Cryptography

Lecture 2

Byte-wise shift cipher

- Work with an alphabet of *bytes* rather than (English, lowercase) *letters*
  - Works natively for arbitrary data!

- Use XOR instead of modular addition
  - Essential properties still hold

Hexadecimal (base 16)

Hex	Bits (“nibble”)	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Hex	Bits (“nibble”)	Decimal
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Hexadecimal (base 16)

- 0x10
  - 0x10 = 16*1 + 0 = 16
  - 0x10 = 0001 0000
- 0xAF
  - 0xAF = 16*A + F = 16*10 + 15 = 175
  - 0xAF = 1010 1111

ASCII

- Characters (often) represented in ASCII
  - 1 byte/char = 2 hex digits/char

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Source: http://benborowiec.com/2011/07/23/better-ascii-table/

ASCII

- ‘1’ = 0x31 = 0011 0001
- ‘F’ = 0x46 = 0100 0110

- Note that writing 0x00 to a file is different from writing “0x00” to a file
  - 0x00 = 0000 0000 (1 byte)
  - “0x00” = 0x30 78 30 30
    = 0011 0000 0111 1000… (4 bytes)

Useful observations

- Only 128 valid ASCII chars (128 bytes invalid)
- 0x20-0x7E printable
- 0x41-0x7a includes upper/lowercase letters
  - Uppercase letters begin with 0x4 or 0x5
  - Lowercase letters begin with 0x6 or 0x7

Byte-wise shift cipher

- $\mathcal{M}$ = {strings of bytes}
- Gen: choose uniform byte $k \in \mathcal{K}$ = {0, …, 255}
- $\text{Enc}_k(m_1 \ldots m_t)$: output $c_1 \ldots c_t$, where
  $$c_i := m_i \oplus k$$
- $\text{Dec}_k(c_1 \ldots c_t)$: output $m_1 \ldots m_t$, where
  $$m_i := c_i \oplus k$$

- Verify that correctness holds…

Code for byte-wise shift cipher

```
// read key from key.txt (hex) and message from ptext.txt (ASCII);
// output ciphertext to ctext.txt (hex)
#include <stdio.h>

main(){
  FILE *keyfile, *pfile, *cfile;
  int i;
  unsigned char key;
  char ch;

  keyfile = fopen("key.txt", "r"), pfile = fopen("ptext.txt", "r"), cfile = fopen("ctext.txt", "w");

  if (fscanf(keyfile, "%2hhX", &key)==EOF) printf("Error reading key.\n");

  for (i=0; ; i++){
    if (fscanf(pfile, "%c", &ch)==EOF) break;
    fprintf(cfile, "%02X", ch^key);
  }

  fclose(keyfile), fclose(pfile), fclose(cfile);
}
```

Is this cipher secure?

- No -- only 256 possible keys!
  - Given a ciphertext, try decrypting with every possible key
  - If ciphertext is long enough, only one plaintext will "make sense"
- Can further optimize
  - First nibble of plaintext likely 0x4, 0x5, 0x6, 0x7 (assuming letters only)
  - Can reduce exhaustive search to 26 keys (how?)

Sufficient key space principle

- The key space must be large enough to make exhaustive-search attacks impractical
  - How large do you think that is?
- Note: this makes some assumptions…
  - English-language plaintext
  - Ciphertext sufficiently long so only one valid plaintext

The Vigenère cipher

- The key is now a *string*, not just a character
- To encrypt, shift each character in the plaintext by the amount dictated by the next character of the key
  - Wrap around in the key as needed
- Decryption just reverses the process

```
tellhimaboutme
cafecafecafeca
--------------
veqpjiredozxoe
```

The Vigenère cipher

- Size of key space?
  - If keys are 14-character strings over the English alphabet, then key space has size $26^{14} \approx 2^{66}$
  - If variable length keys, even more…
  - Brute-force search infeasible

- Is the Vigenère cipher secure?

- (Believed secure for many years…)

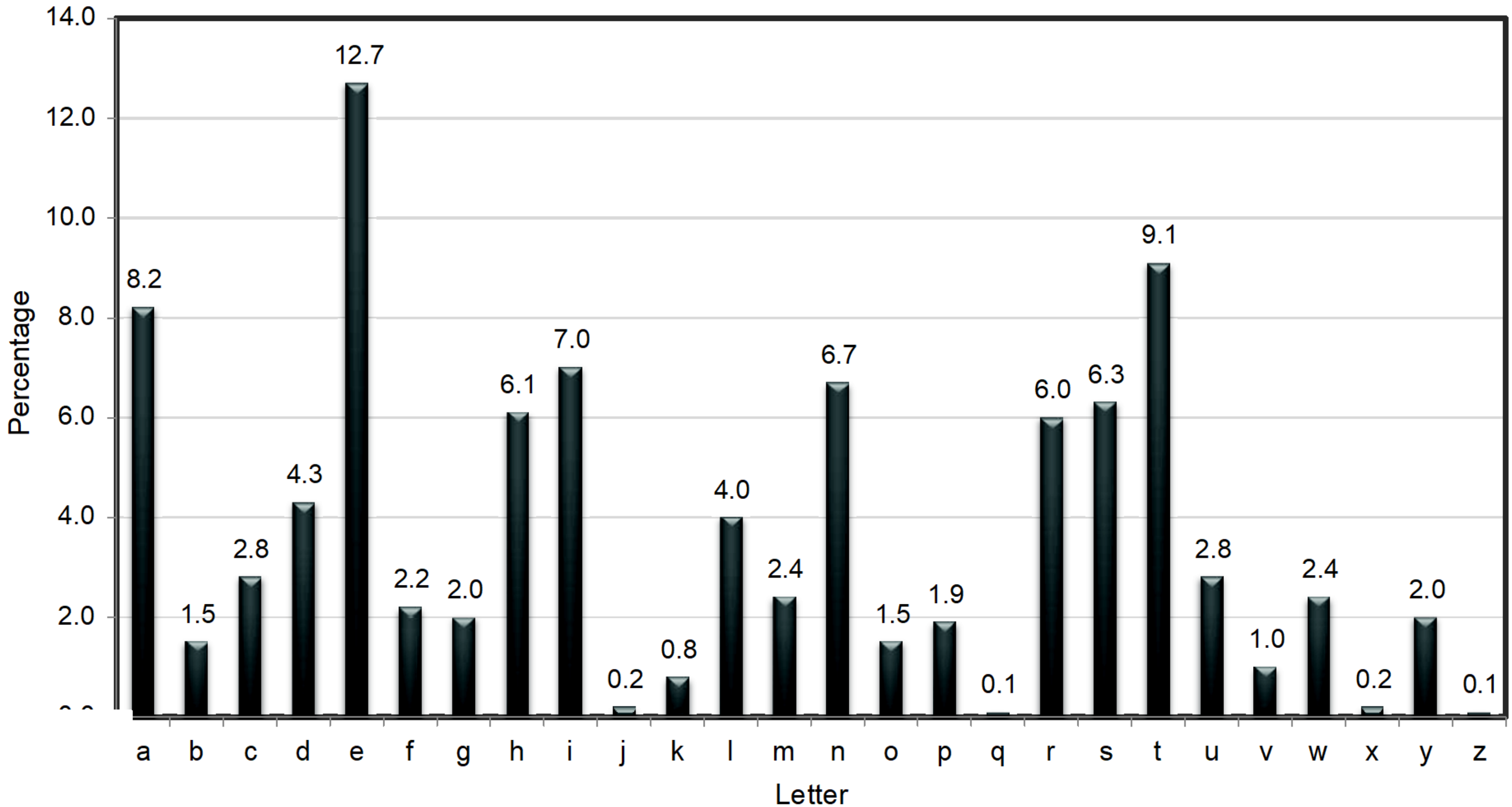
Attacking the Vigenère cipher

- (Assume a 14-character key)
- Observation: every 14th character is "encrypted" using the same shift

veqpjiredozxoe**u**alpcmsdjqu
iqn**d**nossoscdcusoa**k**jqmxpqr
hyycjq**o**qqodhjcciowie**i**i

- Looki… (almost) like looking at ciphertext encrypted with the shift cipher
  - Though a direct brute-force attack doesn't work…
  - Why not?

Using plaintext letter frequencies



Attacking the Vigenère cipher

- Look at every 14th character of the ciphertext, starting with the first
  - Call this a "stream"
- Let α be the most common character appearing in this stream
- Most likely, this character corresponds to the most common plaintext character ('e')
  - Guess that the first character of the key is α - 'e'
- Repeat for all other positions

- This is somewhat haphazard…

A better attack

- Let p_i ($0 \le i \le 25$) denote the frequency of the i^{th} English letter in general text
  - One can compute that $\Sigma_i \, p_i^2 \approx 0.065$
- Let q_i denote the observed frequency of the i^{th} letter in a given stream of the ciphertext
- If the shift for a stream is j, expect $q_{i+j} = p_i$ for all i
  - So expect $\Sigma_i \, p_i \, q_{i+j} \approx 0.065$
- Test for every value of j to find the right one
  - Repeat for each stream

Finding the key length

- When using the correct key length, the ciphertext frequencies $\{q_i\}$ of a stream will be shifted versions of the $\{p_i\}$
  - So $\Sigma\, q_i^2 = \Sigma\, p_i^2 \approx 0.065$
- When using an incorrect key length, expect (heuristically) that the $\{q_i\}$ are equal
  - So $\Sigma\, q_i^2 = \Sigma\, (1/26)^2 = 1/26 = 0.038$
- In fact, good enough to find the key length N that maximizes $\Sigma\, q_i^2$
  - Can check with other streams…

Byte-wise Vigenère cipher

- The key is a string of bytes
- The plaintext is a string of bytes
- To encrypt, XOR each character in the plaintext with the next character of the key
  - Wrap around in the key as needed
- Decryption just reverses the process

Example

- Say plaintext is "Hello!" and key is 0xA1 2F
- "Hello!" = 0x48 65 6C 6C 6F 21
- XOR with 0xA1 2F A1 2F A1 2F
- 0x48 $\oplus$ 0xA1
  - 0100 1000 $\oplus$ 1010 0001 = 1110 1001 = 0xE9

- Ciphertext: 0xE9 4A CD 43 CE 0E

Attacking the (variant) Vigenère cipher

- Two steps:
  - Determine the key length
  - Determine each byte of the key

- Same principles as before…

Determining the key length

- Let p_i (for $0 \le i \le 255$) be the frequency of **byte** i in general English text
  - I.e., $p_i = 0$ for $i < 32$ or $i > 127$
  - I.e., p_{97} = frequency of 'a'
  - The distribution is far from uniform
- If the key length is N, then every N^{th} character of the plaintext is encrypted using the same "shift"
  - If we take every N^{th} character and calculate frequencies, we should get the p_i's in permuted order
  - If we take every M^{th} character (M not a multiple of N) and calculate frequencies, we should get something close to uniform

Determining the key length

- How to distinguish these two?
- For some candidate key length, tabulate $q_0, \ldots, q_{255}$ and compute $\Sigma\, q_i^2$
  - If close to uniform, $\Sigma\, q_i^2 \approx 256 \cdot (1/256)^2 = 1/256$
  - If a permutation of p_i, then $\Sigma\, q_i^2 \approx \Sigma\, p_i^2$
    - Could compute $\Sigma\, p_i^2$ (but somewhat difficult)
    - Key point: will be much larger than 1/256
- Compute $\Sigma\, q_i^2$ for each possible key length, and look for maximum value
  - Correct key length should yield a large value for every stream

Determining the i^{th} byte of the key

- Assume the key length N is known
- Look at every N^{th} character of the ciphertext, starting with the i^{th} character
  - Call this the i^{th} ciphertext "stream"
  - Note that all bytes in this stream were generated by XORing plaintext with the same byte of the key
- Try decrypting the stream using every possible byte value B
  - Get a candidate plaintext stream for each value

Determining the i^{th} byte of the key

- Could use $\{p_i\}$ as before, but not easy to find
- When the guess B is correct:
  - All bytes in the plaintext stream will be between 32 and 127
  - Frequencies of lowercase letters (as a fraction of all lowercase letters) should be close to known English-letter frequencies
    - Tabulate observed letter frequencies $q'_0, \ldots, q'_{25}$ (as fraction of all lowercase letters)
    - Should find $\Sigma\, q'_i\, p'_i \approx \Sigma\, p'^2_i \approx 0.065$, where p'_i corresponds to English-letter frequencies
    - In practice, take B that maximizes $\Sigma\, q'_i\, p'_i$, subject to caveat above (and possibly others)

Attack time?

- Say the key length is between 1 and L
- Determining the key length: $\approx 256\ L$
- Determining all bytes of the key: $< 256^2\ L$

- Brute-force key search: $\approx 256^L$

The attack in practice

- Attack is more reliable as the ciphertext length grows larger

- Attack still works for short(er) ciphertexts, but more “tweaking” and manual involvement can be needed

First programming assignment

- Decrypt ciphertext (provided online) that was generated using the Vigenère cipher

So far…

- “Heuristic” constructions; construct, break, repeat, …

- Can we *prove* that some encryption scheme is secure?

- First need to *define* what we mean by “secure” in the first place…

Historically…

- Cryptography was an *art*
  – Heuristic design and analysis

- This isn’t very satisfying
  – How do we know when a scheme is secure?

Modern cryptography

- In the late ‘70s and early ‘80s, cryptography began to develop into more of a *science*
- Based on three principles that underpin most crypto work today

Core principles of modern crypto

- Formal definitions
  - Precise, mathematical model and definition of what security means
- Assumptions
  - Clearly stated and unambiguous
- Proofs of security
  - Move away from design-break-patch