Everything you need to know about cryptography in 1 hour

Colin Percival
Tarsnap
cperciva@tarsnap.com

May 13, 2010
Lots of people get cryptography wrong:
- Google Keyczar (timing side channel). ← Stupidity
- SSL (session renegotiation). ← Stupidity
- Amazon AWS signature method 1 ← Using a tool wrong (non-collision-free signing).
- Flickr API signatures ← Using the wrong tool wrong (hash length-extension).
- Intel HyperThreading ← Unusual environment (architectural side channel).
- WEP, WPA, GSM... (various flaws). ← Unusual environment

Cryptography is usually broken for one of three reasons:
- Stupidity.
- Using the wrong tools or using them in the wrong way.
- Unusual environments.
Conventional wisdom: Don’t write cryptographic code!
- Use SSL for transport.
- Use GPG for protecting data at rest.
- “If you’re typing the letters A-E-S into your code, you’re doing it wrong.” — Thomas Ptacek

Reality: You’re going to write cryptographic code no matter what I say, so you might as well know what you’re doing.

Reality: Most applications only need a small set of well-understood standard idioms which are easy to get right.

55 minutes from now, you should:
- Know what to do in 99% of the situations you’ll encounter.
- Know where some of the common mistakes are.
- Know when you’re doing something non-standard and you really need to consult a cryptographer.
Cryptography protects against *some* attacks, but not all.
- “Three Bs”: Bribery, Burglary, Blackmail.
- Fourth B: (Guantanamo) Bay.

Attacking people is often more expensive than attacking data.

Attacking people is almost always more dangerous than attacking data.
- Data doesn’t hold press conferences to complain that it was tortured!
  - (The information, not the android.)

The purpose of cryptography is to force the US government to torture you.
- Hopefully they’ll decide that your information isn’t that important.
Cryptography has three major purposes: Encryption, Authentication, and Identification.

- Encryption prevents evil people from reading your data.
- Authentication (aka. Signing) prevents evil people from modifying your data without being discovered.
- Identification prevents evil people from pretending to be you.

Sometimes Authentication and Identification are performed in a single step: “this message hasn’t been modified since I wrote it” and “I’m Colin” are replaced by a single “this message hasn’t been modified since Colin wrote it”.

In most cases you will want to put together two or more cryptographic components.
The *plaintext* is the data we care about.

The *ciphertext* is the data evil people get to see.

A *key* is used to convert between these. Sometimes we need several keys.

*Symmetric* cryptography is when converting plaintext to ciphertext uses the same key as converting ciphertext to plaintext.

*Asymmetric* cryptography is when the two directions use different keys.

*Ideal* cryptographic components don’t really exist, but if a cryptographic component is recognizably non-ideal, it is generally considered to be broken.
An ideal hash function $H(x)$ is a function mapping arbitrary-length inputs to $n$-bit outputs which is:

- Collision-resistant, and
- One-way.

(Collision-resistant) means that it takes $\approx 2^{n/2}$ time to find two inputs which have the same hash.

(One-way) means that given a hash, it takes $\approx 2^n$ time to find an input which has that hash.

Nothing else is guaranteed!

In particular, knowing $H(x)$ might allow an attacker to compute $H(y)$ for some values of $y$. 

Colin Percival  Tarsnap  cperciva@tarsnap.com  Everything you need to know about cryptography in 1 hour
DO: Use SHA-256.
DO: Consider switching to SHA-3 within the next 5-10 years (once NIST decides what it is, probably in 2012).
DO: Use a hash when you can securely distribute $H(x)$ and want to validate that a value $x'$ which you received insecurely is in fact equal to $x$.
DON’T: Use MD2, MD4, MD5, SHA-1, RIPEMD.
DON’T: Put FreeBSD-8.0-RELEASE-amd64-disc1.iso and CHECKSUM.SHA256 onto the same FTP server and think that you’ve done something useful.
DON’T: Try to use a hash function as a symmetric signature.
Symmetric authentication is performed by providing a message authentication code (MAC).

An ideal message authentication code $f_k(x)$ uses a key to map arbitrary-length inputs to $n$-bit outputs such that it takes $\approx 2^n$ time for an attacker to generate any pair $(y, f_k(y))$ even if given arbitrary pairs $(x, f_k(x))$.

- Sometimes called a “random function”.

Unlike hashing, knowing $f_k(x)$ does not allow you to compute $f_k(y)$ for some other $y$.

- The Flickr API used hashing to authenticate API requests where they should have used a MAC.
DO: Use HMAC-SHA256.

DO: Guarantee that you cannot have two different messages result in the same data being input to HMAC-SHA256.
  - Amazon and Flickr both got this wrong.

AVOID: CBC-MAC.
  - Theoretically secure, but exposes your block cipher to attacks.

AVOID: Poly1305.
  - If your name is Daniel Bernstein, go ahead and use this. Otherwise, you’re never going to produce a secure and correct implementation.

DON’T: Leak information via timing side channels when you verify a signature.
A *side channel* is any way that an attacker can get information other than the ciphertext.

- Cryptosystems are defined by their mathematical design, whereas side channels are inherently artifacts of how cryptosystems are implemented.

The most common side channel is timing – how long it takes for you to encrypt/decrypt/sign/verify a message.

Other side channels include electromagnetic emissions ("TEMPEST"), power consumption, and microarchitectural features (e.g., L1 data cache eviction on Intel CPUs with HyperThreading).
DO: Consult a cryptographer if you’re planning on giving evil people physical access to anything which does cryptography (e.g., smartcards).

DO: Consult a cryptographer if you’re planning on allowing evil people to run code on the same physical hardware as you use for cryptography (e.g., virtualized systems).

DO: Consult a cryptographer if you’re planning on releasing a CPU which leaks information in new and exciting ways.

  - Intel probably got this wrong.

DON’T: Write code which leaks information via how long it takes to run.
Timing attacks

- **AVOID:** Key-dependent or plaintext-dependent table lookups.
- **DON’T:** Have key-dependent or plaintext-dependent branches (if, for, while, foo ? bar : baz).
- **DON’T EVEN DREAM ABOUT:** Writing the following code:
  ```c
  for (i = 0; i < MACLEN; i++)
     if (MAC_computed[i] != MAC_received[i])
        return (MAC_IS_BAD);
  return (MAC_IS_GOOD);
  ```
- **DO:** Write the following code:
  ```c
  for (x = i = 0; i < MACLEN; i++)
     x |= MAC_computed[i] - MAC_received[i];
  return (x ? MAC_IS_BAD : MAC_IS_GOOD);
  ```
- Google Keyczar got this wrong.
Symmetric encryption is usually built out of *block ciphers*.

An **ideal block cipher** uses a key to bijectively map \( n \)-bit inputs \( x \) to \( n \)-bit outputs \( E_k(x) \) such that knowing pairs \((x, E_k(x))\) doesn’t allow you to guess \((x', E_{k'}(x'))\) for any \((x', k') \neq (x, k)\) with probability non-negligibly higher than \(2^{-n}\).

- Sometimes called a “random permutation”.

Usually all we care about is that \( E_k(x) \) doesn’t reveal information about \( E_{k'}(x') \) for \( x' \neq x \).

- If an attacker can get useful information about a block cipher by looking at how it handles different (but related) keys, the block cipher is said to be vulnerable to a *related-key attack*.
DO: Use AES-256.
   AES-256 is vulnerable to a related-key attack, but this will never matter as long as you get other things right.
   AES-128 is theoretically strong enough, but block ciphers are hard to implement without side channels, and the extra key bits will help if some key bits get exposed.

DON’T: Use blowfish.

DON’T EVEN DREAM ABOUT: Using DES.

AVOID: Triple-DES.

DON’T: Use a block cipher “raw”; instead, use it in an established mode of operation.
A block cipher mode of operation tells you how to use a block cipher to protect stream(s) of data.

In many cases, the plaintext needs to be padded to a multiple of the block size; the block cipher mode of operation will tell you how to do this.

Modes of operation usually have funky initialisms: ECB, CBC, CFB, OFB, CTR, IAPM, CCM, EAX, GCM...

Please don’t ask me how to expand all of these.

Most modes of operation provide only encryption; some provide authentication as well.
DO: Use CTR mode.

DON’T: Use modes which provide both encryption and authentication.

DON’T EVEN DREAM ABOUT: Using ECB mode.

DO: Use a MAC (i.e., HMAC-SHA256) to authenticate your encrypted data.
   - If you think you don’t need this, consult a cryptographer. He’ll tell you that you’re wrong.

DO: Verify the authenticity of your encrypted data before you decrypt it.
An asymmetric authentication scheme uses a signing key to transform plaintext into ciphertext and a verification key to transform ciphertext into either the plaintext or “invalid signature”.

- The signing key cannot be computed from the verification key, but the verification key can usually be computed from the signing key.
- The ciphertext usually consists of the plaintext plus a signature.

An asymmetric authentication scheme is considered to be broken if an attacker with access to the verification key can generate any valid ciphertext, even if he can convince you to sign arbitrary other plaintexts.
Asymmetric authentication

- **DO:** Use RSASSA-PSS (RSA signing with Probabilistic Signature Scheme padding).
- **DO:** Use a 2048-bit RSA key, a public exponent of 65537, and SHA256.
- **DON’T:** Use PKCS v1.5 padding.
- **DON’T EVEN DREAM ABOUT:** Using RSA without message padding.
- **PROBABLY AVOID:** DSA.
- **PROBABLY AVOID:** Elliptic Curve signature schemes.
- **DON’T EVEN DREAM ABOUT:** Using the same RSA key for both authentication and encryption.
Asymmetric encryption is like asymmetric signing, except the opposite way around: Plaintext is converted to ciphertext using a public key, but converting ciphertext to plaintext requires the private key.

An asymmetric encryption scheme is considered to be broken if an attacker can decrypt a given ciphertext, even if he can convince you to decrypt arbitrary other ciphertexts.

Most asymmetric encryption schemes have a fairly low limit on the size of the message which can be encrypted.
DO: Use RSAES-OAEP (RSA encryption with Optimal Asymmetric Encryption Padding).

DO: Use a 2048-bit RSA key, a public exponent of 65537, SHA256, and MGF1-SHA256.

DON’T: Use PKCS v1.5 padding.

DON’T: Use RSA without message padding.

DO: Generate a random key and apply symmetric encryption to your message, then apply asymmetric encryption to your symmetric encryption key.

DO: Be especially careful to avoid timing side channels in RSAES-OAEP.
Passwords / passphrases are often used directly for Identification, but can also be used for Encryption or Authentication.

DO: Avoid using passwords whenever possible.

DO: Use a key derivation function to convert passwords into keys as soon as possible.
  - DO: Use PBKDF2 if you want to be buzzword-compliant.
  - DO: Use scrypt if you want to be \( \approx 2^8 \) times more secure against serious attackers.

DON’T EVEN DREAM ABOUT: Storing your users’ passwords on your server.
  - No, not even if they’re encrypted.
SSL is a horrible system.
- SSL is far too complex to be implemented securely.
- SSL gives attackers far too many options for where to attack.
- SSL requires that you decide which certificate authorities you want to trust.
  - Do you trust the Chinese government?
- Unfortunately, SSL is often the only option available.
- **DO:** Distribute an asymmetric signature verification key (or a hash thereof) with the client side of client-server software, and use that to bootstrap your cryptography.
- **DO:** Use SSL to secure your website, email, and other public standard Internet-facing servers.
- **DO:** Think very carefully about which certificate authorities you want to trust.
**DO: Consult a cryptographer if...**

- Your cryptography is going to be on hardware which attackers have physical access to (e.g., smartcards).
- You need to use the minimum possible amount of power (e.g., on mobile phones).
- You need to process the maximum possible data rate (e.g., 10 Gbps IPSec tunnels).
- You need to transmit the minimum possible number of bits (e.g., communicating with a nuclear submarine).
- You want to ignore any of the advice I’ve given in this talk.
Questions?