CS484
Computer and Network Security

Part I: Cryptography

Part I: Crypto

Crypto

- **Cryptology** — The art and science of making and breaking "secret codes"
- **Cryptography** — making "secret codes"
- **Cryptanalysis** — breaking "secret codes"
- **Crypto** — all of the above (and more)

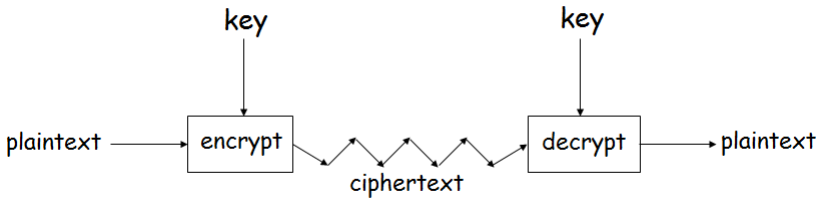
How to Speak Crypto

- A *cipher* or *cryptosystem* is used to *encrypt* the *plaintext*
- The result of encryption is *ciphertext*
- We *decrypt* ciphertext to recover plaintext
- A *key* is used to configure a cryptosystem
- A *symmetric key* cryptosystem uses the same key to encrypt as to decrypt
- A *public key* cryptosystem uses a *public key* to encrypt and a *private key* to decrypt

Crypto

- Basic assumptions
  - The system is completely known to the attacker
  - Only the key is secret
  - That is, crypto algorithms are not secret
- This is known as **Kerckhoffs' Principle**
- Why do we make such an assumption?
  - Experience has shown that secret algorithms tend to be weak when exposed
  - Secret algorithms never remain secret
  - Better to find weaknesses beforehand

Crypto as Black Box



A generic view of symmetric key crypto

Simple Substitution

- Plaintext: fourscoreandsevenyearsago
- Key:

Plaintext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Ciphertext:
  IRXUVFRUHDQGVHYHQBHDUVDJR
- Shift by 3 is "Caesar's cipher"

Ceasar's Cipher Decryption

- Suppose we know a Caesar's cipher is being used:

Plaintext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Given ciphertext:
  VSRQJHEREVTXDUHSDQWV
- Plaintext: spongebobsquarepants

Cryptanalysis I: Try Them All

- A simple substitution (shift by n) is used
  - But the key is unknown
- Given ciphertext: CSYEVIXIVQMREXIH
- How to find the key?
- Only 26 possible keys — try them all!
- **Exhaustive key search**
- Solution: key is n = 4

Simple Substitution: General Case

- In general, simple substitution key can be any **permutation** of letters
  - Not necessarily a shift of the alphabet
- For example

Plaintext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext	J	I	C	A	X	S	E	Y	V	D	K	W	B	Q	T	Z	R	H	F	M	P	N	U	L	G	O

- Then $26! > 2^{88}$ possible keys

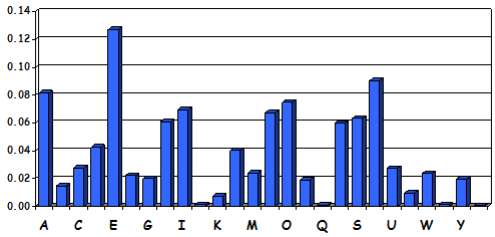
Cryptanalysis II: Be Clever

- We know that a simple substitution is used
- But not necessarily a shift by n
- Find the key given the ciphertext:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQ
WAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFX
FQVXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPB
FTIXPFHXZHVFAGFOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHF
QPBQWAQJJTODXQHFOQPWTBDHHIXQVAPBFZQHCFWPFHPBFIPBQW
KFABVYYDZBOTHPBQPQJTQOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZ
BVFOJIWFFACFCCFHQWAUVWFLQHGFXVAFXQHFUFHILTTAVWAFFAW
TEVOITDHFHFQAITIXPFHXAFQHEFZQWGFLVWPTOFFA

Cryptanalysis II

- Cannot try all 2^{88} simple substitution keys
- Can we be more clever?
- English letter frequency counts…



Types of Cryptography

- Symmetric Key encryption
  - Same key for encryption and decryption
  - Modern types: Stream ciphers, Block ciphers
- Public Key encryption (or "asymmetric")
  - Two keys, one for encryption (public), and one for decryption (private)
  - And digital signatures — nothing comparable in symmetric key crypto

Symmetric Encryption

- The universal technique for providing confidentiality for transmitted or stored data
- Also known as conventional encryption or single-key encryption
- Examples: DES, AES.
- Two requirements for secure use:
  - Need a strong encryption algorithm:
    - Even if the attacker knows it and has some ***ciphertexts***
  - Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure

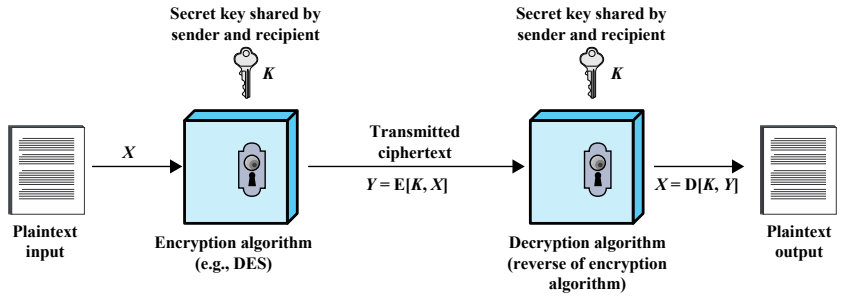


Figure 2.1 Simplified Model of Symmetric Encryption

- Five components:
  - Plain text, encryption algorithm, secret key, ciphertext, decryption algorithm

Attacking Symmetric Encryption

Cryptanalytic Attacks

- Rely on:
  - Nature of the algorithm
  - Some knowledge of the general characteristics of the plaintext
  - Some sample plaintext-ciphertext pairs
- Exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used
  - If successful all future and past messages encrypted with that key are compromised

Brute-Force Attacks

- Try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
  - On average half of all possible keys must be tried to achieve success

Table 20.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	•Encryption algorithm •Ciphertext to be decoded
Known plaintext	•Encryption algorithm •Ciphertext to be decoded •One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	•Encryption algorithm •Ciphertext to be decoded •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

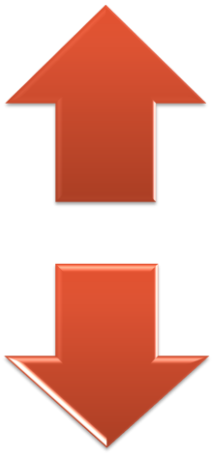
Table 2.1

	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

DES = Data Encryption Standard
AES = Advanced Encryption Standard

Comparison of Three Popular Symmetric Encryption Algorithms

Data Encryption Standard (DES)



- Until recently was the most widely used encryption scheme (1977)
  - FIPS PUB 46
  - Referred to as the Data Encryption Algorithm (DEA)
  - Uses 64 bit plaintext block and 56 bit key to produce a 64 bit ciphertext block

- Strength concerns:
  - Concerns about the algorithm itself
    - DES is the most studied encryption algorithm in existence
  - Concerns about the use of a 56-bit key
    - The speed of commercial off-the-shelf processors makes this key length inadequate

Table 2.2

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/s	Time Required at 10^{13} decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns = 1.125 years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/s	Time Required at 10^{13} decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns = 1.125 years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years

$7.2 \times 10^{16} / 10^9 = 7.2 \times 10^7$ seconds

$72000000 / 3600 = 20000$ hours

$20000 / 24 / 365 / 2 \approx 1.125$ years

Computationally Secure Encryption

- Encryption is computationally secure if:
  - Cost of breaking cipher exceeds value of information
  - Time required to break cipher exceeds the useful lifetime of the information
- Usually very difficult to estimate the amount of effort required to break
- Can estimate time/cost of a brute-force attack

DES – a step by step example

DES: Plain text and Key

- Let M= 0123456789ABCDEF

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

- Let K= 133457799BBCDFF1

0001 0011 0011 0100 0101 0111 0111 1001
1001 1011 1011 1100 1101 1111 1111 0001

DES: Two steps

1. Key generation
2. Plain text encryption

1. Key generation

- Based on a 56 key, we need to generate 16 sub-key of 48 bits each.
- In fact, the original DES key is 64 bits, but 8 of them are neglected!

```
00010011 00110100 01010111 01111001
10011011 10111100 11011111 11110001
```

- We shortly show how this is done

1.A Apply PC^{-1} permutation

- We permute K according to the following:

PC^{-1}

```
57 49 41 33 25 17  9  1 58 50 42 34 26 18
10  2 59 51 43 35 27 19 11  3 60 52 44 36
63 55 47 39 31 23 15  7 62 54 46 38 30 22
14  6 61 53 45 37 29 21 13  5 28 20 12  4
```

- **K**= 00010011 00110100 01010111 01111001
  10011011 10111100 11011111 11110001
- **κ**= 1111000 0110011 0010101 0101111
  0101010 1011001 1001111 0001111

1.B Split κ evenly

- **κ**= 1111000 0110011 0010101 0101111
  0101010 1011001 1001111 0001111
- **κ**= L_0R_0

1.C Generate L_i, R_i

- L_i = #$_i$ left shifts of L_{i-1}
- R_i = #$_i$ left shifts of R_{i-1}

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#i	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

- Given that **L_0= 1111000011001100101010101111**
- **Then L_1= 1110000110011001010101011111**

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#i	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

1111000011001100101010101111 01010101011001100111100011111
1110000110011001010101011111 10101010110011001111000111110
1100001100110010101010111111 01010101100110011110001111101
0000110011001010101011111111 01010110011001111000111101010
0011001100101010101111111100 01011001100111100011110101010
1100110010101010111111110000 01100110011110001111010101010
0011001010101011111111000011 10011001111000111101010101010
1100101010101111111100001100 01100111100011110101010101100
0010101010111111110000110011 10011110001111010101010110011
0101010101111111100001100110 00111100011110101010101100110
0101010111111110000110011001 11110001111010101010110011001
0101011111111000011001100101 11000111101010101011001100110
0101111111100001100110010101 00011110101010101100110011111
0111111110000110011001010101 01111010101010110011001111100
1111111000011001100101010101 11101010101011001100111100011
1111100001100110010101010111 10101010101100110011110001111
1111000011001100101010101111 01010101011001100111100011111

1.D merge L_iR_i to get $\acute{K}_i$

1. 11100001100110010101010111111101010101100110011110001111 0
2. 11000011001100101010101111111010101011001100111100011110 1
3. 00001100110010101010111111111010101100110011110001111010 1
4. 00110011001010101011111111100010110011001111000111101010 1
5. 11001100101010101111111110000011001100111100011110101010 1
6. 00110010101010111111111000011100110011110001111010101010 1
7. 11001010101011111111100001100011001111000111101010101011 0
8. 00101010101111111110000110011100111100011110101010101100 1
9. 01010101011111111100001100110001111000111101010101011001 1
10. 01010101111111100001100110011111000111101010101011001100
11. 01010111111110000110011001011100011110101010101100110011
12. 01011111111000011001100101010001111010101010110011001111
13. 01111111100001100110010101010111101010101011001100111100
14. 11111110000110011001010101011110101010101100110011110001
15. 11111000011001100101010101111010101010110011001111000111
16. 11110000110011001010101011110101010101100110011110001111

1.E Apply PC^{-2} permutation

- We permute each of $\acute{K}_i$ according to the following:

PC^{-2}

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

- $\acute{K}_i$ = 11100001100110010101010111111101010101100110011110001111 0
- k_i= 0 0011011000000101110111111111100011100000111 0010

Final results of Step 1.

1. 000110 110000 001011 101111 111111 000111 000001 110010
2. 011110 011010 111011 011001 110110 111100 100111 100101
3. 010101 011111 110010 001010 010000 101100 111110 011001
4. 011100 101010 110111 010110 110110 110011 010100 011101
5. 011111 001110 110000 000111 111010 110101 001110 101000
6. 011000 111010 010100 111110 010100 000111 101100 101111
7. 111011 001000 010010 110111 111101 100001 100010 111100
8. 111101 111000 101000 111010 110000 010011 101111 111011
9. 111000 001101 101111 101011 111011 011110 011110 000001
10. 101100 011111 001101 000111 101110 100100 011001 001111
11. 001000 010101 111111 010011 110111 101101 001110 000110
12. 011101 010111 000111 110101 100101 000110 011111 101001
13. 100101 111100 010111 010001 111110 101011 101001 000001
14. 010111 110100 001110 110111 111100 101110 011100 111010
15. 101111 111001 000110 001101 001111 010011 111100 001010
16. 110010 110011 110110 001011 000011 100001 011111 110101

Two steps:

1. Key generation
2. Plain text encryption

- Recall `M= 0123456789ABCDEF`

```
0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111
```

2.A Apply an *initial permutation* **IP**

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

- M= 0000 0001 0010 0011 0100 0101 0110 0111
  1000 1001 1010 1011 1100 1101 1110 1111
- IP= 1100 1100 0000 0000 1100 1100 1111 1111
  1111 0000 1010 1010 1111 0000 1010 1010

2.B Split **IP** evenly

- IP= 1100 1100 0000 0000 1100 1100 1111 1111
  1111 0000 1010 1010 1111 0000 1010 1010
- IP = L_0R_0

2.C.1 generate L_i, and prepare R_i

- $L_n = R_{n-1}$
- $R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$ where:
  - $\oplus$ denotes XOR
  - $f(R_{n-1}, K_n) = P(\text{S-BOX}(E(R_{n-1}) \oplus K_n))$
  - $E(R_{n-1})$ permutates and ***extends*** R_{n-1} ***as follows:***

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

2.C.2 still preparing R_i

- R_0 = 1111 0000 1010 1010
  1111 0000 1010 1010
- $E(R_0)$ = 011110 100001 010101 010101
  011110 100001 010101 010101
- K_1 = 000110 110000 001011 101111
  111111 000111 000001 110010
- $K_1 \oplus E(R_0)$ = 011000 010001 011110 111010
  100001 100110 010100 100111

2.C.3 yet still preparing Ri

- $K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8$
- $f(R_{n-1}, K_n)$

  $= P(\text{S-BOX}(E(R_{n-1}) \odot K_n))$

  $= P(S1(B_1)S2(B_2)S3(B_3)S4(B_4)S5(B_5)S6(B_6)S7(B_7)S8(B_8))$

S1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

2.C.3 yet still preparing Ri

- $K_1 \odot E(R_0) =$ 011000 010001 011110 111010 100001 100110 010100 100111
- $S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) =$ 0101 1100 1000 0010 1011 0101 1001 0111

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

2.C.4 almost there!

- Apply a permutation **P** to get $f(R_{n-1},K_n) = P(\text{S-BOX}(E(R_{n-1}) \oplus K_n))$
- **P**

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

- 0101 1100 1000 0010 1011 0101 1001 0111
- becomes: 0010 0011 0100 1010 1010 1001 1011 1011

2.C.5 Finally R_i

- $R_1 = L_0 \odot f(R_0, K_1) =$

1100 1100 0000 0000 1100 1100 1111 1111

$\odot$

0010 0011 0100 1010 1010 1001 1011 1011

=

1110 1111 0100 1010 0110 0101 0100 0100

2.D Reverse the last pair

- The last pair $L_{16}R_{16}$ becomes $R_{16}L_{16}$
- L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
  R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101
- $R_{16}L_{16}$ = 00001010 01001100 11011001 10010101
  01000011 01000010 00110010 00110100

2.E Apply a permutation

- $R_{16}L_{16}$ = 00001010 01001100 11011001 10010101
  01000011 01000010 00110010 00110100

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

- IP^{-1} = 10000101 11101000 00010011 01010100
  00001111 00001010 10110100 00000101

85E813540F0AB405

Here is your cipher

85E813540F0AB405

M= 0123456789ABCDEF

Permutations of DES (IP, IP^{-1})

- What security do they add?
- It does not!!
- Proof?

Triple DES (3DES)

- Repeats basic DES algorithm three times using either two or three unique keys
- First standardized for use in financial applications in ANSI standard X9.17 in 1985
- Attractions:
  - 168-bit key length overcomes the vulnerability to brute-force attack of DES
  - Underlying encryption algorithm is the same as in DES
- Drawbacks:
  - Algorithm is sluggish in software
  - Uses a 64-bit block size

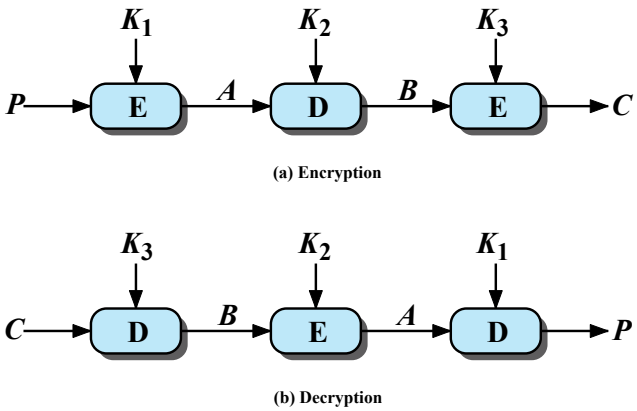


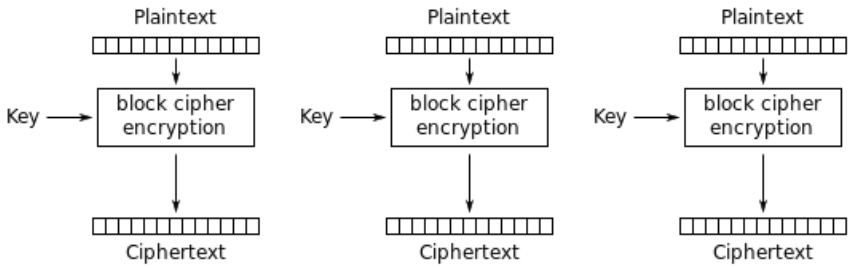
Figure 20.2 Triple DES

Practical Security Issues

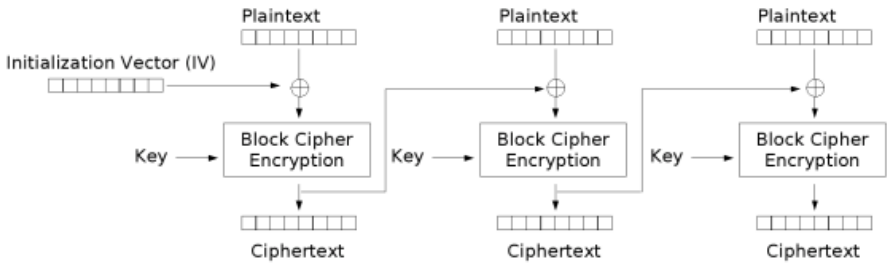
- Typically symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block
- Electronic codebook (ECB) mode is the simplest approach to multiple-block encryption
  - Each block of plaintext is encrypted using the same key
  - Cryptanalysts may be able to exploit regularities in the plaintext
- Modes of operation
  - Alternative techniques developed to increase the security of symmetric block encryption for large sequences
  - Overcomes the weaknesses of ECB

Table 20.3
Block Cipher Modes of Operation

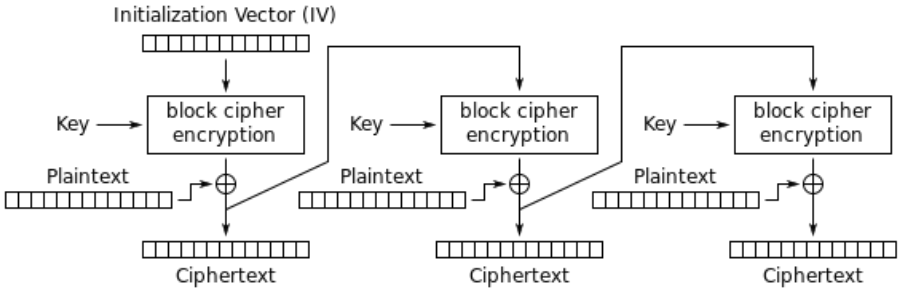
Mode	Description
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.



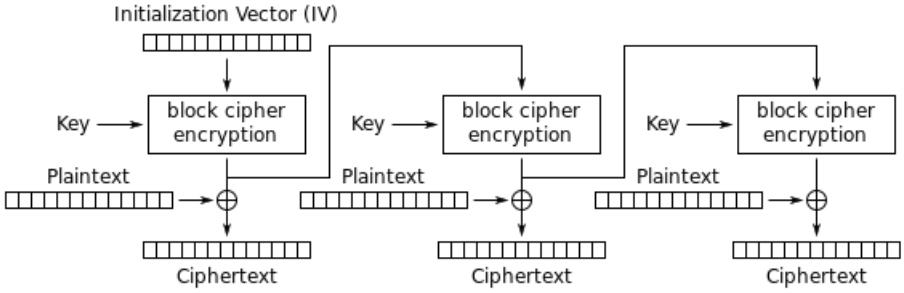
Electronic Codebook (ECB) mode encryption



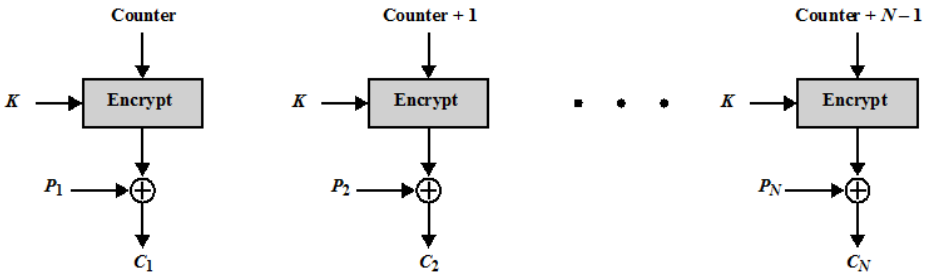
Cipher Block Chaining (CBC) mode encryption



Cipher Feedback (CFB) mode encryption



Output Feedback (OFB) mode encryption



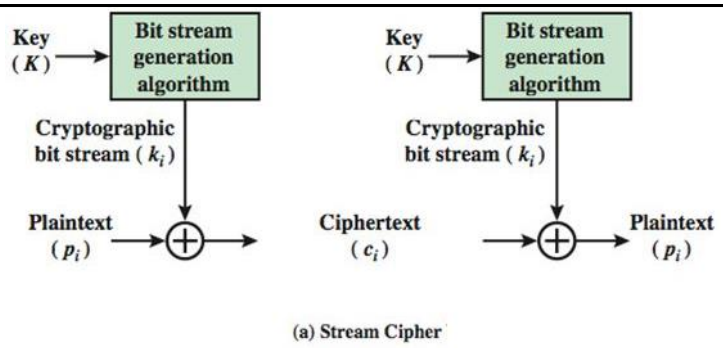
Block & Stream Ciphers

Block Cipher

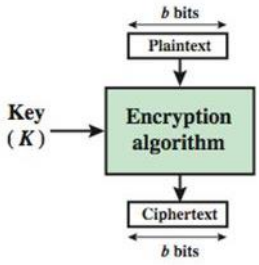
- Processes the input one block of elements at a time
- Produces an output block for each input block
- Can reuse keys
- More common

Stream Cipher

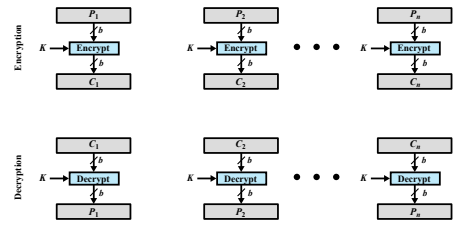
- Processes the input elements continuously
- Produces output one element at a time
- Primary advantage is that they are almost always faster and use far less code
- Encrypts plaintext one byte at a time
- Pseudorandom stream is one that is unpredictable without knowledge of the input key



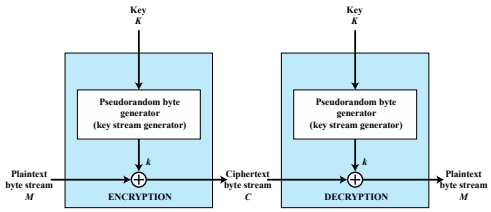
(a) Stream Cipher



(b) Block Cipher



(a) Block cipher encryption (electronic codebook mode)



(b) Stream encryption

Figure 2.2 Types of Symmetric Encryption

Advanced Encryption Standard (AES)

Needed a replacement for 3DES

3DES was not reasonable for long term use

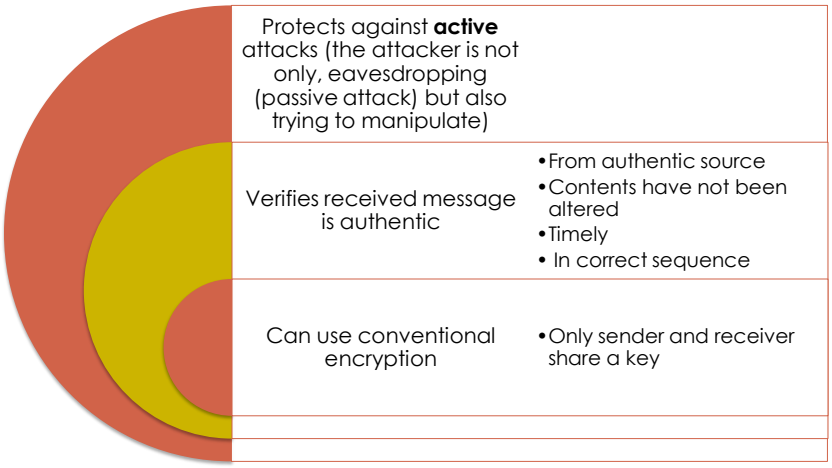
NIST called for proposals for a new AES in 1997

- Should have a security strength equal to or better than 3DES
- Significantly improved efficiency
- Symmetric block cipher
- 128 bit data and 128/192/256 bit keys

Selected Rijndael in November 2001

Published as FIPS 197

Message Authentication



Message Authentication Without Confidentiality

- Message encryption by itself does not provide a secure form of authentication
- It is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag:
  - Error-detection + sequence + timestamp

Message Authentication Without Confidentiality

- Typically message authentication is provided as a separate function from message encryption
- Situations in which message authentication without confidentiality may be preferable include:
  - There are a number of applications in which the same message is broadcast to a number of destinations some times not encrypted
  - An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages (only few from time to time)
  - Authentication of a computer program in plaintext is an attractive service and no need to encrypt the whole program

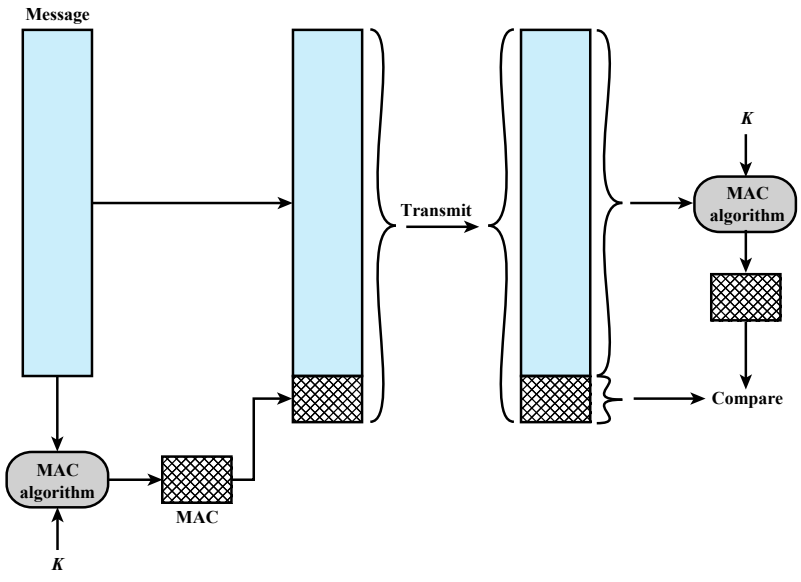


Figure 2.3 Message Authentication Using a Message Authentication Code (MAC).

Hash functions

- Integrity
- A message digest with no need for keys!
- A finger print of a message, file, program, etc.
- Send only an authenticated hash, and the receiver checks for integrity.
  - Symmetric or asymmetric
- Keyed hash: Compute D= H(K . M . K), then send MD
  - Provides authentication!

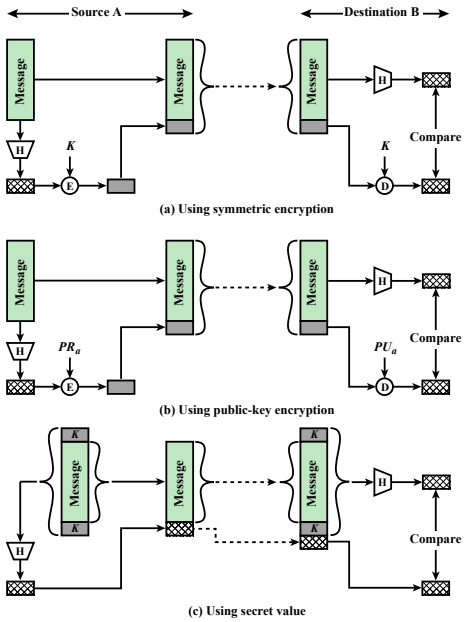


Figure 2.5 Message Authentication Using a One-Way Hash Function.

Requirements on Hash functions

- To be useful for message authentication, a hash function H must have the following properties:

1. Can be applied to a block of data of any size
2. Compression: Produces a fixed-length output much smaller than its input
3. H(x) is relatively easy to compute for any given x
4. Pre-image resistant (One-way)
   - Computationally infeasible to find x such that H(x) = h
5. Weak collision resistance
   - For a given x, it is computationally infeasible to find y ≠ x such that H(y) = H(x)
6. Strong collision resistance
   - it is computationally infeasible to find any pair (x,y) such that H(x) = H(y)

Security of Hash Functions

There are two approaches to attacking a secure hash function:

- Cryptanalysis
  - Exploit logical weaknesses in the algorithm
- Brute-force attack
  - Strength of hash function depends solely on the length of the hash code produced by the algorithm

SHA most widely used hash algorithm

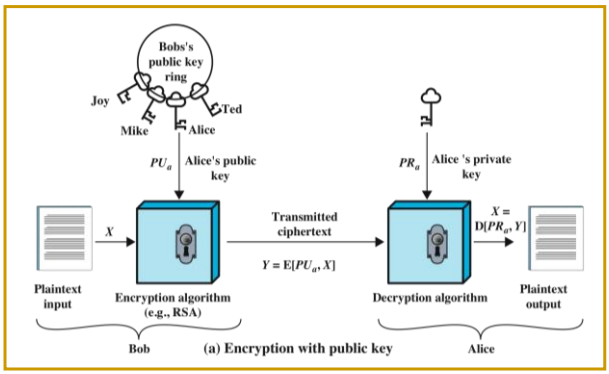
MD2, MD4, MD5

Additional secure hash function applications:

- Passwords
  - Hash of a password is stored by an operating system
- Intrusion detection
  - Store H(F) for each file on a system and secure the hash values

Public-Key Encryption Structure

- Publicly proposed by Diffie and Hellman in 1976
- Based on mathematical functions
- Asymmetric
  - Uses two separate keys
  - Public key and private key
  - Public key is made public for others to use
- Some form of protocol is needed for distribution



- **Plaintext**
  - Readable message or data that is fed into the algorithm as input
- **Encryption algorithm**
  - Performs transformations on the plaintext
- **Public and private key**
  - Pair of keys, one for encryption, one for decryption
- **Ciphertext**
  - Scrambled message produced as output
- **Decryption key**
  - Produces the original plaintext

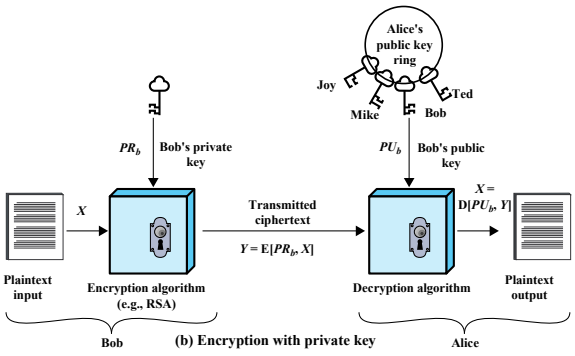


Figure 2.6 Public-Key Cryptography

- User encrypts data using his or her own private key
- Anyone who knows the corresponding public key will be able to decrypt the message

Diffie-Hellman Key Exchange

- First published public-key algorithm
- By Diffie and Hellman in 1976 along with the exposition of public key concepts
- Used in a number of commercial products
- Practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- Security relies on difficulty of computing discrete logarithms

DH realistic example

- Participants: A, B, and *as expected* i
- Common knowledge: n, and g:
  - 315791951375393537137595555337555955191395351995195751755791151795317131135377351919777977373317997317733397199751739199735799971153399111973979977771537137371797357935195553135595739995397713957733739311195177913515117135537117337933757391519397371511397177931573171379357959553351119739999331371993975955117517533779531733395731377975535199115193333715755551757577311599577519951355333733513711
  - 5
- A → B: g^X
  - 274896333250738003987756655558178864202509626681103519395660238884978361258681524065733848121609888166361358396716008309721231543266983475439082379842229520755262549240362420033996781492998212085476914229474198518119740287510455605815485750540439959705032341258897916057748891104776214031140121127131909573145615110962806985892213748176027161799511011873162811236693577815238035096643174972077 29
  -
- B → A: g^Y
  - 187132866656582599554679836612833231300795071533171674971320459038848254471031175750172147136785747373317428228505838107642382626253961500175115579624143507548242830091759253041099058354690908023855631737836740635469845806280809831455689612760312507248666064672970159211722264785314419970237008127843524060431608614374970036610540502843158026669848931373327228822920269911326990948064558159970 6251
- Relies on the difficulty of DLP (discrete logarithm problem) that is NP.

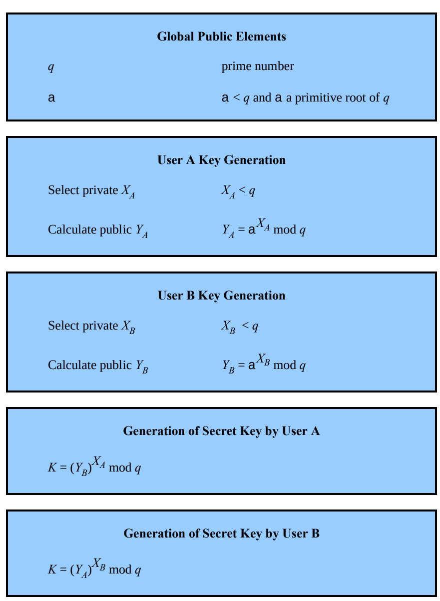


Figure 21.9 The Diffie-Hellman Key Exchange Algorithm

DH Example

Have
- Prime number $q = 353$
- Primitive root $\alpha = 3$

A and B each compute their public keys
- A computes $Y_A = 3^{97} \bmod 353 = 40$
- B computes $Y_B = 3^{233} \bmod 353 = 248$

Then exchange and compute secret key:
- For A: $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$
- *For B:* $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$

Attacker must solve:
- $3^a \bmod 353 = 40$ which is hard
- Desired answer is 97, then compute key as B does

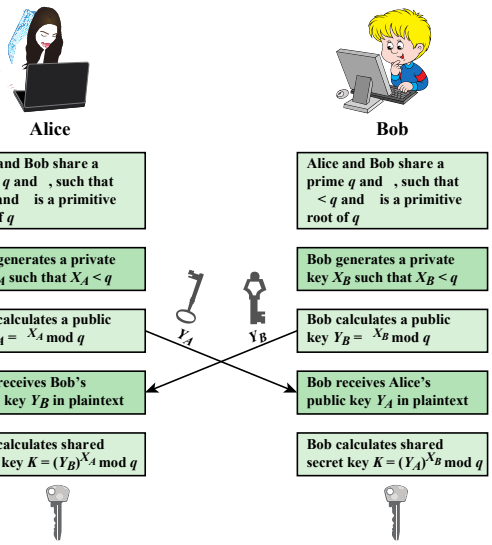


Figure 21.10 Diffie-Hellman Key Exchange

Man-in-the-Middle Attack

- Attack is:
  1. Darth generates private keys X_{D1} and X_{D2}, and their public keys Y_{D1} and Y_{D2}
  2. Alice transmits Y_A to Bob
  3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2
  4. Bob receives Y_{D1} and calculates K1
  5. Bob transmits X_A to Alice
  6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1
  7. Alice receives Y_{D2} and calculates K2
- All subsequent communications compromised

DH problem

- The version of DH that we just introduced is called anonymous DH.
- A man-in-the-middle attack can be easily launched on anonymous DH.
- Therefore, anonymous DH lacks authentication.
- Two other versions of DH are supported with authentication, but they both rely on PKI.
  - Static DH
  - Ephermal DH
- Let's leave DH for now, and we continue on its authenticated versions later.

RSA Public-Key Encryption

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key algorithm
- Uses exponentiation of integers modulo a prime
- Encrypt: $C = M^e \bmod n$
- *Decrypt:* $M = C^d \bmod n = (M^e)^d \bmod n = M$
- Both sender and receiver know values of n and e
- Only receiver knows value of d
- Public-key encryption algorithm with public key PU = {e, n} and private key PR = {d, n}

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

Figure 21.7 The RSA Algorithm

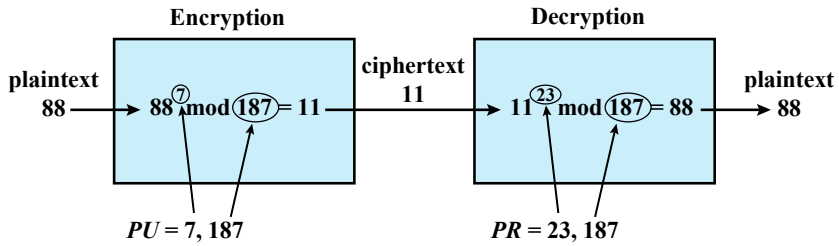


Figure 21.8 Example of RSA Algorithm

Security of RSA

Brute force

- Involves trying all possible private keys

Mathematical attacks

- There are several approaches, all equivalent in effort to factoring the product of two primes

Timing attacks

- These depend on the running time of the decryption algorithm

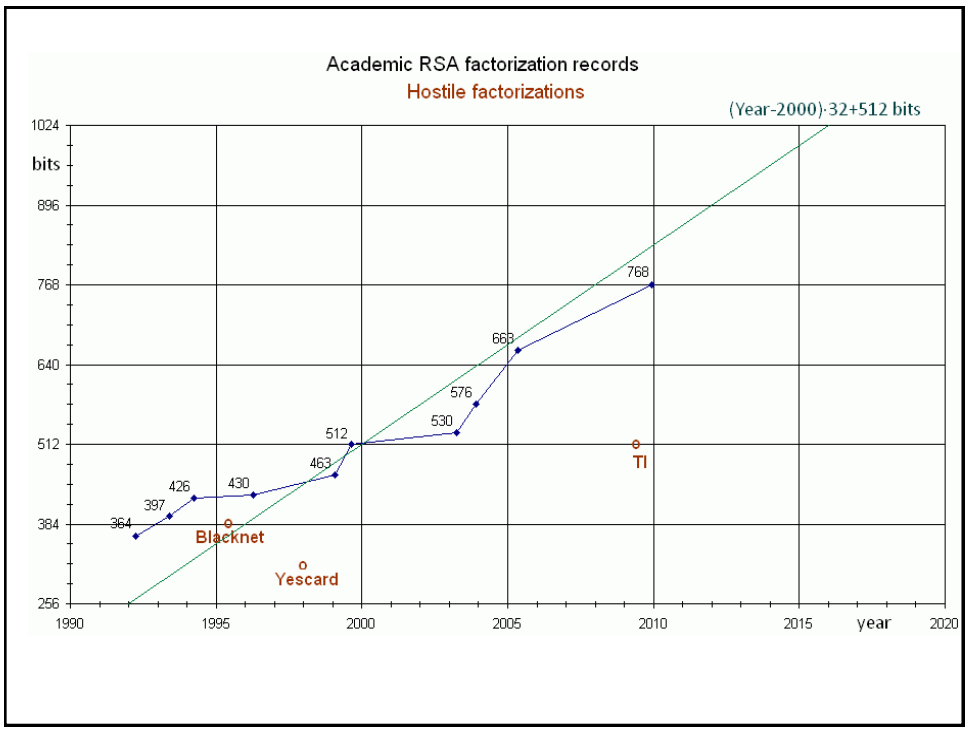
Chosen ciphertext attacks

- This type of attack exploits properties of the RSA algorithm

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Table 21.2

Progress in Factorization



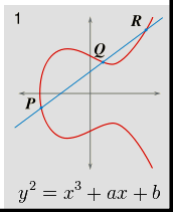
Other Public-Key Algorithms

Digital Signature Standard (DSS)

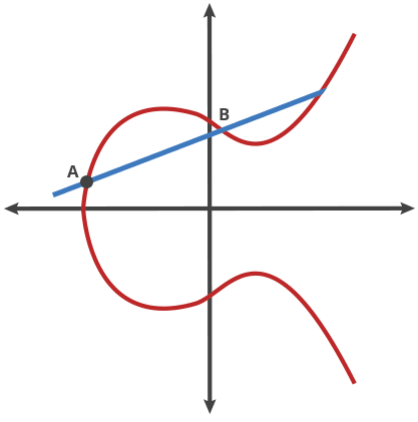
- FIPS PUB 186
- Makes use of SHA-1 and the Digital Signature Algorithm (DSA)
- Originally proposed in 1991, revised in 1993 due to security concerns, and another minor revision in 1996
- Cannot be used for encryption or key exchange
- Uses an algorithm that is designed to provide only the digital signature function

Elliptic-Curve Cryptography (ECC)

- Equal security for smaller bit size than RSA (10%)
- Seen in standards such as IEEE P1363
- Confidence level in ECC is not yet as high as that in RSA
- Based on a mathematical construct known as the elliptic curve



ECC



Public Key: Point A (start), Point E (end)
Private Key: Number of "jumps"/"dots" from A to E

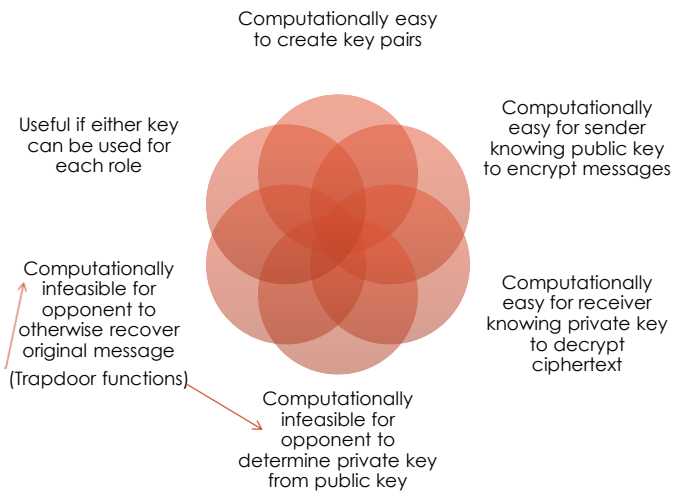
Common concept of DH, RSA, and ECC

- Trap door function:
  - *Difficult* to get x from g^x
  - *Difficult* to get p, q, from n
  - *Difficult* to get n from E = n . A

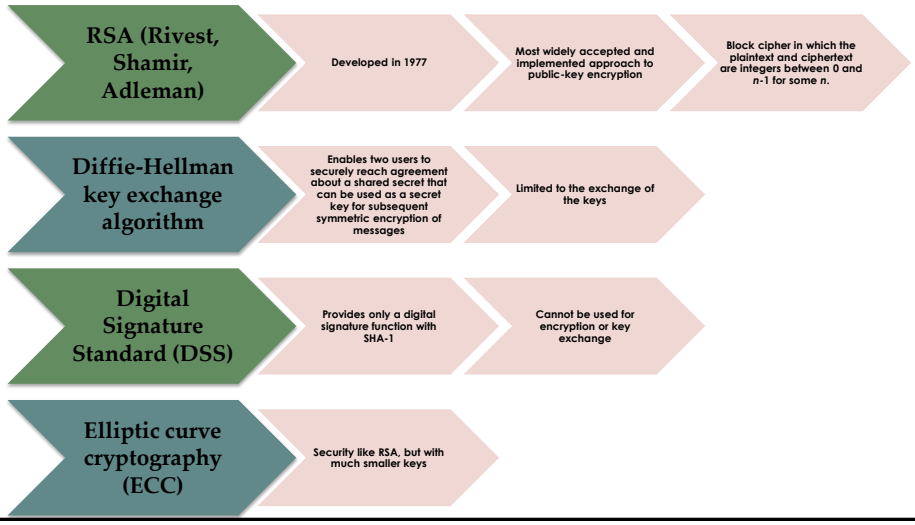
Applications for Public-Key Cryptosystems

Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes

Requirements for Public-Key Cryptosystems



Asymmetric Encryption Algorithms



Digital Signatures

- NIST FIPS PUB 186-4 defines a digital signature as:

  "The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation."

- Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block
- FIPS 186-4 specifies the use of one of three digital signature algorithms:
  - Digital Signature Algorithm (DSA)
  - RSA Digital Signature Algorithm
  - Elliptic Curve Digital Signature Algorithm (ECDSA)

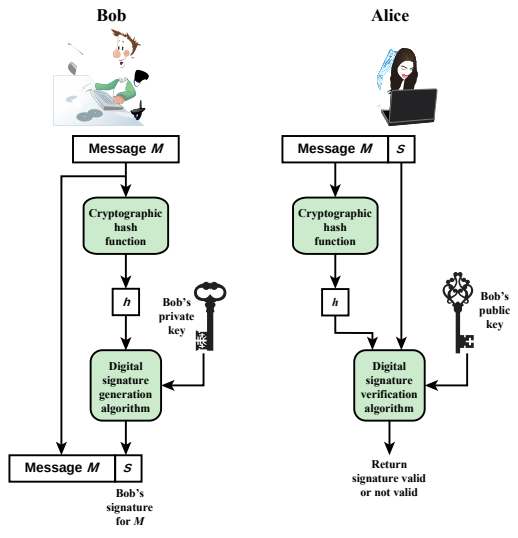


Figure 2.7 Simplified Depiction of Essential Elements of Digital Signature Process

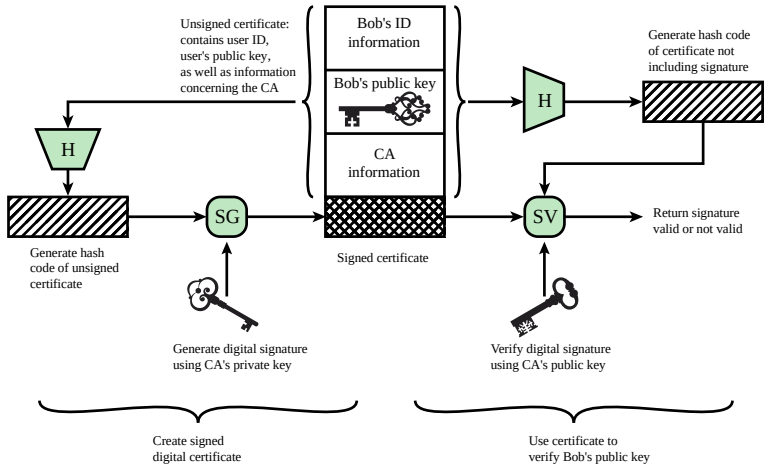


Figure 2.8 Public-Key Certificate Use

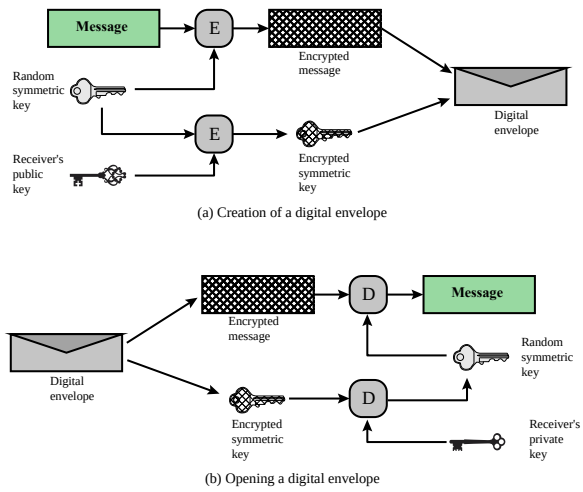


Figure 2.9 Digital Envelopes

Authenticated DH

- Ephermal DH: Alice (and Bob) signs its half-key.
- Static DH: TTP (Trusted Third Party) signs a static half key for Alice (and Bob)

Summary

- Symmetric encryption
  - Symmetric encryption
  - Symmetric block encryption algorithms
  - Stream ciphers
- Message authentication and hash functions
  - Authentication using symmetric encryption
  - Message authentication without message encryption
  - Secure hash functions
  - Other applications of hash functions
- Public-key encryption
  - Structure
  - Applications for public-key cryptosystems
  - Requirements for public-key cryptography
  - Asymmetric encryption algorithms
- Digital signatures and key management
  - Digital signature
  - Public-key certificates
  - Symmetric key exchange using public-key encryption
  - Digital envelopes


DATA SECURITY
IN THIS CORNER, WE HAVE FIREWALLS, ENCRYPTION, ANTIVIRUS SOFTWARE, ETC. AND IN THIS CORNER, WE HAVE DAVE!!
HUMAN ERROR

Network Security Protocols Part I

- Security protocols are three-line programs that people still manage to get wrong.

  - Roger M. Needham

Some notation

- scrypt(k,m): symmetric encryption of the message m using the key k
- crypt(k,m) : asymmetric encryption of the message m using the key k
- pk(a): public key of a

Needham-Schroeder Public Key protocol NSPK

Alice

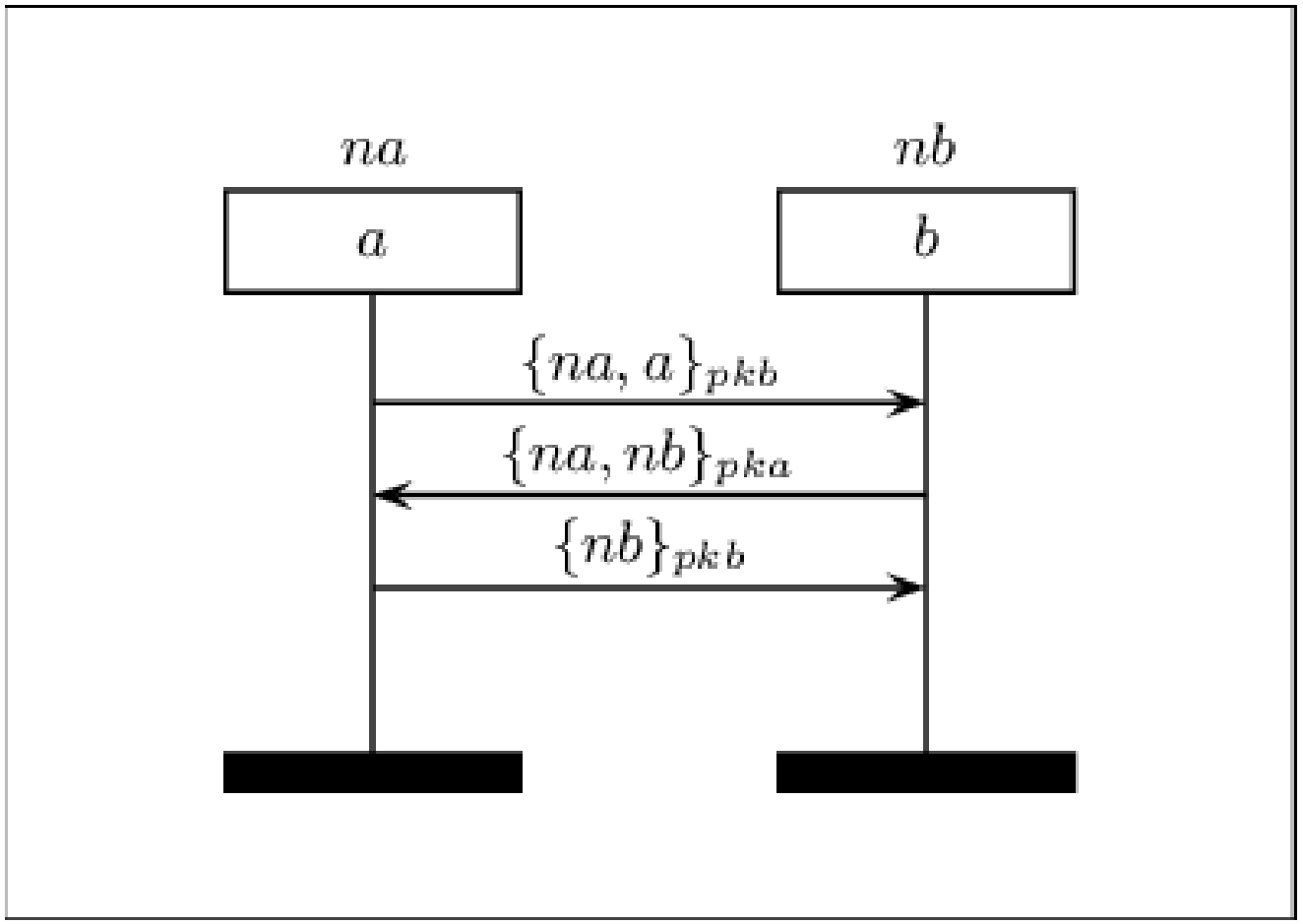
1. crypt(pk(B), Na, A)

2. crypt(pk(A), Na, Nb)

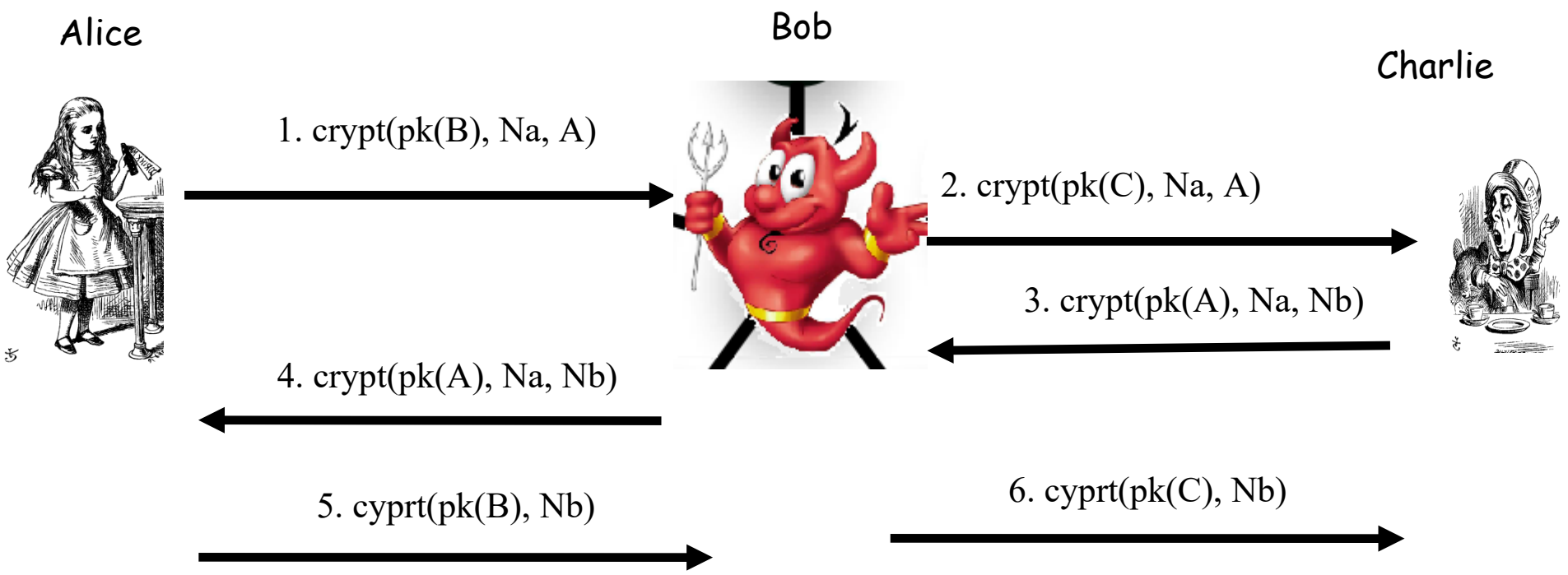
3. cyprt(pk(B), Nb)

Bob

NSPK

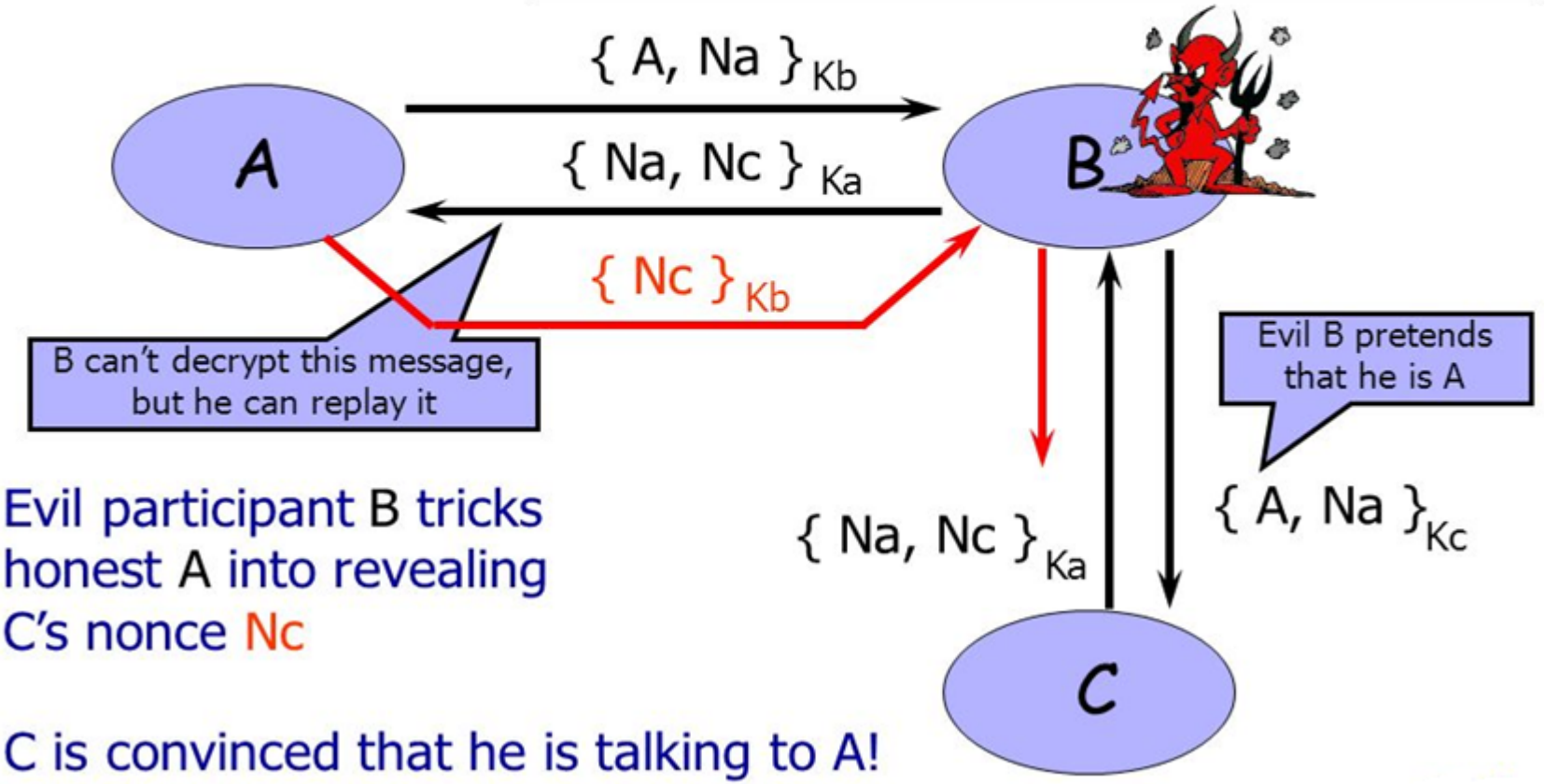


Lowe's Attack on NSPK



Lowe's Attack on NSPK

[Lowe. "Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR". TACAS 1996]



Evil participant B tricks honest A into revealing C's nonce Nc

C is convinced that he is talking to A!

Lowe's Fix of broken Needham - Schroeder

Alice

1. crypt(pk(B), Na, A)

2. crypt(pk(A), Na, Nb, B)

3. cyprt(pk(B), Nb)

Bob

Protocol

- Human protocols — the rules followed in human interactions
  - Example: Asking a question in class
- Networking protocols — rules followed in networked communication systems
  - Examples: HTTP, FTP, etc.
- Security protocol — the (communication) rules followed in a security application
  - Examples: NSPK, SSL, IPSec, Kerberos, etc.

Protocols

- Protocol flaws can be very ***subtle***
- Several well-known security protocols have significant flaws
  - NSPK, WEP, GSM, and IPSec
- Implementation errors can also occur
  - Recently, IE implementation of SSL
- Not easy to get protocols right…

Network Security Protocols Part II

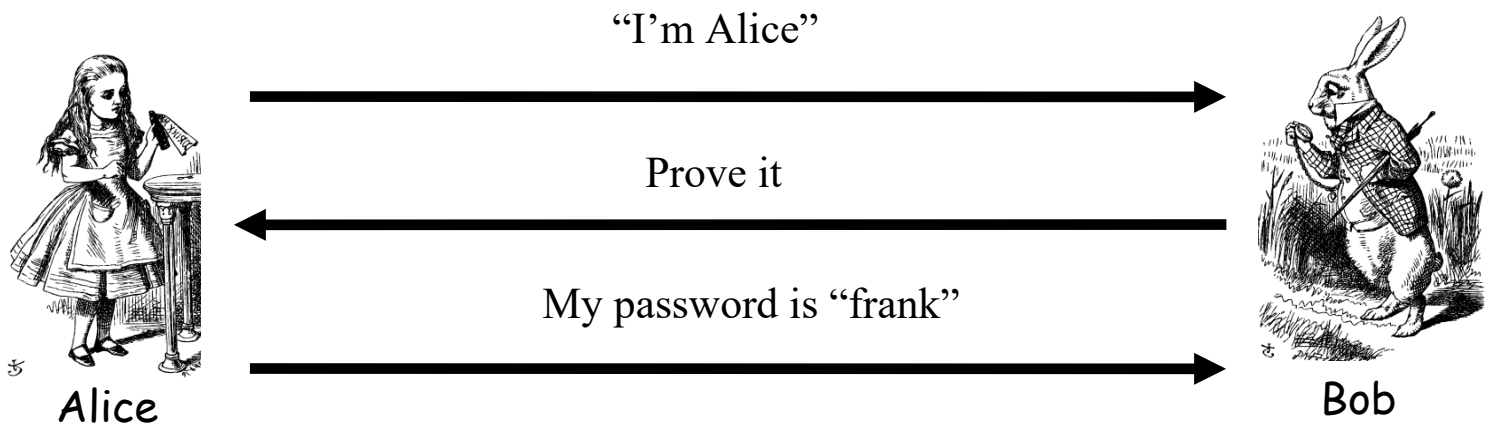
Authentication

- Alice must prove her identity to Bob
  - Alice and Bob can be humans or **computers**
- May also require Bob to prove he's Bob (mutual authentication)
- Probably need to establish a **session key**
- May have other requirements, such as
  - Public keys, symmetric keys, hash functions, …

Authentication

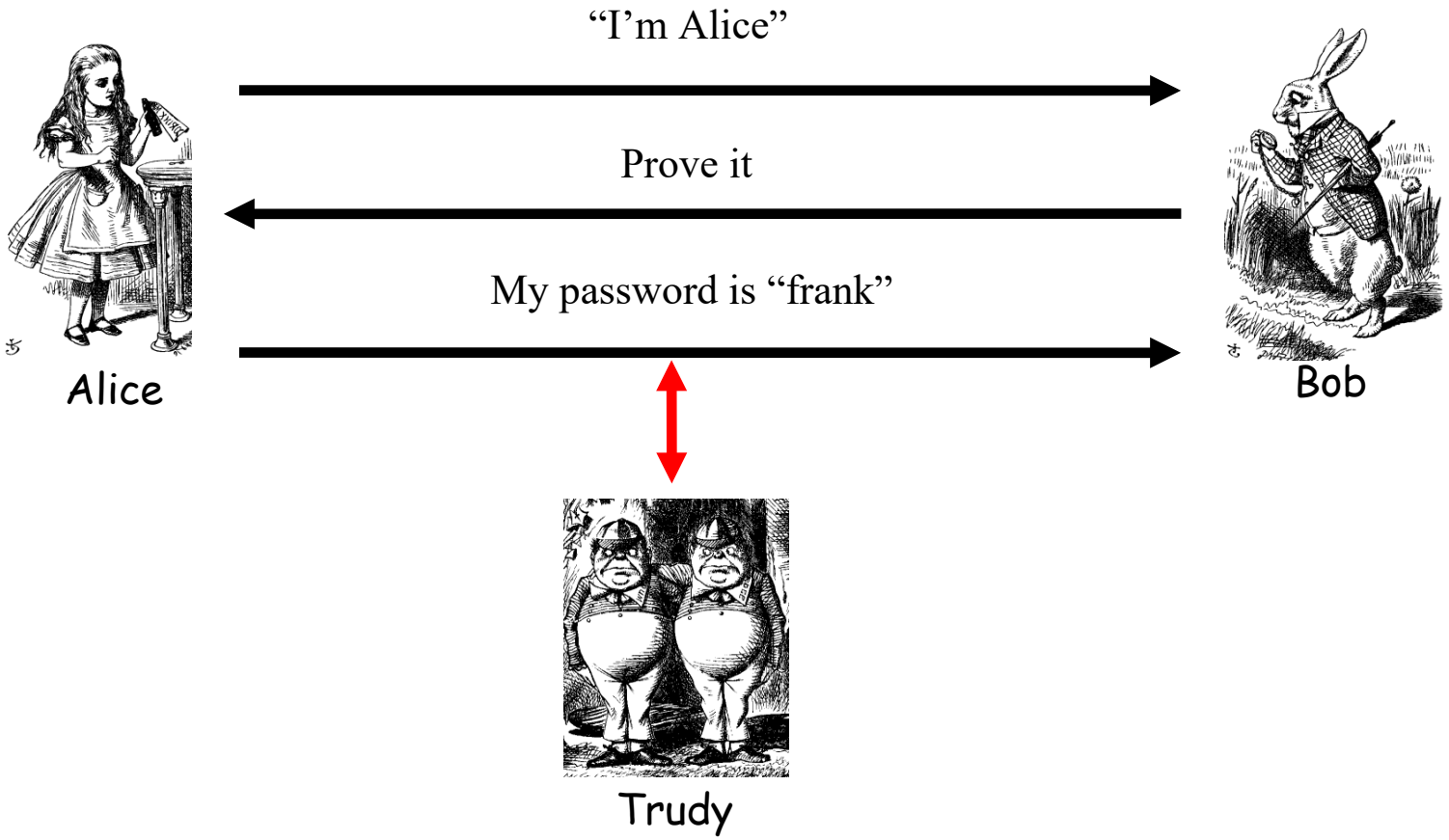
- Alice must prove her identity to Bob
  - Alice and Bob can be humans or **computers**
- May also require Bob to prove he's Bob (mutual authentication)
- Probably need to establish a **session key**
- May have other requirements, such as
  - Public keys, symmetric keys, hash functions, …

Simple Authentication

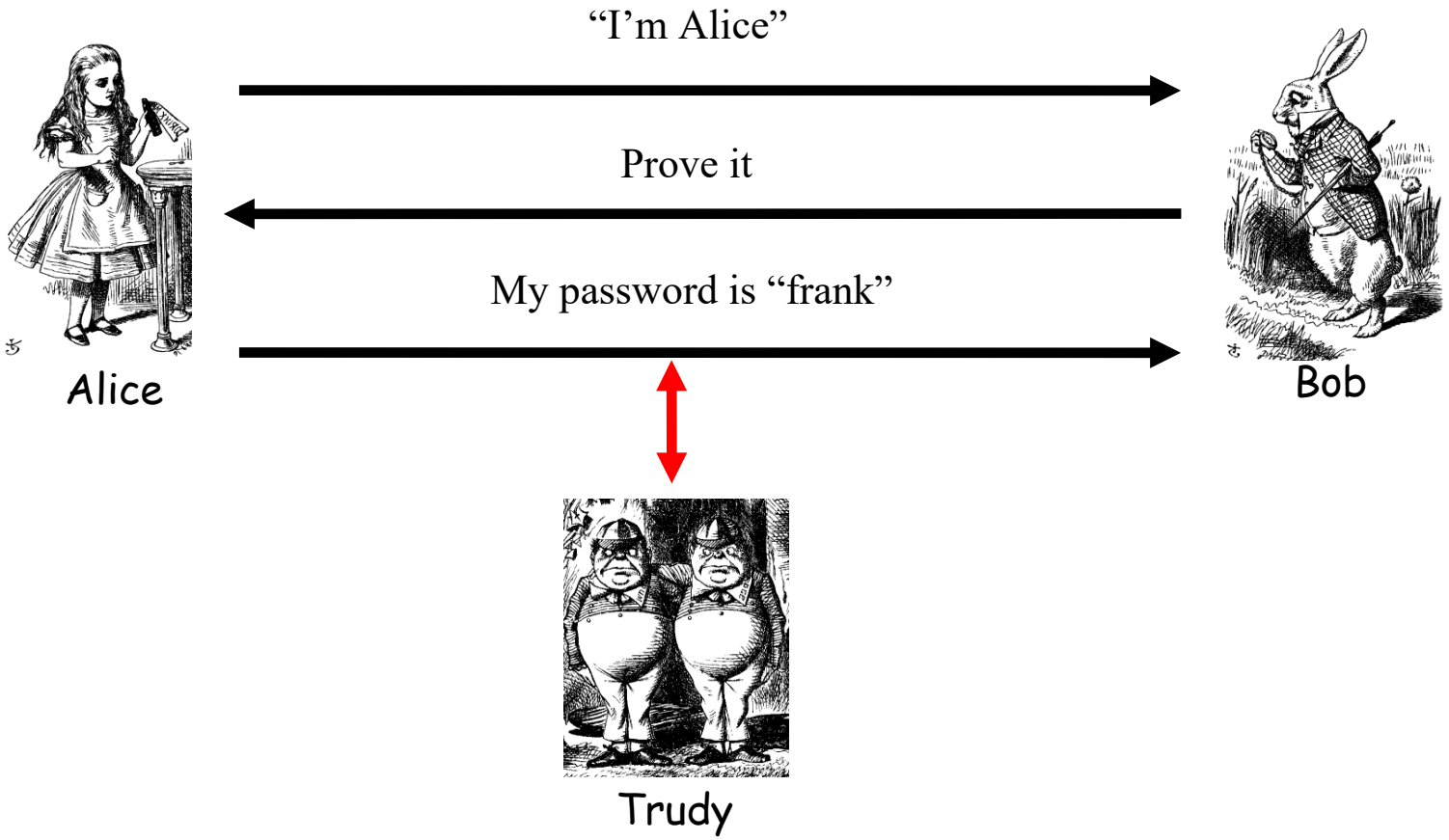


- Simple and may be OK for standalone system
- But highly insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
  - Also, Bob must know Alice's password

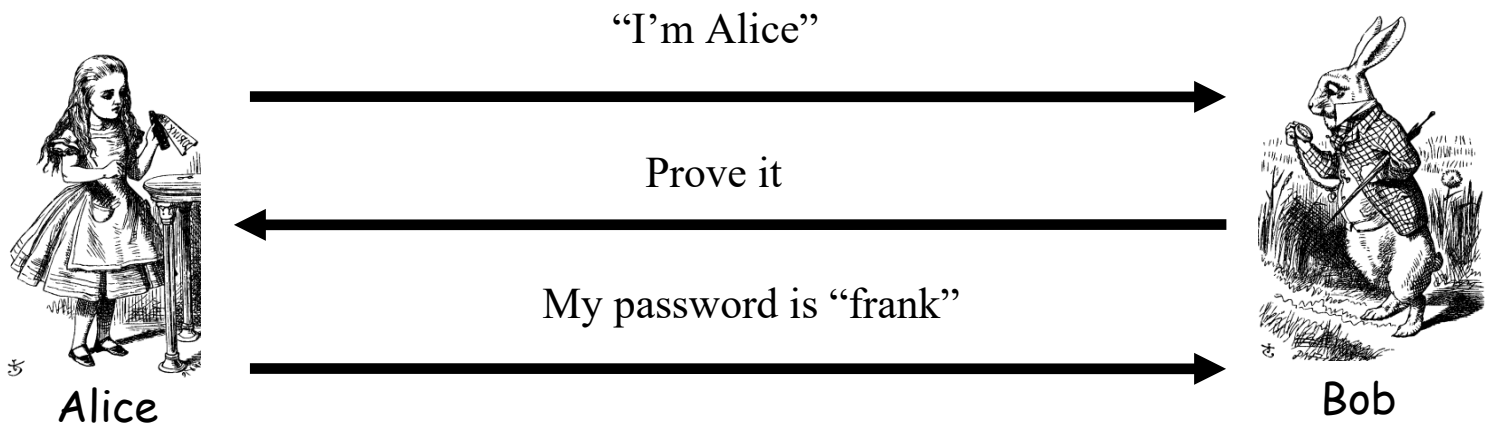
Authentication Attack



Authentication Attack



Simple Authentication



- Simple and may be OK for standalone system
- But highly insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
  - Also, Bob must know Alice's password

Challenge-Response

- To prevent replay, use ***challenge-response***
  - Goal is to ensure "freshness"
- Suppose Bob wants to authenticate Alice
  - ***Challenge*** sent from Bob to Alice
- Challenge is chosen so that…
  - Replay is not possible
  - Only Alice can provide the correct ***response***
  - Bob can verify the response

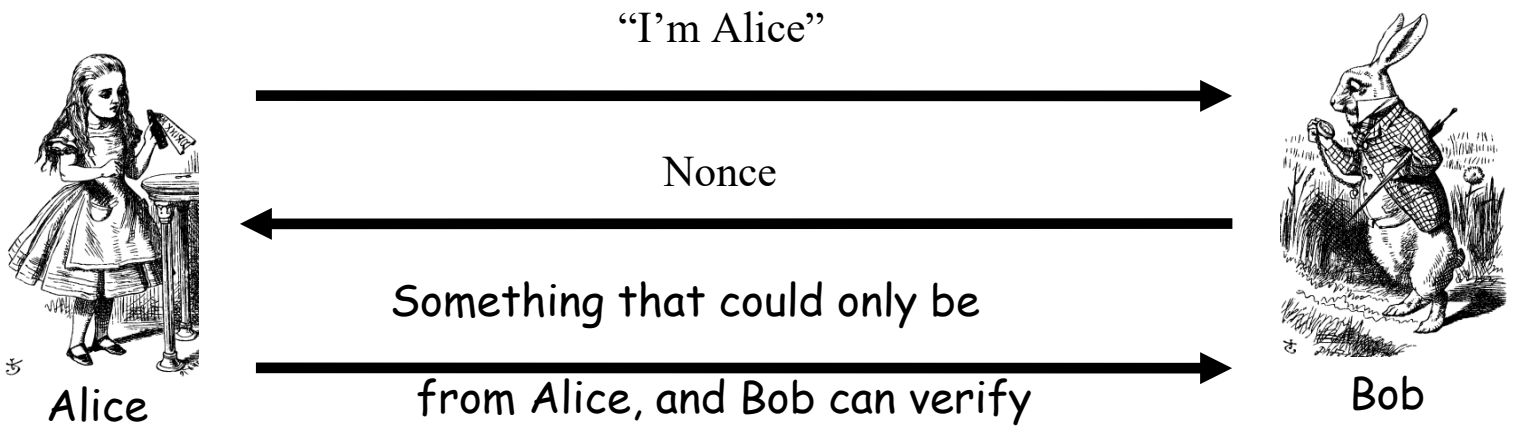
Nonce

- To ensure freshness, can employ a **nonce**
  - Nonce == **n**umber used **once**
- What to use for nonces?
  - That is, what is the challenge?
- What should Alice do with the nonce?
  - That is, how to compute the response?
- How can Bob verify the response?
- Should we use passwords or keys?

Challenge-Response

- To prevent replay, use ***challenge-response***
  - Goal is to ensure "freshness"
- Suppose Bob wants to authenticate Alice
  - ***Challenge*** sent from Bob to Alice
- Challenge is chosen so that…
  - Replay is not possible
  - Only Alice can provide the correct ***response***
  - Bob can verify the response

Generic Challenge-Response



- In practice, how to achieve this?
- Hashed password works, but…
- …encryption is much better here (why?)

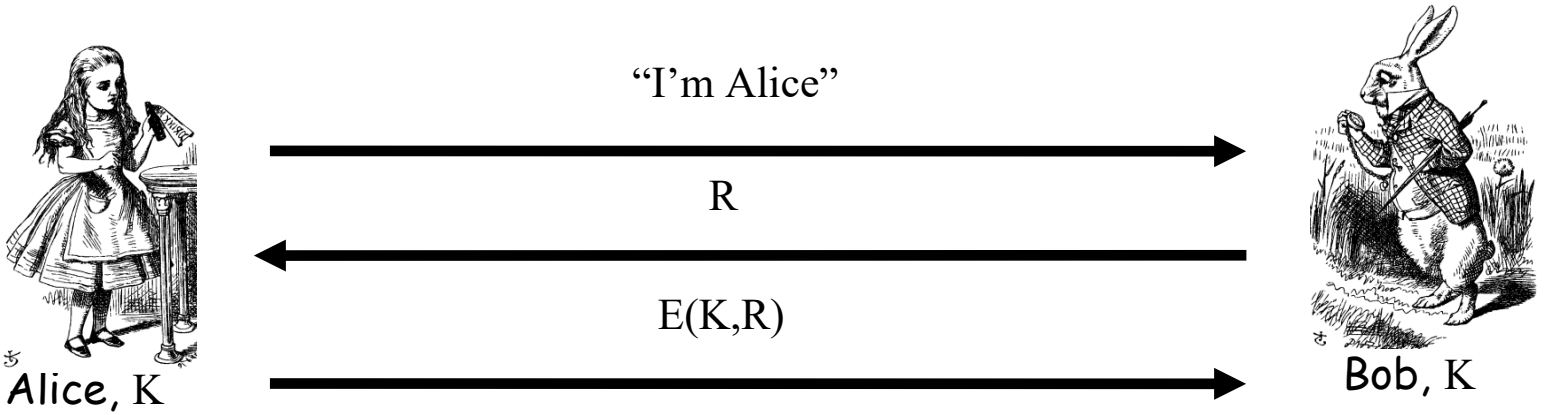
Symmetric Key Notation

- Encrypt plaintext P with key K
- $$C = E(K,P)$$
- Decrypt ciphertext C with key K
- $$P = D(K,C)$$
- Here, we are concerned with attacks on protocols, ***not*** attacks on cryptography
  - So, we assume crypto algorithms are secure

Authentication: Symmetric Key

- Alice and Bob share symmetric key K
- Key K known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, …

Authenticate Alice Using Symmetric Key



- Secure method for Bob to authenticate Alice
- But, Alice does not authenticate Bob
- So, can we achieve mutual authentication?

Network Security Protocols

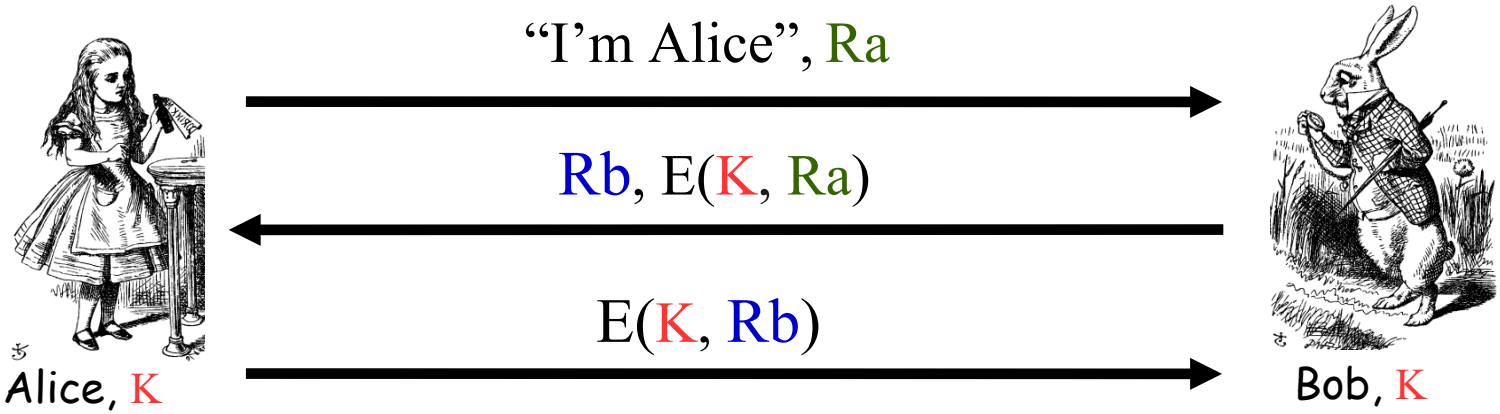
Part III

Mutual Authentication

Mutual Authentication

- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob

Mutual Authentication



- Unfortunately, this protocol is subject to a **reflection** attack

Mutual Authentication Attack

Trudy

1. “I’m Alice”, Ra

2. Rb, E(K, Ra)

5. E(K, Rb)

Bob, K

Trudy

3. “I’m Alice”, Rb

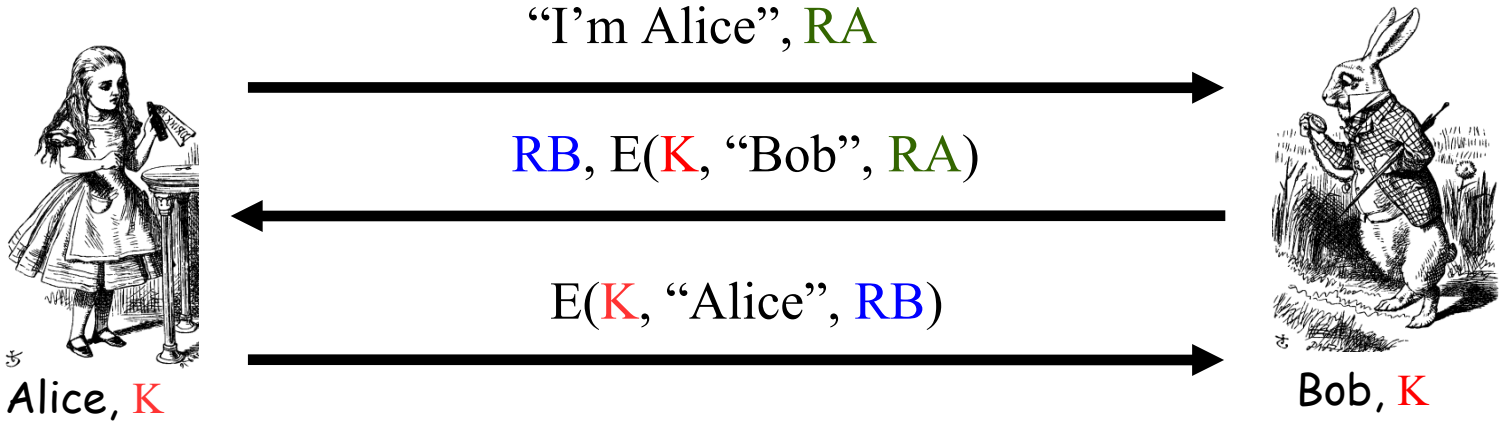
4. Rc, E(K, Rb)

Bob, K

Mutual Authentication

- Our one-way authentication protocol is **not** secure for mutual authentication
  - Protocols are subtle!
  - In this case, "obvious" solution is not secure
- Also, if assumptions or environment change, protocol may not be secure
  - This is a common source of security failure
  - For example, Internet protocols

Symmetric Key Mutual Authentication



- Do these "insignificant" changes help?
- Yes!

Network Security Protocols

Part IV

Perfect Forward Secrecy

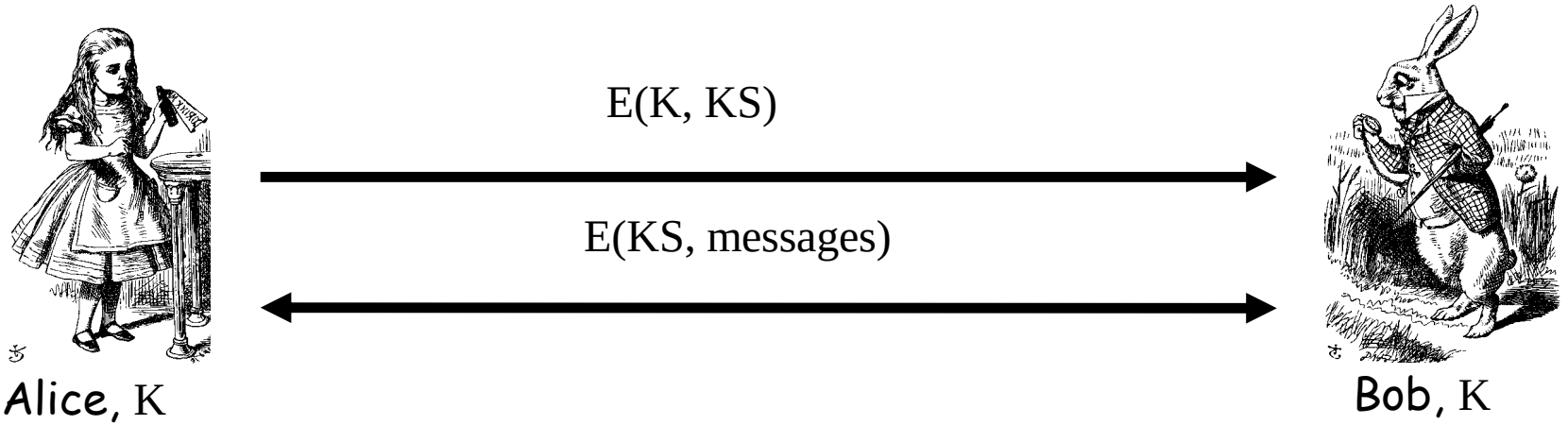
Perfect Forward Secrecy

- Consider this "issue"…
  - Alice encrypts message with shared key K and sends ciphertext to Bob
  - Trudy records ciphertext and later attacks Alice's (or Bob's) computer to recover K
  - Then Trudy decrypts recorded messages
- **Perfect forward secrecy (PFS):** Trudy cannot later decrypt recorded ciphertext
  - Even if Trudy gets key K or other secret(s)
- Is PFS possible?

Perfect Forward Secrecy

- Suppose Alice and Bob share key K
- For perfect forward secrecy, Alice and Bob cannot use K to encrypt
- Instead they must use a session key KS and forget it after it's used
- Can Alice and Bob agree on session key KS in a way that provides PFS?

Naïve Session Key Protocol



- Trudy could record E(K, KS)
- If Trudy later gets K then she can get KS
  - Then Trudy can decrypt recorded messages
- ***No*** perfect forward secrecy in this case

Perfect Forward Secrecy

- We can use **Diffie-Hellman** for PFS
- Recall: public g and p

Alice, a

$g^a \bmod p$

$g^b \bmod p$

Bob, b

- But Diffie-Hellman is subject to MiM attack
- How to get PFS and prevent MiM?

Perfect Forward Secrecy

Alice: K, a

E(K, g^a mod p)

E(K, g^b mod p)

Bob: K, b

- Session key KS = g^{ab} mod p
- Alice **forgets** a, Bob **forgets** b
- Recall: **Ephemeral Diffie-Hellman**
- Neither Alice nor Bob can later recover KS
- Are there other ways to achieve PFS?

Mutual Authentication, Session Key and PFS

Alice

"I'm Alice", RA

RB, [RA, g^b mod p]Bob

[RB, g^a mod p]Alice

Bob

- Session key is K = g^{ab} mod p
- Alice forgets a and Bob forgets b
- If Trudy later gets Bob's and Alice's secrets, she cannot recover session key K

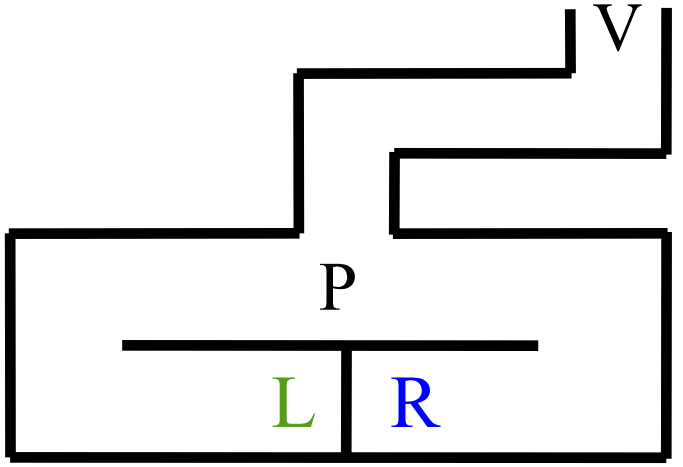
Network Security Protocols
Part V
Zero-knowledge proofs (protocols)

Zero-Knowledge Proof (ZKP)

- Alice wants to prove that she knows a secret without revealing **any** info about it
- Bob must verify that Alice knows secret
  - But, Bob gains no information about the secret
- Process is probabilistic
  - Bob can verify that Alice knows the secret to an arbitrarily high probability
- An "interactive proof system"

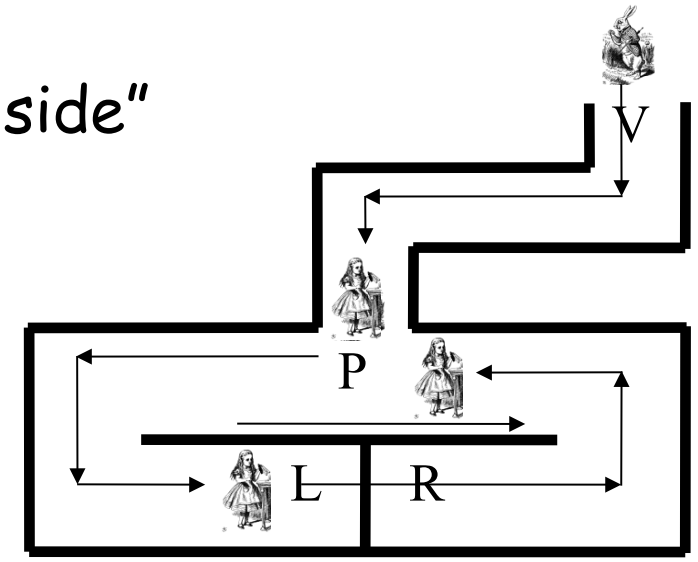
Bob's Cave

- Alice knows secret phrase to open path between L and R ("افتح يا سمسم")
- Can Alice (P) convince Bob(V) that she knows the secret without revealing phrase?



Bob's Cave

- Bob: "Alice, come out on R side"
- Alice (quietly):
- "افتح يا سمسم"
- If Alice does not know the secret…

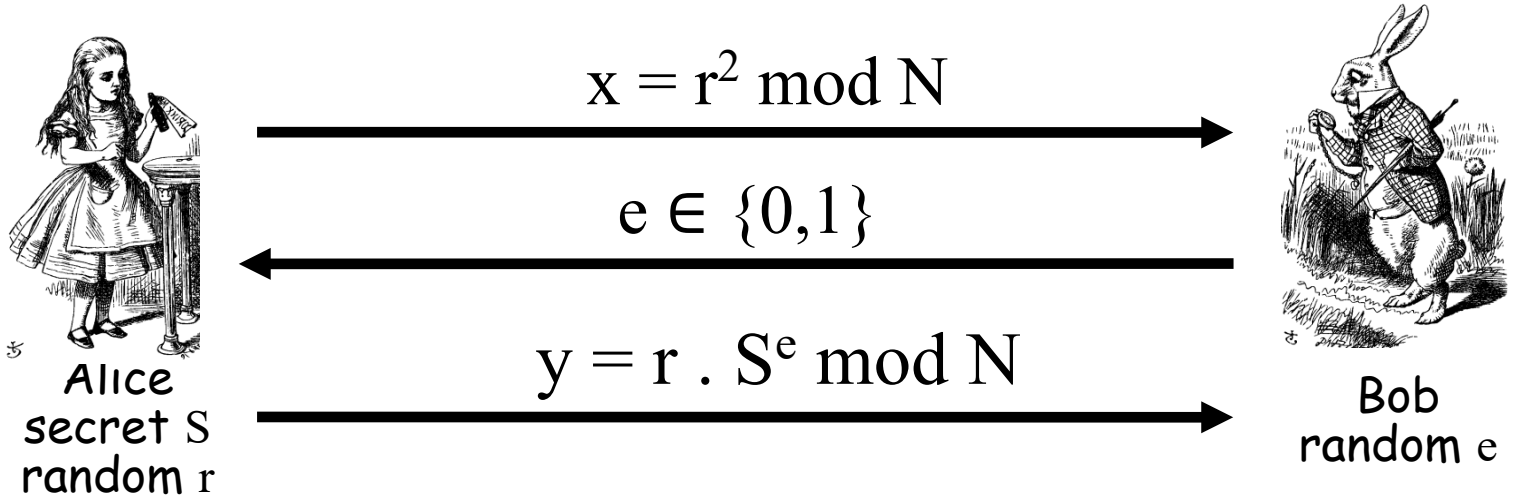


- …then Alice could come out from the correct side with probability $1/2$
- If Bob repeats this n times and Alice does not know secret, she can only fool Bob with probability $1/2^n$

Fiat-Shamir Protocol

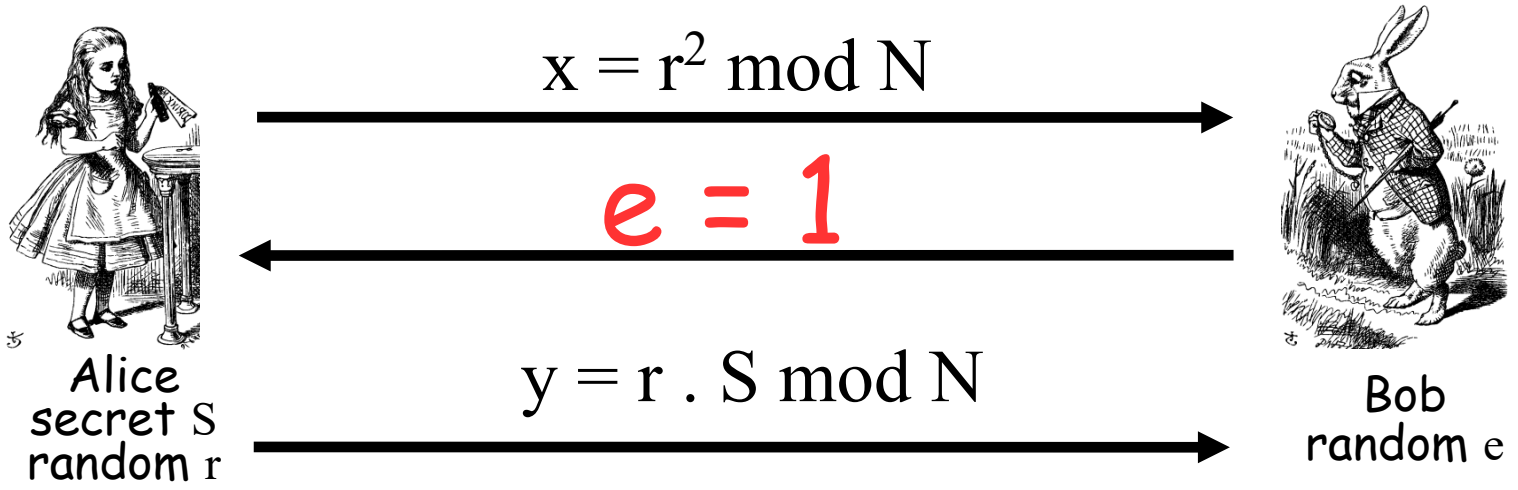
- Cave-based protocols are inconvenient
  - Can we achieve same effect without the cave?
- Finding square roots modulo N is difficult
  - Equivalent to factoring
- Suppose $N = pq$, where p and q prime
- Alice has a secret S
- N and $v = S^2 \bmod N$ are **public**, S is **secret**
- Alice must convince Bob that she knows S without revealing any information about S

Fiat-Shamir



- **Public:** N and $v = S^2 \bmod N$
- Alice selects random r, Bob chooses $e \in \{0,1\}$
- Bob verifies: $\mathbf{y^2 = x \cdot v^e \bmod N}$
  - Note that $y^2 = r^2 \cdot S^{2e} = r^2 \cdot (S^2)^e = x \cdot v^e \bmod N$

Fiat-Shamir: e = 1

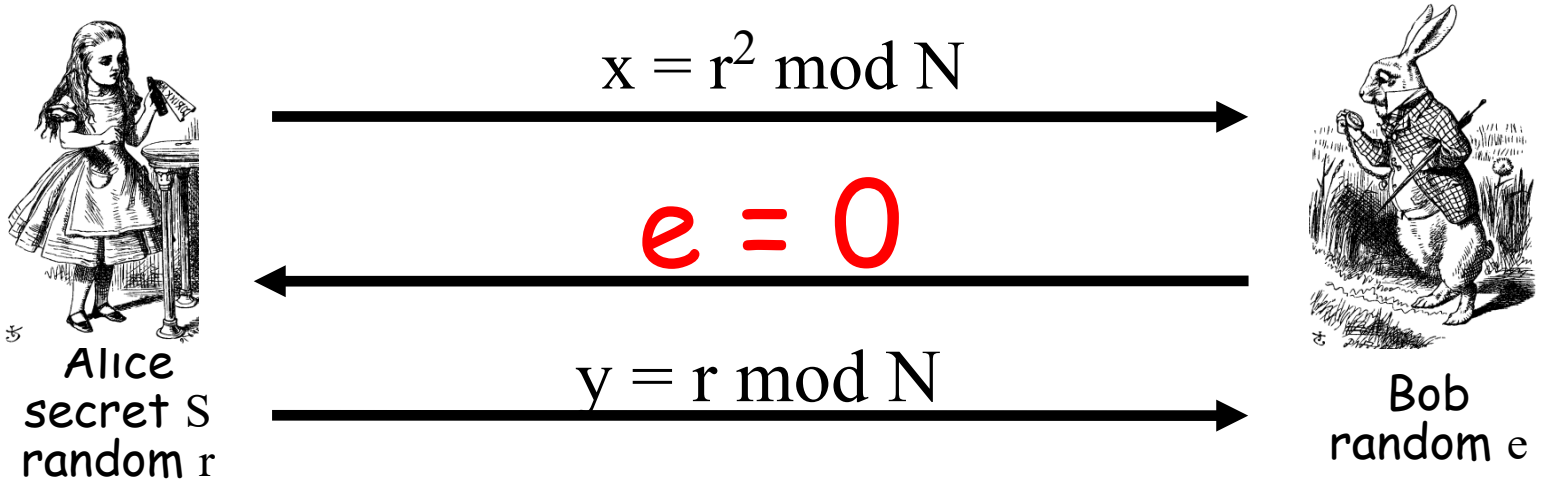


- **Public:** N and $v = S^2 \bmod N$

Alice selects random r, Bob chooses e = 1

- If $y^2 \stackrel{?}{=} x \,.\, v \bmod N$ then Bob accepts it
  - And Alice passes this iteration of the protocol
- Note that Alice must know S in this case

Fiat-Shamir: e = 0



- **Public:** N and $v = S^2 \bmod N$

Alice selects random r, Bob chooses e = 0

- Bob must checks whether $y^2 \stackrel{?}{=} x \bmod N$
- "Alice" does **not** need to know S in this case!

Fiat-Shamir

- **Public:** N and $v = S^2 \bmod N$
- **Secret:** Alice knows S
- Alice selects random r and **commits** to r by sending $x = r^2 \bmod N$ to Bob
- Bob sends **challenge** $e \in \{0,1\}$ to Alice
- Alice **responds** with $y = r \,.\, S^e \bmod N$
- Bob checks whether $y^2 \stackrel{?}{=} x \,.\, v^e \bmod N$
  - Does this prove response is from Alice?

Tricking Bob

- If Trudy expects $e = 0$, she may:
  - send $x = r^2$ in msg 1 and $y = r$ in msg 3
  - Bob verifies: $y^2 \stackrel{?}{=} x \bmod N$
    - $r^2 \stackrel{?}{=} r^2 \Longrightarrow$ Bob is tricked!
- If Trudy expects $e = 1$, she may:
  - send $x = r^2 \cdot v^{-1}$ in msg 1 and $y = r$ in msg 3
  - Bob verifies: $y^2 \stackrel{?}{=} x \cdot v \bmod N$
    - $r^2 \stackrel{?}{=} r^2 \cdot v^{-1} \cdot v \Longrightarrow r^2 \stackrel{?}{=} r^2 \Longrightarrow$ Bob is tricked!
- But, if Bob chooses $e \in \{0,1\}$ at random, Trudy can only trick Bob with probability $1/2$

Fiat-Shamir Facts

- Trudy can trick Bob with probability $1/2$, but…
  - …after n iterations, the probability that Trudy can convince Bob that she is Alice is only $1/2^n$
  - Just like Bob's cave!
- Bob's $e \in \{0,1\}$ must be unpredictable
- Alice must use new r each iteration, or else…
  - If $e = 0$, Alice sends $r \bmod N$ in message 3
  - If $e = 1$, Alice sends $r \cdot S \bmod N$ in message 3
  - Anyone can find S given $r \bmod N$ and $r \cdot S \bmod N$

Fiat-Shamir Zero Knowledge?

- Zero knowledge means that nobody learns ***anything*** about the secret S
  - **Public:** $v = S^2 \bmod N$
  - Trudy sees $r^2 \bmod N$ in message 1
  - Trudy sees $r \,.\, S \bmod N$ in message 3 (if $e = 1$)
- If Trudy finds r from $r^2 \bmod N$, she gets S
  - If Trudy could find modular square roots, she could get S from **public** v
- Protocol does not seem to "help" to find S

ZKP properties

1) Zero-knowledge. If the statement is true, no dishonest verifier learns anything other than the fact that the statement is true.
2) Completeness. If the statement is true, an honest verifier will be convinced by an honest prover.
3) Soundness. If the statement is false, no dishonest prover can convince an honest verifier.

ZKP in the Real World

- zCash uses zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge)
- Able to prove that the conditions for a valid transaction have been satisfied without revealing any crucial information about the addresses or values involved
- Other applications, location, age, bank balance, etc..
- ZKP is **not** just pointless mathematics!

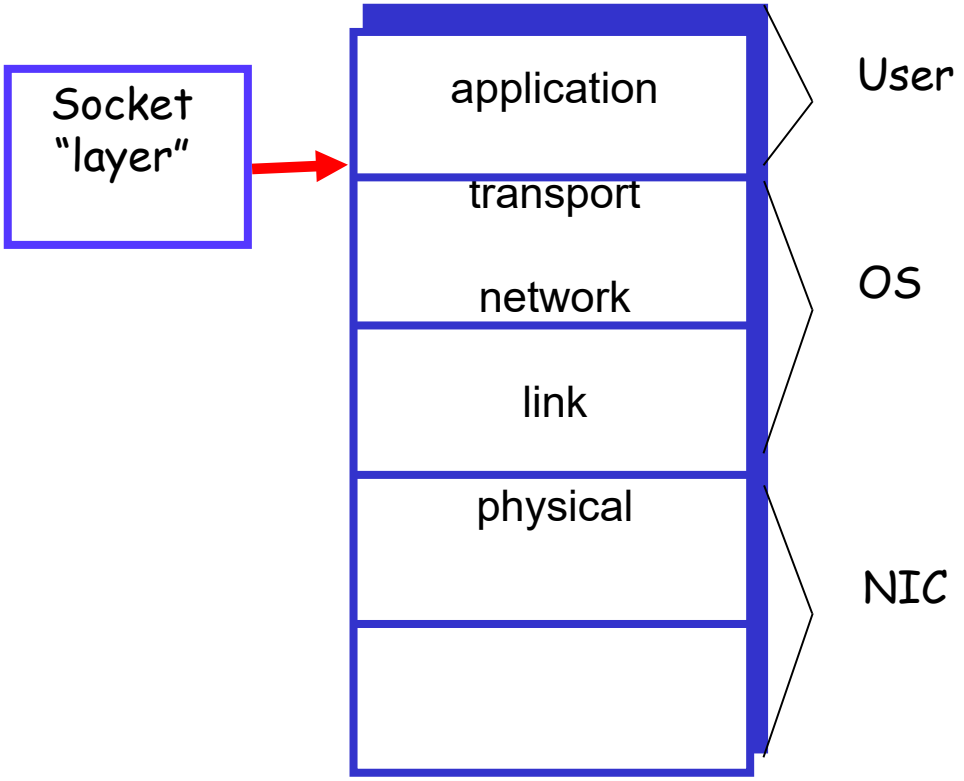
Network Security Protocols
Part VI

Real-world protocols (I)

SSL/TLS
Kerberos


Secure Socket Layer

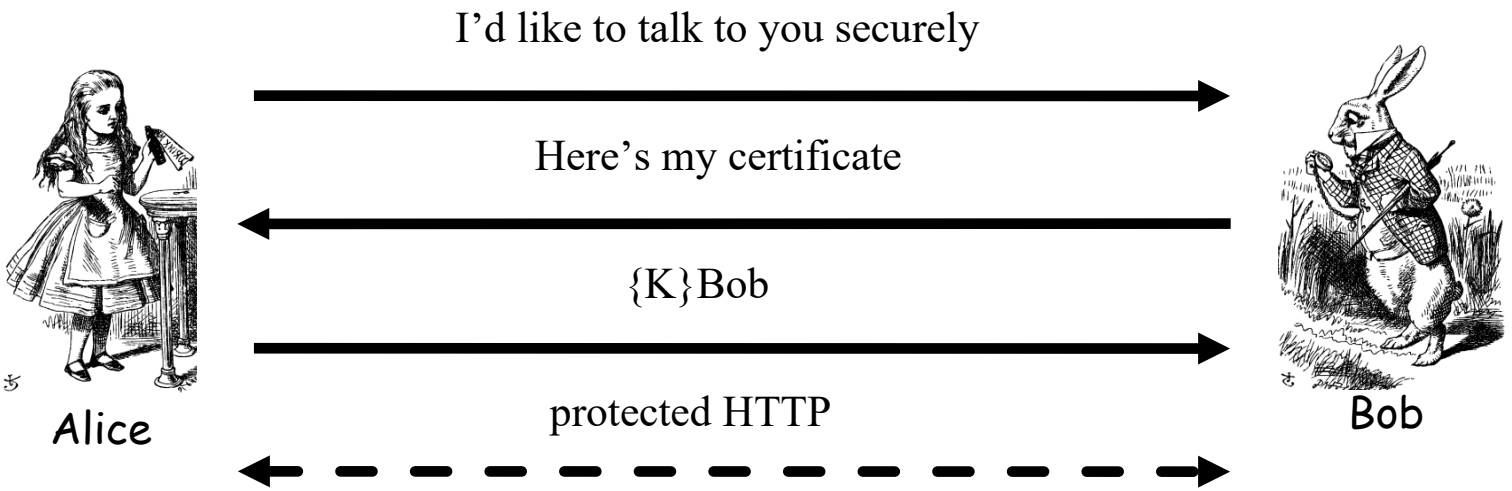
- "Socket layer" lives between application and transport layers
- SSL usually between HTTP and TCP
- Upgraded and standardized to Transport Layer Security (TLS)



What is SSL/TLS?

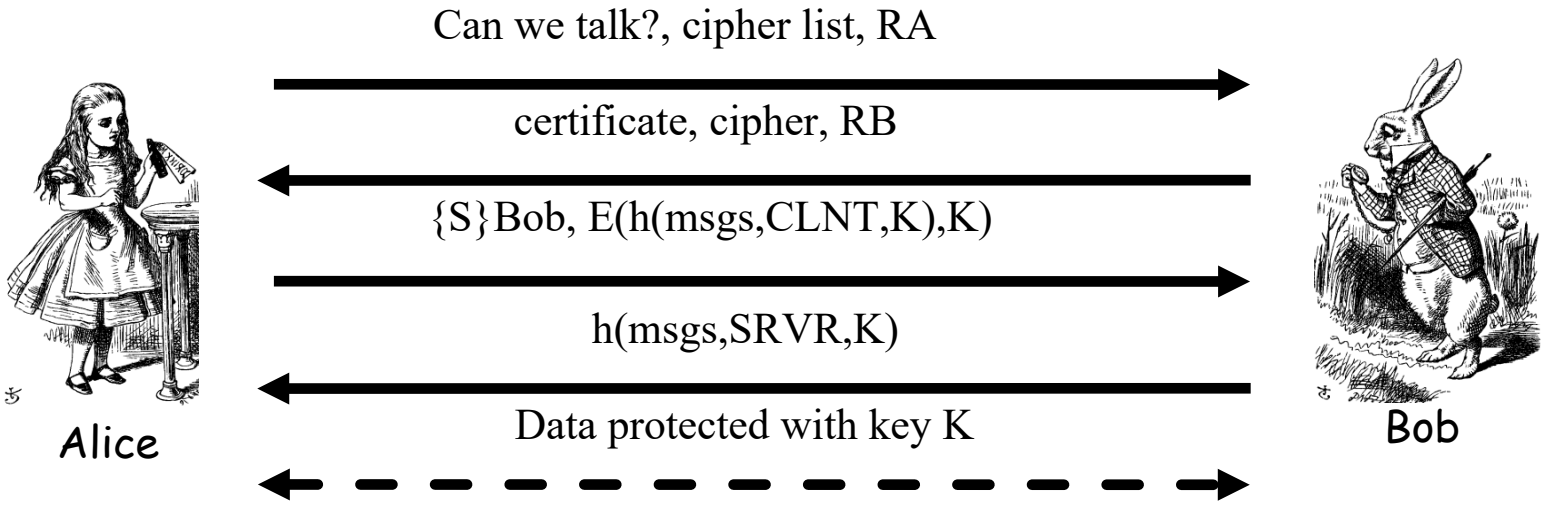
- SSL/TLS is the protocol used for majority of secure Internet transactions today
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - As long as you have money, Amazon does not really care who you are…
  - …so, no need for mutual authentication, you need a secure channel with Amazon,
  - and that's what SSL/TLS is providing

Simple SSL-like Protocol



- Is Alice sure she's talking to Bob?
- Is Bob sure he's talking to Alice?

Simplified SSL Protocol



- S is the so-called **pre-master secret**
- K = h(S,RA,RB)
- "msgs" means all previous messages
- CLNT and SRVR are constants

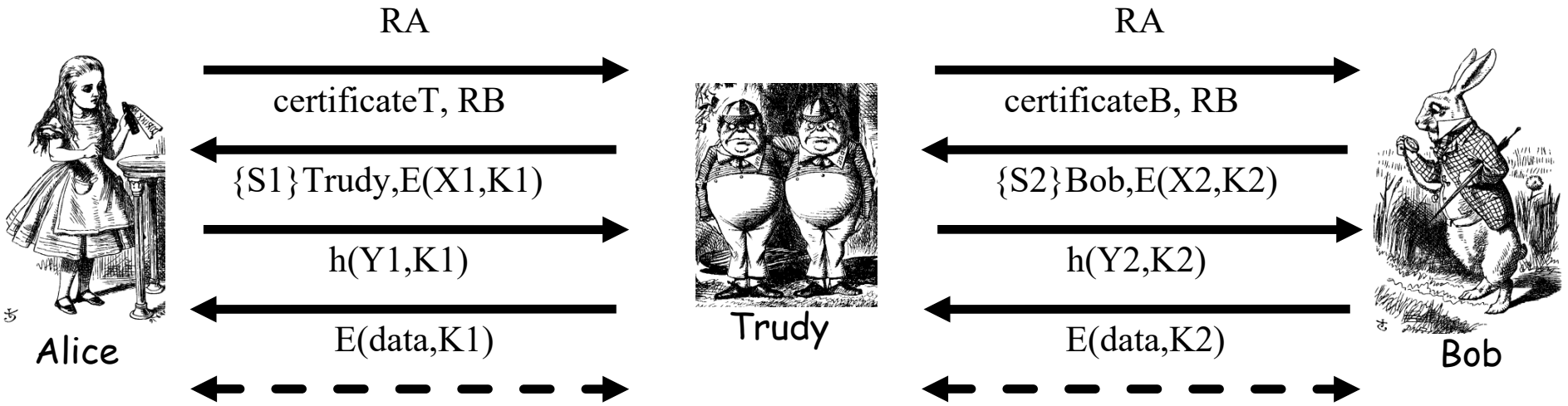
SSL Keys

- 6 "keys" derived from $K = h(S,RA,RB)$
  - 2 encryption keys: client and server
  - 2 integrity keys: client and server
  - 2 IVs: client and server
- Different keys in each direction

SSL Authentication

- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
  - Why would server not authenticate client?
- Mutual authentication is possible: Bob sends **certificate request** in message 2
  - Then client must have a valid certificate
  - But, if server wants to authenticate client, server could instead require password

SSL MiM Attack?



- Q: What prevents this MiM "attack"?
- A: Bob's certificate must be signed by a certificate authority (CA)
- What does browser do if signature not valid?
- What does user do when browser complains?

Network Security Protocols
Part VII

Real-world protocols (II)

Kerberos

- In Greek mythology, Kerberos is 3-headed dog that guards entrance to Hades

Kerberos

- In security, Kerberos is an authentication protocol based on symmetric key crypto
  - Originated at MIT, released 1980
  - Part of Athena project
  - Based on Needham-Schroeder protocol
  - Relies on a **Trusted Third Party (TTP)**

Motivation

- Authentication using public keys
  - N users –> N key pairs
- Authentication using symmetric keys
  - N users requires (on the order of) N^2 keys
- Symmetric key case **does not scale**
- Kerberos based on symmetric keys but only requires N keys for N users
  - Security depends on TTP
  - No PKI is needed

Key Distribution Center

- Kerberos **Key Distribution Center** or **KDC**
  - KDC acts as the Trusted Third Party (TTP)
  - TTP is trusted, so it must not be compromised
- KDC shares symmetric key K_A with Alice, key K_B with Bob, key K_C with Carol, etc.
- KDC holds a master key K_{KDC} known ***only*** to him
- KDC enables:
  - authentication
  - confidentiality and integrity via session keys
- Crypto algorithm was DES, now AES

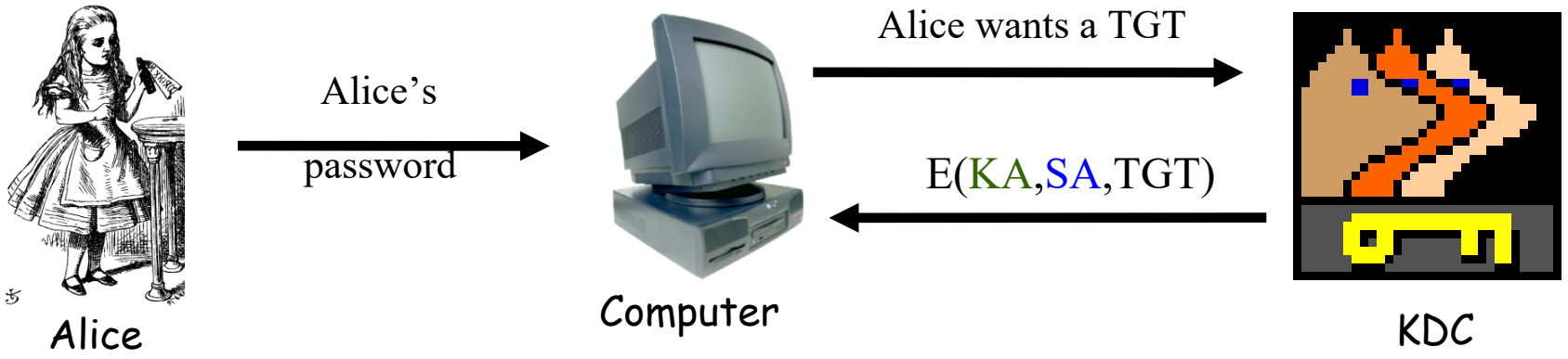
Tickets

- KDC issue **tickets** containing info needed to access network resources
- KDC also issues **Ticket-Granting Tickets** or **TGTs** that are used to obtain tickets
- Each TGT contains
  - Session key
  - User's ID
  - Expiration time
- Every TGT is encrypted with KKDC
  - So, TGT can only be read by the KDC

Login steps

1. Alice enters her password
2. Then Alice's computer does following:
   I. Derives KA from Alice's password
   II. Uses KA to get TGT for Alice from KDC
3. Alice then uses her TGT (credentials) to securely access network resources

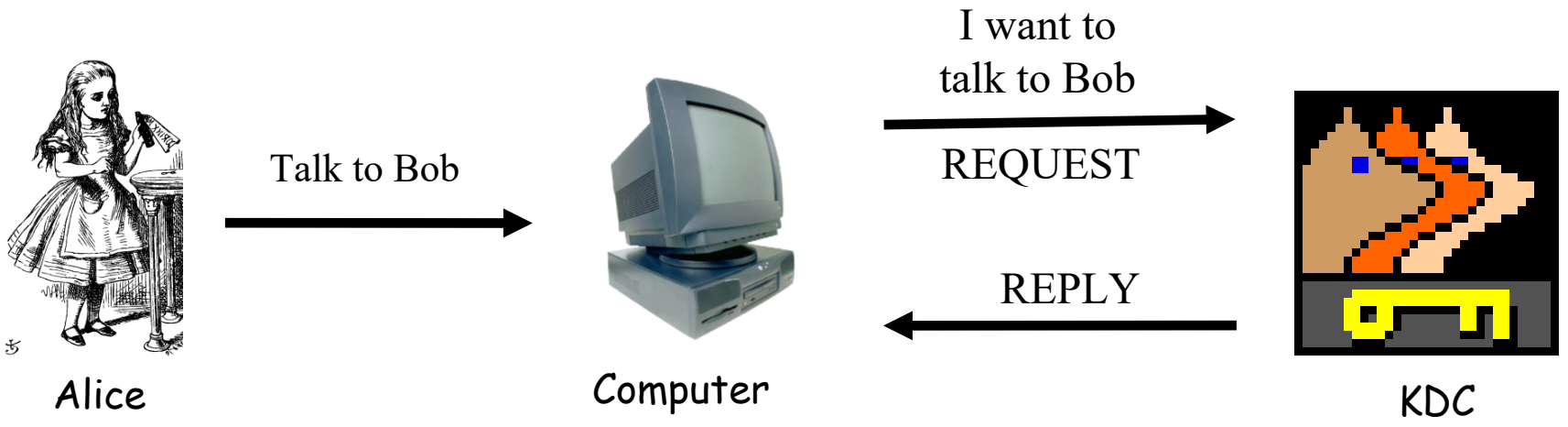
Kerberos Login



- Key KA = h(Alice's password)
- KDC creates session key SA
- Alice's computer decrypts SA and TGT
  - Then it forgets KA

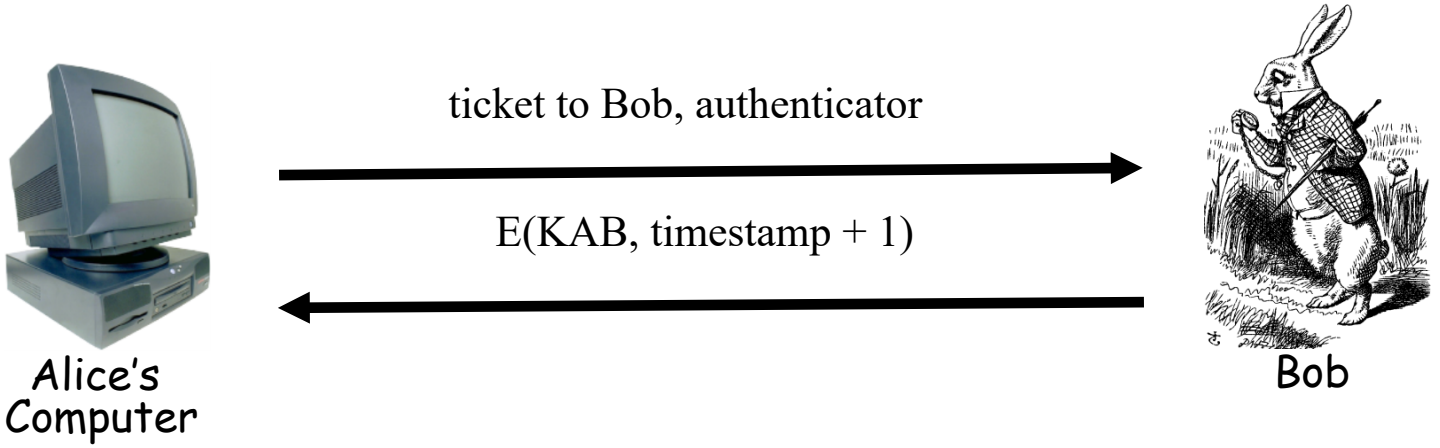
TGT = E(KKDC, "Alice", SA)

Alice Requests "Ticket to Bob"



- REQUEST = (TGT, authenticator)
  - TGT = E(KDCC, "Alice", SA)
  - authenticator = E(SA, timestamp)
- REPLY = E(SA, "Bob", KAB, ticket to Bob)
  - ticket to Bob = E(KB, "Alice", KAB)
- KDC gets SA from TGT to verify timestamp

Alice Uses Ticket to Bob



- ticket to Bob = E(KB, "Alice", KAB)
- authenticator = E(KAB, timestamp)
- Bob decrypts "ticket to Bob" to get KAB which he then uses to verify timestamp

Kerberos

- Key SA used in authentication
  - For confidentiality/integrity
- Timestamps for authentication and replay protection
- Timestamps…
  - Reduce the number of messages, like a nonce that is known in advance
  - But, "time" is a security-critical parameter

Kerberos + -

- **Plus**: Security is transparent to Alice
  - "MIT provides Kerberos in source form so that anyone who wishes to use it may look over the code for themselves and assure themselves that the code is trustworthy"
- **Minus**: KDC ***must*** be secure; it's trusted!

Access Control

Part I

Introduction
&
User Identification

Access Control

- Access control (AC) is the process of by which a (computer) system manages the identities and privileges of its entities (person, machine, etc.).
  - Prevent malicious access to resources.
  - Allow unmalicious access to resources.
- AC has three parts:
  - **Identification**: Who are you?
  - **Authentication:** Are you who you say you are?
  - **Authorization:** Are you allowed to do your request?
- Note: AC often used as synonym for Authorization

A “physical” example

- Me going to the office of Dr. Omar Alrazzaz and saying to the guard: “Hello, I am Omar Amousa, I have an appointment with Dr. Alrazzaz”
  - Identification
- The guard asking me for a proof of identity, me providing it (e.g., my ID), and the guard checking it.
  - Authentication
- The guard checking the appointments of Dr. Alrazzaz and allowing me in
  - Authorization

A digital example

- Me going to JUST employees' portal and inserting: omaralmousa@cit.just.edu.jo, in *userid* field then hitting sign-in.
  - Identification
- The system asking for my password, me providing it, and the system checking it.
  - Authentication
- The system allowing me to access the marks of my students.
  - Authorization

Identification

- Identification is to specify a character.
  - A claim, an assertion without proof
  - Examples:
    - A user entering his user-name
    - A person standing in front of a receptionist saying: “I am Frankenstein”

Identification mechanisms

- Common mechanisms:
  - User-name:
    - Unique identifier
    - Not secret
  - Bio-metrics:
    - May serve identification / authentication
    - Discuss later in authentication
  - Access card:
    - May serve identification / authentication
    - Discuss later in authentication

Which one to choose?

- Regardless of the used mechanism/technology, the identification system MUST identify users uniquely.

Registration

- Users must obtain (be given) initial credentials.
- Registration consists of 4 steps.

1)Requesting credentials.

2)Approving the request.

  - This person should be someone different from the requester. (**why**? Next slide)
  - An approval authority should be clearly defined in the organization's security policy.

3)Proving identity

4)Issuing credentials

Separation of duties

- Critical procedures (registration) must not be performed by a single individual (two at least!)
- A vital concept in information security.
- Examples?

Access Control

Part II

User Authentication – 1 of 2

Access Control

- Access control (AC) is the process of ensuring that a (computer) system maintains a **correct** view of all (of its) entities and their privileges to (its) resources.
  - Prevent malicious access to resources.
  - Allow unmalicious access to resources.
- AC has three parts:
  - **Identification**: Who are you?
  - **Authentication:** Are you who you say you are?
  - **Authorization:** Are you allowed to do your request?
- Note: AC often used as synonym for Authorization

Authentication

- **Authentication:** Are you who you say you are?
  - Determine whether access is allowed or not
  - Authenticate human to machine
  - Or, possibly, machine to machine
- Authentication can be based on:
  I. Something you **know**, e.g., a password
  II. Something you **have**, e.g., a smartcard
  III. Something you **are**, e.g., your fingerprint, iris

I. Something You Know

- Knowledge-based: Passwords
- Lots of things act as passwords!
  - PIN
  - Social security number
  - Date of birth
- Most popular:
  - **Cost**: passwords are free
  - **Convenience**: easy for admin to reset

Password Issues

"Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations"

- **Crypto keys**
- For 64 bits, then 2^{64} keys
- Choose key at random…
- …then attacker (Trudy) must try about 2^{63} keys

- **Passwords**
- For 8 char. password, and 256 different char, then $256^8 = 2^{64}$ passwords
- Users do not select passwords at random ...
- … Trudy has far less than 2^{63} passwords to try (**dictionary attack**)

Weak vs Strong Passwords

- **Weak passwords**
  - frank
  - Fido
  - Password
  - incorrect
  - Pikachu
  - 102560
  - AustinStamp
- **Strong Passwords?**
  - jflej,43j-EmmL+y
  - 09864376537263
  - P0kem0N
  - FSa7Yago
  - 0nceuP0nAt1m8
  - PokeGCTall150

Password Experiment

- 3 groups, each group **advised** to select passwords:
  - **Group A:** At least 6 chars, 1 non-letter
  - **Group B:** Password based on passphrase
  - **Group C:** 8 random characters
- Results
  - **Group A:** About 30% of passwords easy to crack
  - **Group B:** About 10% cracked, easy to remember
  - **Group C:** About 10% cracked, hard to remember
- User compliance hard to achieve: 1/3rd did not comply
- 10% of passwords are likely easy to crack.
- If passwords not assigned, best advice is…
  - Choose passwords based on passphrase
  - Use password cracking tool to test for weak pwds
  - require periodic password change

Attacks on Passwords

- Attacker could…
  - Target one particular account
  - Target any account on system
  - Target any account on any system
  - Attempt denial of service (DoS) attack
- Common attack path
  - Outsider >> normal user >> administrator
    - Access any account and then upgrade her level of privilege
  - May only require **one** weak password!

Password Retry

- Suppose system locks after 3 bad passwords, how long should it lock?
  - 5 seconds
    - Attacker(Trudy) might cycle through accounts!!
  - 5 minutes
    - Cause denial of service
- No apparent solution to this dilemma.

Password Storage

- Systems need to verify their users
  - Storing plain passwords is a bad idea
- As a solution: store **Hashed** passwords
  - Store y = h(password)
  - To verify, hash the given password and compare
- If Trudy obtains the password file, she does not (directly) obtain passwords
- But Trudy can try a *forward search:* guess x and check whether y = h(x)

Dictionary Attack

- Trudy pre-computes h(x) for all x in a **dictionary** of common passwords
- Suppose Trudy gets access to password file containing hashed passwords
  - She only needs to compare hashes to her pre-computed dictionary
  - After one-time work of computing hashes in dictionary, actual attack is trivial
- Can we prevent this forward search attack? Or at least make it more difficult?

Salt

- Hash password with **salt**
- Choose random salt s and compute
  y = h(password, **salt**)

and store (**salt**,y) in the password file

- Note that the salt is not secret
- Still easy to verify salted password
- But lots more work for Trudy
  - Why?

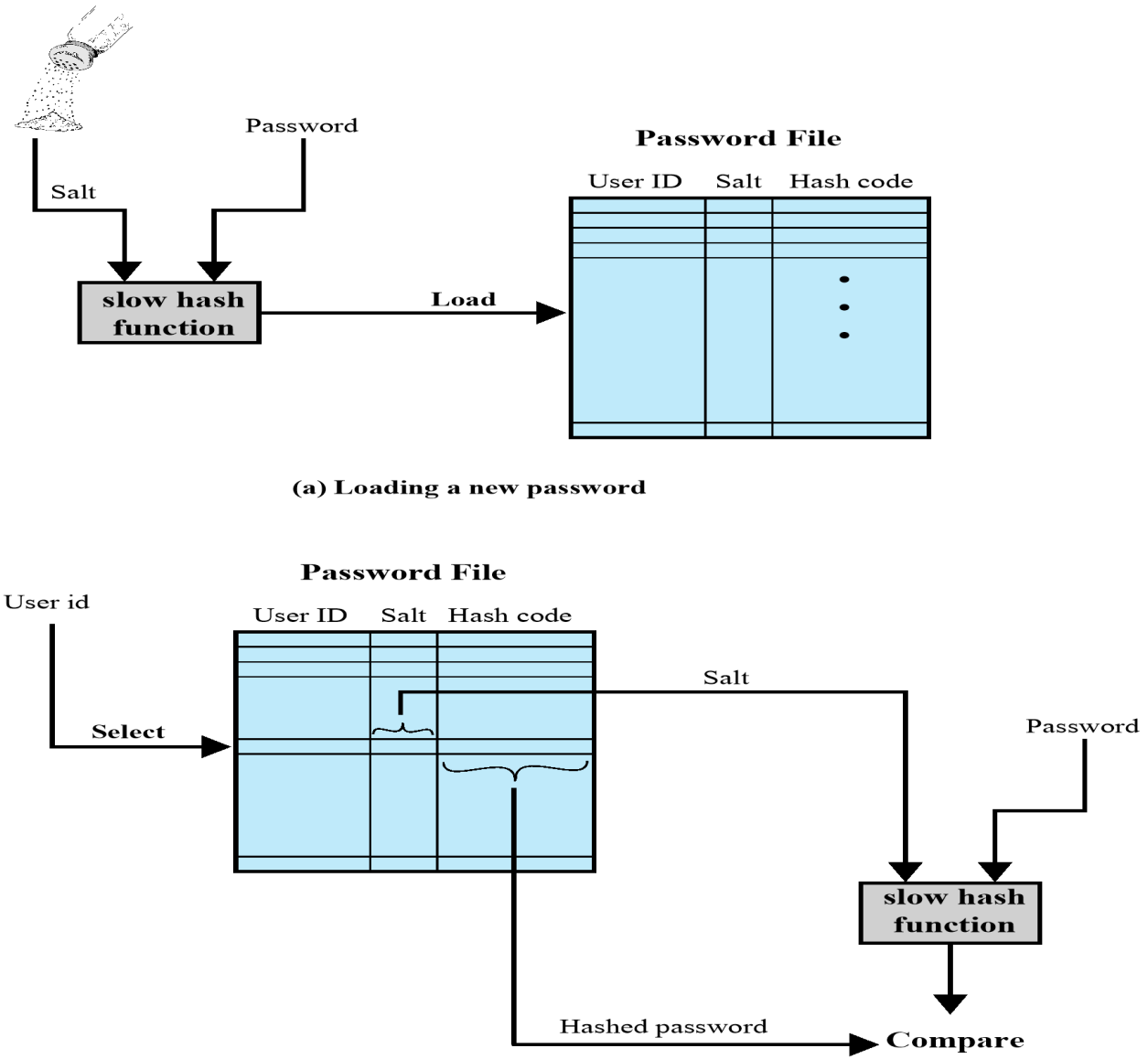


Figure 3.3 UNIX Password Scheme

Other Password Issues

- Too many passwords to remember
  - Results in password reuse
  - Why is this a problem?
- Social engineering
  - 34% of users will give their passwords if asked.
- keystroke logging, spyware, etc.
- Failure to change default passwords
- Who suffers from bad password?
  - ATM weak PIN: you lose
  - Work password: whole company loses

Password Cracking Tools

- Popular password cracking tools
  - Password Crackers
  - Password Portal
  - L0phtCrack and LC4 (Windows)
  - John the Ripper (Unix)
- Admins should use these tools to test for weak passwords since attackers will
- Good articles on password cracking
  - Passwords - Conerstone of Computer Security
  - Passwords revealed by sweet deal

Access Control

Part II

User Authentication – 2 of 2

II. Something You Are Biometrics

“You are your key”

- Examples
  - Fingerprint
  - Handwritten signature
  - Facial recognition
  - Speech recognition
  - Gait (walking) recognition
  - “Digital doggie” (odor recognition)
  - ....

Ideal Biometric

- **Universal:** applies to (almost) everyone
  - In reality, no biometric applies to everyone
- **Distinguishing:** distinguish with certainty
  - In reality, cannot hope for 100% certainty
- **Permanent:** physical characteristic being measured never changes
  - In reality, OK if it remains valid for long time
- **Collectable:** easy to collect required data
  - Depends on whether subjects are cooperative
- Also, safe, user-friendly, and ???

Identification vs Authentication

- **Identification:** Who are you?
  - Compare **one-to-many**
  - Example: suspicious fingerprint is sent to the FBI database of fingerprints for comparison with millions of fingerprints.
- **Authentication:** Are you who you say you are?
  - Compare **one-to-one**
  - Example: Thumbprint mouse
- Identification problem is more difficult
  - More “random” matches since more comparisons
- We are (mostly) interested in authentication

Enrollment vs Recognition

- **Enrollment phase**
  - Subject's biometric info put into database
  - Must carefully measure the required info
  - Because it is one-time work, it is OK if it is slow and multiple measurements are required
  - Must be very precise
- **Recognition phase**
  - Biometric detection, when used in practice
  - Must be quick and simple
  - But must be reasonably accurate

Cooperative Subjects?

- **Authentication**: cooperative subjects
- **Identification**: uncooperative subjects
- For example, facial recognition
  - Used in Las Vegas casinos to detect known cheaters (also, terrorists in airports, etc.)
  - Often, less than ideal enrollment conditions
  - Subject will try to confuse in recognition phase
- Cooperative subject makes it much easier
  - We are focused on authentication
  - So, we can assume subjects are cooperative

Biometric Errors

- **Fraud rate** versus **insult rate**
  - Fraud: occurs when a dishonest user is incorrectly granted access as an honest user (or granted access that must be denied)
    - Trudy authenticated as Alice (False Acceptance Rate-FAR)
  - Insult: occurs when an honest user is incorrectly denied access by that system
    - Alice not authenticated as Alice (False Rejection Rate-FRR)
- For biometrics, can decrease fraud or insult, but other one will increase
- For example
  - 99% voiceprint match: low fraud (FAR), high insult (FRR)
  - 30% voiceprint match: high fraud (FAR), low insult (FRR)
- **Equal error rate:** rate where fraud = insult

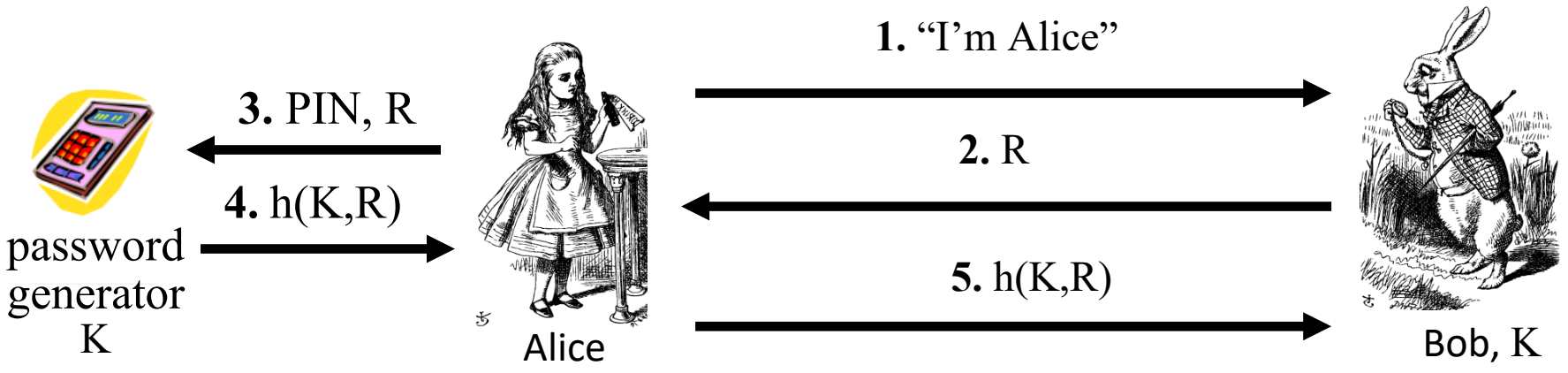
III. Something You Have

Figure 7.7: A Smartcard Reader (Courtesy of Athena, Inc.)

- Something you possess
- Examples include following…
  - Access cards: ATM card, smartcard, etc.
  - Car key
  - Laptop computer (or MAC address)
  - Password generators
  - Tokens (physical or soft)

- A smartcard is a credit card sized device that includes a small mount of memory and computing resources, so that it is able to store cryptographic keys or other secrets, and perhaps even do some computations on the card.
- A special purpose smartcard reader is used to read the key stored on the card.
- Then the key can be used to authenticate the user. Since a key is used, and keys are selected at random, password guessing attacks can be eliminated

Password Generator

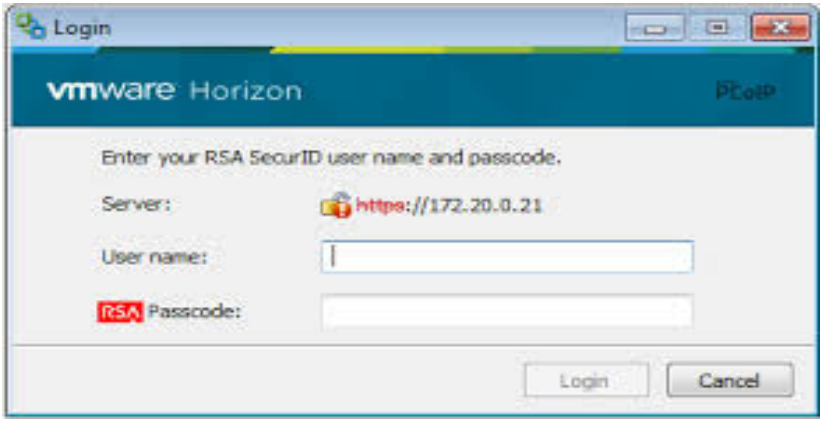


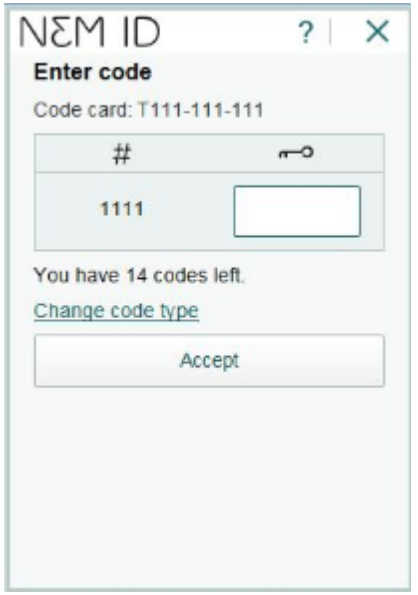
- Alice receives random "challenge" R from Bob
- Alice enters PIN and R in password generator
- Password generator hashes symmetric key K with R
- Alice sends "response" h(K,R) back to Bob
- Bob verifies response
  - Both Bob and the password generator have to have the key K
- Note: Alice **has** pwd generator and **knows** PIN

Tokens

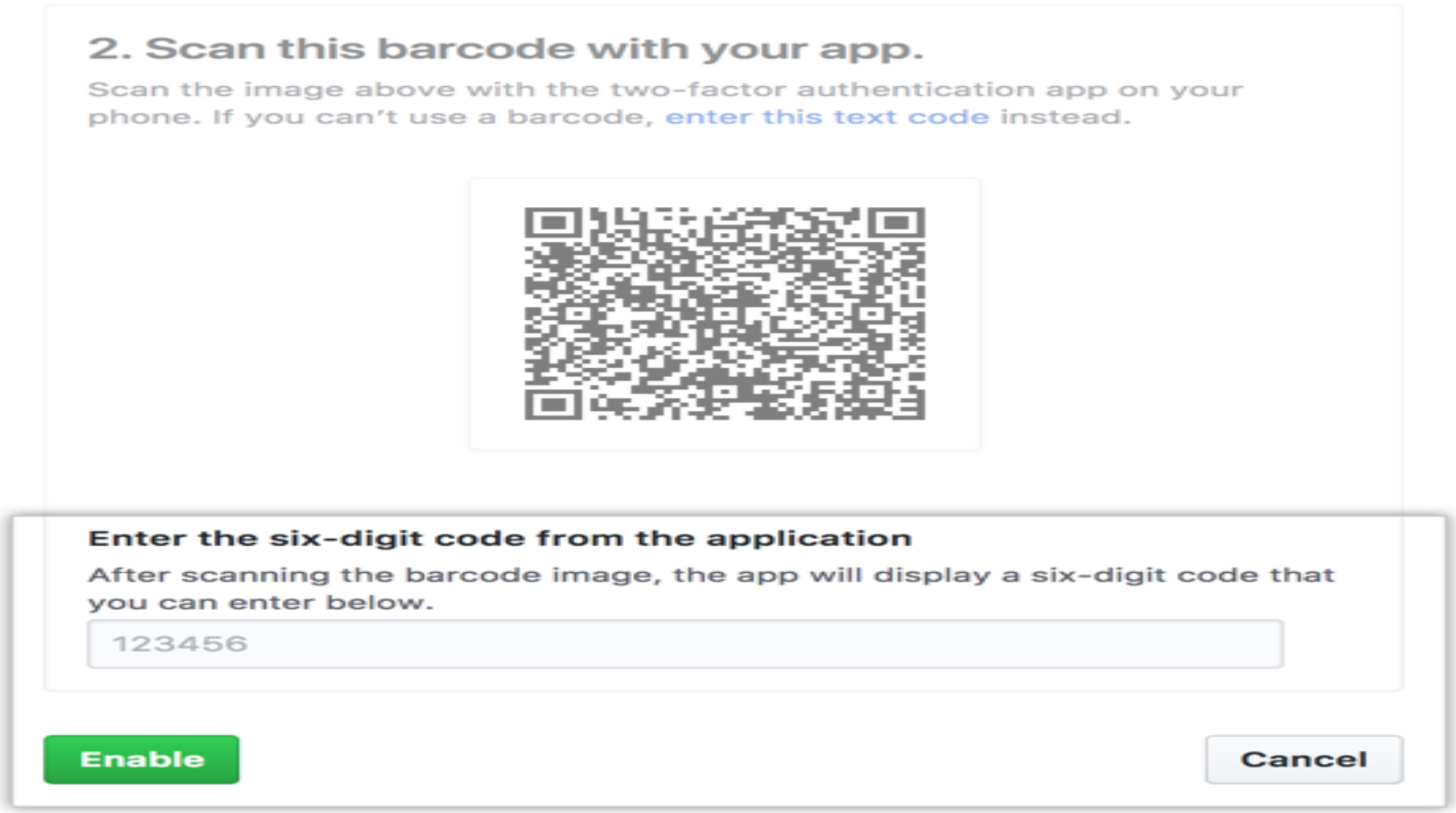
- Physical or smart (as an App)
- They provide One-Time-Password (OTP)
- Two protocols to generate OTPs:
  - HMAC-based OTP (HOTP): shared secret between the token and the server and a counter
    - New OTP every button push
  - Time-based OTP (TOTP): shared secret between the token and the server and time synchronization
    - New OTP every time interval (1 minute)
- Smart tokens replacing physical ones (google authenticator)

Physical Tokens





Smart Tokens



Multi-factor Authentication

- Combine authentication techniques from multiple factors, such as combining something you know with something you have (two-factor)
- Two-factor: requires any 2 out of 3 of
  - Something you **know**
  - Something you **have**
  - Something you **are**
- Examples
  - ATM: Card and PIN
  - Credit card: Card and signature
  - Smartcard with password/PIN
  - JUST servers: password + SMS verification code

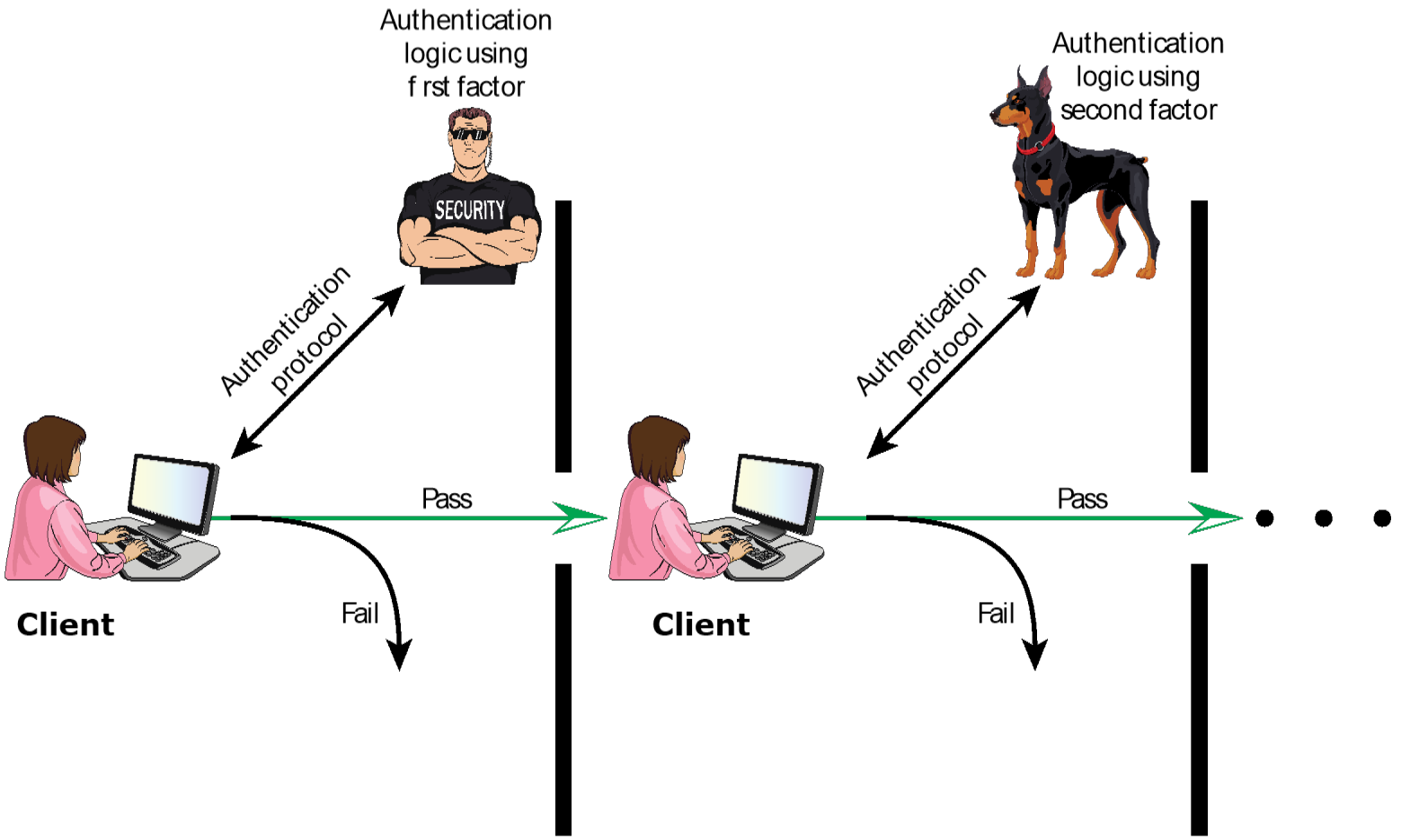


Figure 3.2 Multifactor Authentication

Usability vs. Security

Federated identity management

- People have many (online) accounts
  - It is inconvenient to enter password(s) repeatedly
  - It is inconvenient to remember all passwords
- The concepts of federation and single sign-on aim to reduce the inconvenience (security vs. usability) .
- Federated identity management occurs when organizations agree to share some of their authentication information.
- log-on to websites by your Google, Facebook, or Twitter account

Single Sign-on

- Single sign-on (SSO) shares authenticated sessions across systems.
- Many organizations create SSO solutions within their organizations to help users avoid the burden of repeatedly authenticating.
- Kerberos: a single sign-on protocol

SAML

- Security Assertion Markup Language (SAML) allows browser-based single-sign-on across a variety of web systems.
- Three actors in a SAML request:
  1. Principal: an end user wants to use web-based services
  2. Identity provider: the organization providing the proof of identity, usually the end user's employer, or other account provider.
  3. Service provider: the provider of the web-based service that the end user wants to access.

SAML (example)

1. Principal (PR) requests a resource from a service provider.
2. Service provider (SP) checks if principal already has a valid session:
   - if YES: SP grants access.
   - If NO:
     a) SP redirects PR to a SSO from identity provider (IP).
     b) PR tries to authenticate to the IP
     c) IP creates an XHTML form customized for the SP.
     d) PR then has to use this form to request what is called a security assertion from the SP.
     e) SP then validates the request and creates a security context with the desired service and redirects the user to that service.
     f) PR then requests that desired service and the resource service responds by granting access.

Web Cookies

- Cookie is provided by a website and stored on user's machine
- Numerical values that are stored and managed by a web browser.
- Cookie indexes a database at website and retains information about a user.
- Cookies **maintain state** across sessions
  - Web uses a stateless protocol: HTTP
  - Cookies also maintain state within a session
- Like a single sign-on for a website
  - But, very, very weak form of authentication
- Cookies also create privacy concerns

Access Control
Part III
Authorization

Access Control

- Access control (AC) is the process of ensuring that a (computer) system maintains a **correct** view of all (of its) entities and their privileges to (its) resources.
  - Prevent malicious access to resources.
  - Allow unmalicious access to resources.
- AC has three parts:
  - **Identification**: Who are you?
  - **Authentication:** Are you who you say you are?
  - **Authorization:** Are you allowed to do your request?
- Note: AC often used as synonym for Authorization

Authentication vs Authorization

- Authentication: Are you who you say you are?
  - Restrictions on who (or what) can access system
- **Authorization:** Are you allowed to do that?
  - Restrictions on actions of authenticated users
- Authorization is a form of **access control**
- Classic view of authorization…
  - Access Control Lists (ACLs)
  - Capabilities (C-lists)

Lampson's Access Control Matrix

- **Subjects** (users) index the rows
- **Objects** (resources) index the columns

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	_	_
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Matrix complexity

- **Access control matrix** has **all** relevant info
- Could be 100's of users, 10,000's of resources
  - Then matrix with 1,000,000's of entries
- Managing such a matrix is cumbersome
- Note: We need to check this matrix before access to any resource by any user
- How to make this efficient/practical?

Access Control Lists (ACLs)

- ACL: store access control matrix by **column**
- Example: ACL for **insurance data** is in **blue**

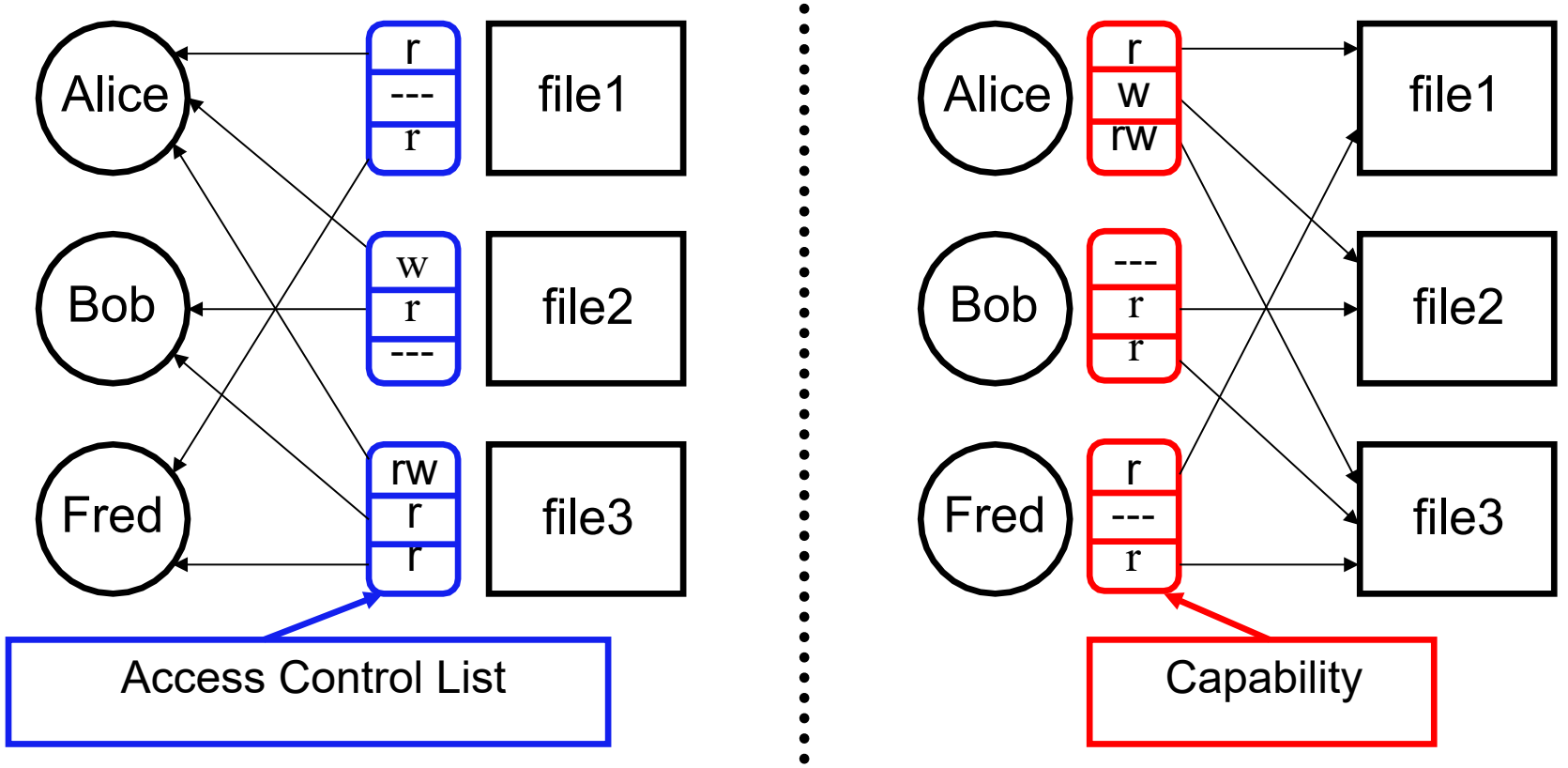
	OS	Accounting program	Accounting data	**Insurance data**	Payroll data
Bob	rx	rx	r	_	_
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

- Store access control matrix by **row**
- Example: Capability for **Alice** is in **red**

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	_	_
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs Capabilities



- Note that arrows point in opposite directions…
- With ACLs, still need to associate users to files

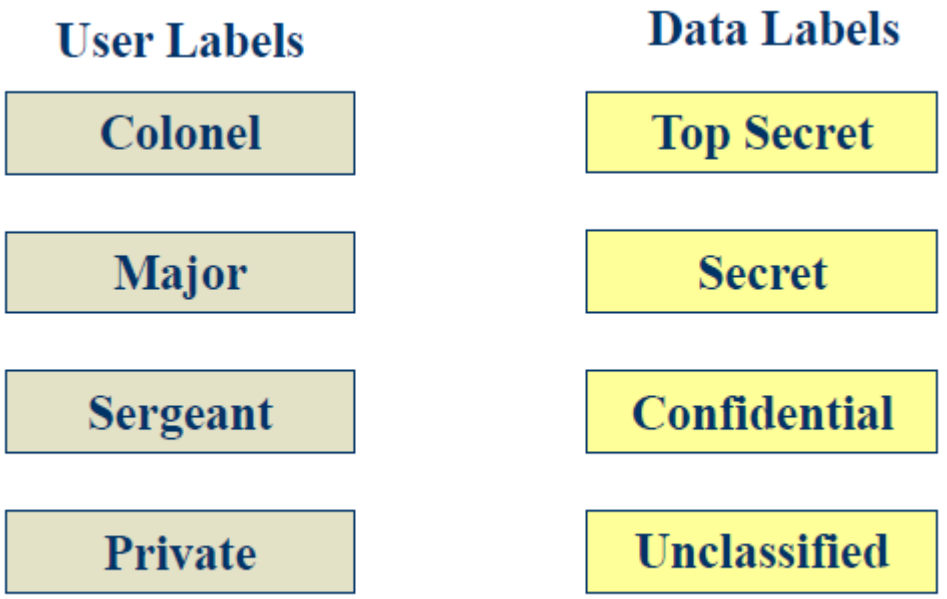
Access Control Models

- Mandatory AC (MAC)
- Discretionary AC (DAC)
- Role-based AC (RBAC)
- Attribute-based AC (ABAC)

Mandatory AC (MAC)

- Forced by OS
- Based on comparing security labels with security clearances
- Users have security clearances, data has security labels
  - security labels: how sensitive or critical a system resource is
  - security clearances: which entities are eligible to access certain resources
- Need to restrict the access according to security level of user and data
- Useful for environments with hierarchical propagation of information

Mandatory AC (MAC)



Mandatory AC (MAC)

- Each subject and object are classified into one of the security classifications
- Entities cannot enable other entities to access their resources
- Many examples, mainly two:
  - Bell-LaPadula model : for confidentiality
  - Biba model: for integrity

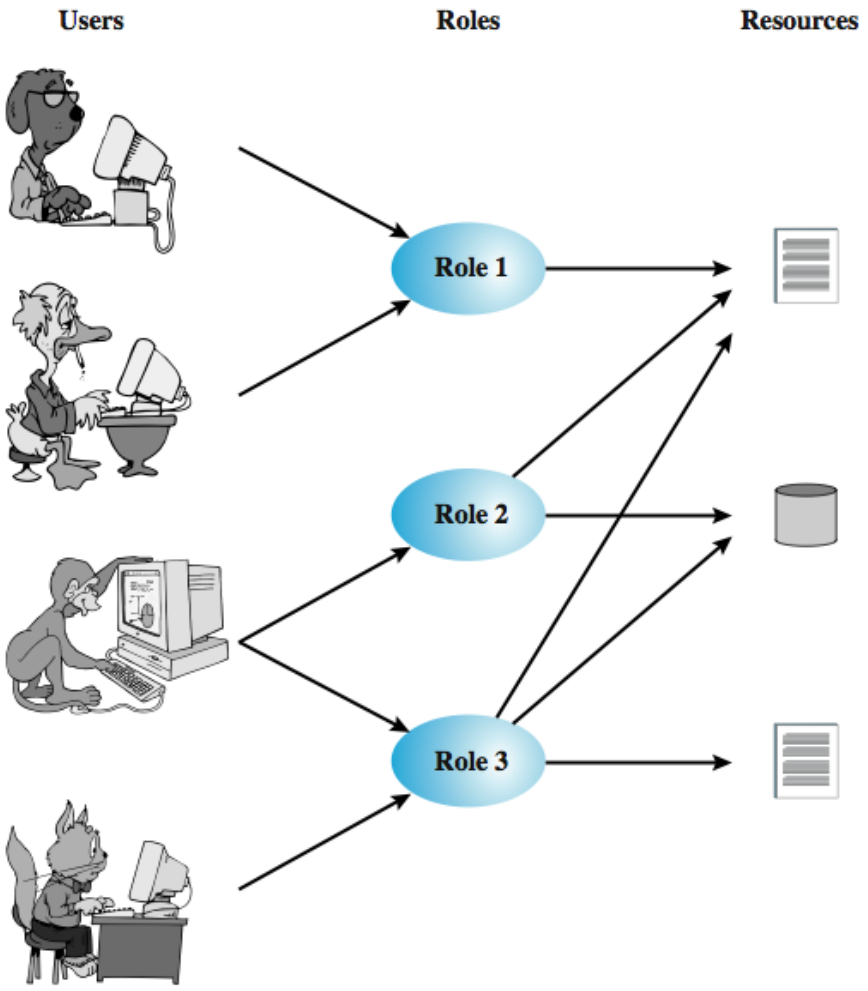
Discretionary AC (DAC)

- Based on the identity of the requestor and access rules
- Entities enable other entities to access their resources
- Often provided using an access matrix (recall)

Role-Based AC (RBAC)

- Based on user roles instead of identities.
- An alternative to traditional DAC and MAC policies.
- For access control purposes, it is more important to know what a user's organizational responsibilities are rather than who the user is.

Role-Based AC (RBAC)



Attribute-Based AC (ABAC)

- Based on the attributes of the user, the resources and the current environment
- Traditional access control systems were based on the identity or the role of the requestor.
- Not effective on the Internet.
- The access decision should be based on properties (attributes) of the requester and of the resource.

Attribute-Based AC (ABAC)

Not all access control decisions are identity-based or role-base, but rather based on digital credentials; this is more suitable for an open communication infrastructure.

RBAC vs ABAC

- an Online movies streaming service needs to enforce an access control policy that is based on the users' age and the movie content ratings dispayed in the following table:

Movie Rating	Users Allowed
R	Age 21 or older
PG-13	Age 13 or older
G	Everyone

The RBAC approach-I

- There are pre-defined roles created for users: *Adult, Juvenile and Child*.
- Permissions:
  1. Can view R rated movies,
  2. Can view PG-13 rated movies, and
  3. Can view G rated movies.

The RBAC approach-II

- Access control policies:

  I. The *Adult* role gets assigned with all three permissions (permission number 1, 2, and 3),

  II. The *Juvenile* role gets permission number 2, and 3.

  III. The *Child* role only gets permission number 3.

- Both the user-to-role assignment and the permission-to-role assignment are manually administrative tasks.

The ABAC approach:

- There is no need to explicitly define roles.
- Whether a user ***u*** can access or view a movie ***m*** (in a security environment ***e*** which is ignored here) would be resolved by evaluation a policy rule such as:
  - R3: can_access(***u,m, e***) ←

  (Age(***u***) >= 21 Λ Rating (***m***) ∈ { R, PG−13,G }) V

  (Age(***u***) >= 13 Λ Age(***u***) < 21 Λ Rating(***m***) ∈ { PG−13,G }) V

  (Age(***u***) < 13 Λ Rating(***m***) ∈ { G })
  - where Age and Rating are the subject attribute and the resource attribute, respectively.

RBAC vs ABAC

- ABAC model eliminates:
  - the definition and management of static roles,
  - the need for the administrative tasks for user-to-role assignment and permission-to-role assignment.
- ABAC becomes more manageable and scalable than RBAC for finer-grained access control policies that often involve multiple subject and object attributes.

OWASP
Top 10 Attacks
In a nutshell

The Ten Most Critical Web Application Security Risks

Reference: OWASP

OWASP

- The Open Web Application Security Project is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.
- At OWASP, you'll find free and open:
  - Application security tools and standards.
  - Complete books on application security testing, secure code development, and secure code review.
  - Presentations and videos.
  - Cheat sheets on many common topics.
  - Standard security controls and libraries.
  - Local chapters worldwide.
  - Cutting edge research.
  - Extensive conferences worldwide.
  - Mailing lists.

A1: Injection

- Definition: Sending untrusted data (as part of a command or a query) to an interpreter
  - The untrusted data tricks/fools the interpreter causing it executing of commands or accessing of data without proper authorization.
- Main Types: SQL injection, and Light-weight Directory Access Protocol LDAP injection.
- Main Vulnerabilities: Not having proper input validation/ filtering.
- Main Prevention: Perform proper input validation

A1: Code Example

User Input

Search Caroline

User Input

Search Caroline SELECT c1,c2,.. FROM t1 WHERE cn LIKE '%Caroline%';

User Input

Search Caroline%' UNION SELECT userID, pwd, userType FROM users; --

A2: Broken Authentication

- Definition: Targeting and thus bypassing weak authentication systems.
- Main Types:
  A. Brute-force
  B. Credential stuffing
  C. Guessing of weak and default passwords.
- Main Vulnerabilities:
  I. Allow A and B above.
  II. Use weak and default passwords.
- Main Prevention:
  - Do not I, II, and III .
  - Implement multi-factor authentication.

A3: Sensitive Data Exposure

- Definition: Stealing, modification, or publishing weakly protected sensitive data such as: financial, healthcare, and personal data.
- Main Types:
  - Credit card fraud
  - Identity theft
- Main Vulnerabilities:
  - Weak protection of data at rest or in transit (poor encryption or none at all!).
- Main Prevention:
  - Force proper encryption at rest or in transit.
  - Do not store sensitive data unless necessary.

A4: XML External Entities (XXE)

- Definition: Evaluating malicious external entity references within XML documents.
- Main Types:
  - Disclose or share internal files using the file URI handler
  - Remote code execution.
- Main Vulnerabilities:
  - Accepting XML from untrusted sources.
  - Using old versions of Simple Object Access Protocol (SOAP).
- Main Prevention:
  - Use simple data formats such as JSON instead of XML.
  - Update SOAP to version 1.2 or above.
  - Use XXE detection tools.

A4 Example: Billion Laughs attack

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

A5: Broken Access Control

- Definition: Accessing unauthorized functionalities or data in weak authorization systems.
- Main Types:
  - Insecure Direct Object Reference.
  - Missing function-level access control.
- Main Vulnerabilities:
  - Not forcing access control checks properly.
  - Permitting viewing or editing someone else's account.
- Main Prevention:
  - Deny access (request) unless public (resource).
  - Invalidate tokens on the server after logout.
  - Implement and enforce (properly) an access control policy.

A5 Examples

- Insecure Direct Object Reference:
  - http://funnysite/whatever/account?id=omar
  - http://funnysite/whatever/account?id=admin
- Missing function-level access control
  - http://funnysite/whatever/adminDetails

A6: Security Misconfiguration

- Definition: the lack of proper security configuration in a system including: OS, web browsers, ISs, DBs, etc..
- Main Types:
  - No security configuration.
  - Poor security configuration.
- Main Vulnerabilities:
  - Keeping default accounts.
  - Enabling unnecessary features (ports, privileges, accounts).
  - Missing security hardening across the application stack.
- Main prevention:
  - Disable default accounts or force password changes.
  - Use a minimal platform without any unnecessary features.

A6 Example

Account name:

`admin`

Account password:

`admin`

A7: Cross-Site Scripting (XSS)

- Definition: Executing scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
- Main Types: HTML, JavaScript.
- Main Vulnerabilities:
  - Allow the execution of arbitrary JavaScript.
  - Storing and using un-sanitized user input.
  - Allowing untrusted HTTP requests.
- Main Prevention:
  - Using frameworks that automatically such as React JS.
  - Escaping untrusted HTTP requests data based on the HTML content.

A7 Example

User Input

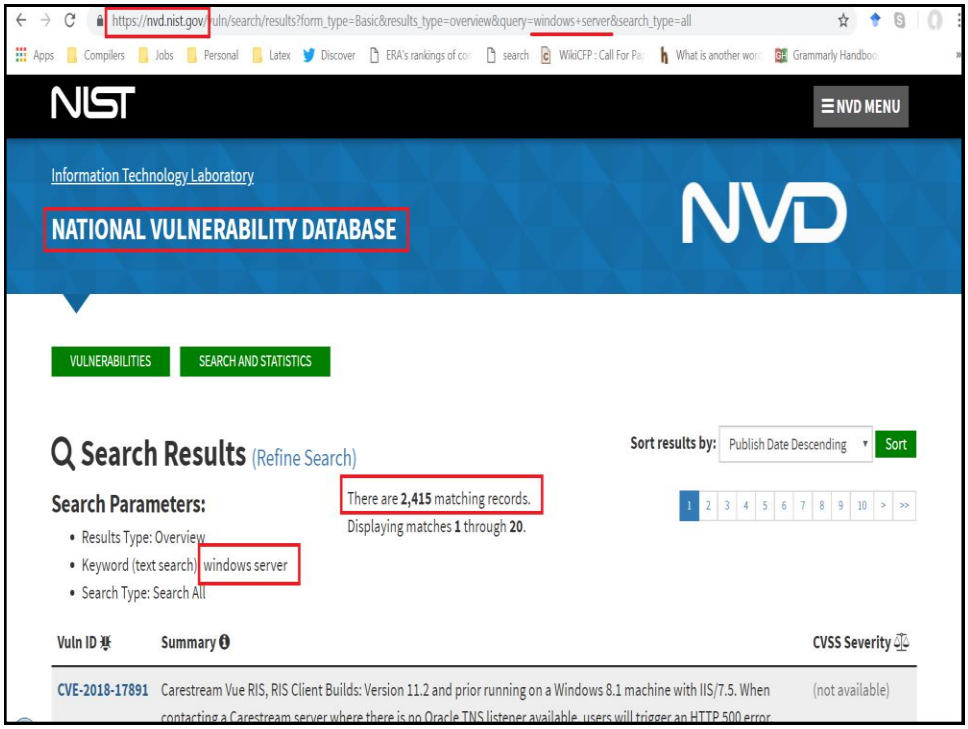
Name	Caroline
Occupation	Vice President, Security
# Years	<script> do bad things </script>

A8: Insecure Deserialization

- Preliminary definitions:
  - Serialization: converting an object to a simpler form for later use.
    - Example, storing an object as a byte[] for later transmission.
  - Deserialization: reversing serialization, i.e., converting simple form of data to an object.
    - Example: converting the received byte[] into the original object again.
- Definition: Insecure deserialization is making the receiver to deserialize untrusted data into an object.
- This may lead to different attacks such as: remote code execution, injection attacks, and privilege escalation attacks.
- Vulnerability: deserialize untrusted data.
- Prevention: Apply integrity checks on data, enforce strict type checks, and isolate deserialized object in low privilege environments.

A9: Using Vulnerable Components

- Definition: (attack name suffice!) Inheriting the known weakness of used components.
- Component Types: servers, libraries, frameworks.
- Main Vulnerabilities:
  - Not knowing the used components, or their vulnerabilities.
  - Not fixing or upgrading used components on regular basis.
- Main Prevention:
  - Continuously monitor sources like CVE and NVD for vulnerabilities in the components.
  - Subscribe to email alerts for security vulnerabilities related to components you use.
  - Only obtain components from official sources over secure links.



A10: Insufficient Logging & Monitoring

- Definition: allowing attackers to perform continuous attempts on a system due to the absence or the improper logging and monitoring of incidents .
  - In average, an attack takes >200 days to be detected!.
  - The detection mainly happen externally not internally!.
- Main Types: (1) No logging at all! (2) No monitoring of logged data. (3) No alerts on suspicious events.
- Main Vulnerabilities:
  - Not logging events such as logins, failed logins, and high-value transactions.
  - Warnings and errors generate unclear log messages.
  - Logs are only stored locally.
- Main Prevention: (1) Log important events, (2) Monitor logs, and (3) Apply effective response policy for suspicious events.

Malware and countermeasures

Omar Almousa

Jordan University of Science and Technology, Jordan

CIA

- Called security triad and represents main security goals.
- Confidentiality
- Integrity
- Availability

Malware

- Malicious Software: code used to steal data, bypass access control, or cause harm to a system.
- Common types:
  - Spyware
  - Adware
  - Bot
  - Scareware
  - Virus
  - Ransomware
  - Trojans
  - Worms
  - ...

Malware types I

- Spyware: a malware that spies on the user, including activity trackers, keystroke collection, and data capture.
- Adware: Advertising supported software is designed to automatically deliver advertisements. Installed with some versions of software to deliver advertisements and may come with spyware.
- Bot: From the word robot, a bot is a malware designed to automatically perform action, usually online.
  - An increasing use of malicious bots is botnets.
  - A botnet is the existing of several computers that are infected with bots and programmed to wait (silently) for commands provided by the attacker.

Malware Types II

- Scareware: a malware designed to persuade the user to take a specific action based on fear/scare.
  - pop-up windows that resemble operating system dialogue boxes stating the system is at risk or needs the execution of a specific program to return to normal operation.
  - In reality, no problems were assessed or detected and if the user agrees and clears the mentioned program to execute, his or her system will be infected with malware.
- Virus: a malicious executable code that is attached to other executable files, often legitimate programs.
  - Require end-user activation and can activate at a specific time or date.
  - Viruses can be harmless and simply display a picture or they can be destructive, such as those that modify or delete data.
  - Spread by USB drives, optical disks, network shares, or email.

Malware Types III

- Ransomware: a malware that captures/holds some resource until a payment/ransom is paid.
  - Usually by encrypting data with a key unknown to the user.
- Trojan horse: a malware that carries out malicious operations under the guise of a desired operation.
  - Trojans are found in image files, audio files or games; they from a viruses because they bind themselves to non-executable files.
- Worms: malware that replicate themselves by independently exploiting vulnerabilities in networks.
  - Worms usually slow down networks.
  - Unlike a virus that requires a host program to run, a worm can run by themselves.

Social engineering

- Social engineering is an attack that attempts to manipulate individuals into performing actions or reveal confidential information.
- Rely on people's willingness to be helpful or people's weaknesses.
- For example, an attacker could call an authorized employee with an urgent problem that requires immediate network access.
- These are some types of social engineering attacks:
  - Pretexting: This is when an attacker calls an individual and lies to them in an attempt to gain access to privileged data. An example involves an attacker who pretends to need personal or financial data in order to confirm the identity of the recipient.
  - Tailgating: This is when an attacker quickly follows an authorized person into a secure location.
  - Something for Something (Quid pro quo):This is when an attacker requests personal information from a party in exchange for something, like a free gift.

Phishing

- Phishing is when an attacker sends a fraudulent email disguised as being from a legitimate, trusted source.
  - The message intent is to trick the recipient into installing malware on their device, or into sharing personal or financial information.
  - An example of phishing is an email forged to look like it was sent by a retail store asking the user to click a link to claim a prize. The link may go to a fake site asking for personal information, or it may install a virus.
- Spear phishing (whaling) is a highly targeted phishing attack.
  - While phishing and spear phishing both use emails to reach the victims, spear phishing emails are customized to a specific person.
  - The attacker researches the target's interests before sending the email.
  - For example, an attacker learns the target is interested in cars, and has been looking to buy a specific model of car. The attacker joins the same car discussion forum where the target is a member, forges a car sale offering and sends email to the target. The email contains a link for pictures of the car. When the target clicks on the link, malware is installed on the target's computer.

Denial Of Service DOS

- Denial-of-Service (DoS) attacks are a type of network attack on availability that results in an interruption of network services.
- There are two major types of DoS attacks:
  1. Overwhelming Quantity of Traffic: A DoS attack that happens by sending an enormous quantity of data at a rate which the system (network, host, or application) cannot handle. This causes a slowdown in transmission or response, or a crash of a system.
  2. Maliciously Formatted Packets: A DoS attack that occurs by sending maliciously formatted packets to a system making the receiver unable to handle them. For example, an attacker forwards packets containing errors that cannot be identified by the application, or forwards improperly formatted packets. This causes the receiving system to run very slowly or crash.
- DoS attacks are considered a major risk because they can easily interrupt communication and cause significant loss of time and money. These attacks are relatively simple to conduct, even by an unskilled attacker.

Distributed DOS (DDos)

- Similar to a DoS attack but originates from multiple, coordinated sources.
- As an example, a DDoS attack could proceed as follows:
  - An attacker builds a network of infected hosts, called a botnet.
  - The infected hosts are called zombies. The zombies are controlled by handler systems.
  - The zombie computers constantly scan and infect more hosts, creating more zombies.
  - When ready, the attacker instructs handler systems to make the botnet of zombies carry out a DDoS attack.

Firewalls

- A firewall is software/hardware designed to control, or filter, which communications are allowed in and which are allowed out of a device or network.
- A firewall can be installed on a single computer with the purpose of protecting that one computer (host-based firewall), or it can be a stand-alone network device that protects an entire network of computers and all of the host devices on that network (network-based firewall).
- Over the years, as computer and network attacks have become more sophisticated, new types of firewalls have been developed which serve different purposes in protecting a network.

Firewalls Types

- Network Layer Firewall: filtering based on source and destination IP addresses.
- Transport Layer Firewall: filtering based on source and destination data ports, and filtering based on connection states.
- Application Layer Firewall: filtering based on application, program or service.
- Context Aware Application Firewall: filtering based on the user, device, role, application type, and threat profile.
- Proxy Server: filtering of web content requests like URL, domain, media, etc.
- Reverse Proxy Server: placed in front of web servers, reverse proxy servers protect, hide, offload, and distribute access to web servers.
- Network Address Translation (NAT) Firewall: hides or masquerades the private addresses of network hosts.
- Host-based Firewall: filtering of ports and system service calls on a single computer operating system.

IDS and IPS

- An Intrusion Detection System (IDS): is a system that scans data against a set of rules (attack signatures) that represent malicious traffic.
  - If a match is detected, the IDS will log the detection, and create an alert for a network administrator.
  - The Intrusion Detection System does not take action when a match is detected so it does not prevent attacks from happening. The job of the IDS is merely to detect, log and report.
- Intrusion Prevention System (IPS): An Intrusion Prevention System (IPS) is an IDS with the ability to block or deny traffic based on a positive rule or signature match.
- One of the most well-known IPS/IDS systems is Snort and the commercial version of Snort is Cisco's Sourcefire.

Conclusive recommendations

- Update your OS, AV, ...
- Install and maintain firewalls, anti-virus systems.
- Back-up frequently
- Encrypt data
- Use strong/different passwords (long phrases better than short complicated)
- Be aware of social engineering and do not share much
- .....

Software Security

Why Software?

- Why is software as important to security as crypto, access control, protocols?
- Virtually all information security features are implemented in software
- If your software is subject to attack, your security can be broken
  - Regardless of strength of crypto, access control, or protocols
- Software is a poor ***foundation*** for security

Software Flaws

If automobiles had followed the same development cycle as the computer,
a Rolls-Royce would today cost $100, get a million miles per gallon,
and explode once a year, killing everyone inside.
— Robert X. Cringely

My software never has bugs. It just develops random features.
— Anonymous

Bad Software is Ubiquitous

- NASA Mars Lander (cost $165 million)
  - Crashed into Mars due to…
  - …error in converting English and metric units of measure
  - Believe it or not
- Denver airport
  - Baggage handling system — very buggy software
  - Delayed airport opening by 11 months
  - Cost of delay exceeded $1 million/day
  - What happened to person responsible for this fiasco?
- MV-22 Osprey
  - Advanced military aircraft
  - Faulty software can be fatal

Software Issues

Alice and Bob

- Find bugs and flaws by accident
- Hate bad software…
- …but they learn to live with it
- Must make bad software work

Trudy

- Actively looks for bugs and flaws
- Likes bad software…
- …and tries to make it misbehave
- Attacks systems via bad software

Complexity

- "Complexity is the enemy of security", Paul Kocher, Cryptography Research, Inc.

System	Lines of Code (LOC)
Netscape	17 million
Space Shuttle	10 million
Linux kernel 2.6.0	5 million
Windows XP	40 million
Mac OS X 10.4	86 million
Boeing 777	7 million

- A new car contains more LOC than was required to land the Apollo astronauts on the moon

Lines of Code and Bugs

- Conservative estimate: 5 bugs/10,000 LOC
- **Do the math**
  - Typical computer: 3k exe's of 100k LOC each
  - Conservative estimate: 50 bugs/exe
  - Implies about 150k bugs per computer
  - So, 30,000-node network has 4.5 billion bugs
  - Maybe only 10% of bugs security-critical and only 10% of those remotely exploitable
  - Then "only" 45 million critical security flaws!

Software Security Topics

- Malicious software (intentional) – we covered this already
  - Viruses
  - Worms
  - Other breeds of malware
- Program flaws (unintentional)
  - Buffer overflow
  - Incomplete mediation
  - Race conditions

Program Flaws

- An **error** is a programming mistake
  - To err is human
- An error may lead to incorrect state: **fault**
  - A fault is internal to the program
- A fault may lead to a **failure**, where a system departs from its expected behavior
  - A failure is externally observable

Example

```
char array[10];
for(i = 0; i < 10; ++i)
     array[i] = `A`;
array[10] = `B`;
```

- This program has an **error**
- This error might cause a **fault**
  - Incorrect internal state
- If a fault occurs, it might lead to a **failure**
  - Program behaves incorrectly (external)
- We use the term **flaw** for all of the above

Secure Software

- In software engineering, try to ensure that a program does what is intended
- ***Secure*** software engineering requires that software **does what is intended…**
- **…and nothing more**
  - Well, your are asking a lot more
- Absolutely secure software? Dream on…
  - Absolute security ***anywhere*** is impossible
- How can we manage software risks?

Program Flaws

- Program flaws are **unintentional**
  - But can still create security risks
- We'll consider 3 types of flaws
  - Buffer overflow (smashing the stack)
  - Incomplete mediation
  - Race conditions
- These are the most common flaws

Buffer Overflow

Attack Scenario

- Users enter data into a Web form
- Web form is sent to server
- Server writes data to array called buffer, without checking length of input data
- Data “overflows” buffer
  - Such overflow might enable an attack
  - If so, attack could be carried out by anyone with Internet access

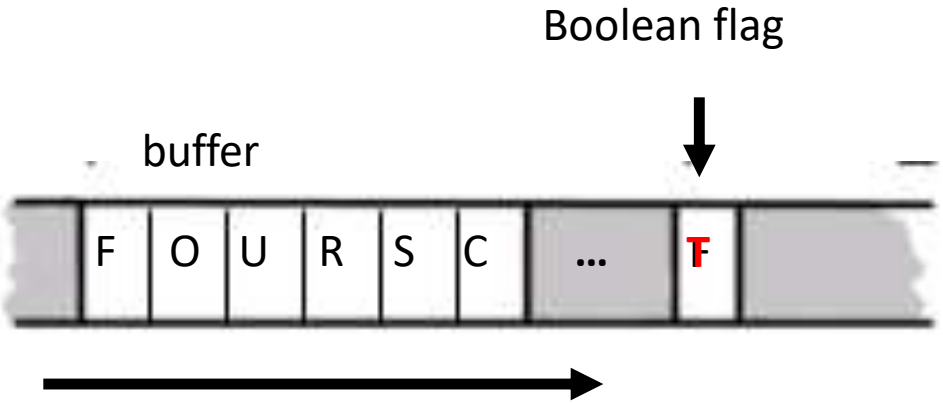
Buffer Overflow

```
int main(){
    int buffer[10];
    buffer[20] = 37;}
```

- **Q:** What happens when code is executed?
- **A:** Depending on what resides in memory at location “buffer[20]”
  - Might overwrite **user** data or code
  - Might overwrite **system** data or code
  - Or program could work just fine

Simple Buffer Overflow

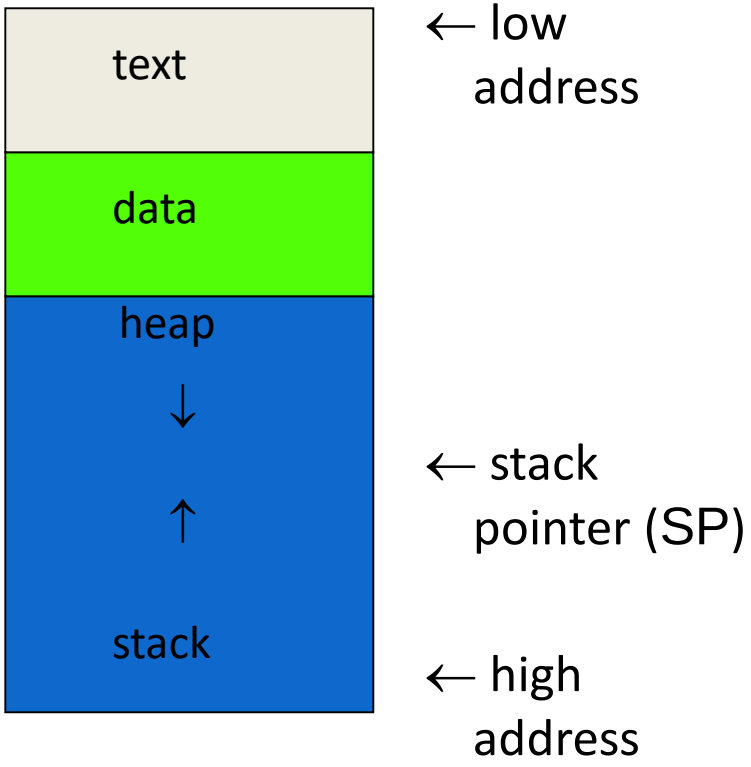
- Consider boolean flag for authentication
- Buffer overflow could overwrite flag allowing anyone to authenticate



- In some cases, Trudy need not be so lucky as in this example

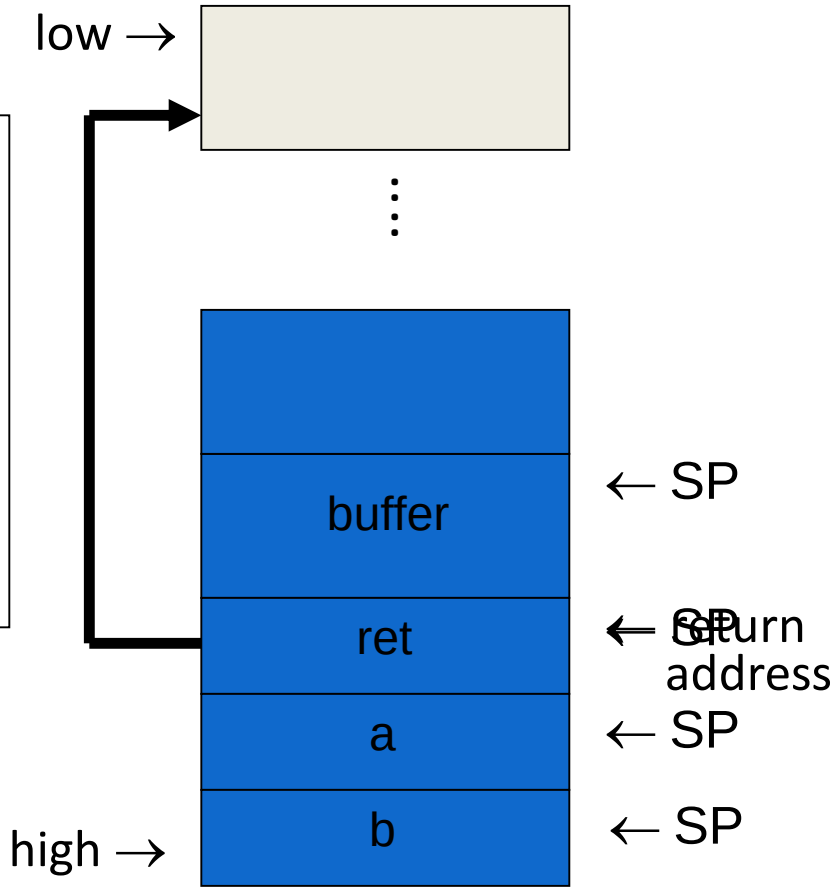
Memory Organization

- **Text** — code
- **Data** — static variables
- **Heap** — dynamic data
- **Stack** — "scratch paper"
  - Dynamic local variables
  - Parameters to functions
  - Return address



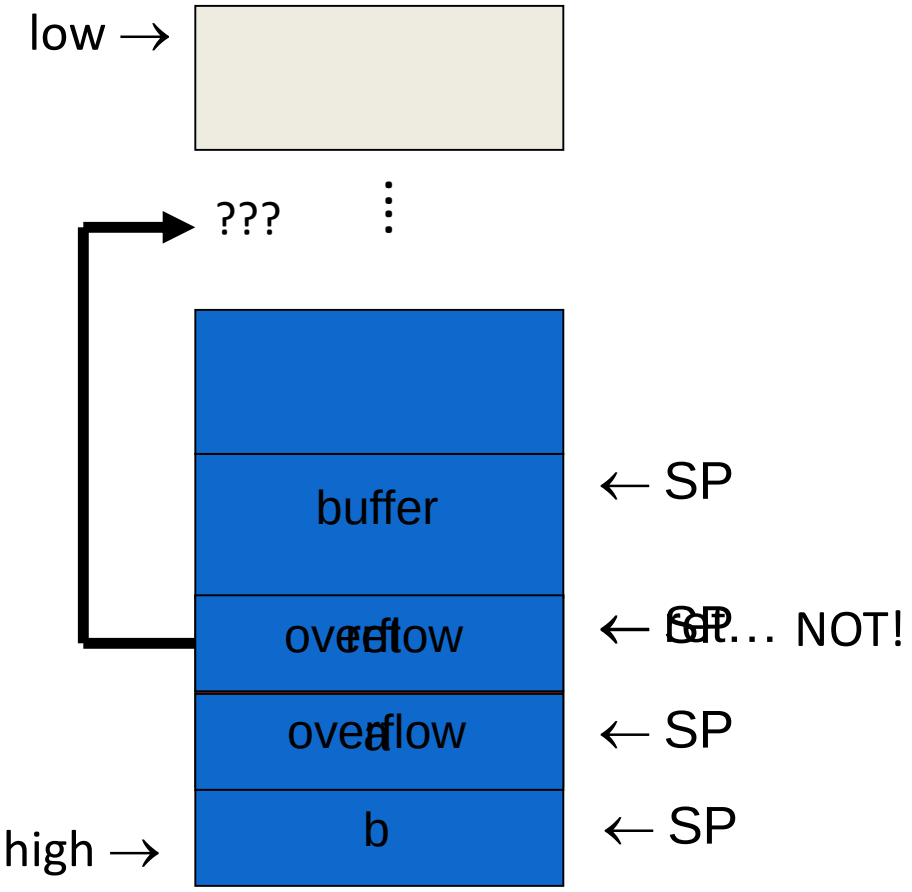
Simplified Stack Example

```
void func(int a, int b){
   char buffer[10];
}
void main(){
   func(1,2);
}
```



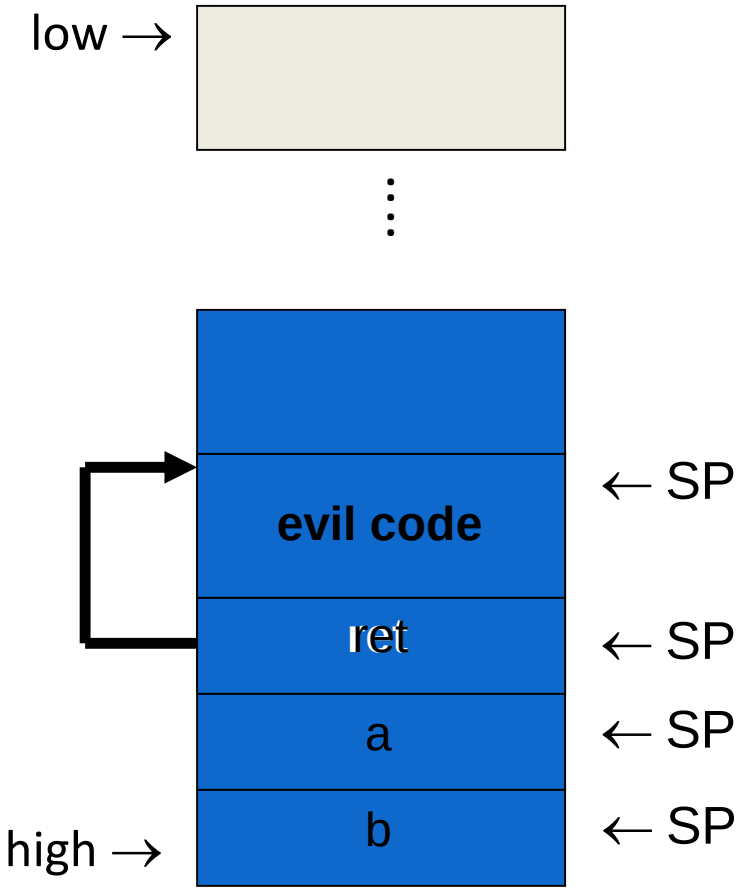
Smashing the Stack

- What happens if buffer overflows?
- Program "returns" to wrong location
- A crash is likely



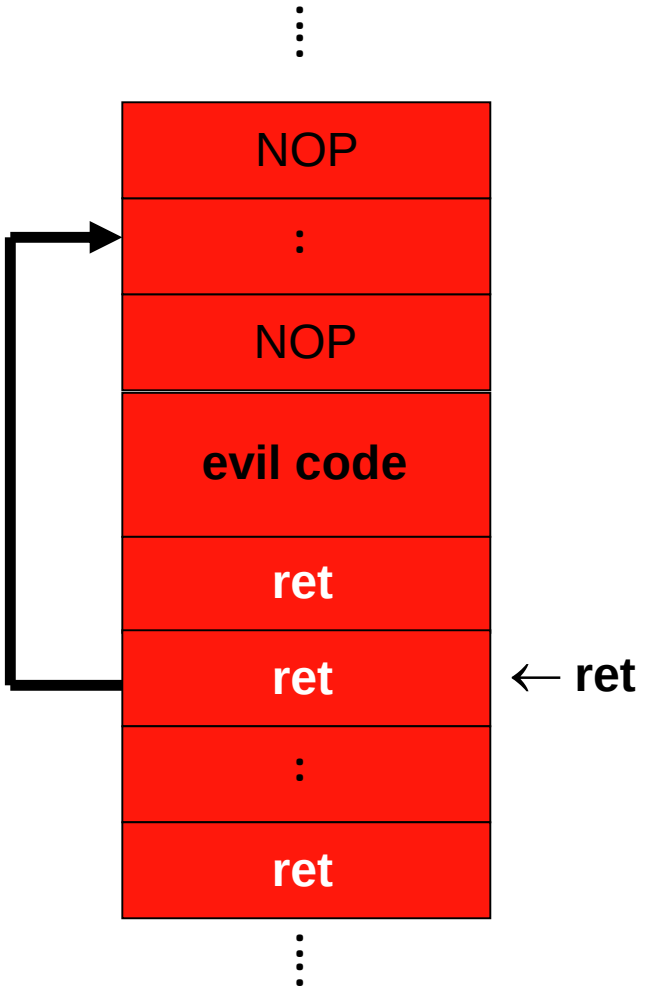
Smashing the Stack

- Trudy has a better idea…
- **Code injection**
- Trudy can run code of her choosing…
  - …on your machine



Smashing the Stack

- Trudy may not know…
  1) Address of evil code
  2) Location of **ret** on stack
- Solutions
  1) Precede evil code with NOP "landing pad"
  2) Insert **ret** many times



ASLR

- Address Space Layout Randomization
  - Randomize place where code loaded in memory
- Makes most buffer overflow attacks probabilistic
- Windows Vista uses 256 random layouts
  - So about 1/256 chance buffer overflow works
- Similar thing in Mac OS X and other OSs
- Attacks against Microsoft's ASLR do exist
  - Possible to "de-randomize"

Buffer Overflow

- A major security threat yesterday, today, and tomorrow
- The good news?
  - It is possible to reduce overflow attacks (safe languages, ASLR, education, etc.)
- The bad news?
  - Buffer overflows will exist for a long time
  - Why? Legacy code, bad development practices, clever attacks, etc.

Incomplete Mediation

Input Validation

- Consider: `strcpy(buffer, argv[1])`
- A buffer overflow occurs if

  `len(buffer) < len(argv[1])`
- Software must **validate** the input by checking the length of `argv[1]`
- Failure to do so is an example of a more general problem: **incomplete mediation**

Input Validation

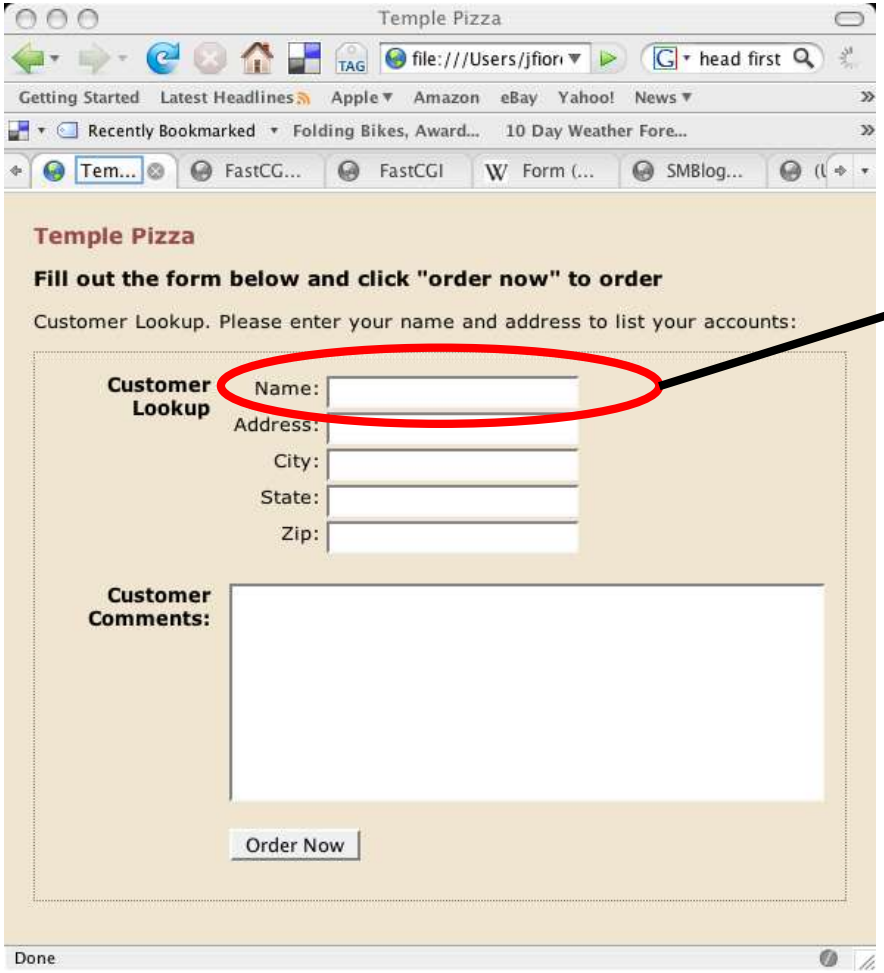
- Consider web form data
- Suppose input is validated on client
- For example, the following is valid

```
http://www.things.com/orders/final&custID=112&
  num=55A&qty=20&price=10&shipping=5&total=205
```

- Suppose input is not checked on server
  - Why bother since input checked on client?
  - Then attacker could send http message

```
http://www.things.com/orders/final&custID=112&
  num=55A&qty=20&price=10&shipping=5&total=25
```

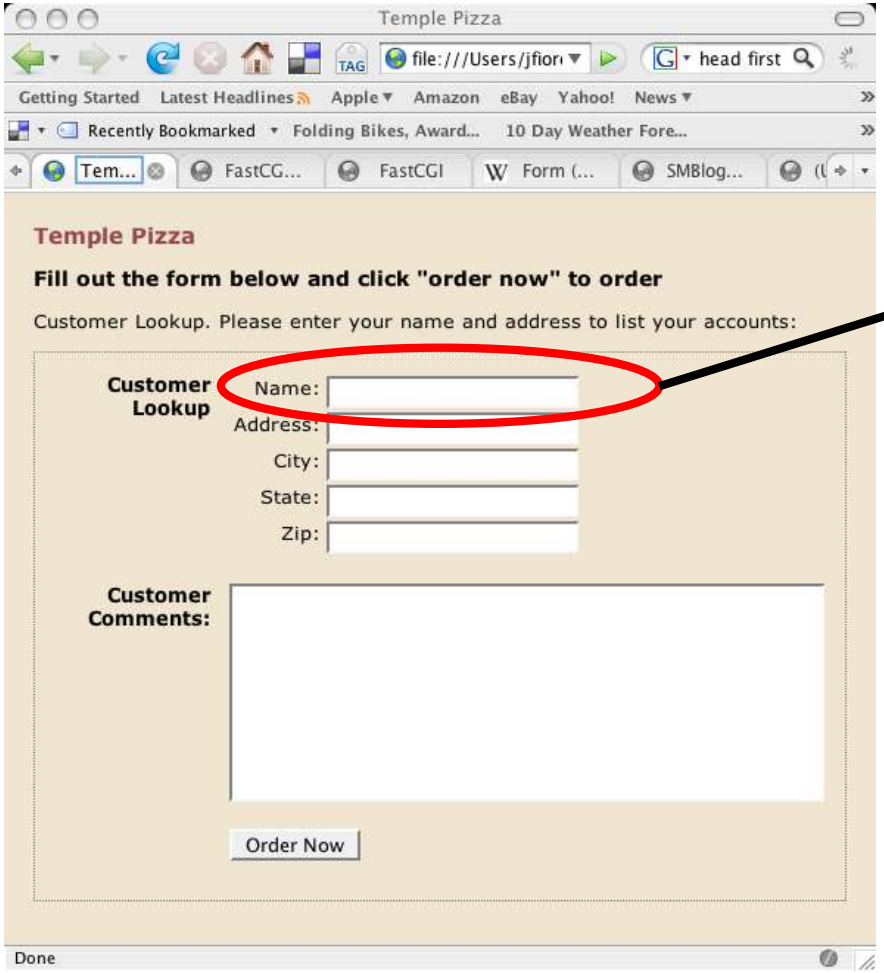
Incomplete Mediation (ex. SQL Injection)



John Fiore

SELECT * from CUSTOMERS
WHERE name = 'John Fiore'

Incomplete Mediation (ex. SQL Injection)



John Fiore' or '1'='1

SELECT * from CUSTOMERS
WHERE name = 'John Fiore'
OR '1'='1'

Race Conditions

Race Condition

- Security processes should be **atomic**
  - Occur “all at once”
- Race conditions can arise when security-critical process occurs in stages
- Attacker makes change between stages
  - Often, between stage that gives authorization, but before stage that transfers ownership

Time-of-Check-to-Time-of-Use Errors(TOCTTOU)

- Real Example:
  - Buy something that costs $100
  - The buyer removes five $20 bills from a wallet, carefully counts them in front of the seller, and lays them on the table
  - Then the seller turns around to write a receipt
  - While the seller's back is turned, the buyer takes back one $20 bill
  - When the seller turns around, the buyer hands over the stack of bills, takes the receipt, and leaves
  - Between the time the security was checked (counting the bills) and the access (exchanging the sculpture for the bills), a condition changed
    - What was checked is no longer valid when the object (that is, the sculpture) is accessed

Time-of-Check-to-Time-of-Use Errors(TOCTTOU)

- Computing Example:
  - “file” is a symbolic link for a file that can be opened normally
  - The attacker, after access is called, can change the “file”

The program with TOCTTOU vulnerability	Attacker

`if (access("file", W_OK) != 0) {`
`exit(1);`
`}`
 	
`//writing over /etc/passwd`
 	
`fd = open("file",O_WRONLY);`
`write(fd, buffer, sizeof(buffer));`
	`// After the access check`
	`// and Before the open, "file" points to the password database`
	
	`symlink("/etc/passwd", "file");`

Race Conditions

- Race conditions are common
- Race conditions may be more prevalent than buffer overflows
- But race conditions harder to exploit
  - Buffer overflow is “low hanging fruit” today
- To prevent race conditions, make security-critical processes atomic
  - Occur all at once, not in stages
  - Not always easy to accomplish in practice

Course Summary

- Crypto
  - Symmetric key, public key, hash functions, cryptanalysis
- Protocols
  - Simple auth., SSL, Kerberos, ZKP, PFS
- Access Control
  - Identification, authentication, authorization
- Software
  - Malware, attacks, flaws