CSE 40567 / 60567: Computer Security

Cryptography 4
Homework #2 has been released. It is due Thursday, Feb. 7th at 11:59PM

See Assignments Page on the course website for details
Office Hours Change for Thursday (2/7)

Office Hours will be held from 10am-12pm
Block Ciphers

Some encryption algorithms divide a message into a sequence of parts, or blocks, and encipher each block with the same key.

Simplest mode of operation (ECB):
Block Ciphers

Definition: Let $E$ be an encryption algorithm, and let $E_k(X)$ be the encryption of message $X$ with key $k$. Let a message $X = x_1x_2\ldots$, where each $x_i$ is of a fixed length. Then a block cipher is a cipher for which $E_k(X) = E_k(x_1)E_k(x_2)\ldots$.

Example: AES is a block cipher. It breaks the message into 128-bit blocks and uses the same 128-, 192- or 256-bit key to encipher each block.
The ECB Penguin

Image Credit: https://blog.filippo.io/the-ecb-penguin/
Cipher Block Chaining (CBC)

ECB should not be used if encrypting more than one block of data with the same key (advice: just avoid it)

Alternative: make each ciphertext block dependent on all blocks processed up until that point
Initialization Vector Pitfall

- Predictable IVs for each transaction
  - CBC mode: enables online attacks with chosen-plaintext

Example: Alice’s medical record for a specific condition

1. Assume Mallory can predict IVs: $IV_M$ and $IV_A$
2. Mallory’s chosen plaintext: $X_M = IV_M \oplus IV_A \oplus \text{“false”}$
3. Application encrypts: $\{X_M\} = E_k(IV_M \oplus X_M) = E_k(IV_M \oplus (IV_M \oplus IV_A \oplus \text{“false”}))$
   
   $\{X_M\} = E_k(IV_A \oplus \text{“false”})$

4. Mallory compares ciphertexts: $\{X_M\} = \{X_A\}$?
Choosing Initialization Vectors

Always use a random IV for each transaction

The IV can be made public after encryption. Why is this secure?

The IV is only used to ensure that the same plaintext encrypts to different ciphertexts. After it is used, there is no harm in releasing it, as it leaks no information about the plaintext.
Advanced Encryption Standard (AES)

• If you need a symmetric key algorithm, this is the one to use

• Based on Rijndael, winner of the 2001 NIST AES competition

• Key sizes: 128-, 192- or 256-bit

• Block size: 128-bit

• Rounds: 10, 12 or 14 (depending on key size)
AES is a substitution-permutation network

- Works via a combination of both substitution and permutation operations
  - Fast in both hardware and software
- Operates on 4x4 column-major order matrix of bytes (state)

\[
\begin{bmatrix}
  b_0 & b_4 & b_8 & b_{12} \\
  b_1 & b_5 & b_9 & b_{13} \\
  b_2 & b_6 & b_{10} & b_{14} \\
  b_3 & b_7 & b_{11} & b_{15}
\end{bmatrix}
\]
High-level overview of AES

1. **KeyExpansions** — round keys are derived from $k$ using a key schedule

2. **InitialRound**
   1. AddRoundKey

3. **Rounds**
   1. SubBytes
   2. ShiftRows
   3. MixColumns
   4. AddRoundKey

4. **Final Round (no MixColumns)**
   1. SubBytes
   2. ShiftRows
   3. AddRoundKey
SubBytes Step

Each byte in the state is replaced with its entry in a fixed 8-bit lookup table (a substitution box), $S$: $b_{i,j} = S(a_{i,j})$. 
ShiftRows Step

Bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.
MixColumns Step

Each column of the state is multiplied with a fixed polynomial $c(x)$. 
AddRoundKey Step

Each byte of the state is XORed with a byte of the round subkey.
AES Compatible Stream Modes

Cipher Feedback (CFB) Mode, a variation of CBC that turns a block cipher into a self-synchronizing stream cipher

Downside: Slow
AES Compatible Stream Modes

Output Feedback (OFB) Mode also turns a block cipher into a self-synchronizing stream cipher.

Downside: Slow
AES Compatible Stream Modes

Counter (CTR) Mode generates the next keystream block by encrypting successive values of a counter combined with a nonce (IV)

*Use this mode when a stream cipher is needed*
AES functions in LibreSSL / OpenSSL

#include <openssl/aes.h>

int AES_set_encrypt_key(const unsigned char *userKey, const int bits, AES_KEY *key);
int AES_set_decrypt_key(const unsigned char *userKey, const int bits, AES_KEY *key);

int private_AES_set_encrypt_key(const unsigned char *userKey, const int bits, AES_KEY *key);
int private_AES_set_decrypt_key(const unsigned char *userKey, const int bits, AES_KEY *key);

void AES_cbc_encrypt(const unsigned char *in, unsigned char *out, size_t length, const AES_KEY *key, unsigned char *ivec, const int enc);

void AES_ctr128_encrypt(const unsigned char *in, unsigned char *out, size_t length, const AES_KEY *key, unsigned char ivec[AES_BLOCK_SIZE], unsigned char ecount_buf[AES_BLOCK_SIZE], unsigned int *num);
Public Key Cryptography

- A symmetric algorithm is like a safe
  - The key is the combo
  - Anyone with the combo can open the safe
  - Anyone without the combo must learn safecracking

1976: Whitfield Diffie and Martin Hellman introduce alternative paradigm with two keys:

\[ k_P \] Public (sharable)

\[ k_S \] Private (secret)
Sending a message using public key crypto

1. Alice and Bob agree on a public key crypto system

2. Bob sends Alice his public key

\[ k_{P,B} \]
Sending a message using public key crypto

3. Alice encrypts her message using Bob’s public key and sends it back to Bob

\[ X, k_{P,B} \rightarrow \{X\}_{k_{P,B}} \]

4. Bob decrypts Alice’s message using his private key

\[ \{X\}_{k_{P,B}, k_{S,B}} \rightarrow X \]
Sending a message using public-key crypto and a central key repository

• The protocol we just saw is clunky: Alice needs to contact Bob before sending him a message

• If public keys are stored in an accessible database, the protocol is simplified to three steps:

1. Alice gets Bob’s key from the database

\[ k_{P,B} \]

\[ B \]
Sending a message using public-key crypto and a central key repository

2. Alice encrypts her message using Bob’s public key and sends it back to Bob

\[ X, k_{P,B} \rightarrow \{X\}_{k_{P,B}} \]

3. Bob decrypts Alice’s message using his private key

\[ \{X\}_{k_{P,B}}, k_{S,B} \rightarrow X \]

Bob isn’t involved until he reads his message
Practical public key crypto application: email encryption via GPG

- Free implementation of OpenPGP standard (RFC4880)
- Public key encryption and signing of data and communications
- Versatile key management system

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.9 (GNU/Linux)

mQg:BEK9勿8R8AC9h4q4ueWr8skKk8DTP+XKvZW6HSTYFkfxjAULITyZpGBajYZuX0y/t7THAkqgS/eSX9VErNFriUIkXho8ED/F0x7tq5tBQv22i/ehoakeww9RNYy6e yUSYrCPh4Ktsz02LXIAOSHFEmk18ow6oHT8PKBNzoz2Yyo/nFv7z67GlwcghzKt muJ6dd70N2vpGbMBEI/0JsEAly7dpB99IMXzaukLo9cmKmL2P75K8ysQZUvc2L 7X7dI4J0TWYKi9Sj0XjQNA7CJbvv10Uk3JwLsH/VzX+MD3P91BDEqAwu4mLGI kxYMU/rP0juC7arPDMl47+N/sS/2qH021lkf2Su2kry57ikczkt5czLztWFC9I fKwF3IC1b2HAD1s8An19dL8HfNfaqJ12rowfw+gOktiYEEBEECAAYFAkZ3/9oA CgqQlW0fayU+w00050CqCpoeF3nPPhcwK+ft0V9UnvRAlAOsK50k0kgf3 Eph34 LKYpe9sI6mP8CBJ0ozvN0jUz94Vux8tV8eygE0kL1bSYodDSHTKq+wKXXITdgy+/kmj1L9Sy8vficH0AfHhfVJyx80dEXIzBnFx8r8E822h9Nqt5M9y90By MfF0f77YvLky/yYocTwQA7B4AyFyLe7v8SsEJC7xO4zTt2/9/k1SFI3mgAsJKe 43B2i1nmC5gqVQh3Z/HRNnU1pC8whp/u2z8s8767B8yF59mX4q5q160Ar33g3IF XEHwLh3B8eAQaAJgBQClq/W5AsMAAoJEAMKQZT2AUuWYMa2NBMHIOJMcnj80o F1gyxGXB5lCAJ4zcU7/4RbQuP+UV/hPf20L7SeQ0A==
=q2SM
-----END PGP PUBLIC KEY BLOCK-----

Image Credit: http://www.oekonux-conference.org/documentation/texts/Meyer.html

# apt-get install gnupg
How does public key crypto work?

- How do we generate two keys that work together?
- How do we make sure the public key doesn’t reveal any information about the private key?
- How can we design the algorithm to be resilient to chosen plaintext attacks: an attacker can choose any message to encrypt?
Trapdoor Function

$f$: easy

$\text{Domain}$

$\text{Range}$

$f^{-1}$: hard

$f_t^{-1}$: easy with trapdoor $t$
RSA

• First full-fledged public key encryption algorithm
  - 1978: Ron Rivest, Adi Shamir, and Leonard Adleman

• If you need a public key encryption algorithm, this is one to use

• Simple to understand and implement
  - Security comes from the difficulty of factoring large numbers


RSA Key Generation

1. Choose two random large prime numbers of equal length, \( p \) and \( q \).

2. Compute the product (modulus): \( n = p \cdot q \)

3. Randomly choose the encryption key \( e \) such that \( e \) and \( (p - 1)(q - 1) \) are relatively prime

4. Use the extended Euclidean algorithm to compute the decryption key, \( d \), such that:

\[
ed \equiv 1 \mod (p - 1)(q - 1)
\]

\[
d = e^{-1} \mod ((p - 1)(q - 1))
\]
RSA Key Generation

- $d$ and $n$ are also relatively prime
- The numbers $e$ and $n$ are the public key $k_P$
- The number $d$ is the private key $k_S$
- The two primes, $p$ and $q$, are no longer needed. They must be discarded and never revealed.
RSA Encryption

1. Divide a message $X$ into numerical blocks $x_i$ that are smaller than $n$ (with binary data, choose the largest power of 2 less than $n$).

   The encrypted messages, $\{X\}$, will be made up of similarly sized message blocks $\{x_i\}$, of the same length.

2. Apply the encryption formula:

   $$\{x_i\} = x_i^e \mod n$$
RSA Decryption

1. Take each encrypted block \( \{x_i\} \) and apply the decryption formula:

\[
x_i = \{x_i\}^d \mod n
\]

The message could have also been encrypted with \( d \) and decrypted with \( e \). The use of the keys is arbitrary!