How I Learned to Stop Worrying and Trust Crypto Again

GRAHAM STEEL

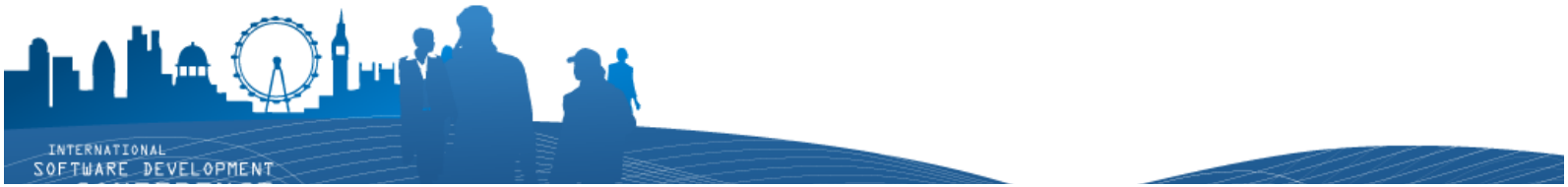
graham@cryptosense.com

@graham_steel

Cryptosense

QCon

Please evaluate my talk via the mobile app!

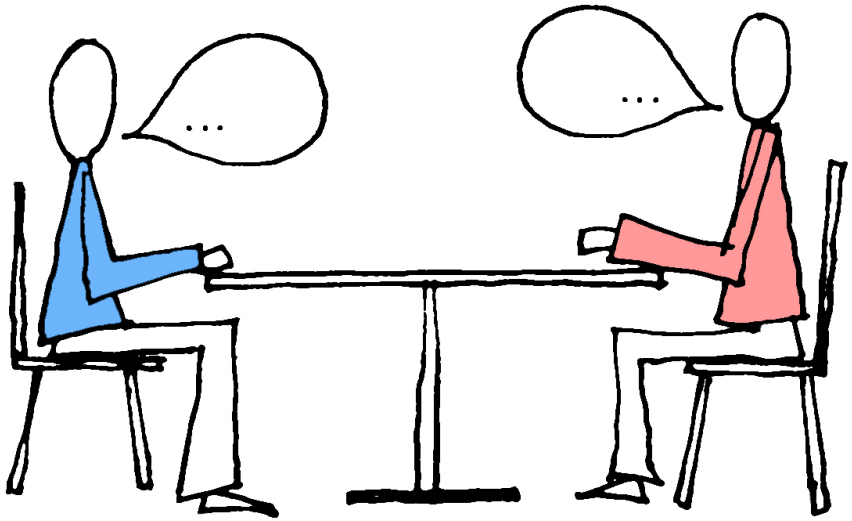


2013: The Paranoids Were Right

BUT “Properly implemented strong crypto systems are one of the few things that you can rely on"

How does cryptography get deployed?

Scenario 1: Cryptographer's dream



Step 1: Project manager and client evaluate security goals and threat scenario

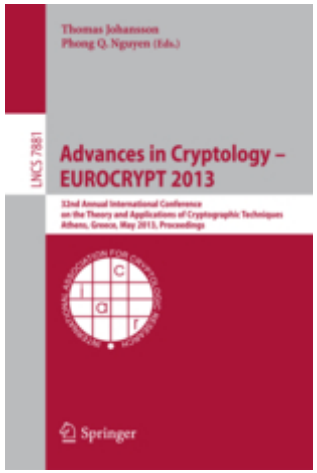
How does cryptography get deployed?

Scenario 1: Cryptographer's dream

Engineers and PM choose appropriate security notion

How does cryptography get deployed?

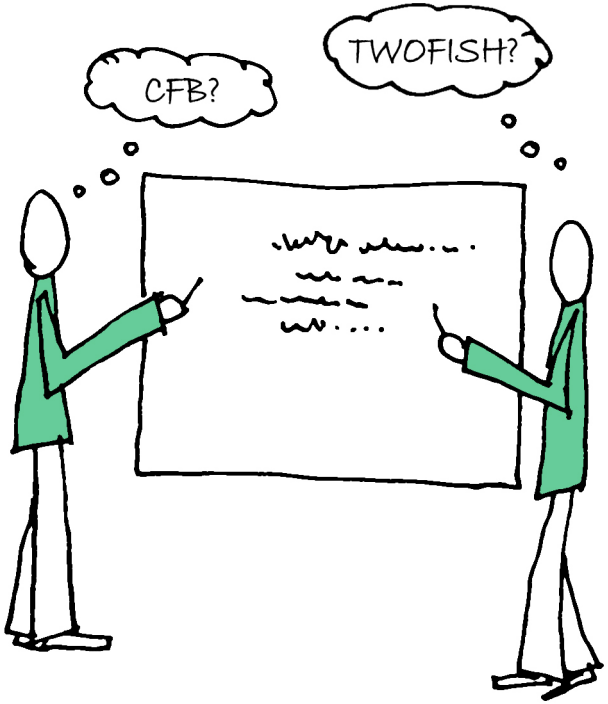
Scenario 1: Cryptographer's dream



Engineers consult the literature

How does cryptography get deployed?

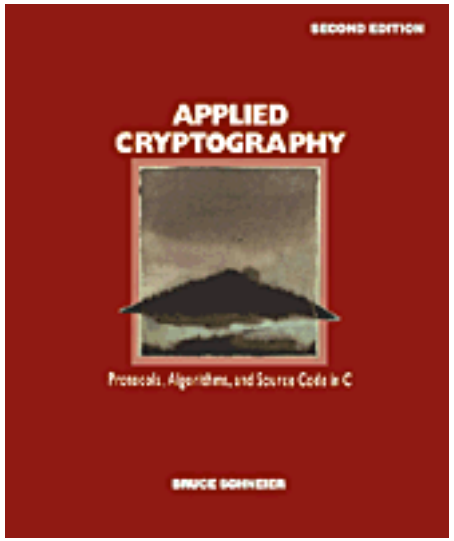
Scenario 2: Cryptographer's nightmare



Engineers plan some fun crypto stuff

How does cryptography get deployed?

Scenario 2: Cryptographer's nightmare



Engineers consult the literature

How does cryptography get deployed?

Scenario 2: Cryptographer's nightmare

WhatsApp mobile messaging app in the firing line again over cryptographic blunder

Bad kitty! "Rookie mistake" in Cryptocat chat app makes cracking a snap

Serious Crypto Bug Found In PHP 5.3.7

24/0

Posted in General Technology by DragonMicro | Comments (0)

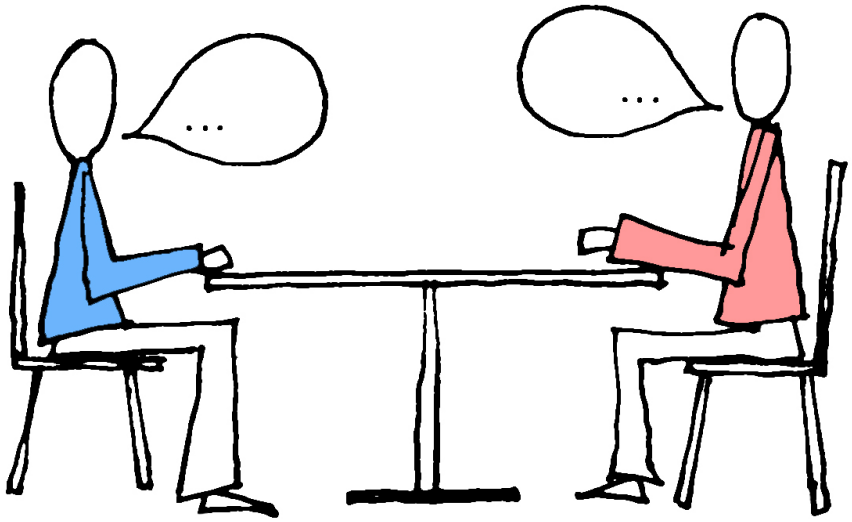
"The maintainers of the PHP scripting language are warning u
problem in the latest release and advising them not to upgrade
resolved. PHP 5.3.7 was just released last week and that vers
security vulnerabilities. But now a serious flaw has been found
to the way that one of the cryptographic functions handles inputs. In some cases, when the crypt() function is called using MD5 salts, the function will return only the salt value."

Apple fixes critical crypto bug in iOS, OS X fix to be released "soon"

Posted on 24 February 2014.

How does cryptography get deployed?

Scenario 3: Reality



Client tells PM project must use certain crypto APIs for compliance/hardware interop/legacy reasons

How does cryptography get deployed?

Scenario 3: Reality

PM tells engineer to use that API

How does cryptography get deployed?

Scenario 3: Reality

What happens next is the subject of this talk:

- How good are the common APIs?
- How can we use them securely?

Lesson 1: Beware of Proprietary APIs

What makes a good crypto API?

At least the following:

1. Is it open and subject to review?
2. Does it support modern cryptographic primitives?
3. Does it support flexible, secure key management?
4. Is it mistake-resistant?
5. Will it interoperate with other stuff?

PKCS#11

Almost ubiquitous for cryptographic hardware

Originally an RSA standard, moved to OASIS in 2013

Version 1.0 of PKCS#11 1995, current version 2.20 2004(!), v2.40 due RSN

Defined as a C header file + 400 page document describing what the functions should do

The standard is quite loose: every implementation is different

Assessment of PKCS#11

1. Since 2013 the API is open (before not clear), but actual implementations may be closed
2. v2.40 will have some more modern crypto (e.g. GCM) but still lots of legacy stuff
3. Key management is provided for but has many pitfalls (see BCFS CCS '10)
4. Not very mistake resistant (low level)
5. Good interoperability across platforms

Java JCA/JCE

The standard Java crypto interface

Consists of APIs (java.security, javax.crypto etc.) and providers that implement those interfaces (SunRSASign, SunJCE, Bouncy Castle etc.)

Widespread adoption in enterprise Java world

Hardware support limited (needs proprietary extensions or alternative APIs to handle login, sessions, most key management functions).

Assessment of Java JCE/JCA

1. The API is under Oracle's control. Providers sometimes open (like Bouncy Castle)
2. API extensible, providers support some modern crypto (e.g. GCM) but also much legacy stuff (e.g. RC2)
3. Some key management via Keystore. Not much public security analysis of this.
4. Not especially mistake resistant
5. Interoperability between providers good, also can have e.g. PKCS#11 provider

OpenSSL

Originally an open source implementation for TLS/SSL, now used for almost anything

1. Source available, but decision process murky
2. Contains legacy crypto (as does SSL!) as well as some modern extensions
3. Minimal key-management facilities
4. Very easy to get wrong (see references)
5. Compiled everywhere, TLS interop ok

MS CAPI/CNG

CNG slowly replacing CAPI on MS platforms

1. Proprietary, most providers closed
2. CNG contains some modern crypto, CAPI not so much
3. No management of persistent symmetric keys
4. Plenty of things to get wrong (e.g. contains the NSA backdoored RNG)
5. Not very interoperable

W3C Crypto API

A new project of the W3C to make crypto available in the browser to HTML5 apps

Contains modern crypto, but despite fresh start, also contains legacy crypto.

Some key-management (details still being hammered out)

Some effort to make it easier to use (SubtleCrypto vs WorkerCrypto, IV management,…)

Other libraries

NaCl – a crypto lib by Dan Bernstein

Nice high level design but favours Dan Berstein primitives (which may well be secure but have not been subject to as much review as e.g. AES)

cryptlib – C crypto library by Peter Guttman

Supports many standard protocols with nice high level interface, but no OAEP, no SHA-2

Crypto API Gotchas

So you've chosen a "good" API, a sound implementation, what could possibly go wrong?

Everything.

Given good APIs, even domain experts make mistakes in implementing cryptography.

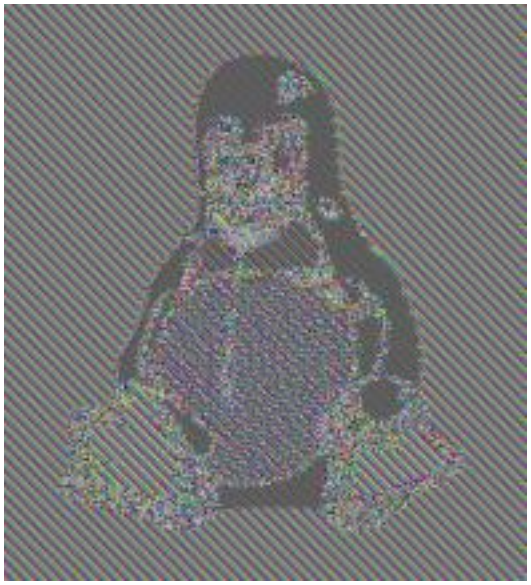
The following is a list of common mistakes. Avoiding these *does not mean* your crypto is secure, but it should reduce your consultancy bill ;-)

- will also demonstrate that security is fractal (Butler Lampson)

Examples of crypto API gotchas

- Legacy algorithms, modes, often used by default
  - e.g. ECB “mode”

  “All these other modes need initialization vectors. ECB doesn’t. I’ll use ECB”



More Crypto API Gotchas

- Other Block cipher modes: IV management often left to application. Easy to get wrong.
  - Requirements are different for each mode
- Unauthenticated encryption
  - Don't think you need it? You need it.
  - Less error-prone (IMO) to use an authenticated mode (like AES-GCM or RSA-OAEP) than patch a MAC around AES-CBC or RSA-PKCS#1v1.5
- Random number generators incorrectly initialized
  - Famous recent attacks (Debian OpenSSL, GCD RSA..)

Protocols

Typically crypto operations are part of a protocol

(think SSH, TLS,…)

If you roll your own protocol, even if all your crypto is secure when considered in isolation, probability of a security error in the first draft is 1.

Even mature protocols have flaws (e.g. TLS – yet another set of flaws announced Monday)

But you're still better off following a standard protocol than making one up.

Conclusions

To maximise your chances of producing “strong crypto properly implemented”

- Favour open API standards over proprietary
- Look out for legacy
- Check out the well-known pitfalls, avoid them
- Review, review, review!

References

Egele et al., “An empirical study of cryptographic misuse in android applications.” CCS 2013.

Fahl et al. “Why eve and mallory love android: An analysis of android SSL (in)security” CCS 2012

Bortolozzo et al. “Attacking and Fixing PKCS#11 Security Tokens” CCS 2010

Meyer & Schwenk “Lessons Learned from Previous SSL/TLS Attacks” S&P 2013

Most recent TLS attacks secure-resumption.com

Thanks

graham@cryptosense.com

@graham_steel