Crypto Android


```
Function scrypt(Passphrase,Salt,N,p,dkLen):
    (B0 ... Bp-1) ← PBKDF2HMAC_SHA256(Passphrase, Salt, 1, p * MFLen)
    for i = 0 to p-1 do
        Bi ← SMix(Bi,N)
    end for
    Output ← PBKDF2HMAC_SHA256(Passphrase, B0 || B1 ... Bp-1, 1, dkLen)
```

```
Function SMix(B,N):
    X ← B
    for i = 0 to N - 1 do
       Vi ← X
       X ← BlockMix(X)
    end for
    for i = 0 to N - 1 do
        j ← Integerify(X) mod N
        X ← BlockMix(X ⊕ Vj)
    end for
    Output ← X
```

SMix richiede un vettore di lunghezza N. Visto che la memoria è costosa, implementare Scrypt in hardware è costoso [**?**]

Trusted Execution Environment

I Trusted Execution Environment (TEE) sono ambienti di esecuzione isolati dalle altre applicazioni e sistemi operativi

- Solitamente implementati come processori separati fisicamente dal processore primario

Poichè sono intoccabili dalle applicazioni, si può utilizzare un TEE per operazioni crittografiche isolate persino dal kernel di sistema

- Spesso includono in hardware chiavi uniche, in modo da garantire che solo un preciso processore possa decifrare i dati

TrustZone

TrustZone è l' implementazione del concetto di TEE in processori ARM
Si definiscono due domini di esecuzione:

- RichOS, dove viene eseguito il sistema operativo classico
- TrustOS, dove viene eseguito un firmware "sicuro", spesso impresso in hardware

TrustZone usa stesso processore fisico, stesso ISA ma due aree di memoria completamente isolate per i due domini

Android

Android è una specifica per un sistema operativo pensato per l' utilizzo mobile [**?**]

AOSP (Android Open Source Project) è un implementazione di Android rilasciata sotto Apache 2.0

Per questo studio ci siamo concentrati su come Android cifri il disco e gestisca l' autentificazione delle applicazioni nella AOSP 6.0.1b

Disk Encryption

Android permette, dalla versione 3.0, la cifratura del disco di sistema

- Android 5.0 ha modificato alcuni punti dell' algoritmo utilizzato

Il codice relativo alla cifratura del disco è in singolo file, cryptofs.c

Android opera a livello di blocchi direttamente nel kernel

- Il device è cifrato al primo avvio di sistema
- Trasparente alle applicazioni e all' utente
- Non può funzionare su file system che non usano un approccio a blocchi
  - Ufficialmente supporta solo etx4 e f2f [**?**]

Ogni blocco è cifrato in maniera indipendente con la stessa chiave (DEK) [**?**]

- AES-CBC a 128 bit
- IV = AES128 (SHA256(DEK), SN)

Key generation

Come viene generato DEK ?

```
fd = open("/dev/urandom", O_RDONLY|O_CLOEXEC);
read(fd, key_buf, sizeof(key_buf));
read(fd, salt, SALT_LEN);
close(fd);
```

- Insieme a DEK viene generato anche un salt (S)
- Nessun check sulla presenza e sulla "qualità" di /dev/urandom...

Key store

DEK è salvata su disco, ma non in chiaro: è cifrata da una chiave che dipende

- Dalla password dell' utente
  - Questo permette all' utente di cambiare periodicamente la propria password senza dover cifrare nuovamente l' intero disco
- Dalla chiave hardware del TEE del processore
  - In questa maniera solo il device che ha cifrato il disco può decifrarlo

Key store algorithm

- Viene generata IK1, a 256 bit, come SCRYPT della password utente e di S
  - Se l' utente non possiede una password, "`default_password`"
- IK1 viene esteso a 256 byte con il formato 00 IK1 00 ... 00
- IK1 esteso viene cifrato con la chiave privata del TEE, generando IK2
- Genero IK3, a 256 bit, come SCRYPT di IK2 ed S
- I primi 128 bit di IK3 sono KEK, gli altri 128 bit sono IV
- Cifro DEK con AES128 CBC, usando KEK ed IV

Considerazioni e curiosità

Durante i test e lo studio del codice son sorti diversi punti e curiosità:

- Android, per ridurre il tempo di cifratura iniziale, cifra solo i blocchi attivi nella partizione
- DEK è generata tramite /dev/urandom, ergo viene generata dal kernel linux sottostante Android
- Visto che DEK è spesso generata al primo boot del sistema, sarebbe interessante studiare come si comporta il generatore su più dispositivi della stessa famiglia

Android Application Certificate

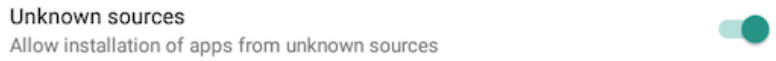
Le applicazioni Android son distribuite tramite un file .apk
Ogni applicazione Android DEVE possedere un certificato

- In mancanza di un certificato, non si può installare l' applicazione

I certificati certificano non solo il codice, ma anche i metadati e tutte le risorse nell' apk

- Android utilizza RSA 2048 e SHA1

Fonti note e non note



Android presenta un flag che consente di installare applicazioni da fonti non note. Alcune domande sorgono spontanee:

- Chi sono le fonti note ?
- Come capisce che un' applicazione arriva da una fonte nota ?

Facendo una rapida ricerca, il flag viene controllato solo in un punto del codice

```
String callerPackage = getCallingPackage();
    if (callerPackage != null && intent.getBooleanExtra(
            Intent.EXTRA_NOT_UNKNOWN_SOURCE, false)) {
        try {
            mSourceInfo = mPm.getApplicationInfo(callerPackage, 0);
            if (mSourceInfo != null) {
                if ((mSourceInfo.flags&ApplicationInfo.FLAG_SYSTEM) != 0) {
                    // System apps don't need to be approved.
                    initiateInstall();
                    return;
                }
            }
        } catch (NameNotFoundException e) {
        }
    }
    if (!isInstallingUnknownAppsAllowed()) {
         //ask user to enable setting first
         showDialogInner(DLG_UNKNOWN_APPS);
         return;
     }
    initiateInstall();
```


Dal codice notiamo che, prima di iniziare l' installazione, controlla solo se l' applicazione è un' applicazione "di sistema"

- Questo significa che, in qualche modo, le app di sistema sono le fonti note

Solo le app di sistema possono iniziare il processo di installazione. Questo permette ai produttori di costruire un proprio "store" di applicazioni per i loro utenti

- Sta poi allo store ed al suo produttore introdurre eventuali controlli sull' identità degli sviluppatori

Ma allora a cosa serve firmare le applicazioni ?

- E' possibile installare un update per un' app solo se firmato con la stessa chiave RSA
- E' possibile specificare canali sicuri tra le applicazioni, in cui un' app comunica solo con una che possiede una determinata firma
- Solo app firmate dalla stessa chiave possono essere eseguite dallo stesso processo

Giochiamo un po'

Abbiamo provato a "giocare" con Android ed i suoi certificati

E' stato estremamente semplice realizzare uno script che rimuovesse la firma originale e firmasse l' applicazione con la nostra chiave

- Abbiamo potuto installare la nostra "fakeApp" sul sistema

Abbiamo poi generato un certificato "scaduto" e abbiamo firmato l' applicazione

- Android ha installato l' applicazione senza dare un avviso del problema

Considerazioni e curiosità

Come visto, il sistema di app signing su Android non serve tanto a garantire l' identità dello svilluppatore, quanto a garantire che solo lo sviluppatore di un' app possa rilasciarne update

- Approccio totalmente diverso alla concorrenza
- Il controllo dei certificati è da parte del produttore del dispositivo
- E' estremamente semplice rimuovere la firma di un .apk e firmarlo con un proprio certificato
- Android accetta anche certificati scaduti senza dare alcun segnale

References

[1] Colin Percival, *Stronger key derivation via sequential memory-hard functions*, 2009.

[2] Google Inc., *Android 6.0 Compatibility Definition*, 16 ottobre 2015.

[3] *Android Full Disk Encryption*, Web. 21 gennaio 2016. https://source.android.com/security/encryption/

[4] *Android App Signing*, Web. 21 gennaio 2016. https://developer.android.com/tools/publishing/app-signing.html