Thinking about Cryptography: Crypto Flaws and How to Avoid Them

Jonathan Katz
Some good news and some bad news...
The bad news

Implementing currently

\[ \text{Crypto is hard...} \]

...leading to many flaws
The good news

Stupid encryption mistakes criminals make
Blown cover: Malware authors show how easy it is to get encryption wrong and, in the process, help security pros crack their code
Crypto flaws ⇒ exploits

• Netscape
• Debian
• MSCHAP
• WEP
• Android/Bitcoin wallets
• PS3, XBox
• Flickr, Visa
• Yahoo, LinkedIn
• ...

Scope of this talk

• In scope
  – Developer misuse of crypto (libraries)

• Out of scope
  – Crypto backdoors
  – Vulnerabilities in crypto libraries (incl. timing attacks, programming errors)
  – Clever attacks on complex protocols (BEAST, CRIME, ...)
Examples...

- Using deterministic encryption
- Using nonrandom IV with CBC-mode encryption
- Using nonrandom encryption keys
- Using non-salted hashes
- Seeding PRNGs with fixed constants or low-entropy seeds
None of this is new!

• Anderson, “Why Cryptosystems Fail,” ACM CCCS 1993
• Gutmann, “Lessons Learned in Implementing and Deploying Crypto Software,” USENIX Security 2002
IEEE Top-10 Security Design Flaws

- Never assume trust
- Use authentication that cannot be bypassed
- Authorize after authentication
- Separate data and control
- Validate all data
- Use cryptography correctly
- Identify sensitive data and how to handle it
- Consider users
- Understand how external components affect attack surface
- Be flexible with future changes to objects/actors
The problem is still here...


![Graph showing prevalence of cryptographic vulnerabilities by CWE (percentage of applications affected)]

Crypto problems more common than XSS, SQL injection
The problem is still here...

  - 88% of Android applications using crypto APIs make mistakes
  - E.g.,
    - Weak keys/parameters
    - Deprecated/weak algorithms
    - Insufficient entropy
    - Ineffective password hashing
But it hasn’t gone away...

- Lazar et al., “Why does cryptographic software fail?” APSys 2014
  - 269 crypto vulnerabilities in CVE database (Jan 2011 – May 2014)
  - 83% due to crypto misuse
  - E.g.,
    - Weak keys/parameters
    - Deprecated/weak algorithms
    - Insufficient entropy
    - Neglecting to encrypt sensitive data
Avoiding crypto flaws

• Don’t use weak keys/parameters
• Don’t use deprecated/weak algorithms
• Use sufficient entropy
• ...

Avoiding crypto flaws

• Don’t use a keyed hash as a MAC
• Don’t use textbook RSA signatures/encryption
• Don’t use unauthenticated encryption
• Implement schemes exactly as defined
• Avoid timing vulnerabilities
• ...
Common themes?
Common themes

• Unusable cryptography

• Misunderstanding cryptography
Same underlying cause!

• Incomplete separation between specification (API) and implementation (mechanism)
Public-key encryption

Key encapsulation
- RSA-OAEP
- DHIES

Symmetric-key encryption
- AEAD
  - CBC-mode
  - CBC-MAC

- AES-128
Usable cryptography?
Usable by whom?

• The users

• Developers!

• Let’s look at a problem developers face...
Private-key encryption

Given $c$, attacker learns nothing about $m$
Message authentication

Given $c_1$, ..., attacker cannot generate $c^*$ that fools receiver to output new $m^*$
CBC-mode encryption
CBC-MAC
CBC in OpenSSL

• Provides a “CBC” function that takes as input a key, an IV, and a message
  – CBC: choose IV at random, then call \( \text{CBC}_k(IV, m_1, \ldots, m_n) \) and return \( IV + \) the result
  – CBC-MAC: call \( \text{CBC}_k(0\ldots0, |m|, m_1, \ldots, m_n) \) and return the last blocksize bytes of the result

• What could go wrong?
CBC-mode encryption

- CBC-mode encryption is only secure when used with an *unpredictable* IV
  - $\text{CBC}_k(0...0, m_1, ..., m_n)$ is not secure
  - $\text{CBC}_k(\text{ctr}, m_1, ..., m_n)$ is not secure

- Correctness
  - $\text{CBC}_k(\text{IV}, m_1, ..., m_n)$ without the IV is not (fully) decryptable
CBC-MAC

• CBC-MAC is very brittle!
  – Returning $\text{CBC}_k(0...0, |m|, m_1, ..., m_n)$ is insecure
    • Returning too many or too few bytes from the end can also be insecure
  – Returning last block of $\text{CBC}_k(0...0, m_1, ..., m_n)$ is insecure
  – Using $\text{CBC}_k(\text{IV}, |m|, m_1, ..., m_n)$ can be insecure and/or incorrect
Interactions

• Using CBC-mode encryption and CBC-MAC (properly!) with the same key is insecure

• Even if CBC-mode/CBC-MAC correct, need to use with secure block cipher, random key, ...
CBC in OpenSSL

- OpenSSL documentation:
  - Normally, [CBC] is found as the function `alg_cbc_encrypt()`. Be aware that `alg_cbc_encrypt()` is not really CBC (it does not update the IV); use `alg_ncbc_encrypt()` instead
  - The use of different starting variables prevents the same plaintext enciphering to the same ciphertext
CBC in OpenSSL

- OpenSSL wiki:
  - [Set] up a 256-bit key and a 128-bit IV ... Make sure you use the right key and IV length for [the] cipher ... or it will go horribly wrong!!

  - [Set] up a buffer for the ciphertext...It is important to ensure that this buffer is sufficiently large ... or you may see a program crash (or potentially introduce a security vulnerability ...)

Developers need help


• App developers often turn to StackOverflow for programming help

• How valuable/dangerous is this?
  – Only 17% of StackOverflow posts had secure code
  – User study: developers using StackOverflow have higher chance of writing *functional* code, but lower chance of writing *secure* code
Fundamental problem

• Why should a developer need to know *anything* about the low-level crypto details?

• Should instead be provided with an interface to *encrypt* and an interface to *authenticate*  
  – Newer crypto libraries (NaCl, Keyczar) work this way  
  – Safer defaults

• API documentation needs to improve  
  – Listed in OpenSSL roadmap since mid-2015
Misunderstanding cryptography
Core principles of modern crypto

- Formal definitions
  - Precise, mathematical model and definition of what security means

- Assumptions
  - Clearly stated and unambiguous

- Proofs of security
  - Move away from design-break-patch cycle
Formal definitions

• Defines the syntax of an API...

• Allows for an understanding of the security guarantees provided by that API
  – I.e., semantically secure encryption ensures that a passive adversary learns nothing about the message from the ciphertext

• This is all that should be exposed by a proper crypto library
Formal definitions

• Definitions are *central* to modern crypto

• Definitions as a “first-class object”

• Specifying a threat model is critical
Assumptions

• (Most) modern cryptography relies on unproven assumptions
  – Get over it

• Be vaguely aware when an assumption is (close to being) falsified
  – E.g., RSA-1024

• Should be handled automatically if using a proper crypto library
Proofs

• Crypto schemes can be *proven* secure!
  – Relative to some definition
  – Based on certain assumption(s)

• These proofs are brittle
  – Schemes must be implemented as described
  – Don’t design your own crypto
Constructions?

• No!

• When teaching crypto to a general audience, focus on *definitions* and *secure usage*
  – Details of schemes *irrelevant* to average developer
Avoiding flaws?

- “CBC-mode encryption produces random-looking ciphertexts”
- “View encryption as a black box that completely scrambles the message”

⇒ Encryption provides message integrity!
   (Many authentication protocols built/broken based on this incorrect reasoning)
How could this have gone?

• What is the desired security (threat model)?
  – Impersonator should be unable to properly encode a random challenge from verifier

• Does semantically secure encryption imply it?
  – Nothing in the definition says it does...

• What is the appropriate primitive here?
  – Message authentication code
Avoiding flaws?

• “Hash functions produce completely random-looking output”
• “Changing even one bit of input results in completely unpredictable output”

⇒ Hash(key, m) must be a good MAC!

   (Many integrity mechanisms built/broken based on this incorrect reasoning)
How could this have gone?

• What is the desired security (threat model)?
  – Message integrity

• Does a collision-resistant hash imply it?
  – Nothing in the definition says it does...

• What is the appropriate primitive here?
  – Message authentication code
Summary

• Many crypto flaws result from developers’ exposure to *implementations* rather than *specifications*
  – Schemes vs. clean, easy, clear APIs
  – Constructions vs. definitions
• Crypto libraries should *only* expose an API for desired high-level tasks
• Developers should *focus on security goals* rather than low-level details about schemes
Questions?