Symmetric Encryption: Building Blocks
With material from Dave Levin, Jon Katz, David Brumley
Symmetric crypto

Recall:

- $k = k_e = k_d$
- Everyone who knows $k$ knows the whole secret
Perfect secrecy (Shannon)

a.k.a. Information Theoretic Secrecy

Recall:

Alice

Public channel

Bob

Eve

1

Guess: $m_1$

Eve

2

Guess: $m_2$

Goal: Eve2 is no better off than Eve1

$(Pr[m=m_1] = Pr[m=m_2])$
Random functions
• Terminology note:

• Capital letters \((X,Y,F)\) = sets of things

• Lowercase letters \((x,y,f)\) = individual things in set
Concept model: $f(x)$

- $f(x)$ maps inputs $X$ to outputs $Y$ ($X = Y$ or $X \neq Y$)
- For reasonably-sized $X$ and $Y$, LOTS of possibilities!

\[ F = \text{set of all such possible functions } f(x) \]
Draw \( f \) from \( F \) at random

- \( \Pr[f(x) = y] = 1/|Y| \)

- If \( f(3) = 8 \), then what is \( f(4) \)?
  - We don’t know! And there is no way to predict (unless you know \( f \)). \textit{True for all values} \( x \).

- Given that \( f(x) = 7 \), what is \( x \)?
  - Can’t find out without brute-forcing.
    - How long will this take?
  - This is called a \textit{one-way function}
Why do we care?

*Because one-way functions provide confidentiality!*
• If Alice writes $f(x)$ instead of $x$ to a file, no one can recover the plaintext without brute-forcing.

• Including Alice! (This is a problem.)
(Efficiently) Recovering $x$

- If everyone can invert $f$, no confidentiality
- Instead, we want a **one-way trapdoor function**:  
  - $F(k,x) = y$
  - If you know $y$ and $k$, you can recover $x$ easily
  - If you don’t know $k$, you must brute-force

This is starting to resemble our cryptosystem model!
This is all imaginary

• Storing all the possible fs in F would be hard

• No true one-way trapdoor has been found
  • Unclear whether it’s possible

• Instead: **Approximate** this with **pseudo-random functions** (PRF)
  • These are really hard to create correctly!
Pseudo-Random Functions
Pseudo-random function family (PRF)

- **F**: *family* of functions $f(x)$
  - All have the same domain and range $X, Y$
- **Randomly** choose one function $f_k(x)$
  - Recall, $k$ is our trapdoor
    - $k$ is which function in the family we chose!
- Cannot distinguish between a true random function, and a randomly chosen function in $F$
  - Family is public!
Eve's job: Provide $x$. Figure out which world we are in. With very high probability, she can’t do better than random guessing.
• Note — if attacker is wrong most of the time (rather than half the time), that indicates *insecurity*.

• She should switch guesses
Block ciphers
Block cipher basics

• Start with a PRF
  • That operates on fixed-length blocks: input size == output size
  • Each function is a permutation (it’s invertible)
  • Each function, inverse is efficiently computable

• Key length: related to how many functions there are

• Block length: size of input/output block
Security goal

- Every $f_k$ in $F$ has the same range
  - Every output (ciphertext) belongs to some input
  - But the permutation is different for each $k$

- Goal: If you don’t know $k$, you cannot distinguish!
  - $k$ is the only secret!
Example block cipher

- Let block length == key length
- $E_k(m) = k \oplus x = c; \quad D_k(c) = k \oplus c = x$
Beyond the block

- Block ciphers operate on a (small) fixed size block
  - AES = 128 bits
  - This is not enough for real applications
- Instead: Break input up into blocks
  - Strategy for doing this = encryption mode
  - Different modes = different security, performance
Block cipher modes
Electronic code book (ECB)

Plain: \[ \begin{array}{cccccc}
    m_1 & m_2 & m_3 & m_4 & \ldots & m_n \\
\end{array} \]

E(k,m_i): \[ \downarrow \downarrow \downarrow \downarrow \downarrow \]

Cipher: \[ \begin{array}{cccccc}
    c_1 & c_2 & c_3 & c_4 & \ldots & c_n \\
\end{array} \]

- What is the problem here?
CPA revisited

Uh oh.
• CPA-resistance is **mandatory**

• Deterministic schemes **cannot be CPA-secure**
  
  • Nor are they secure to send multiple messages

• Moral: *Always use randomized encryption!*
  
  • Which builds in a varying value per message

  • Never use ECB mode!
Randomizing block ciphers

- \( r \) is an **Initialization Vector (IV)**
- \( c_i = E(k, m \text{ XOR } r_i) \); \( m = r_i \text{ XOR } D(k, c_i) \)
- \( r \) can be *random* or *unique* (see next slides)
Random IV

- A new string chosen at random per message
- Must send r along with c; use more bandwidth
Unique IV

• Should not repeat

• New IV for every \textit{message}

• Both sender and receiver can predict which IV is used for the next message; must remain synch’d.
  • Naive counter (not very secure, more later)
    • $IV_{i+1} = IV_i + 1$
  • Better: Calculate from previous: $IV_{i+1} = E(k, IV_i)$

• Less bandwidth but requires \textit{in-order} delivery
• How to generate randomness at each block?
• Lucky for us, E provides built-in randomness
Cipher-block chaining (CBC)

Decryption formula:
\[ m_i = D(k, c_i) \text{ XOR } c_{i-1} \]
CBC continued

- Solves the randomization/CPA problem
- Encryption is not parallelizable anymore
  - But decryption still is
- Other similar approaches: PCBC, OFB, CFB
  - XOR in more/different stuff
  - Different performance characteristics
Avoid synching IV

First block of output is gibberish, but we don’t care!
Attacking predictable IV

• Assume:
  • Options for m are known (e.g., \{yes, no\})
  • Last IV (for \(m_n\)) is known
  • Next IV is predictable
  • CPA attack

• Choose \(m_{n+1} = \text{“yes” XOR IV}_n \text{ XOR IV}_{n+1}\)
  • If \(c_{n+1} == c_n\) then \(m_n\) was “yes” … Why?
Stream cipher

- Inspired by the one time pad
- Generate ongoing series of bits
  - XOR each bit with next bit of $m$
- Synchronous: Sender, receiver use same bitstream
  - Requires synchronizing where to start, no drops
- Musts: Good randomness, long period
- Several popular ones but partially busted
  - Especially used for GSM
Output feedback (OFB)

• Turns a block cipher into a stream cipher
  • Generate **key stream** which is XORed with data

• Can generate all keys ahead, encrypt in parallel

Used improperly, can cycle too fast!
Counter mode (CTR)

\[ \text{Encrypt: } \text{IV} \rightarrow E_{k} \rightarrow \text{Ciphertext} \]

\[ m_{i} = D(k, \text{IV} + i) \oplus c_{i} \]

Decrypt? \[ m_{i} = D(k, \text{IV} + i) \oplus c_{i} \]
More on counter mode

• Also converts block to stream cipher
  • Like OFB, encrypt in parallel

• Generate keystream as a sequence
  • Sequence guaranteed not to repeat for a while

• Possible attacks:
  • If initial IV is not random and not transformed or concatenated
  • When used properly, very secure
• This can cause trouble if you naively increment the IV every time between messages.

• For message A, the input to the block cipher would be \( IV_A, IV_A + 1, IV_A + 2, \) etc.

• Then if for message B, you use \( IV_B = IV_A + 1 \)
  • Input to blocks will be equivalent to \( IV_A + 1, IV_A + 2, \) etc.

• In effect you are reusing inputs to the block cipher in a predictable way, which is bad.
Block cipher padding

• If your message doesn’t divide evenly into blocks, then what?
  • Doesn’t apply in streaming mode (e.g. CTR)
  • Pad with extra bytes in some known pattern
• Susceptible to padding oracle attacks
  • Brute-force one byte at a time (from end) in $m_{n-1}$
  • Because of XOR, affects padding at the end of $m_n$
  • If you guessed right, padding will be valid

Lesson: Don’t return informative errors
• Cool walkthrough of padding oracle attack:

• http://robertheaton.com/2013/07/29/padding-oracle-attack/
Common block ciphers
Data Encryption Standard (DES)

- Developed at IBM/NSA in 1970s (non-public process)
- 56-bit key, 64-bit block length

Concerns:
- Short key length — can be brute-forced in days
- Short block length — repeat blocks too often
Triple DES

- Triple the key length: \( k = (k_1, k_2, k_3) \)
  - Still short block length
  - How much does brute-force increase by?

- New block cipher:
  - \( E_3(k, m) = E(k_1, D(k_2, E(k_3, m))) \)
  - One version: \( k_1 = k_3 \)
    - Effective key length = 112

- Fairly slow, but used in practice (back compatible)
Advanced Encryption Standard (AES)

- Public contest at NIST, 1997
  - 15 candidates, winner selected in 2000
  - Lots of analysis of each candidate
- Efficiency + security considered
  - “Most secure” didn’t win!
- Supports keys: 128/192/256 bits
  - Nanoseconds since big bang: \( \sim 2^{90} \)
Summing up (so far)

• Symmetric crypto is very fast in practice
  • Especially stream ciphers

• If \textit{used properly} it can be very secure

• Next time:
  • Message authentication
  • Key exchange