Cryptographic Foibles and Missteps

The misuse and Abuse of Good Security Primitives (and How to do it Properly)

Joshua Hill
InfoGard Laboratories
josh-ccc@untruth.org
http://www.untruth.org

Pleased to meet you…

- B.S. Computer Science from Cal Poly
- M.S. Mathematics from Cal Poly
- Worked at InfoGard Labs for 10 years
  - FIPS 140, CC, VISA PED, Postal, Systems, etc
  - Device and System security evaluations
  - Reviewed hundreds of products
    - Source code, documentation, etc.

What's This Talk About?

- Good Primitives
  - No proprietary algorithms
  - No weak primitives
  - High security assurance primitives
    - Misunderstood
    - Used incorrectly
    - Abused
  - "It is impossible to make things fool proof…"
- I've seen every problem here at least once

What is this talk NOT about?

- Snake Oil, Inc.
  - “One Time Pad!”
  - “1,000,000 bit keys!”
  - “Completely Unbreakable!”
  - “Revolutionary!”
  - “Inverse N-dimensional permutation matrix routed through the exhaust manifold and the main deflector dish, with a twist of lemon.”
- Such schemes have no assurance of security
  - New, secret and unproven systems are **expected** to be flawed

I came here for an argument!

- Ciphers provide confidentiality
- Ciphers do **not** (in general) provide integrity
- It is not generally possible to identify “gibberish” programmatically
- Example Modes
  - ECB
  - CBC
  - CTR

Exact Change Only, Please

- ECB is the raw mode of the cipher
- A given data block always encrypts to the same ciphertext block
  - This can expose structure in the plaintext
- Any bit level change should flip half (on average) of the other bits in the block
  - Sounds great, but can you tell?

Order Matters!

- Blocks can be reordered
- Reordering or repetition of blocks can change the message
- Any ciphertext encrypted with the same key can be used as a source
- The security of the session is dependent on the plaintext data formatting!

The Rosencrantz & Guildenstern Attack

From: King Claud
ius To: The King
of England Plea
se help me kill
my nephew Hamlet
. Please send me
evidence with my
loyal chattel:
my men Rosencra
ntz & Guildenste
rn.

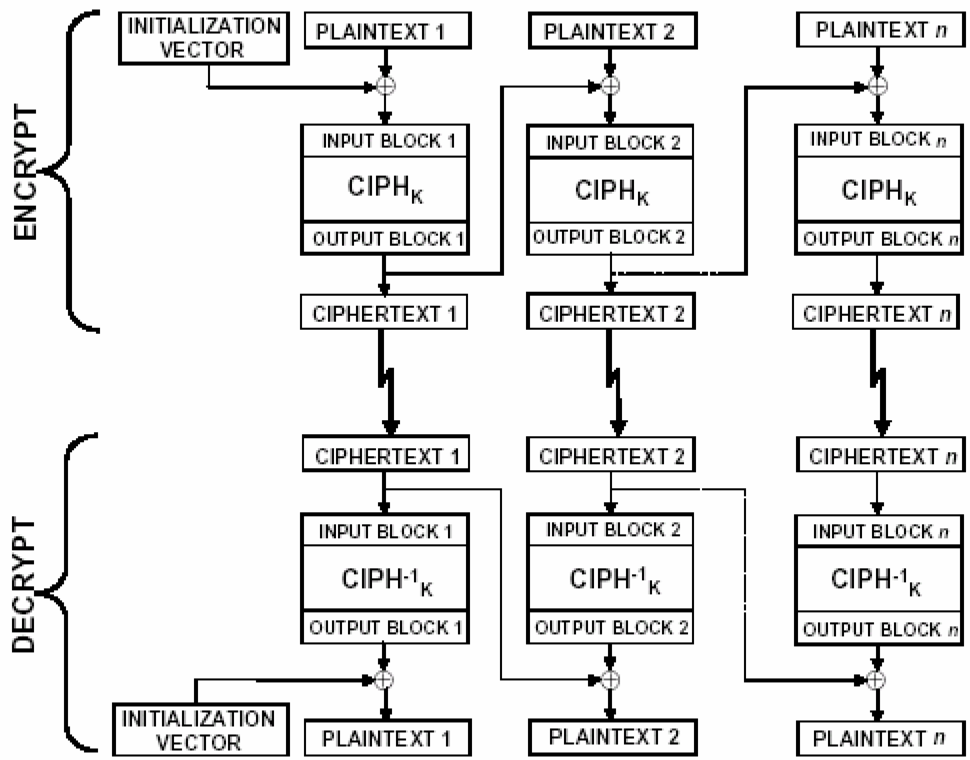
Becomes

From: King Claud
ius To: The King
of England Plea
se help me kill
my men Rosencra
ntz & Guildenste
rn.
. Please send me
evidence with my
loyal chattel:
my nephew Hamlet

Cipher Block Chaining (CBC) Mode

- CBC Structure
  - Uses Initialization Vector (IV)
  - IV is XORed with plaintext before encryption
  - Chains ciphertext output to next block's IV
- Error propagation
  - Plaintext block corresponding to modified ciphertext is corrupted
  - Second block has corruptions dependent on initial corruption
  - Third block is not corrupted

CBC Mode



Reference: NIST SP800-38a

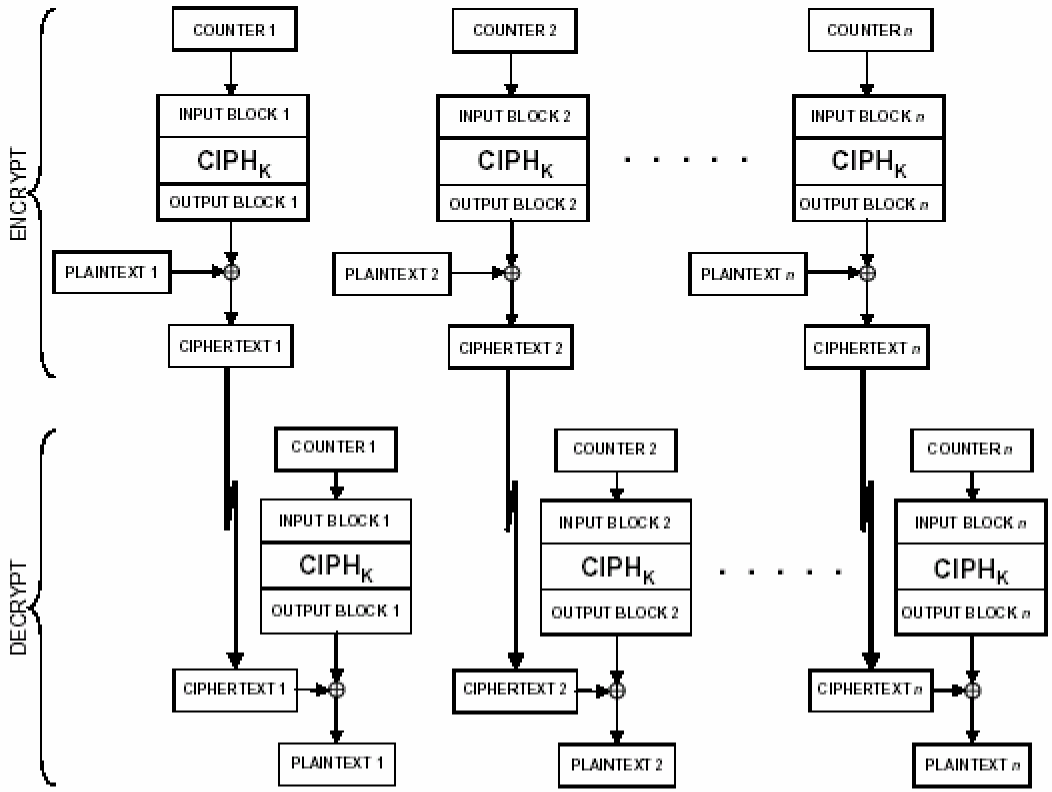
CBC Shenanigans!

- An attacker can change any number of targeted bits in a single plaintext block
- The prior block is corrupted
  – Can you tell?
- What if the attacker can alter the IV?
  – Change the first block of text
  – No consequences!

Counter (CTR) Mode

- CTR Mode Structure
  - Stream cipher like mode
  - Uses encrypted counter to make keystream
    - Ciphertext is keystream XOR plaintext
  - Only uses cipher in encrypt mode
- Notes
  - Counter must be unique
  - Counter need not be secret but integrity should be assured
  - No error propagation
    - Attacker can target bitwise changes anywhere with no consequences

CTR Mode



Reference: SP800-38a

CTR Shenanigans!

- An attacker can change… anything.
- If an attacker can force a repeated counter to be selected under the same key…
  - Step 1: Specify a repeated counter
  - Step 2: Just XOR the ciphertexts together
  - Step 3: Profit!

Postmortem (Integrity)

- Integrity desired, but no integrity protection is included
  - Without cryptographic integrity protection, you’re left with data dependent “protections”
- Errors of assumption:
  - “Cryptography” is not magic security pixie dust!
  - “Confidentiality” isn’t the only goal out there!

Bonanza!

- RSA is fragile
- Fragile things must be used carefully
- Therefore, RSA must be used carefully.

Next up: Socrates is a man, baby!

RSA

- Encryption/Decryption
  - Encrypt with public key
  - Decrypt with secret key
- Signing/Verifying
  - Sign with private key
  - Verify with public key
- Signing looks like decryption, and verifying looks like encryption (for RSA only)
- Strength based on the difficult of factoring

RSA Parameters

- Primes p, q
- Modulus n = pq
- Phi: $\phi(n) = (p-1)(q-1)$
- e (the public exponent), $\gcd(e, \phi(n)) = 1$
  - Common selections are 3, 17, 65537
- d (the private exponent), $ed \equiv 1 \bmod \phi(n)$
- The public key is (n, e)
- The private key is d

Encrypt/Decrypt

- Encrypt
  - The Message, m
    - View the message as a positive integer
    - m must be smaller than n
  - Ciphertext, c
    - Calculated: $c = m^e \bmod n$
- Decrypt
  - Use the private key to decrypt
    - Calculated: $m' \equiv c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^1 \equiv m \bmod n$

We have… prime numbers?

- It is hard to generate numbers that are definitely prime
  - Techniques do exist, but they are slow
- There are an infinite number of primes…
  - But, they get more sparse as numbers get bigger
- Generally we use probabilistic techniques for finding primes
  - Miller-Rabin Test with n rounds
    - Generally cited lower bound of failure of $(1/4)^n$

RSA Cake

- A problem with a small modulus
  - If the same message is encrypted and sent to *e* different distinct parties, the attacker can decrypt the message
  - This can be overcome through random padding

RSA Sorbet

- Another problem with a small modulus
  - If the recipient is forgiving when doing a signature verification an attacker can forge signatures
  - PKCS#1 v1.5 padding

    `00 01 FF FF FF ... FF 00  ASN.1  HASH`
  - Number of padding ‘`FF`’ bytes is important!
  - This can be overcome through strict enforcement of padding format

RSA Pudding

- Things to watch for in key generation
  - p, q may not actually be prime
  - d may be “small” (half as long as n)
  - Every key must have a unique modulus
  - p, q should have “the right form”
    - About the same size, but not too close to each other
    - p-1, p+1, q-1, q+1 should have large factors

Strawberry Tart

(… well, it's got **some** RSA in it…)

- Important System Characteristics
  - Use RSA keys for either sign/verify or encrypt/decrypt, **not both**
    - If you do both with the same key, you risk doing the attackers job for them
  - Only operate on properly padded messages!
  - On encrypt/sign, do the padding yourself!
  - On decrypt/verify, test the padding yourself!
  - Never give back error messages that indicate why the operation failed

Postmortem (RSA)

- Be afraid… Be **very** afraid!
- Use RSA only in well implemented, evaluated protocols
- Use implementations of these protocols made by cryptographic experts

Some Random Notes

- Most cryptographic processes involve random values (e.g., keys)
- These values must not be guessable
- Strength should be quantifiable
  - Uncertainty estimated as “entropy”

“True” RNGs

- True RNGs (TRNGs)
  - Are difficult to characterize
  - Fail subtly
  - Are difficult to make dependable
  - Are generally not full entropy
- Good practice is to pass “through” a good Pseudo RNG (PRNG)
  - Masks failures
  - Good designs “accumulate” entropy
  - Most designs are not good!

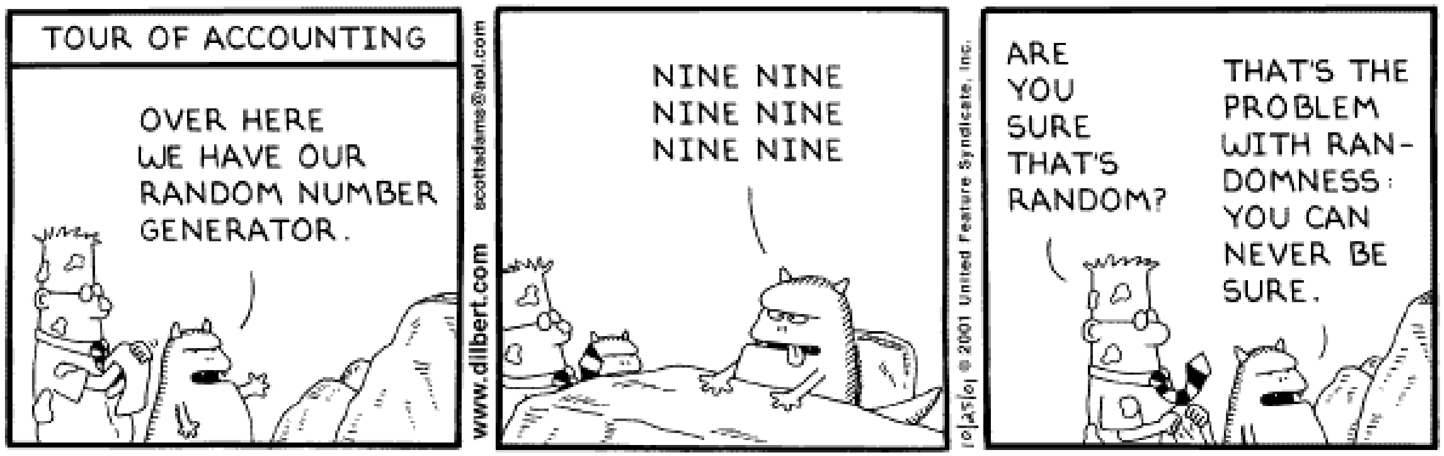
Entropy Sources

- Physical Sources
  - Noisy diodes
  - Ring oscillators
  - Radioactive sources
  - Quantum effects
  - Air turbulence
  - Audio / radio / CCD noise
- "Other" sources
  - Big complex system behavior
    - Scheduling patterns
    - Large scale clock jitter (i.e. sampling jitter)
    - Network packet arrival timing
    - Booting randomness
  - Human Sources (Typing / mouse activity)

Entropy Estimation

- A “Hard Problem”
  - Not **theoretically** possible to completely answer
- Entropy is always calculated with respect to an observer
- Best analysis provides two approaches
  - Entropy Estimate
    - Based on a statistical model
  - Entropy Bound Measurement
    - Based on statistical testing

To quote Scott Adams…



You know you're in trouble if there's a Dilbert cartoon!

Obvious Issues

- Understand your entropy source
  - Quantify your entropy (Thanks Netscape!)
  - Know your enemy (and where they are in the system)
- Don’t just use a TRNG
- Entropy source should be periodically tested
- Make sure an attacker can’t track your PRNG state
  - Seeds must have sufficient entropy to foil guessing
  - Reseed frequently, but accumulate entropy
- Use a good PRNG design
  - There aren’t many
  - SP800-90 is an excellent source for designs
  - Don’t deviate from good designs (thanks Microsoft!)
- Don’t allow your attacker to specify your seed values

Postmortem (Final)

- Know the schemes that you use
  - Some requirements are there for a reason
  - If you have a requirement, design for that requirement
- Be careful with fragile systems
  - Systems must be carefully designed to “cushion” the fragile design
- Use professional help
  - Use (well made) pre-implemented libraries
  - Use well understood protocols