Lecture for February 10, 2016

ECS 235A

UC Davis

Matt Bishop

Supporting Crypto

- All parts of SSL use them
- Initial phase: public key system exchanges keys
  - Messages enciphered using classical ciphers, checksummed using cryptographic checksums
  - Only certain combinations allowed
    - Depends on algorithm for interchange cipher
  - Interchange algorithms: RSA, Diffie-Hellman, Fortezza

RSA: Cipher, MAC Algorithms in SSL

Interchange cipher	*Classical cipher*	*MAC Algorithm*
RSA, key ≤ 512 bits	*none*	MD5, SHA
	RC4, 40-bit key	MD5
	RC2, 40-bit key, CBC mode	MD5
	DES, 40-bit key, CBC mode	SHA
RSA	*None*	MD5, SHA
	RC4, 128-bit key	MD5, SHA
	IDEA, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA

RSA: Cipher, MAC Algorithms in TLS

Interchange cipher	*Classical cipher*	*MAC Algorithm*
RSA	*None*	MD5, SHA, SHA256
	DES, EDE mode, CBC mode	SHA
	AES (128-bit key), CBC mode	SHA, SHA256
	AES (256-bit key), CBC mode	SHA, SHA256

Diffie-Hellman: Types

- Diffie-Hellman: certificate contains D-H parameters, signed by a CA
  - DSS or RSA algorithms used to sign
- Ephemeral Diffie-Hellman: DSS or RSA certificate used to sign D-H parameters
  - Parameters not reused, so not in certificate
- Anonymous Diffie-Hellman: D-H with neither party authenticated
  - Use is "strongly discouraged" as it is vulnerable to attacks

D-H: Cipher, MAC Algorithms in SSL

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Diffie-Hellman, DSS or RSA Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA
Diffie-Hellman, key ≤ 512 bits RSA Certificate	DES, 40-bit key, CBC mode	SHA

D-H: Cipher, MAC Algorithms in TLS

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Diffie-Hellman, DSS or RSA Certificate	DES, EDE mode, CBC mode	SHA
	AES, 128-bit key, CBC mode	SHA, SHA256
	AES, 256-bit key, CBC mode	SHA, SHA256

Ephemeral D-H: Cipher, MAC Algorithms in SSL

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Ephemeral Diffie-Hellman, DSS Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA
Ephemeral Diffie-Hellman, key ≤ 512 bits, RSA Certificate	DES, 40-bit key, CBC mode	SHA

Ephemeral D-H: Cipher, MAC Algorithms in TLS

<table>
<tr><th>Interchange cipher</th><th>Classical cipher</th><th>MAC Algorithm</th></tr>
<tr><td rowspan="3">Ephemeral Diffie-Hellman,
DSS or RSA Certificate</td><td>DES, EDE mode, CBC mode</td><td>SHA</td></tr>
<tr><td>AES, 128-bit key, CBC mode</td><td>SHA, SHA256</td></tr>
<tr><td>AES, 256-bit key, CBC mode</td><td>SHA, SHA256</td></tr>
</table>

Anonymous D-H: Cipher, MAC Algorithms in SSL

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Anonymous D-H, DSS Certificate	RC4, 40-bit key	MD5
	RC4, 128-bit key	MD5
	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA
AnonymousDiffie-Hellman, key ≤ 512 bits, RSA Certificate	RC4, 40-bit key	MD5
	DES, 40-bit key, CBC mode	SHA

Anonymous D-H: Cipher, MAC Algorithms in TLS

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Anonymous D-H, DSS Certificate	DES, EDE mode, CBC mode	SHA
	AES, 128-bit key, CBC mode	SHA, SHA256
	AES, 256-bit key, CBC mode	SHA, SHA256

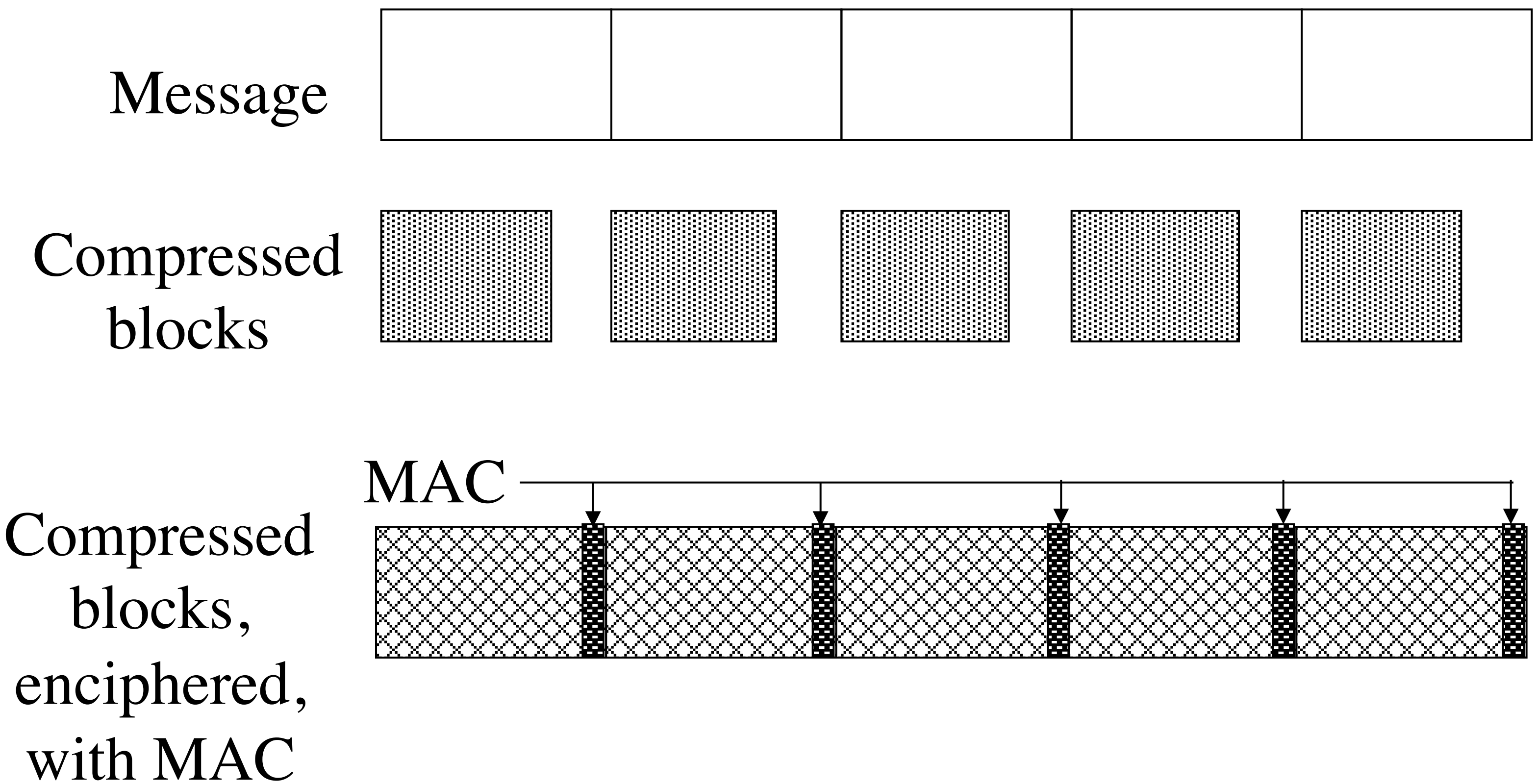
Fortezza: Cipher, MAC Algorithms

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Fortezza key exchange	*none*	SHA
	RC4, 128-bit key	MD5
	Fortezza, CBC mode	SHA

Digital Signatures

- RSA
  - Concatenate MD5 and SHA hashes
  - Sign with public key
- Diffie-Hellman, Fortezza
  - Compute SHA hash
  - Sign appropriately

SSL Record Layer



Record Protocol Overview

- Lowest layer, taking messages from higher
  - Max block size 16,384 bytes
  - Bigger messages split into multiple blocks
- Construction
  - Block b compressed; call it b_c
  - MAC computed for b_c
    - If MAC key not selected, no MAC computed
  - b_c, MAC enciphered
    - If enciphering key not selected, no enciphering done
  - SSL record header prepended

SSL MAC Computation

- Symbols
  - h hash function (MD5 or SHA)
  - k_w write MAC key of entity
  - $ipad$ = 0x36, $opad$ = 0x5C
    - Repeated to block length (from HMAC)
  - seq sequence number
  - SSL_comp message type
  - SSL_len block length
- MAC

$$h(k_w \,||\, opad \,||\, h(k_w \,||\, ipad \,||\, seq \,||\, SSL\_comp \,||\, SSL\_len \,||\, block))$$

TLS MAC Computation

- Symbols
  - h hash function (SHA256)
  - k_w MAC write key of entity
  - *seq* sequence number
  - *TLS_comp* message type
  - *TLS_vers* version of TLS
  - *TLS_len* block length
- MAC

$$h(k_w \,||\, seq \,||\, TLS\_comp \,||\, TLS\_vers \,||\, TLS\_len \,||\, block)$$

SSL Handshake Protocol

- Used to initiate connection
  - Sets up parameters for record protocol
  - 4 rounds
- Upper layer protocol
  - Invokes Record Protocol
- Note: what follows assumes client, server using RSA as interchange cryptosystem

Overview of Rounds

1. Create SSL connection between client, server
2. Server authenticates itself
3. Client validates server, begins key exchange
4. Acknowledgments all around

Handshake Round 1

$\{ v_C \,||\, r_1 \,||\, sid \,||\, ciphers \,||\, comps \}$

Client $\longrightarrow$ Server

$\{ v \,||\, r_2 \,||\, sid \,||\, cipher \,||\, comp \}$

Client $\longleftarrow$ Server

v_C	Client's version of SSL
v	Highest version of SSL that Client, Server both understand
r_1, r_2	nonces (timestamp and 28 random bytes)
s_1	Current session id (0 if new session)
s_2	Current session id (if s1 = 0, new session id)
ciphers	Ciphers that client understands
comps	Compression algorithms that client understand
cipher	Cipher to be used
comp	Compression algorithm to be used

Handshake Round 2

Client $\xleftarrow{\{certificate\}}$ Server

Client $\xleftarrow{\{mod \,||\, exp \,||\, \{\, h(r_1 \,||\, r_2 \,||\, mod \,||\, exp) \,\}\, k_S \,\}}$ Server

Client $\xleftarrow{\{ctype \,||\, gca \,\}}$ Server

Client $\xleftarrow{\{er2\,\}}$ Server

Note: if Server not to authenticate itself, only last message sent; third step omitted if Server does not need Client certificate

k_S	Server's private key
ctype	Certificate type requested (by cryptosystem)
gca	Acceptable certification authorities
er2	End round 2 message

Handshake Round 3

- Both parties compute a master secret from a given *premaster*
  – Used to generate keys for reading and writing

Handshake Round 3, SSL master

$$master = \mathrm{MD5}(pre \,\|\, \mathrm{SHA}(\text{‘A’} \,\|\, pre \,\|\, r_1 \,\|\, r_2) \,\|$$
$$\mathrm{MD5}(pre \,\|\, \mathrm{SHA}(\text{‘BB’} \,\|\, pre \,\|\, r_1 \,\|\, r_2) \,\|$$
$$\mathrm{MD5}(pre \,\|\, \mathrm{SHA}(\text{‘CCC’} \,\|\, pre \,\|\, r_1 \,\|\, r_2)$$
$$key\_block = \mathrm{MD5}(master \,\|\, \mathrm{SHA}(\text{‘A’} \,\|\, master \,\|\, r_1 \,\|\, r_2)) \,\|$$
$$\mathrm{MD5}(master \,\|\, \mathrm{SHA}(\text{‘BB’} \,\|\, master \,\|\, r_1 \,\|\, r_2)) \,\|$$
$$\mathrm{MD5}(master \,\|\, \mathrm{SHA}(\text{‘CCC’} \,\|\, master \,\|\, r_1 \,\|\, r_2)) \,\|$$
$$\ldots$$

Handshake Round 3, TLS master

A(i) = HMAC_hash(secret, A(i–1)); A(0) = *seed*

P_hash(x, seed) = HMAC_hash(secret, A(1) || seed) ||

HMAC_hash(secret, A(2) || seed) ||

HMAC_hash(secret, A(3) || seed) || …

PRF(secret, label, seed) = P_hash(secret, label || seed)

master = PRF(pre, "master secret", r1 || r2)

key_block = PRF(master, "key expansion", r1 || r2)

Handshake Round 3

$$\text{Client} \xrightarrow{\{\, pre \,\}K_{\text{server}}} \text{Server}$$

Both Client, Server compute master secret *master* as in the previous slides

$$\text{Client} \xrightarrow{\{\, h(master \,\|\, opad \,\|\, h(msgs \,\|\, master \,\|\, ipad))\,\}} \text{Server}$$

msgs	Concatenation of previous messages sent/received this handshake
opad, *ipad*	As above

Handshake Round 4

Client sends “change cipher spec” message using that protocol

Client $\longrightarrow$ Server

$\{ h(master \,||\, opad \,||\, h(msgs \,||\, \text{0x434C4E54} \,||\, master \,||\, ipad)) \}$

Client $\longrightarrow$ Server

Server sends “change cipher spec” message using that protocol

Client $\longleftarrow$ Server

$\{ h(master \,||\, opad \,||\, h(msgs \,||\, master \,||\, ipad)) \}$

Client $\longleftarrow$ Server

msgs Concatenation of messages sent/received this handshake in *previous* rounds (does notinclude these messages)

opad, *ipad*, *master* As above

SSL Change Cipher Spec Protocol

- Send single byte
- In handshake, new parameters considered "pending" until this byte received
  - Old parameters in use, so cannot just switch to new ones

SSL Alert Protocol

- Closure alert
  - Sender will send no more messages
  - Pending data delivered; new messages ignored
- Error alerts
  - Warning: connection remains open
  - Fatal error: connection torn down as soon as sent or received

SSL Alert Protocol Errors

- Always fatal errors:
  - unexpected_message, bad_record_mac, decompression_failure, handshake_failure, illegal_parameter
- May be warnings or fatal errors:
  - no_certificate, bad_certificate, unsupported_certificate, certificate_revoked, certificate_expired, certificate_unknown

SSL Application Data Protocol

- Passes data from application to SSL Record Protocol layer

SSL Issues

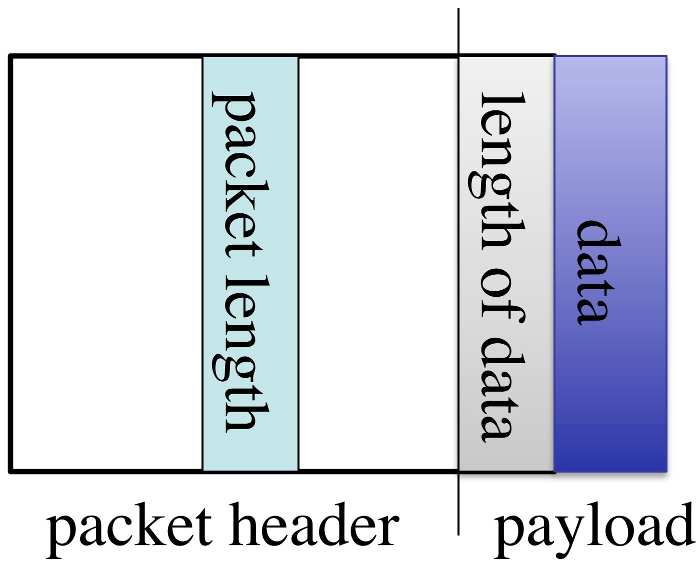
- Heartbleed
  - Implementation bug
- FREAK
  - Exploits a crypto protocol with 40-bit keys
- POODLE
  - Exploits random padding

Heartbleed

- SSL clients, servers may send a message asking "are you alive"?
  - Called a *heartbeat*
- Packet body is:
  - *length of data* || *data*
- Recipient sends back the *data* in the packet

The Attack

- Send a heartbeat packet with length of data set to a large number and the actual data much smaller



What Happens

- Recipient loads packet into buffer
  - Key: the buffer is not cleared!
- Recipient reads length of data *from the payload field*
  - Not the packet header!
- Recipient returns that much data from the buffer
  - Typically contains cookies, passwords, other good stuff

FREAK

- Goal: force client to use export-approved encryption
  - This means an RSA key under 512 bits and a classical cryptosystem with a key length of 40 bits
  - The RSA key can be factored in hours

Background

- Some SSL cipher suites designed for use when crypto was export-controlled
  - Maximum key length allowed: 40 bits for classical, 512 bits for RSA
- Modern clients don't offer it
  - Export controls don't apply here
- But many accept it if it's the only one the server offers

Man-In-The-Middle Attack

Client $\xleftarrow{\{\mathit{certificate}\}}$ Server

Client $\xleftarrow{\{ mod \,\|\, exp \,\|\, \{\, h(r_1 \,\|\, r_2 \,\|\, mod \,\|\, exp) \,\}\, k_S \,\}}$ Server

Client $\xleftarrow{\{ ctype \,\|\, gca \,\}}$ Server

Client $\xleftarrow{\{ er2 \,\}}$ Server

Man-In-The-Middle Attack

$\{ v_C \,\|\, r_1 \,\|\, sid \,\|\, ciphers \,\|\, comps \}$

Client $\longrightarrow$

$\{ v_C \,\|\, r_1 \,\|\, sid \,\|\, export\text{-}grade \,\|\, comps \}$

Attacker $\longrightarrow$ Server

$\{ v \,\|\, r_2 \,\|\, sid \,\|\, export\text{-}grade \,\|\, comp \}$

Client $\longleftarrow$ Server

Client $\xrightarrow{\{ pre \} export\text{-}grade_{\text{server}}}$ Server

The Attack

- In step 2, client accepts export-grade key
  - Attacker then factors the modulus, computes private key
- In step 3, attacker deciphers message to get *pre*; can then compute *master*, *key_block*, and hence all keys

A Helpful Error

- It can take hours to factor the modulus!
- But … many servers generate export key *only* when they start
  - So compute that, and the results are good until the server stops and restarts
  - Apache was one of these

POODLE

- This one finished off SSL 3.0
  - Fixing it requires change to protocol *and* implementation!
- Goal: grab "secure" HTTPS cookies, other interesting tokens (HTTP Authorization header contents)

Assumptions

- Using CBC encryption
- Cipher block padding is not deterministic
  - Nor is the padding included in the MAC
  - Meaning: cannot verify integrity of *padding*
- Padding is 1 block of L bytes
  - Last byte contains L–1
  - This assumption is for exposition only!
- Size of cookie known

On Receiving Message …

- Receives ciphertext is $C_1 \ldots C_n$, with IV C_0
- Deciphers it as $P_i = d_k(C_i) \oplus C_{i-1}$
- Get length of padding from last block of P_n
  - Discard padding
- Check MAC
  - If it matches, accept ciphertext
  - Otherwise, reject ciphertext

The Attack

- Replace Cn by Ci, where Ci is beginning of interesting data
  - Like a cookie
- Ciphertext accepted if $d_k(C_i) \oplus C_{n-1}$ has L–1 as its value
  - On average, happens 1 out of 2^8 = 256 times

How To Do This

- Attacker injects JavaScript program into victim's browser
  - Or somehow gets a cookie-bearing HTTPS request
- SSL records for message modified so that:
  - Padding fills an entire block C_n
  - Cookie's first byte appears as final byte in earlier block C_i
- Replace C_n by C_i and forward message
- If rejected, try with a new request

Why It Works

- Assume each block C_i has 16 bytes $C_i[0] \ldots C_i[15]$
- If server accepts modified ciphertext, last block will be 15
  - As padding is 15 bytes + last one
- So $d_k(C_i[15]) \oplus C_{n-1}[15] = 15$
- So $P_i[15] = 15 \oplus C_{n-1}[15] \oplus C_i[15]$
  - And this is first byte of cookie!

Take It From There

- As request path, request body under control of attacker, change message so size is the same but position of headers shifts appropriately

Results

- RFC 7568:
  - “SSLv3 MUST NOT be used. Negotiation of SSLv3 from any version of TLS MUST NOT be permitted.”
- TLS not vulnerable as padding is not random
  - Each byte contains length of padding
  - Recipient *must* check these values

Background: Entropy

- Random variables
- Joint probability
- Conditional probability
- Entropy (or uncertainty in bits)
- Joint entropy
- Conditional entropy
- Applying it to secrecy of ciphers

Random Variable

- Variable that represents outcome of an event
  - X represents value from roll of a fair die; probability for rolling n: $p(X = n) = 1/6$
  - If die is loaded so 2 appears twice as often as other numbers, $p(X = 2) = 2/7$ and, for $n \neq 2$, $p(X = n) = 1/7$
- Note: $p(X)$ means specific value for X doesn't matter
  - Example: all values of X are equiprobable

Joint Probability

- Joint probability of X and Y, $p(X, Y)$, is probability that X and Y simultaneously assume particular values
  - If X, Y independent, $p(X, Y) = p(X)p(Y)$
- Roll die, toss coin
  - $p(X = 3, Y = \text{heads}) = p(X = 3)p(Y = \text{heads}) = 1/6 \times 1/2 = 1/12$

Two Dependent Events

- X = roll of red die, Y = sum of red, blue die rolls

$p(Y=2) = 1/36$	$p(Y=3) = 2/36$	$p(Y=4) = 3/36$	$p(Y=5) = 4/36$
$p(Y=6) = 5/36$	$p(Y=7) = 6/36$	$p(Y=8) = 5/36$	$p(Y=9) = 4/36$
$p(Y=10) = 3/36$	$p(Y=11) = 2/36$	$p(Y=12) = 1/36$	

- Formula:
  - $p(X=1, Y=11) = p(X=1)p(Y=11) = (1/6)(2/36) = 1/108$

Conditional Probability

- Conditional probability of X given Y, $p(X|Y)$, is probability that X takes on a particular value given Y has a particular value
- Continuing example …
  - $p(Y=7|X=1) = 1/6$
  - $\mathrm{p}(Y=7|X=3) = 1/6$

Relationship

- $p(X, Y) = p(X \mid Y)\, p(Y) = p(X)\, p(Y \mid X)$
- Example:
  - $p(X{=}3, Y{=}8) = p(X{=}3 \mid Y{=}8)\, p(Y{=}8) = (1/5)(5/36) = 1/36$
- Note: if X, Y independent:
  - $p(X \mid Y) = p(X)$

Entropy

- Uncertainty of a value, as measured in bits
- Example: X value of fair coin toss; X could be heads or tails, so 1 bit of uncertainty
  – Therefore entropy of X is $H(X) = 1$
- Formal definition: random variable X, values $x_1, \ldots, x_n$; so Σ_i p$(X = x_i) = 1$
  $H(X) = -\Sigma_i\, p(X = x_i) \lg p(X = x_i)$

Heads or Tails?

- $H(X) = -p(X\text{=heads}) \lg p(X\text{=heads})$
  $\quad - \mathrm{p}(X\text{=tails}) \lg p(X\text{=tails})$
  $= -(1/2) \lg (1/2) - (1/2) \lg (1/2)$
  $= -(1/2)(-1) - (1/2)(-1) = 1$
- Confirms previous intuitive result

n-Sided Fair Die

$H(X) = -\Sigma_i \, p(X = x_i) \lg p(X = x_i)$

As $p(X = x_i) = 1/n$, this becomes

$H(X) = -\Sigma_i \, (1/n) \lg (1/ n) = -n(1/n) (-\lg n)$

so

$H(X) = \lg n$

which is the number of bits in n, as expected

Ann, Pam, and Paul

Ann, Pam twice as likely to win as Paul

W represents the winner. What is its entropy?

- w_1 = Ann, w_2 = Pam, w_3 = Paul
- $p(W= w_1) = p(W= w_2) = 2/5$, $p(W= w_3) = 1/5$

- So $H(W) = -\Sigma_i\, p(W = w_i) \lg p(W = w_i)$

$= -(2/5) \lg (2/5) - (2/5) \lg (2/5) - (1/5) \lg (1/5)$

$= -(4/5) + \lg 5 \approx -1.52$

- If all equally likely to win, $H(W) = \lg 3 = 1.58$

Joint Entropy

- X takes values from $\{ x_1, \ldots, x_n \}$
  - $\Sigma_i\, p(X=x_i) = 1$
- Y takes values from $\{ y_1, \ldots, y_m \}$
  - $\Sigma_i\, p(Y=y_i) = 1$
- Joint entropy of X, Y is:
  - $H(X, Y) = -\Sigma_j\, \Sigma_i\, p(X=x_i, Y=y_j) \lg p(X=x_i, Y=y_j)$

Example

X: roll of fair die, Y: flip of coin

$p(X=1, Y=\text{heads}) = p(X=1)\, p(Y=\text{heads}) = 1/12$

– As X and Y are independent

$$H(X, Y) = -\Sigma_j \Sigma_i\, p(X=x_i, Y=y_j) \lg p(X=x_i, Y=y_j)$$
$$= -2 [6 [(1/12) \lg (1/12)]] = \lg 12$$

Conditional Entropy

- X takes values from $\{ x_1, \ldots, x_n \}$
  - $\Sigma_i \, p(X=x_i) = 1$
- Y takes values from $\{ y_1, \ldots, y_m \}$
  - $\Sigma_i \, p(Y=y_i) = 1$
- Conditional entropy of X given $Y=y_j$ is:
  - $H(X \mid Y=y_j) = -\Sigma_i \, p(X=x_i \mid Y=y_j) \lg p(X=x_i \mid Y=y_j)$
- Conditional entropy of X given Y is:
  - $H(X \mid Y) = -\Sigma_j \, p(Y=y_j) \, \Sigma_i \, p(X=x_i \mid Y=y_j) \lg p(X=x_i \mid Y=y_j)$

Example

- X roll of red die, Y sum of red, blue roll
- Note $p(X{=}1|Y{=}2) = 1$, $p(X{=}i|Y{=}2) = 0$ for $i \neq 1$
  - If the sum of the rolls is 2, both dice were 1
- $H(X|Y{=}2) = -\Sigma_i\, p(X{=}x_i|Y{=}2) \lg p(X{=}x_i|Y{=}2) = 0$
- Note $p(X{=}\mathrm{i},Y{=}7) = 1/6$
  - If the sum of the rolls is 7, the red die can be any of 1, …, 6 and the blue die must be 7–roll of red die
- $H(X|Y{=}7) = -\Sigma_i\, p(X{=}x_i|Y{=}7) \lg p(X{=}x_i|Y{=}7)$

$$= -6\ (1/6) \lg (1/6) = \lg 6$$

Perfect Secrecy

- Cryptography: knowing the ciphertext does not decrease the uncertainty of the plaintext
- $M = \{ m_1, \ldots, m_n \}$ set of messages
- $C = \{ c_1, \ldots, c_n \}$ set of messages
- Cipher $c_i = E(m_i)$ achieves *perfect secrecy* if $H(M \mid C) = H(M)$