Symmetric key Crypto I
and public key Crypto

CSCI 2070
Symmetric Key Crypto

- **Stream cipher** — like a one-time pad
  - Except that key is relatively short
  - Key is stretched into a long **keystream**
  - Keystream is used just like a one-time pad

- **Block cipher**
  - Block cipher key determines a codebook
  - Each key yields a different codebook
Stream Cipher

• Today, not as popular as block ciphers
• We’ll discuss two stream ciphers...
• **A5/1**
  - Based on shift registers
  - Used in GSM mobile phone system
• **RC4**
  - Based on a changing lookup table
  - Used many places (e.g. SSL and WEP protocols)
• **Stream cipher:**
  - Takes a key $K$ of $n$ bits in length and stretches it into a long keyStream

---

• **Encryption:**

Plaintext $P = p_0, p_1, p_2, p_3, \ldots, p_n$
KeyStream $S = s_0, s_1, s_2, s_3, \ldots, s_n$
Ciphertext $C = c_0 = p_0 \oplus s_0, c_1 = p_1 \oplus s_1, c_2 = p_2 \oplus s_2, \ldots$
  $\ldots, c_n = p_n \oplus s_n$
A5/1 uses 3 shift registers

- **X**: 19 bits \((x_0, x_1, x_2, \ldots, x_{18})\)

- **Y**: 22 bits \((y_0, y_1, y_2, \ldots, y_{21})\)

- **Z**: 23 bits \((z_0, z_1, z_2, \ldots, z_{22})\)

- Key length is 64 bits
• At each step: \( m = \text{maj}(x_8, y_{10}, z_{10}) \)
  – Examples: \( \text{maj}(0,1,0) = 0 \) and \( \text{maj}(1,1,0) = 1 \)
• If \( x_8 = m \) then \( X \) steps
  – \( t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18} \)
  – \( x_i = x_{i-1} \) for \( i = 18,17,\ldots,1 \) and \( x_0 = t \)
• If \( y_{10} = m \) then \( Y \) steps
  – \( t = y_{20} \oplus y_{21} \)
  – \( y_i = y_{i-1} \) for \( i = 21,20,\ldots,1 \) and \( y_0 = t \)
• If \( z_{10} = m \) then \( Z \) steps
  – \( t = z_{7} \oplus z_{20} \oplus z_{21} \oplus z_{22} \)
  – \( z_i = z_{i-1} \) for \( i = 22,21,\ldots,1 \) and \( z_0 = t \)
• Keystream \textbf{bit} is \( x_{18} \oplus y_{21} \oplus z_{22} \)
A5/1: Shift Registers

Each variable here is a single bit
Key is used as initial fill of registers
Each register steps (or not) based on $\text{maj}(x_8, y_{10}, z_{10})$
Keystream bit is XOR of rightmost bits of registers
A5/1 Example

In this example, \( m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(1,0,1) = 1 \)

- Register X steps, Y does not step, and Z steps.
- Keystream bit is XOR of right bits of registers.
- Here, keystream bit will be \( 0 \oplus 1 \oplus 0 = 1 \)
Shift Register Crypto

- Shift register crypto efficient in hardware
- Often, slow if implement in software
- In the past, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used some
  - Common in wireless and high error-rate apps
RC4

- A self-modifying lookup table
- Table always contains a permutation of $0,1,\ldots,255$
- Initialize the permutation using key
- Key length is between 1 – 256 bytes
- Key is used to initialized the permutation $s$

![Diagram of RC4 encryption process]
• At each step, RC4 does the following
  ▪ Swaps elements in current lookup table
  ▪ Selects a keystream byte from table
• Each step of RC4 produces a **byte**
  ▪ Efficient in software
• Each step of A5/1 produces only a bit
  ▪ Efficient in hardware
RC4 Initialization

- \( S[] \) is permutation of 0,1,...,255

\[
\begin{array}{cccccc}
23 & 166 & 10 & \cdots & 22 & 200 & 77 & 12
\end{array}
\]

- Key contain \( N \) bytes

\[
\begin{array}{cccccc}
0 & & & \cdots & & \text{N}
\end{array}
\]

\[
\text{for } i = 0 \text{ to } 255
\]

\[
\begin{align*}
S[i] &= i \\
K[i] &= \text{key}[i \mod N] \\
i++;
j &= 0
\end{align*}
\]

\[
\text{for } i = 0 \text{ to } 255
\]

\[
\begin{align*}
j &= (j + S[i] + K[i]) \mod 256 \\
\text{swap}(S[i], S[j])
j++;
i &= j = 0
\end{align*}
\]
RC4 Keystream

• For each keystream byte, swap elements in table and select byte

\[
i = (i + 1) \mod 256 \\
\quad j = (j + S[i]) \mod 256 \\
\quad \text{swap}(S[i], S[j]) \\
\quad t = (S[i] + S[j]) \mod 256 \\
\quad \text{keystreamByte} = S[t]
\]

• Use keystream bytes like a one-time pad

• **Note:** first 256 bytes should be discarded
  – Otherwise, related key attack exists
Stream Cipher

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
Block Ciphers

- Plaintext and ciphertext consist of fixed-sized blocks

\[
\text{PlainText} \quad \begin{array}{c}
\phantom{X} \\
\phantom{X} \\
\phantom{X} \\
\phantom{X}
\end{array}
\quad \begin{array}{c}
\phantom{X} \\
\phantom{X} \\
\phantom{X} \\
\phantom{X}
\end{array}
\quad \text{CipherText} \quad \begin{array}{c}
\phantom{X} \\
\phantom{X} \\
\phantom{X} \\
\phantom{X}
\end{array}
\]

- Ciphertext obtained from plaintext by iterating a **round function**

\[
\text{Key} \quad \begin{array}{c}
\phantom{X} \\
\phantom{X} \\
\phantom{X} \\
\phantom{X}
\end{array}
\quad \text{PlainText} \quad \begin{array}{c}
\phantom{X} \\
\phantom{X} \\
\phantom{X} \\
\phantom{X}
\end{array}
\quad \text{CipherText}
\]

- Input to round function consists of key and the output of previous round

- Usually implemented in software
Feistel Cipher: Encryption

- **Feistel cipher** is a type of block cipher design, not a specific cipher
- Split plaintext block into left and right halves: \( P = (L_0, R_0) \)
- For each round \( i = 1, 2, \ldots, n \), compute
  \[
  L_i = R_{i-1} \\
  R_i = L_{i-1} \oplus F(R_{i-1}, K_i), \text{ where } F \text{ is round function and } K_i \text{ is subkey}
  \]
- Ciphertext: \( C = (L_n, R_n) \)
Feistel Cipher: Decryption

• Start with Ciphertext $C = (L_n, R_n)$
• For each round $i = n, n-1, \ldots, 1$, compute
  
  $R_{i-1} = L_i$
  
  $L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$

  where $F$ is round function and $K_i$ is subkey

• Plaintext: $P = (L_0, R_0)$

• Formula “works” for any function $F$
  
  ▪ But only secure for certain functions $F$
Tiny Encryption Algorithm

- 64 bit block, 128 bit key
- Assumes 32-bit arithmetic
- Number of rounds is variable (32 is considered secure)
- Uses “weak” round function, so large number of rounds required
TEA Encryption

Assuming 32 rounds:

\[(K[0],K[1],K[2],K[3]) = 128 \text{ bit key}\]
\[(L,R) = \text{plaintext (64-bit block)}\]
\[\text{delta} = 0x9e3779b9\]
\[\text{sum} = 0\]

for i = 1 to 32

\[
\text{sum} += \text{delta} \\
L += ((R<<4)+K[0]) \oplus (R+\text{sum}) \oplus ((R>>5)+K[1]) \\
R += ((L<<4)+K[2]) \oplus (L+\text{sum}) \oplus ((L>>5)+K[3])
\]

next i

ciphertext = (L,R)
TEA Decryption

Assuming 32 rounds:

\[(K[0],K[1],K[2],K[3]) = 128\text{ bit key}\]
\[(L,R) = \text{ciphertext (64-bit block)}\]
\[\text{delta} = 0x9e3779b9\]
\[\text{sum} = \text{delta} \ll 5\]

for \(i = 1\) to \(32\)

\[R := ((L\ll4)+K[2]) \oplus (L+\text{sum}) \oplus ((L\gg5)+K[3])\]
\[L := ((R\ll4)+K[0]) \oplus (R+\text{sum}) \oplus ((R\gg5)+K[1])\]
\[\text{sum} := \text{delta}\]

next \(i\)

plaintext = (L,R)
TEA comments

• **Almost** a Feistel cipher
  – Uses + and - instead of $\oplus$ (XOR)

• Simple, easy to implement, fast, low memory requirement, etc.

• Possibly a “related key” attack

• Simplified TEA (STEA) — insecure version used as an example for cryptanalysis
DES Numerology

• DES is a Feistel cipher with...
  ▪ 64 bit block length
  ▪ 56 bit key length
  ▪ 16 rounds
  ▪ 48 bits of key used each round (subkey)
• Each round is simple (for a block cipher)
• Security depends heavily on “S-boxes”
  – Each S-boxes maps 6 bits to 4 bits
One Round of DES
• Input 32 bits

\[0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \quad 17 \quad 18 \quad 19 \quad 20 \quad 21 \quad 22 \quad 23 \quad 24 \quad 25 \quad 26 \quad 27 \quad 28 \quad 29 \quad 30 \quad 31\]

• Output 48 bits

\[31 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \quad 15 \quad 16 \quad 17 \quad 18 \quad 19 \quad 20 \quad 19 \quad 20 \quad 21 \quad 22 \quad 23 \quad 24 \quad 23 \quad 24 \quad 25 \quad 26 \quad 27 \quad 28 \quad 27 \quad 28 \quad 29 \quad 30 \quad 31 \quad 0\]
• 8 “substitution boxes” or S-boxes
• Each S-box maps 6 bits to 4 bits
• S-box number 1

input bits (0,5)
↓
input bits (1,2,3,4)

<table>
<thead>
<tr>
<th>0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>
**DES P-box**

- **Input 32 bits**
  
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  
  16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- **Output 32 bits**
  
  15  6 19 20 28 11 27 16  0 14 22 25  4 17 30  9  
  1  7 23 13 31 26  2  8 18 12 29  5 21 10  3 24
• 56 bit DES key, numbered 0,1,2,...,55
• Left half key bits, $L_K$
  49 42 35 28 21 14  7
  0 50 43 36 29 22 15
  8 1 51 44 37 30 23
 16 9  2 52 45 38 31
• Right half key bits, $R_K$
  55 48 41 34 27 20 13
  6 54 47 40 33 26 19
 12 5 53 46 39 32 25
 18 11  4 24 17 10  3
For rounds $i=1,2,\ldots,16$

- Let $L_K = (L_K \text{ circular shift left by } r_i)$
- Let $R_K = (R_K \text{ circular shift left by } r_i)$
- Left half of subkey $K_i$ is of $L_K$ bits
  
  \[
  13 \ 16 \ 10 \ 23 \ 0 \ 4 \ 2 \ 27 \ 14 \ 5 \ 20 \ 9 \\
  22 \ 18 \ 11 \ 3 \ 25 \ 7 \ 15 \ 6 \ 26 \ 19 \ 12 \ 1
  \]

- Right half of subkey $K_i$ is $R_K$ bits
  
  \[
  12 \ 23 \ 2 \ 8 \ 18 \ 26 \ 1 \ 11 \ 22 \ 16 \ 4 \ 19 \\
  15 \ 20 \ 10 \ 27 \ 5 \ 24 \ 17 \ 13 \ 21 \ 7 \ 0 \ 3
  \]
• Bits 8, 17, 21, 24 of LK omitted each round
• Bits 6, 9, 14, 25 of RK omitted each round
• Compression permutation yields 48 bit subkey $K_i$ from 56 bits of LK and RK
• Key schedule generates subkey

\[ r_i = \begin{cases} 
1 & i \in \{1, 2, 9, 16\} \\
2 & \text{Otherwise} 
\end{cases} \]
• An initial permutation before round 1
• Halves are swapped after last round
• A final permutation (inverse of initial perm) applied to $(R_{16}, L_{16})$
• None of this serves security purpose
Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- Thirty+ years of intense analysis has revealed no “back door”
- Attacks essentially exhaustive key search
• How to encrypt multiple blocks?
• Do we need a new key for each block?
  ▪ As bad as (or worse than) a one-time pad!
• Encrypt each block independently?
• Make encryption depend on previous block, i.e., “chain” the blocks together?
Modes of Operation

• Many modes — we discuss 3 most popular
• Electronic Codebook (ECB) mode
  ▪ Encrypt each block independently
  ▪ Most obvious, but has a serious weakness
• Cipher Block Chaining (CBC) mode
  ▪ Chain the blocks together
  ▪ More secure than ECB, virtually no extra work
• Counter Mode (CTR) mode
  ▪ Acts like a stream cipher
  ▪ Popular for random access
ECB Mode

- Notation: $C = E(P, K)$
- Given plaintext $P_0, P_1, \ldots, P_m, \ldots$
- Most obvious way to use a block cipher:

<table>
<thead>
<tr>
<th>Encrypt</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0 = E(P_0, K)$</td>
<td>$P_0 = D(C_0, K)$</td>
</tr>
<tr>
<td>$C_1 = E(P_1, K)$</td>
<td>$P_1 = D(C_1, K)$</td>
</tr>
<tr>
<td>$C_2 = E(P_2, K)$</td>
<td>$P_2 = D(C_2, K)$</td>
</tr>
</tbody>
</table>

- For fixed key $K$, this is “electronic” version of a codebook cipher (without additive)
ECB Cut and Paste

• Suppose plaintext is
  Alice digs Bob. Trudy digs Tom.

• Assuming 64-bit blocks and 8-bit ASCII:
  \( P_0 = \text{“Alice digs Bob.”}, \ P_1 = \text{“Trudy digs Tom.”} \)

• Ciphertext: \( C_0, C_1, C_2, C_3 \)

• Trudy cuts and pastes: \( C_0, C_3, C_2, C_1 \)

• Decrypts as
  Alice digs Tom. Trudy digs Bob.
• Suppose $P_i = P_j$
• Then $C_i = C_j$ and Trudy knows $P_i = P_j$
• This gives Trudy some information, even if she does not know $P_i$ or $P_j$
• Trudy might know $P_i$
• Is this a serious issue?
Alice Likes CBC Mode

- Alice’s uncompressed image, and ECB encrypted (TEA)

- Same plaintext block ⇒ same ciphertext!
CBC Mode

- Blocks are “chained” together
- A random initialization vector, or IV, is required to initialize CBC mode
- IV is random, but not secret

Encryption
\[ C_0 = E(IV \oplus P_0, K), \]
\[ C_1 = E(C_0 \oplus P_1, K), \]
\[ C_2 = E(C_1 \oplus P_2, K), \ldots \]

Decryption
\[ P_0 = IV \oplus D(C_0, K), \]
\[ P_1 = C_0 \oplus D(C_1, K), \]
\[ P_2 = C_1 \oplus D(C_2, K), \ldots \]
CBC Mode

- Identical plaintext blocks yield different ciphertext blocks
- If $C_1$ is garbled to, say, $G$ then
  
  $P_1 \neq C_0 \oplus D(G, K), \ P_2 \neq G \oplus D(C_2, K)$
- But $P_3 = C_2 \oplus D(C_3, K), \ P_4 = C_3 \oplus D(C_4, K), \ldots$
- Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)
Alice Likes CBC Mode

- Alice’s uncompressed image, Alice CBC encrypted (TEA)

- Same plaintext yields different ciphertext!
Counter Mode (CTR)

- CTR is popular for random access
- Use block cipher like stream cipher

**Encryption**

\[
C_0 = P_0 \oplus E(IV, K), \quad P_0 = C_0 \oplus E(IV, K),
\]

\[
C_1 = P_1 \oplus E(IV+1, K), \quad P_1 = C_1 \oplus E(IV+1, K),
\]

\[
C_2 = P_2 \oplus E(IV+2, K), \ldots \quad P_2 = C_2 \oplus E(IV+2, K), \ldots
\]

**Decryption**

- CBC can also be used for random access
Data Integrity

- **Integrity** — detect unauthorized writing (i.e., modification of data)
- Example: Inter-bank fund transfers
  - Confidentiality is nice, but integrity is critical
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc.
MAC

• Message Authentication Code (MAC)
  ▪ Used for data **integrity**
  ▪ Integrity **not** the same as confidentiality

• MAC is computed as **CBC residue**
  ▪ That is, compute CBC encryption, but only save the final ciphertext block
MAC Computation

- MAC computation (assuming N blocks)
  \[ C_0 = E(IV \oplus P_0, K), \]
  \[ C_1 = E(C_0 \oplus P_1, K), \]
  \[ C_2 = E(C_1 \oplus P_2, K), \ldots \]
  \[ C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = MAC \]

- MAC sent with IV and plaintext

- Receiver does same computation and verifies that result agrees with MAC

- Receiver must also know the key K
Why does a MAC work?

• Suppose Alice has 4 plaintext blocks
• Alice computes
  \[ C_0 = E(IV \oplus P_0, K), \quad C_1 = E(C_0 \oplus P_1, K), \]
  \[ C_2 = E(C_1 \oplus P_2, K), \quad C_3 = E(C_2 \oplus P_3, K) = \text{MAC} \]
• Alice sends \( IV, P_0, P_1, P_2, P_3 \) and \( \text{MAC} \) to Bob
• Suppose Trudy changes \( P_1 \) to \( X \)
• Bob computes
  \[ C_0 = E(IV \oplus P_0, K), \quad C_1 = E(C_0 \oplus X, K), \]
  \[ C_2 = E(C_1 \oplus P_2, K), \quad C_3 = E(C_2 \oplus P_3, K) = \text{MAC} \neq \text{MAC} \]
• Error propagates into \( \text{MAC} \)
• Trudy can’t change \( \text{MAC} \) to \( \text{MAC} \) without \( K \)
Uses for Symmetric Crypto

• Confidentiality
  ▪ Transmitting data over insecure channel
  ▪ Secure storage on insecure media
• Integrity (MAC)
• Authentication protocols (later...)
• Anything you can do with a hash function
Knapsack
Knapsack Problem

- Given a set of \( n \) weights \( W_0, W_1, \ldots, W_{n-1} \) and sum \( S \), is it possible to find \( a_i \in \{0,1\} \) so that
  \[ S = a_0 W_0 + a_1 W_1 + \ldots + a_{n-1} W_{n-1} \]
  (technically, this is “subset sum” problem)

- **Example**
  - Weights \((62,93,26,52,166,48,91,141)\)
  - Problem: Find subset that sums to \( S=302 \)
  - Answer: \( 62+26+166+48=302 \)

- The (general) knapsack is NP-complete
Knapsack Problem

- **General knapsack** (GK) is hard to solve
- But **superincreasing knapsack** (SIK) is easy
- SIK: each weight greater than the sum of all previous weights
- **Example**
  - Weights (2, 3, 7, 14, 30, 57, 120, 251)
  - Problem: Find subset that sums to $S=186$
  - Work from largest to smallest weight
  - Answer: $120+57+7+2=186$
Knapsack Cryptosystem

1. Generate superincreasing knapsack (SIK)
2. Convert SIK into “general” knapsack (GK)
3. **Public Key:** GK
4. **Private Key:** SIK plus conversion factors

- Ideally...
  - Easy to encrypt with GK
  - With private key, easy to decrypt (convert ciphertext to SIK problem)
  - Without private key, must solve GK
Knapsack Keys

- Start with \((2,3,7,14,30,57,120,251)\) as the SIK
- Choose \(m = 41\) and \(n = 491\) (\(m, n\) relatively prime, \(n\) greater than sum of elements)
- Compute “general” knapsack
  - \(2 \cdot 41 \mod 491 = 82\)
  - \(3 \cdot 41 \mod 491 = 123\)
  - \(7 \cdot 41 \mod 491 = 287\)
  - \(14 \cdot 41 \mod 491 = 83\)
  - \(30 \cdot 41 \mod 491 = 248\)
  - \(57 \cdot 41 \mod 491 = 373\)
  - \(120 \cdot 41 \mod 491 = 10\)
  - \(251 \cdot 41 \mod 491 = 471\)
- “General” knapsack: \((82,123,287,83,248,373,10,471)\)
Knapsack Cryptosystem

• **Private key:** $(2, 3, 7, 14, 30, 57, 120, 251)$
  
  \[
  m^{-1} \mod n = 41^{-1} \mod 491 = 12
  \]

• **Public key:** $(82, 123, 287, 83, 248, 373, 10, 471)$, $n=491$

• Example: Encrypt $10010110$
  
  \[
  82 + 83 + 373 + 10 = 548
  \]

• To decrypt,
  
  – $548 \cdot 12 = 193 \mod 491$
  
  – Solve (easy) SIK with $S = 193$
  
  – Obtain plaintext $10010110$
Knapsack Weakness

- **Trapdoor:** Convert SIK into “general” knapsack using modular arithmetic
- **One-way:** General knapsack easy to encrypt, hard to solve; SIK easy to solve
- This knapsack cryptosystem is **insecure**
  - Broken in 1983 with Apple II computer
- “General knapsack” is not general enough!
- This special knapsack is easy to solve!
Secret Sharing
Shamir’s Secret Sharing

- Two points determine a line
- Give \((X_0, Y_0)\) to Alice
- Give \((X_1, Y_1)\) to Bob
- Then Alice and Bob must cooperate to find secret \(S\)
- Also works in discrete case
- Easy to make “\(m\) out of \(n\)” scheme for any \(m \leq n\)
Shamir’s Secret Sharing

- Give \((X_0, Y_0)\) to Alice
- Give \((X_1, Y_1)\) to Bob
- Give \((X_2, Y_2)\) to Charlie
- Then any two of Alice, Bob and Charlie can cooperate to find secret \(S\)
- But no 1 can find secret \(S\)
- A “2 out of 3” scheme
Shamir’s Secret Sharing

- Give \((X_0, Y_0)\) to Alice
- Give \((X_1, Y_1)\) to Bob
- Give \((X_2, Y_2)\) to Charlie
- 3 pts determine parabola
- Alice, Bob, and Charlie must cooperate to find \(S\)
- A “3 out of 3” scheme
- What about “3 out of 4”?
Secret Sharing Example

- Your symmetric key is \( K \)
- Point \((X_0, Y_0)\) to FBI
- Point \((X_1, Y_1)\) to DoJ
- Point \((X_2, Y_2)\) to DoC
- To recover your key \( K \), two of the three agencies must cooperate
- No one agency can get \( K \)
Visual Cryptography

- Another form of secret sharing...
- Alice and Bob “share” an image
- Both must cooperate to reveal the image
- Nobody can learn anything about image from Alice’s share or Bob’s share
  - That is, both shares are required
- Is this possible?
Visual Cryptography

- How to share a pixel?
- Suppose image is black and white

Then each pixel is either black or white

We can split pixels as shown

<table>
<thead>
<tr>
<th></th>
<th>Share 1</th>
<th>Share 2</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Visual Cryptography

- If a pixel is white, randomly choose a or b for Alice’s/Bob’s shares
- If a pixel is black, randomly choose c or d
- Trudy: no info from one share

<table>
<thead>
<tr>
<th>Pixel</th>
<th>Share 1</th>
<th>Share 2</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Visual Crypto Example

- Alice’s share
- Bob’s share
- Overlaid shares
Visual Crypto

• How does visual “crypto” compare to regular crypto?

• In visual crypto, no key...
  – Or, maybe both images are the key?

• With usual encryption, exhaustive search is always possible

• Exhaustive search on visual crypto?
  – No exhaustive search!
Visual Crypto

• Visual crypto — no exhaustive search...

• So, is visual crypto better or worse than the usual crypto?
  
  – Visual crypto is “information theoretically” secure — true of other secret sharing schemes
  
  – With regular encryption, goal is to make cryptanalysis computationally infeasible

• Visual crypto an example of secret sharing
  
  – Not really a form of crypto, in the usual sense