Symmetric Encryption Security

Christian Grothoff

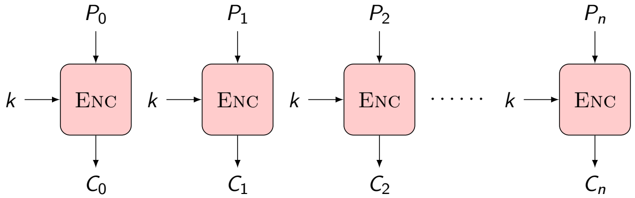
Berner Fachhochschule

3.5.2019

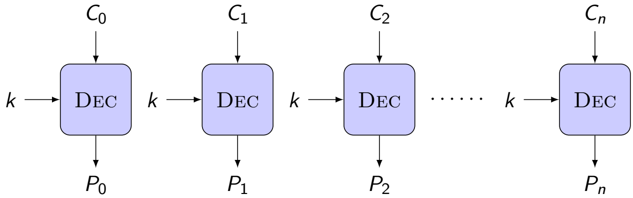
Learning Objectives

- Review: Cipher modes
- Security definitions: IND-CPA
- Beyond IND-CPA
- Case study: Insecurity of WEP
- Real-world use of cryptographic primitives (exercise)

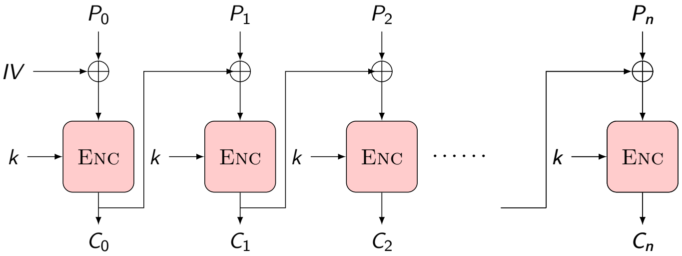
EBC encryption



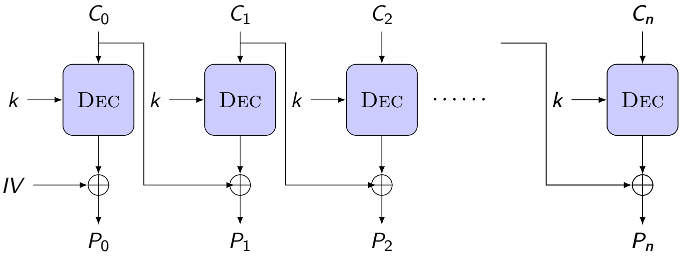
EBC decryption



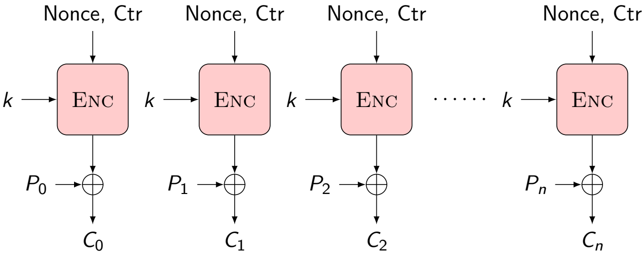
CBC encryption



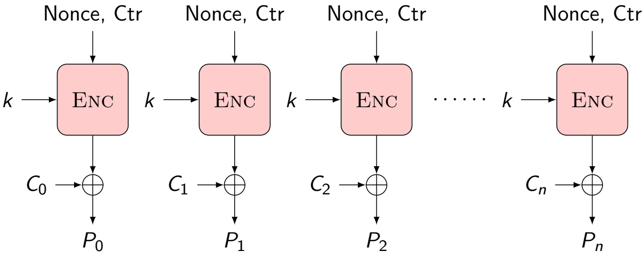
CBC decryption



CTR encryption



CTR decryption



Problem

Which mode is secure?

Problem

Which mode is secure?

How to prove it?

Security Definitions for Symmetric Encryption

Simplistic security definitions would be:

1. It must be impossible for an adversary to find the key from ciphertexts.
2. It must be impossible for an adversary to find the plaintext from a ciphertext.

Security Definitions for Symmetric Encryption

Simplistic security definitions would be:

1. It must be impossible for an adversary to find the key from ciphertexts.
2. It must be impossible for an adversary to find the plaintext from a ciphertext.

These are insufficient as, for example, they do not capture the insecurity of the ECB mode!

Problem

We need a precise, succinct and comprehensive security definition!

Subtle Corner Cases

Given n stocks, the message $m := m_1||m_2||m3||\ldots||m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Subtle Corner Cases

Given n stocks, the message $m := m_1||m_2||m3||\ldots||m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Even partial information leakage about a message is problematic.

Subtle Corner Cases

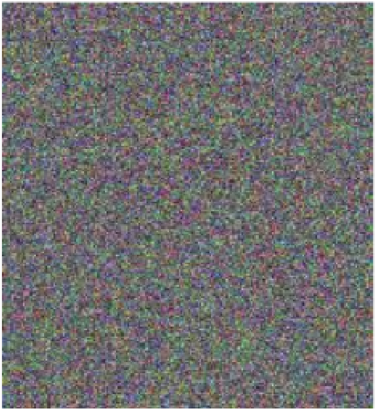
Given n stocks, the message $m := m_1||m_2||m3||\ldots||m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Even partial information leakage about a message is problematic.

In fact, even *probabilistic* leakage is a problem: an adversary that can tell that with probability of 90% whether you are buying or selling might be a problem!

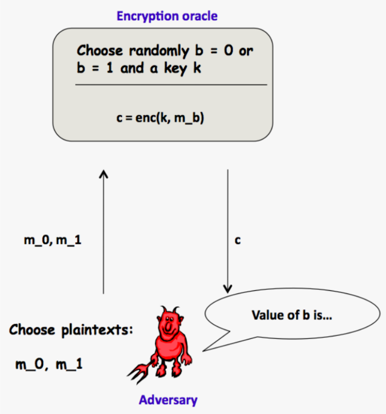
What we want

Our goal is to formalize the intuitive notion of secure encryption shown here:



The picture shows that an adversary does not learn any useful information about a plaintext from a ciphertext.

Indistinguishability under Chosen Plaintext Attacks (IND-CPA)



Indistinguishability under Chosen Plaintext Attacks (IND-CPA)

Security Game: Adversary chooses m_1 and m_2. Defender chooses key k and $b \in \{0, 1\}$. Defender computes $c := \texttt{enc}(k, m_b)$ and gives c to the adversary.

Definition: A symmetric encryption scheme `enc()` is *IND-CPA secure*, if it is impossible for all possible adversaries to tell whether $b = 0$ or $b = 1$. That is, the adversary wins if they can determine the correct b.

Problem

The above definition is incomplete: What if the adversary wins 60% of the time?

Cryptographic Games

An *oracle* is a party in a game that the adversary can call upon to indirectly access information that is otherwise hidden from it. **IND-CPA** can then be formalized like this:

Setup: Generate random key k, select $b \in \{0, 1\}$ for $i \in \{1, \ldots, q\}$.

Oracle: Given M_0 and M_1 (of same length), return $C := \texttt{enc}(k, M_b)$.

The adversary wins, if it can guess b with probability greater than $\frac{1}{2} + \epsilon(\kappa)$ where $\epsilon(\kappa)$ is a negligible function in the security parameter κ.

Restrictions on Oracle use

Many schemes break after an large number of messages. Thus, restrictions are generally imposed on the use of the Oracle by the adversary:

- Best known attack on AES uses birthday attack, 2^{64} queries
- $\Rightarrow$ limit oracle use to say 2^{30} queries of some maximum length, say 2^{13} (1 kB).

Then the resulting *advantage* of the adversary remains "small".

IND-CPA

IND-CPA is a widely accepted definition of secure symmetric encryption.

Practically relevant symmetric encryption schemes (i.e. AES in CTR or CBC mode) are considered IND-CPA secure.

Examples for IND-CPA Insecure Schemes

- Schemes where the plaintext can be recovered from the ciphertext ...
- Schemes where the key can be recovered from the ciphertext ...
- ECB mode encryption ...
- Schemes where the *n*-th plaintext bit can be recovered from ciphertext ...

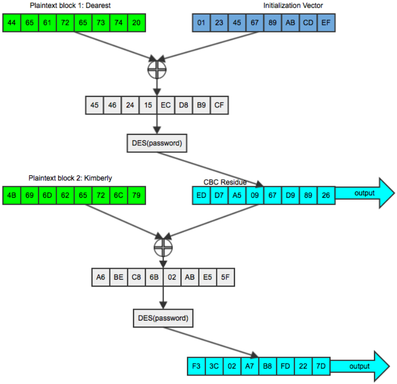
... are all IND-CPA insecure.

Examples for IND-CPA Insecure Schemes

- Any deterministic, stateless encryption scheme is insecure.
- CBC stateful IV mode[1] is IND-CPA insecure

Break

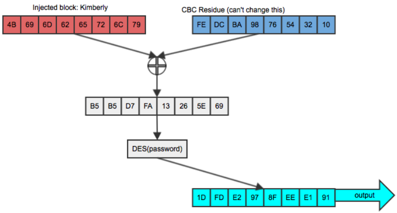
Attacking CBC stateful IV (1/5)[2]



Goal: confirm "Kimberly" was sent!

Attacking CBC stateful IV (2/5)

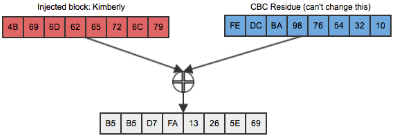
Setup: Get oracle to encrypt “Kimberly”:



Given random CBC residue, this does not help.

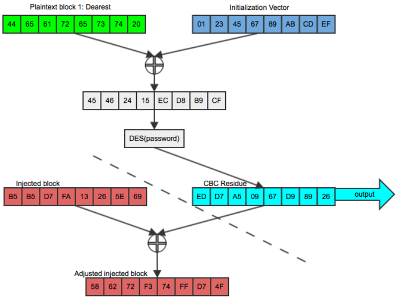
Attacking CBC stateful IV (3/5)

CBC residue is XORed with input, get rid of it first using *predicted* IV:



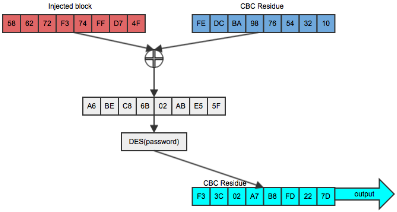
Attacking CBC stateful IV (4/5)

Then add the residue from the original encryption:



Attacking CBC stateful IV (5/5)

Now confirm the output matches:



If output matches, original text was "Kimberly".

Summary

For CBC, if an attacker can:

- guess the plaintext corresponding to any ciphertext block they have seen before, and
- can predict a future IV, and
- can submit a suitable message to be encrypted with that IV,

then they can verify their guess.

Is this attack an issue?

- Requires guessing the entire block
- Requires access to encryption oracle
- Block size is say 8 bytes, so 2^{256} trials

Is this attack an issue?

- Requires guessing the entire block
- Requires access to encryption oracle
- Block size is say 8 bytes, so 2^{256} trials

BEAST (2011) made this attack practical by shifting each unknown plaintext byte to a position in the block just after 7 bytes of known plaintext.

IND-CPA Secure Schemes

- The CTR random IV symmetric encryption scheme is IND-CPA secure.
- The CTR stateful IV encyption scheme (ensuring no IV re-use) is IND-CPA secure.
- The CBC *random* IV symmetric encryption scheme is IND-CPA secure.

Pseudo random functions (PRF)

- A *pseudo random function (PRF)* is a function that is (computationally) indistinguishable from a true random function
- The previous positive results are true under the *assumption* that the block cipher used (e.g. AES) is a PRF.
- Assumption really means that this is a commonly shared belief of the crypto community. No proof exists!
- Breaking any of these schemes thus means breaking the PRF property of the underlying block cipher.

The crucial security property of a secure block cipher is that it is a PRF!

Break

Chosen Ciphertext Attacks

IND-CPA vs. Chosen Ciphertext

IND-CPA is **not** the strongest security model!

- The adversary does not have access to a *decryption* oracle
- With a decryption oracle, an adversary can be allowed to ask for *some* messages of its choice to be decrypted.
- Security is achieved only if *other* messages still remain indistinguishable.

Indistinguishability under Chosen Ciphertext Attacks (IND-CCA)

The adversary's goal is the same as in IND-CPA (determine b given $\texttt{enc}(k, M_b^i)$) for sequences of messages $M_{0,1}^i$).

Setup Generate random key k, select $b \in \{0, 1\}$.

Oracle E Given M, return $C := \texttt{enc}(k, M)$.

Oracle D Given C', return $M := \texttt{dec}(k, C')$.

The additional restriction $C' \neq C$ must be imposed on the use of Oracle D: The adversary is not allowed to ask for decryption of a ciphertext C that was previously returned by the encryption oracle.

Examples for IND-CCA Insecure Schemes

- CTR schemes are IND-CCA insecure

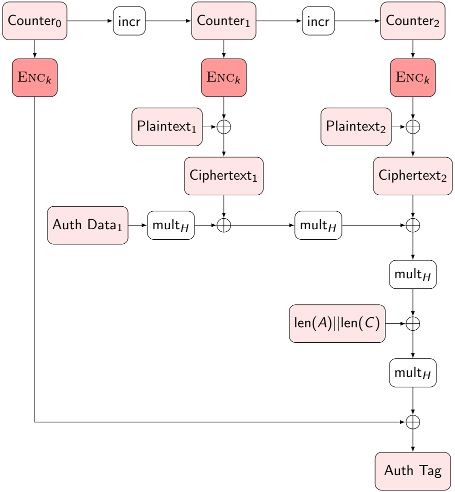
Problem

IND-CCA does not provide authenticity!

Real-world security

Schemes providing authenticated encryption are IND-CCA secure.[3]

GCM encryption



WEP Insecurity

Read the article "Intercepting Mobile Communications: The Insecurity of 802.11" until section 4.2. For each of the attacks, decryption (section 3), message modification (section 4.1) and message injection (section 4.2) explain:

- How does the attack work?
- Why does it work (i.e., what are the flaws that make the attack possible)?

Using encryption APIs

GNU libgcrypt is a C library offering a wide range of cryptographic primitives.

1. `# apt install libgcrypt20-dev`
2. `# apt install gcc gdb valgrind emacs`
3. Download source templates from course Git

Example: AES256 GCM (encrypt.c)

```
char key[256/8], iv[96/8];
char plaintext[] = "Hello world";
char ciphertext[sizeof (plaintext)];
gcry_cipher_hd_t cipher;

gcry_cipher_open (&cipher, GCRY_CIPHER_AES256,
          GCRY_CIPHER_MODE_GCM, 0);
gcry_cipher_setkey (cipher, key, sizeof (key));
gcry_cipher_setiv  (cipher, iv,  sizeof (iv));
gcry_cipher_encrypt (cipher,
       ciphertext, sizeof (ciphertext),
       plaintext,  sizeof (plaintext));
gcry_cipher_close (cipher);
```

Example: AES256 GCM (decrypt.c)

```
char key[256/8], iv[96/8];
char plaintext[1024];
char ciphertext[sizeof (plaintext)];
gcry_cipher_hd_t cipher;

size_t plen = read (STDIN_FILENO,
                    ciphertext, sizeof (ciphertext));
gcry_cipher_open (&cipher, GCRY_CIPHER_AES256,
          GCRY_CIPHER_MODE_GCM, 0);
gcry_cipher_setkey (cipher, key, sizeof (key));
gcry_cipher_setiv  (cipher, iv,  sizeof (iv));
gcry_cipher_decrypt (cipher,
       plaintext,  plen,
       ciphertext, plen);
gcry_cipher_close (cipher);
```

Handling partial reads (decrypt.c)

```
char plaintext[1024];
size_t plen = 0;

while (1) {
  ssize_t inlen = read (STDIN_FILENO,
                        &ciphertext[plen],
                        sizeof (ciphertext) - plen);
  if (-1 == inlen) {
    fprintf (stderr,
             "Failed to read input\n");
    return 1;
  }
  if (0 == inlen)
    break;
  plen += inlen;
}
```

Tasks (1/3)

- Use the provided `encrypt` and `decrypt` programs to encrypt "Hello world" text using AES256+GCM and then decrypt it.
- Study the `libgcrypt` documentation. Use it to switch the program to use AES256+CBC instead.
- Switch back to AES256+GCM. Extend the program to obtain, transmit and verify the authentication tag.
- Extend the program to authenticate additional plaintext data that is not at all encrypted.

Tasks (2/3)

- Write a new program `hash.c` to compute the SHA-256 hash of the data read from `stdin`. Output the result in HEX and compare to `sha256sum`.
- Modify your program to use `SHA-512` instead.
- Write a new program `kdf.c` to compute the SCRYPT key derivation function. Output the result in HEX.

Tasks (3/3)

- Modify your programs to perform 10000 iterations each time before generating any output.
- Measure the time the various operations take.
- Modify your programs to process 1 MB of input instead of the 11 bytes of “Hello world”.
- Again, measure the time the various operations take.
- Change the IV length from 96 bytes to 128 bytes for AES256+GCM and measure again.