Confidentiality

CS 161: Computer Security
Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

February 23, 2017
Review of Where We’re At

• Alice employs an Encryptor $E$ to produce ciphertext from plaintext.

• Bob employs a Decryptor $D$ to recover plaintext from ciphertext.

• So far, both $E$ and $D$ are configured using the same key $K$.

• $K$ is a shared secret between Alice and Bob
  – Eavesdropper Eve doesn’t know it (otherwise, disaster!)

• Use of same secret key for $E$ and $D \Rightarrow \text{“symmetric-key cryptography”}
Block cipher

A function $E : \{0, 1\}^b \times \{0, 1\}^k \rightarrow \{0, 1\}^b$. Once we fix the key $K$ (of size $k$ bits), we get:

$E_K : \{0, 1\}^b \rightarrow \{0, 1\}^b$ denoted by $E_K(M) = E(M,K)$.

(and also $D(C,K)$, $E(M,K)$’s inverse)

- **Three properties:**
  - **Correctness:**
    - $E_K(M)$ is a permutation (bijective function) on $b$-bit strings
    - Bijective $\Rightarrow$ invertible
  - **Efficiency**: computable in $\mu$sec’s
  - **Security:**
    - For unknown $K$, “behaves” like a random permutation

- **Provides a building block** for more extensive encryption
DES (Data Encryption Standard)

- Designed in late 1970s
- Block size 64 bits, key size 56 bits
- NSA influenced two facets of its design
  - Altered some subtle internal workings in a mysterious way
  - Reduced key size 64 bits $\Rightarrow$ 56 bits
    - Made brute-forcing feasible for attacker with massive (for the time) computational resources
- Remains essentially unbroken 40 years later!
  - The NSA’s tweaking hardened it against an attack “invented” a decade later
- However, modern computer speeds make it completely unsafe due to small key size
Today’s Go-To Block Cipher: AES (Advanced Encryption Standard)

- 20 years old
- Block size 128 bits
- Key can be 128, 192, or 256 bits
  - 128 remains quite safe; sometimes termed “AES-128”
- As usual, includes encryptor and (closely-related) decryptor
- How it works is beyond scope of this class
- Not proven secure
  - but no known flaws
  - so we assume it is a secure block cipher
How Hard Is It To Brute-Force 128-bit Key?

- $2^{128}$ possibilities – well, how many is that?
- Handy approximation: $2^{10} \approx 10^3$
- $2^{128} = 2^{10 \times 12.8} \approx (10^3)^{12.8} \approx (10^3)^{13} \approx 10^{39}$
- Say we build massive hardware that can try $10^9$ keys in 1 nsec
  - So $10^{18}$ keys/sec
  - Thus, we’ll need $\approx 10^{21}$ sec
- How long is that?
  - One year $\approx 3 \times 10^7$ sec
  - So need $\approx 3 \times 10^{13}$ years $\approx 30$ trillion years
Issues When Using the Building Block

• Block ciphers can only encrypt messages of a certain size
  – If M is smaller, easy, just pad it (details omitted)
  – If M is larger, can repeatedly apply block cipher
    • Particular method = a “block cipher mode”
    • Tricky to get this right!

• If same data is encrypted twice, attacker knows it is the same
  – Solution: incorporate a varying, known quantity (IV = “initialization vector”)
Electronics Code Book (ECB) mode

- Simplest block cipher mode
- Split message into b-bit blocks $P_1, P_2, \ldots$
- Each block is enciphered independently, separate from the other blocks
  \[ C_i = E(P_i, K) \]
- Since key $K$ is fixed, each block is subject to the same permutation
  - (As though we had a “code book” to map each possible input value to its designated output)
Encryption

Electronic Codebook (ECB) mode encryption
Decryption

Problem: Relationships between $P_i$’s reflected in $C_i$’s
Original image, RGB values split into a bunch of b-bit blocks
Encrypted with ECB and interpreting ciphertext directly as RGB
Later (identical) message again encrypted with ECB
Building a Better Cipher Block Mode

1. Ensure blocks incorporate more than just the plaintext to mask relationships between blocks. Done carefully, *either* of these works:
   - Idea #1: include elements of prior computation
   - Idea #2: include positional information

2. Plus: need some *initial randomness*
   - Prevent encryption scheme from determinism revealing relationships between messages
   - Introduce initialization vector (IV)

- Example: Cipher Block Chaining (CBC)
CBC: Encryption

\[ E(\text{Plaintext}, K) : \]

- If \( b \) is the block size of the block cipher, split the plaintext in blocks of size \( b \): \( P_1, P_2, P_3, \ldots \)
- Choose a random IV (do not reuse for other messages)
- Now compute:

\[ \text{Final ciphertext is } (\text{IV}, C_1, C_2, C_3). \text{ This is what Eve sees.} \]
CBC: Decryption

D(Ciphertext, K):

• Take IV out of the ciphertext
• If $b$ is the block size of the block cipher, split the ciphertext in blocks of size $b$: $C_1, C_2, C_3, \ldots$
• Now compute this:

Output the plaintext as the concatenation of $P_1, P_2, P_3, \ldots$
Original image, RGB values split into a bunch of b-bit blocks
Encrypted with CBC
CBC

Widely used

Issue: sequential encryption, hard to parallelize

Parallelizable alternative: CTR mode

Security: If no reuse of nonce, both are provably secure
(assuming underlying block cipher is secure)
CTR: Encryption

(Nonce = Same as IV)

Important that nonce/IV does not repeat across different encryptions.
Choose at random!
CTR: Decryption

Note, CTR decryption uses block cipher’s encryption, not decryption.
Modern Symmetric-Key Encryption:

Stream Ciphers
Stream ciphers

- Block cipher: fixed-size, **stateless**, requires “modes” to securely process longer messages
- Stream cipher: **keeps state** from processing past message elements, can continually process new elements
- Common approach: “one-time pad on the cheap”:
  - XORs the plaintext with some “random” bits
- But: random bits ≠ the key (as in one-time pad)
  - Instead: output from **cryptographically strong pseudorandom number generator** (PRNG)
Pseudorandom Number Generators (PRNGs)

- Given a seed, outputs sequence of seemingly random bits. (Keeps internal state.)
  \[ \text{PRNG(} \text{seed} \text{)} \Rightarrow \text{“random” bits} \]
- Can output arbitrarily many random bits
- Can a PRNG be truly random?
  - No. For seed length \( s \), it can only generate at most \( 2^s \) distinct possible sequences.
- A cryptographically strong PRNG “looks” truly random to an attacker
  - attacker cannot distinguish it from a random sequence
Building Stream Ciphers

Encryption, given key $K$ and message $M$:
- Choose a random value $IV$
- $E(M, K) = PRNG(K, IV) \oplus M$

Decryption, given key $K$, ciphertext $C$, and initialization vector $IV$:
- $D(C, K) = PRNG(K, IV) \oplus C$

Can encrypt message of any length because PRNG can produce any number of random bits
Using a PRNG to Build a Stream Cipher

Using a Pseudorandom Number Generator (PRNG) to generate a keystream for each message of plaintext.

- **Alice**
  - (Small) K, IV
  - PRNG
  - Keystream
  - $M_i$: $i^{th}$ message of *plaintext*

- **Bob**
  - (Small) K, IV
  - PRNG
  - Keystream

$C_i = M_i \oplus K$
Okay, but how do we build a Cryptographically Strong PRNG?

• Here’s a simple design for a PRNG that generates 128-bit pseudo-random numbers
  – Only state needed is SEED and N (number of calls so far)

• PRNG(SEED) = { return AES-128_{SEED}(++N) }
  – i.e., encrypt counter of # of calls using SEED as key
  – Because AES-128 acts like a random permutation of 128-bit bitstrings, even a tiny change in input such as N vs. N+1 completely and unpredictably changes output
Building a Cryptographically Strong PRNG, con’t

• Here’s a version that incorporates an IV
  – Only state needed is SEED and N (# of calls so far), plus an IV

• PRNG(SEED, IV)
  = \{ \text{return AES-128}_{SEED}(++N \oplus IV) \}
  – i.e., encrypt (counter of # of calls, XOR’d with IV) using SEED as key

• In fact, let’s compare using this PRNG to build a stream cipher with the block cipher “CTR” mode …
Using a PRNG to Build a Stream Cipher

\[ M_i \oplus \text{Keystream} \]

\[ \text{Keystream} \rightarrow \text{Ciphertext} \]

\[ \text{M}_i: \text{ith message of plaintext} \]
Only difference from our stream cipher built on AES-128 is use of a different operator (concatenation vs. XOR) to combine IV and counter. Both are equally secure as long as IV is random.
Eve

"Symmetric-key encryption"

Alice

Bob

M_i?

E(M_i, K) and D(C_i, K) are inverses for the same K

M_i: i^{th} message of plaintext

C_i: i^{th} message of ciphertext

D(C_i, K)

E(M_i, K)
Eve

"Asymmetric-key encryption"

M_i, ?

Alice

K_E

E(M_i, K_E)

C_i: i_th message of ciphertext

Bob

K_D

D(C_i, K_D)

M_i

E(M_i, K_E) and D(C_i, K_D) are inverses for particular K_E and K_D

M_i: i_th message of plaintext

ciphertext
E(Mᵢ, Kₐ) and D(Cᵢ, Kₑ) are inverses for particular Kₑ and Kₑ.

"Asymmetric-key encryption"
Public Key Cryptography

• Having two keys rather than one seems like a step backwards …

• ... However, what if knowing $K_E$ (and $E$ and $D$) doesn’t allow Eve to infer $K_D$?

• If Bob can generate a pair $\langle K_E, K_D \rangle$ that have this property for $E$ and $D$, then Bob can just publish $K_E$ for the world to see
  – No need to pre-exchange keys with Alice!
Eycle:

\[ E(M_i, K_E) \]

C_i: i^{th} message of ciphertext

\[ D(C_i, K_D) \]

M_i: i^{th} message of plaintext

E(M_i, K_E) and D(C_i, K_D) are inverses for particular K_E and K_D

“Public-key encryption”
Public Key Cryptography, con’t

• For Eve, encryption function $E_K(M_i)$ is now fully determined! Surely she can invert it … ?

• $E_K$ needs to be a one-way function, such that computing $E_K^{-1}(x)$ is computationally intractable …

• ... Unless you have some additional knowledge
  – i.e., $K_D$

• Where can we get such a seemingly magic pair of functions $E$ along with $D = E_K^{-1}(x)$?
  – Let’s look at one such public-key approach: RSA
Number Theory Refresher: Efficient Multiplication/Exponentiation

• If ‘a’ and ‘b’ have N bits each:

  Can multiply them in $O(N^2)$ time
  (actually, a bit faster)

  Can exponentiate modulo p
  ($a^b \mod p$ or $b^a \mod p$) in $O(N^3)$ time

• We’re going to care about BIG integers ($N \approx 1000$)
Number Theory Refresher: 
**Totients**

- \( \varphi(n) = \text{totient of } n \)
  
  \[ \text{= \# of } i, \ 0 < i < n: \ i \text{ and } n \text{ are relatively prime} \]

- \( \varphi(p) = p-1 \) if \( p \) is a prime

- \( \varphi(p \cdot q) = (p-1)(q-1) \) if \( p, q \) are distinct primes

- Euler’s theorem:

  Given ‘a’ relatively prime to \( n \), \( a^{\varphi(n)} = 1 \mod n \)
Finding BIG Primes Quickly

Here’s a probabilistic algorithm:

1. Generate a random candidate prime p'
2. Generate random integer a: 1 < a < p' - 1
3. Compute $a^{(p'-1)} \mod p'$. If $\neq 1$, discard p', go to 1
4. Otherwise, go to 2, unless have made enough iterations to have confidence p' “surely” must be prime
   - Enough iterations: while $\exists$ non-primes for which the equation in Euler’s theorem almost always holds, they’re exceedingly rare

• Runs in $O(N^4)$ time for finding an N-bit prime
Putting it all together: RSA

1. Generate random primes p, q
2. Compute $n = p \cdot q$
3. Compute $\varphi(n) = (p-1)(q-1)$
   Important: if Eve sees n, she can’t deduce $\varphi(n)$
   unless she can factor n into p and q
4. Choose $2 < e < \varphi(n)$, where e and $\varphi(n)$ are relatively prime
   Could be something simple like e=3, if rel. prime.
5. Public key $K_E = \{ n, e \}$. Both are Well Known.
6. Compute $d = e^{-1} \mod \varphi(n)$
   d is multiplicative inverse of e, modulo $\varphi(n)$
   easy to find if you know $\varphi(n)$
   (believed) HARD to compute if you don’t know p, q
7. Private key $K_D = \{ d \}$