations

- **How do I update my crypto?**
 - In an installed application
 - With users in the field
 - Gracefully?
 - So that over time, all active accounts are transparently upgraded to the latest secure standards?

- **Examples**
 - New KDF (function, or just parameters) after old KDF found to be weak
 - New crypto library after security bugs fixed

Application Use Cases



Application Use Cases

- **Password handling**
(Storing passwords in a database, using human-intelligible secret keys, ...)
 - Plaintext? **NO NO NO**
 - Old cryptographic hash (MD5)? **NO NO**
 - Old cryptographic hash (MD5) with salt? **NO NO**
 - New cryptographic hash (SHA-2) with salt? **NO**
 - CPU and memory intensive Key Derivation Function?
YES!
 - **bcrypt**
 - **scrypt**

Application Use Cases

- **Encryption of Bulk Data**
(Hiding from users / hiding from network)
 - NaCl / Libsodium default – **YES!**
 - Chacha20 Cipher w/ Poly1305 MAC – **YES!**
 - AES-GCM – **YES!**

- All options provide AEAD
 - Authenticated Encryption with Associated Data

Application Use Cases

- **Symmetric Signatures**
(Authenticating an API, authenticating cookies, but not encrypting)
 - HMAC – **YES!**
 - GMAC – **YES!**

- But are you sure you don't also want to be encrypting the data via AEAD?

Application Use Cases

- **Asymmetric Encryption, i.e. Public Keys
(Sending messages to many people, who are strangers, who need to decrypt and process it offline)**
 - NaCl / Libsodium – **YES!**
- Prefer elliptic curves over RSA
 - Assuming curve parameters are safe, fewer ways to go wrong in implementation
 - Fewer downgrade attacks possible

Application Use Cases

➤ Website security

➤ TLS – **YES!**

➤ BoringSSL (Google)

➤ LibreSSL (OpenBSD)

➤ Client-Server Application Security

➤ TLS! – **YES!**

➤ Consider hard-coding TLS 1.2+, ECDHE, AES-GCM (to prevent downgrade attacks)

➤ Consider self-signing and shipping certificate vs relying on certificate authority