# KeyChain Extension and Integration

## Presentation

Kjell Braden, Marvin Dickhaus, Cassius Puodzius

Fachbereich Informatik
TU Darmstadt

March 27, 2013

# Motivation



- KeyChain is poorly used in current Android Versions
- Lack of integration between key access and its use
- Broader acceptance of encryption

Goals:

- Improve secure storage and authorization handling for keys
- Support easy use of cryptographic functions for apps

# Requirements



Apps should be able to...

- run common crypto operations without accessing keys directly
- generate, store and export symmetric keys
- store and export public keys
- run key agreement protocols
  - store the key agreement parameters in the KeyChain
  - automatically derive and store the shared secret key in the KeyChain

# Existing Android API



Java Cryptography Architecture (JCA)

- offers interfaces for SecretKey/PublicKey/PrivateKey
- offers factories for ciphers, signature schemes etc. to work on the corresponding implementation of keys

native (C++) keystore daemon

- stores key material encrypted using phone lock passphrase, pin or pattern
- offers an OpenSSL engine for loading JCA Key objects from the store without exposing the key material itself, but...
  - there is no real access control
  - only RSA keys are supported
  - attempting to use the keys for anything causes a SIGSEGV in dalvik :-(

# Overview



- a system app for key management, such as
  - key generation
  - import/export/deletion of key pairs
  - granting key access
- a public API which allows
  - encryption / decryption
  - authentication (signature/MAC) / verification
  - generation / import of symmetric keys
  - key agreement protocols

# Key Identification



- each key is referenced by a unique string alias
- keys can be assigned to contacts using the *Key Management* app
- each assignment has a *key usage type identifier*
  - arbitrary string token (application defined)
  - apps can request a key with a given type for a given contact
  - this way the user can easily choose which key to use for which app, and replace keys once they are obsoleted

# API usage



- API calls are forwarded using binder IPC to the *keychain* system app
- system app checks if the calling app is allowed to use the key
- system app may present the user with a dialog to authorize the access
- if authorized, the system app processes the request and sends the result back to the caller

# Considerations



- Separate UI for sending
- Lookup of keys (key usage type)
- Storage of SMS
- Recognize encrypted messages
- Keep it as simple as possible

# Sending and Receiving



## Sending

- Select contact
- Encrypt composed message
- Send Base64-encoded message via SMS
- Store copy of plain message locally

## Receiving

- Capture `SMS_RECEIVED` broadcast
- Recognizing encrypted SMS

# Receiving Workflow



**Require:** all message parts have been received and the full message is reassembled

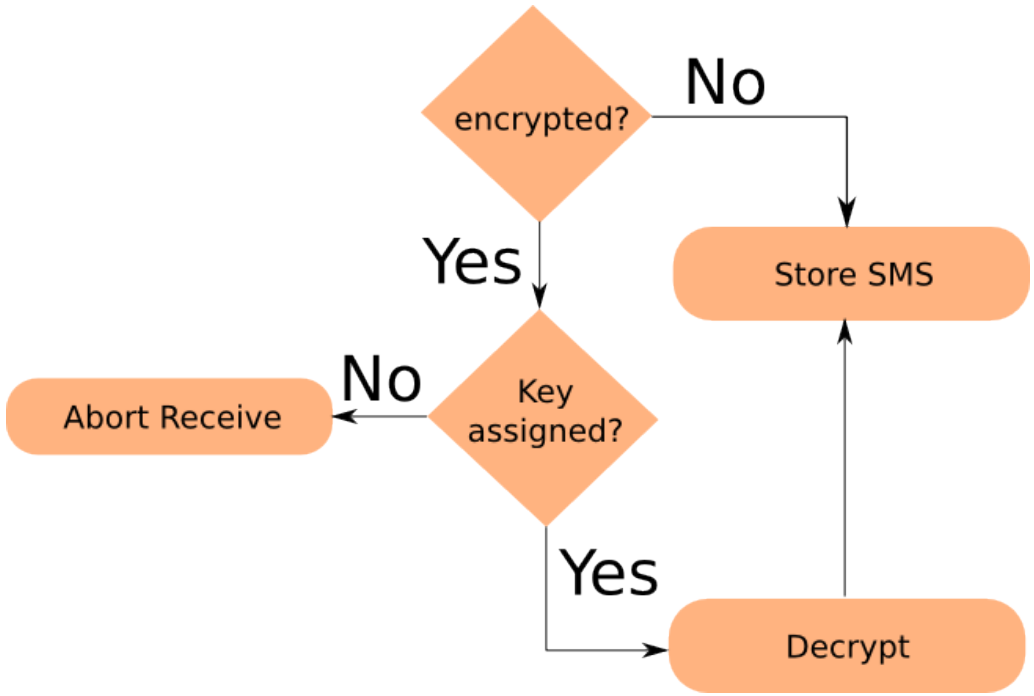


Figure: Receiving workflow

# Trivia



- time spent building full images: approx. 80 hours
- RSA 1024-bit key generation
  - regular MIPS emulator image: 10 minutes
  - x86 emulator image using VT-x/AMD-V: less than 5 seconds