Enhancing Password Based Key Derivation Techniques

PasswordsCon 2014

Presented by Stephen Lombardo & Nick Parker

zatetic

SQLCipher

```
% hexdump -C unencrypted-sqlite.db
00000000  53 51 4c 69 74 65 20 66  6f 72 6d 61 74 20 33 00  |SQLite format 3.|
00000010  04 00 01 01 00 40 20 20  00 00 00 02 00 00 00 03  |.....@  ........|
000003b0  00 00 00 00 00 00 00 00  00 00 00 00 00 41 01 06  |.............A..|
000003c0  17 1b 1b 01 5b 74 61 62  6c 65 73 65 63 72 65 74  |....[tablesecret|
000003d0  73 73 65 63 72 65 74 73  03 43 52 45 41 54 45 20  |ssecrets.CREATE |
000003e0  54 41 42 4c 45 20 73 65  63 72 65 74 73 28 69 64  |TABLE secrets(id|
000003f0  2c 20 70 61 73 73 77 6f  72 64 2c 20 6b 65 79 29  |, password, key)|
00000bd0  00 00 00 00 00 00 00 00  00 00 00 00 00 21 01 04  |.............!..|
00000be0  25 1d 1f 4c 61 75 6e 63  68 20 43 6f 64 65 73 70  |%..Launch Codesp|
00000bf0  61 24 24 77 6f 72 64 70  72 6f 6a 65 74 69 6c 65  |a$$wordprojetile|
```

```
% hexdump -C encrypted-sqlcipher.db
00000000  de ab bc 3a 40 2b 5d 00  b0 d2 9e 3b 75 91 76 73  |...:@+]....;u.vs|
00000010  bc 41 70 0c 8c ab a0 7a  37 eb a2 a8 a9 27 a5 0a  |.Ap....z7....'..|
00000020  38 c9 0b 9c 06 57 78 96  67 a2 e5 78 f8 8c 58 f3  |8....Wx.g..x..X.|
00000030  ea 7c c6 23 14 8a 75 33  d0 a5 2c 30 2e e1 a4 96  |.|.#..u3..,0....|
00000040  b1 c6 5a 21 67 0a 31 bb  3b de a2 d4 80 b4 60 e3  |..Z!g.1.;.....`.|
00000050  05 b0 75 04 f2 26 66 ed  c7 4e 7e 9c ac 2e ec 1d  |..u..&f..N~.....|
00000060  2d fc 31 b4 32 ce 24 0a  d0 23 71 b0 1f 21 12 2c  |-.1.2.$..#q..!.,|
00000070  92 af 8e d9 de ac 76 e6  20 62 56 c6 f5 05 f5 b3  |......v. bV.....|
00000080  53 d0 5f 4c 5e ec 5b 8a  be e7 d1 46 f0 d9 dc b9  |S._L^.[....F....|
00000090  a3 59 d6 63 a4 ae cf d8  e4 82 29 83 dd c7 86 13  |.Y.c......).....|
```

SQLCipher is an open source extension to SQLite that provides transparent 256-bit AES encryption of database files

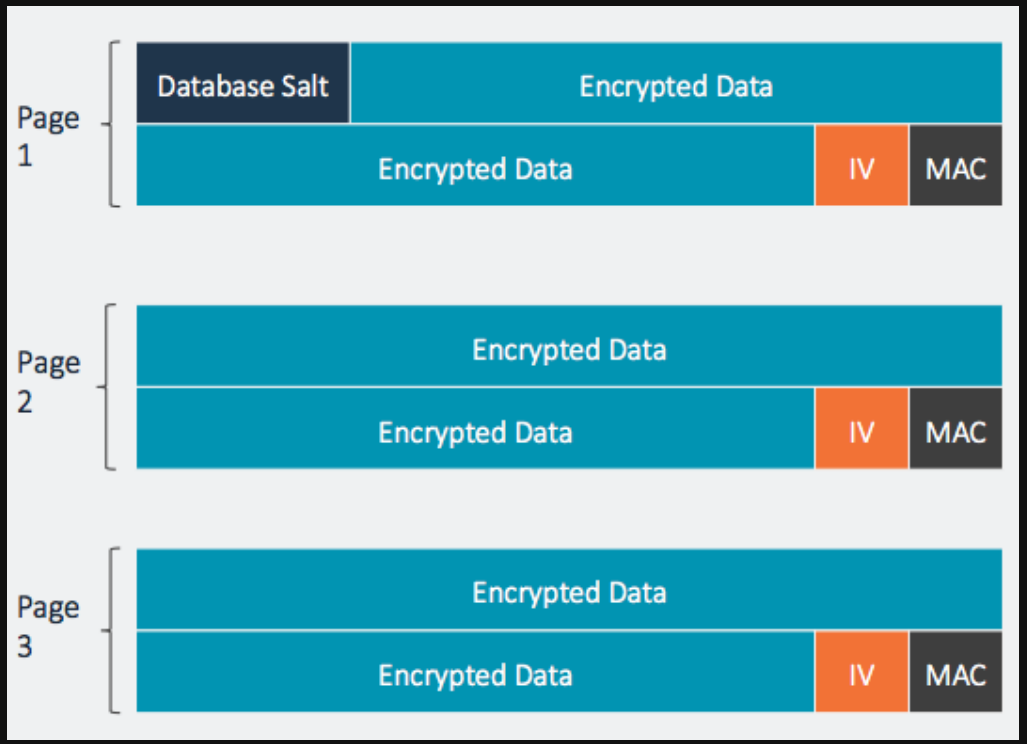
SQLCipher Platform Targets

C/C++, Obj-C, QT, Win32/.NET, Java, Python, Ruby, Linux, Mac OS X, iPhone/iOS, Android, Xamarin.iOS, and Xamarin.Android

Broad spectrum of use cases in both mobile and desktop devices

Our focus on securing user data where part of the key material is provided by the user

How it Works



- Transparent interaction
- On-the-fly
- Multiple crypto providers
- Standard KDF (salt + passphrase)
  - PBKDF2
  - Predates Scrypt

Current State of the Union

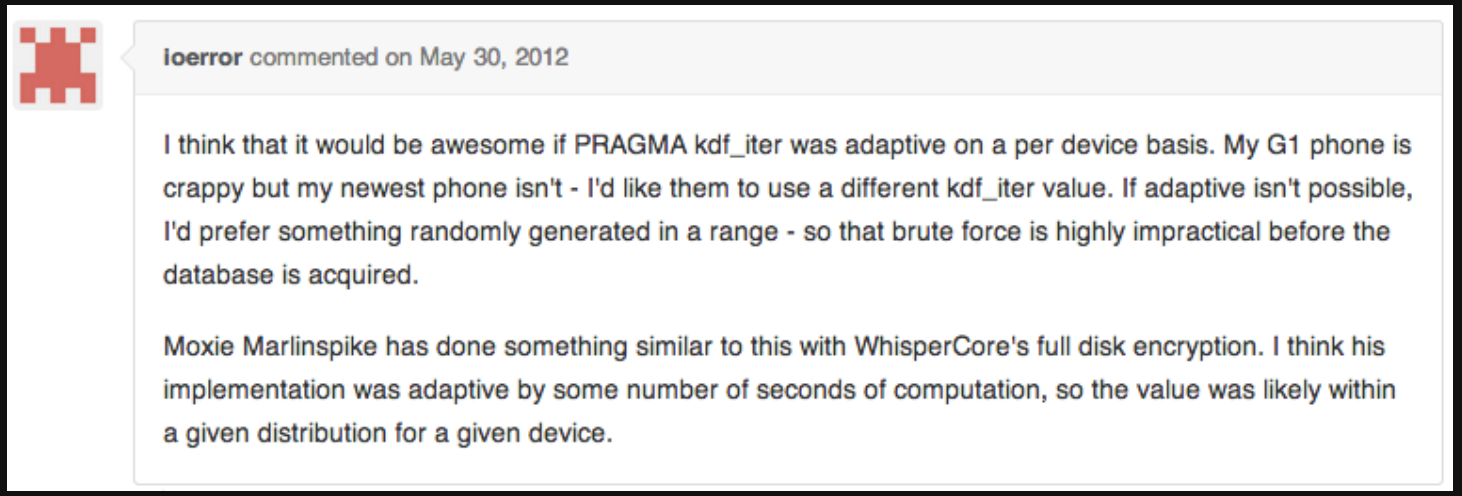
- SQLCipher uses 64,000 iterations when computing a key using PBKDF2
- SQLCipher previously used 4,000 iterations

How Can We Do Better

- Adaptive key derivation work factor
- Multifactor hardware token integration

Device and Platform Challenges

Our world isn't static



ioerror commented on May 30, 2012

I think that it would be awesome if PRAGMA kdf_iter was adaptive on a per device basis. My G1 phone is crappy but my newest phone isn't - I'd like them to use a different kdf_iter value. If adaptive isn't possible, I'd prefer something randomly generated in a range - so that brute force is highly impractical before the database is acquired.

Moxie Marlinspike has done something similar to this with WhisperCore's full disk encryption. I think his implementation was adaptive by some number of seconds of computation, so the value was likely within a given distribution for a given device.

Problems with static KDF length

- Desktop and mobile hardware differ
- Technology evolves (i.e., GPU acceleration)
- Different security requirements / risk profiles / UX experience

Adaptive KDF Goals

- Fast sampling across platforms
- Compute ideal work factor limited by time
- Allow sampling to occur on any platform

Select KDF length By Security Needs

```
sqlcipher> PRAGMA cipher_kdf_compute;
```

- Sample KDF on device
- Compute iteration length based on desired runtime
- Runs by default
- Tunable for time

Tune the Sampling

```
./sqlcipher foo.db
sqlcipher> PRAGMA key = 'foo';
sqlcipher> PRAGMA cipher_kdf_compute;
cipher_kdf_compute
------------------
1,096,007
```

```
./sqlcipher foo.db
sqlcipher> PRAGMA key = 'foo';
sqlcipher> PRAGMA cipher_kdf_compute = 2.0;
cipher_kdf_compute
------------------
2,278,910
```

```
./sqlcipher foo.db
sqlcipher> PRAGMA key = 'foo';
sqlcipher> PRAGMA cipher_kdf_compute = .5;
cipher_kdf_compute
------------------
575,280
```

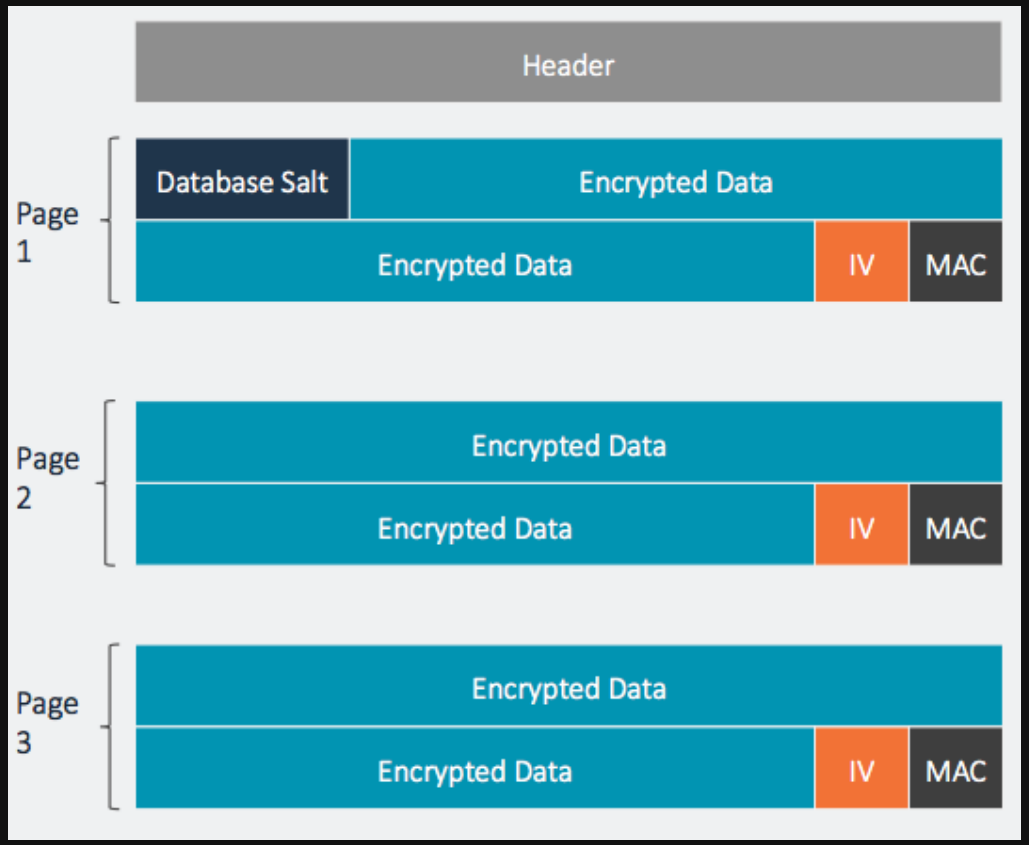
PBKDF2 Sampling Results

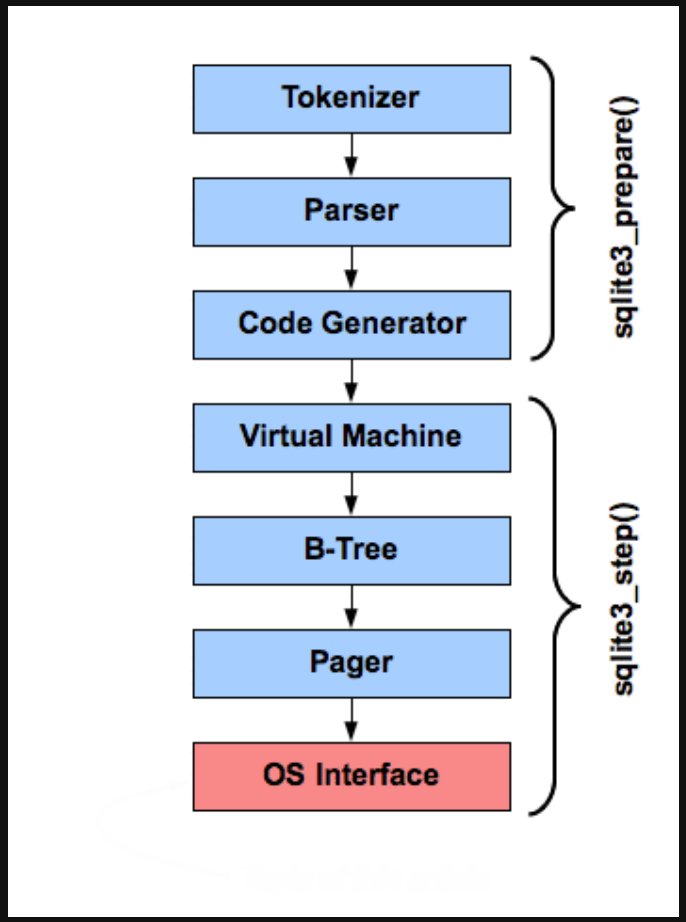
Device	Computed Work Factor
Mac Book Pro (2.3 GHz)	1,161,162
iOS Simulator (7.1)	1,060,260
iPhone 5S	481,882
Android Emulator (4.4.2)	44,139
Android Nexus S (2.3.6)	72,800
Android Galaxy Tab 2 (4.2.2)	80,640

Persisting Configuration

- Previously hard coded KDF work factor
- Now persist KDF work factor

New Database Structure





Adaptive KDF Summary

Pros:

- Fast sampling across platforms
- Compute ideal work factor limited by time
- Allow sampling to occur on any platform

Cons:

- Cross device performance
- Additional complexity within SQLCipher

Multi Factor Key Derivation

- Introduce an addition factor into key derivation process
- - Something you know: Passphrase
  - something you have: Hardware Token

Stepping Back – Current KDF

- Secret database key DKey
- Random database Salt (public) DSalt
- Iterations / Work Factor (adaptive!) I
- Key Length

```
PBKDF2(DKey, DSalt, I, Length)
```

Token Requirements

- Works offline
- Simple interface (USB?)
- Widely available
- Onboard crypto
- Secure key storage
- Multi-use
- Inexpensive

Yubikey

- Long history
- Multiple form factors
- Practically indestructable
- $25 / $40
- http://www.yubico.com/

DaPlug / Plug-Up

- New entrant
- Only Available in Europe
- €8.00 ($110 Shipping!)
- http://www.daplug.io/

Common Denominator

- Onboard HMAC-SHA1 Challenge / Response API
- Programmable write-only key

Simple Implementation

- Onboard Token Key and HMAC
- Permute database salt before use
- Uses SQLCipher provider callback

Simple MFA Process

- Secret database key DKey
- Random database Salt (public) DSalt
- Iterations / Work Factor (adaptive) I
- Key Length
- **Token Key TKey**
- **HMAC-SHA1**

```
PBKDF2(DKey, HMAC-SHA1(TKey,DSalt), I, Length)
```

Results

Pros:

- Database can only be opened with token in place
- Very simple implementation
- Key can't be extracted from token
- Operating on salt does not disclose non-public data to token hardware

Cons:

- USB Required
- Custom code
- API dependencies

More Information

- http://sqlcipher.net
- http://github.com/sqlcipher/sqlcipher/tree/vfs
- http://github.com/sqlcipher/sqlcipher-mfa

Feedback

Join the SQLCipher discussion

https://discuss.zetetic.net/category/sqlcipher

Questions?