CSE P 590 / CSE M 590 (Spring 2010)

Computer Security and Privacy

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

- Software Security (Continued)
  - More attacks / issues
  - Defensive directions
- Cryptography (Intro)
  - Background / history / context / overview
- Research: IMDs

TOCTOU

- TOCTOU == Time of Check to Time of Use

```
int openfile(char *path) {
    struct stat s;
    if (stat(path, &s) < 0)
        return -1;
    if (!S_ISRREG(s.st_mode)) {
        error("only allowed to regular files!");
        return -1;
    }
    return open(path, O_RDONLY);
}
```

- Goal: Open only regular files (not symlink, etc)
- Attacker can change meaning of path between stat and open (and access files he or she shouldn't)

Integer Overflow and Implicit Cast

```
char buf[80];
void vulnerable() {
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if (len > sizeof buf) {
        error("length too large, nice try!");
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dst, const void * src, size_t n);

typedef unsigned int size_t;
```

- If len is negative, may copy huge amounts of input into buf

(from www-inst.eecs.berkeley.edu—implflaws.pdf)

Integer Overflow and Implicit Cast

```
size_t len = read_int_from_network();
char *buf;
buf = malloc(len+5);
read(fd, buf, len);
```

- What if len is large (e.g., len = 0xFFFFFFFF)?
- Then len + 5 = 4 (on many platforms)
- Result: Allocate a 4-byte buffer, then read a lot of data into that buffer.

(from www-inst.eecs.berkeley.edu—implflaws.pdf)

Next

- Randomness
- Timing Attacks

Randomness issues

- Many applications (especially security ones) require randomness
- Explicit uses:
  - Generate secret cryptographic keys
  - Generate random initialization vectors for encryption
- Other "non-obvious" uses:
  - Generate passwords for new users
  - Shuffle the order of votes (in an electronic voting machine)
  - Shuffle cards (for an online gambling site)

C’s rand() Function

- C has a built-in random function: rand()

```
unsigned long int next = 1;
/* rand:  return pseudo-random integer on 0..32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
/* srand:  set seed for rand() */
void srand(unsigned int seed) {
    next = seed;
}
```

- Problem: don’t use rand() for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones

Dr.Dobb's Portal
The World of Software Development

ABOUT US | CONTACT | ADVERTISE | SUBSCRIBE | SOURCE CODE | CURRENT PRINT ISSUE
NEWSLETTERS | RESOURCES | BLOGS | PODCASTS | CAREERS

Windows/.NET

July 22, 2001

Randomness and the Netscape Browser

How secure is the World Wide Web?

Ian Goldberg and David Wagner

No one was more surprised than Netscape Communications when a pair of computer-science students broke the Netscape encryption scheme. Ian and David describe how they attacked the popular Web browser and what they found out.

- Email
- Prin
- Discuss
- Rep

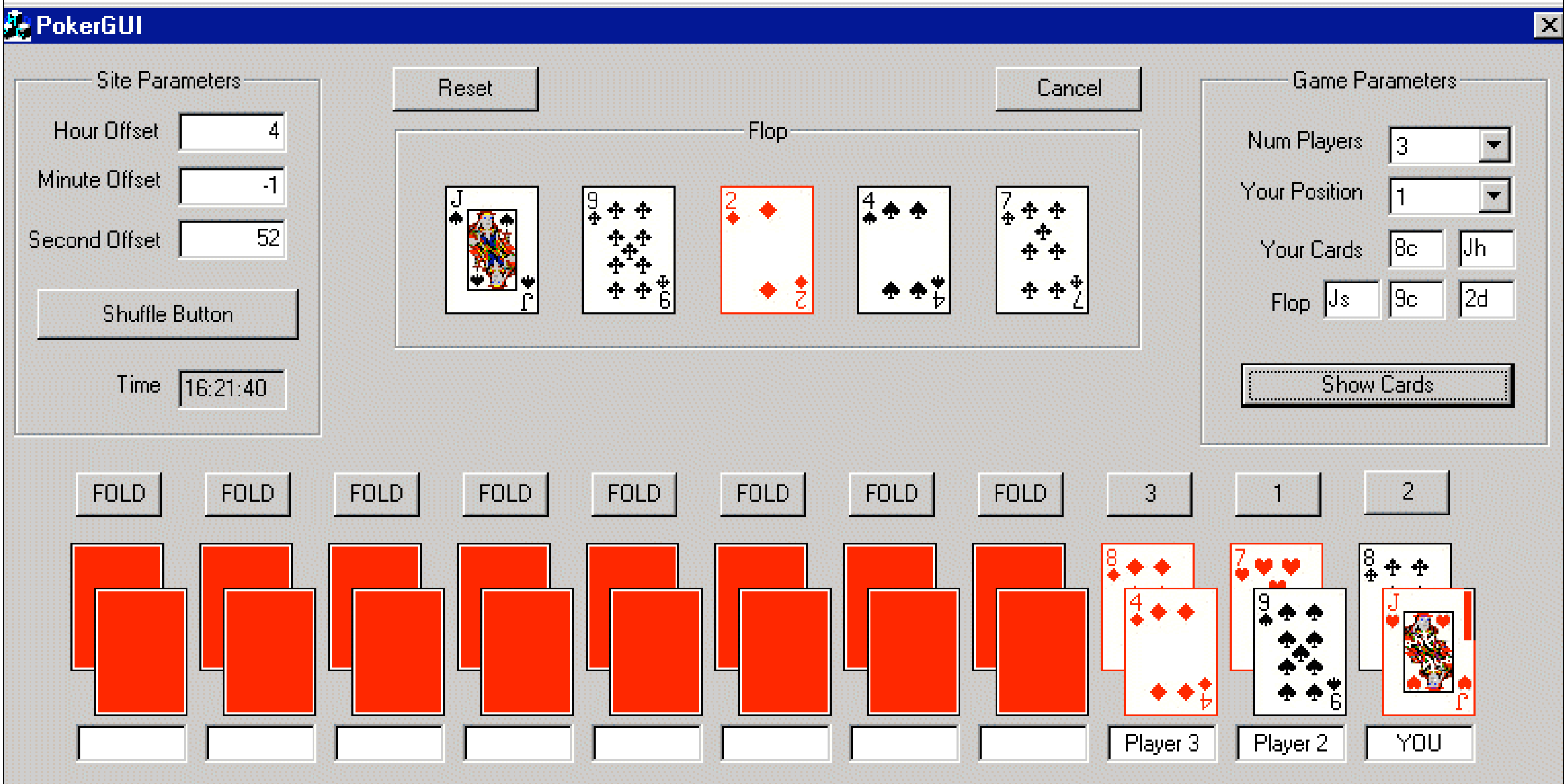
add to:

- Del.icio.us
- Slash
- Digg
- Y!
- Google
- MyWe
- Spurl
- Blin
- Furl

Problems in Practice

- One institution used (something like) rand() to generate passwords for new users
  - Given your password, you could predict the passwords of other users
- Kerberos (1988 - 1996)
  - Random number generator improperly seeded
  - Possible to trivially break into machines that rely upon Kerberos for authentication
- Online gambling websites
  - Random numbers to shuffle cards
  - Real money at stake
  - But what if poor choice of random numbers?

Images from http://www.cigital.com/news/index.php?pg=art&artid=20



Images from http://www.cigital.com/news/index.php?pg=art&artid=20

Images from http://www.cigital.com/news/index.php?pg=art&artid=20

Big news... CNN, etc..

Other Problems

- Live CDs, diskless clients
  - May boot up in same state every time
- Virtual Machines
  - Save state: Opportunity for attacker to inspect the pseudorandom number generator's state
  - Restart: May use same "psuedorandom" value more than once

Obtaining Pseudorandom Numbers

- For security applications, want "cryptographically secure pseudorandom numbers"
- Libraries include:
  - OpenSSL
  - Microsoft's Crypto API
- Linux:
  - /dev/random
  - /dev/urandom
- Internally:
  - Pool from multiple sources (interrupt timers, keyboard, ...)
  - Physical sources (radioactive decay, ...)

Timing Attacks

- Assume there are no "typical" bugs in the software
  - No buffer overflow bugs
  - No format string vulnerabilities
  - Good choice of randomness
  - Good design
- The software may still be vulnerable to timing attacks
  - Software exhibits input-dependent timings
- Complex and hard to fully protect against

Password Checker

- Functional requirements
  - PwdCheck(RealPwd, CandidatePwd) should:
    - Return TRUE if RealPwd matches CandidatePwd
    - Return FALSE otherwise
  - RealPwd and CandidatePwd are both 8 characters long
- Implementation (like TENEX system)

```
PwdCheck(RealPwd, CandidatePwd)   // both 8 chars
    for i = 1 to 8 do
        if (RealPwd[i] != CandidatePwd[i]) then
            return FALSE
    return TRUE
```

- Clearly meets functional description

Attacker Model

```
PwdCheck(RealPwd, CandidatePwd)  // both 8 chars
    for i = 1 to 8 do
        if (RealPwd[i] != CandidatePwd[i]) then
            return FALSE
    return TRUE
```

- Attacker can guess CandidatePwds through some standard interface
- Naive: Try all 256^8 = 18,446,744,073,709,551,616 possibilities

Attacker Model

```
PwdCheck(RealPwd, CandidatePwd)  // both 8 chars
    for i = 1 to 8 do
        if (RealPwd[i] != CandidatePwd[i]) then
            return FALSE
    return TRUE
```

- Attacker can guess CandidatePwds through some standard interface
- Naive: Try all 256^8 = 18,446,744,073,709,551,616 possibilities
- Better: Time how long it takes to reject a CandidatePasswd. Then try all possibilities for first character, then second, then third, ...
  - Total tries: 256*8 = 2048

Other Examples

- Plenty of other examples of timings attacks
  - AES cache misses
    - AES is the "Advanced Encryption Standard"
    - It is used in SSH, SSL, IPsec, PGP, ...
  - RSA exponentiation time
    - RSA is a famous public-key encryption scheme
    - It's also used in many cryptographic protocols and products

Next

- Defensive directions

Toward Preventing Buffer Overflow

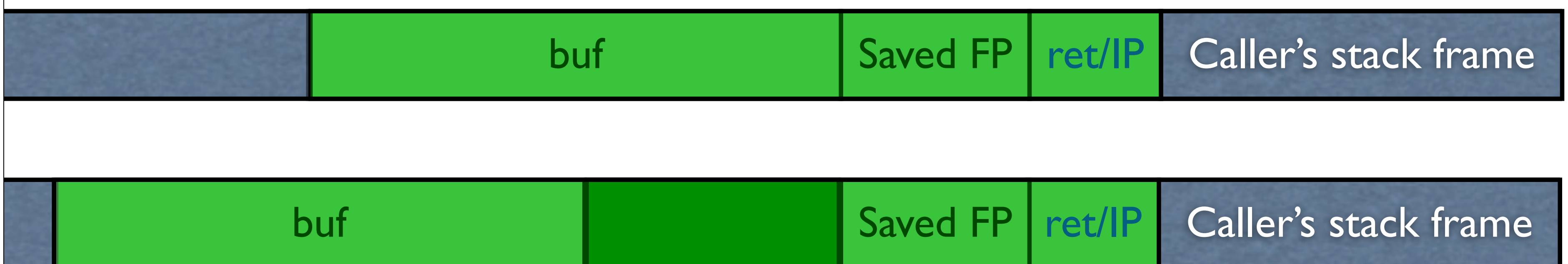
- Use safe programming languages, e.g., Java and C#
  - What about legacy C code?
- Static/dynamic analysis of source code to find overflows
- Black-box testing with long strings
- Mark stack as non-executable
- Randomize stack location or encrypt return address on stack by XORing with random string
  - Attacker won't know what address to use in his or her string
- Run-time checking of array and buffer bounds
  - StackGuard, libsafe, many other tools
- Example companies: Fortify, Coverity

Non-Executable Stack

- NX bit for pages in memory
  - Modern Intel and AMD processors support
  - Modern OS support as well
- Some applications need executable stack
  - For example, LISP interpreters
- Does not defend against return-to-libc exploits
  - Overwrite return address with the address of an existing library function (can still be harmful)
- …nor against heap overflows
- …nor changing stack internal variables (auth flag, ...)

Run-Time Checking: StackGuard

- Embed "canaries" in stack frames and verify their integrity prior to function return
  - Any overflow of local variables will damage the canary



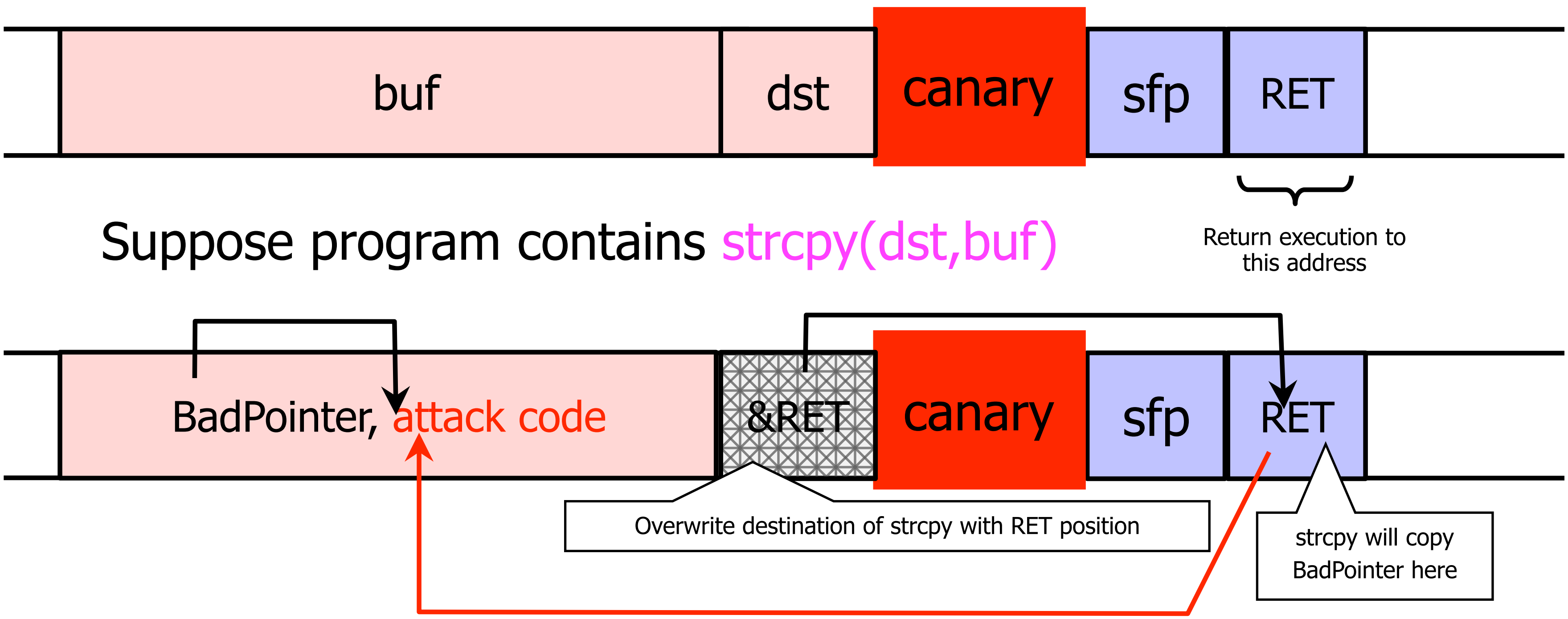
- Choose random canary string on program start
  - Attacker can't guess what the value of canary will be
- Terminator canary: "\0", newline, linefeed, EOF
  - String functions like strcpy won't copy beyond "\0"

StackGuard Implementation

- StackGuard requires code recompilation
- Checking canary integrity prior to every function return causes a performance penalty
- PointGuard also places canaries next to function pointers and setjmp buffers
  - Worse performance penalty
- StackGuard doesn’t completely solve the problem (can be defeated)

Defeating StackGuard (Sketch)

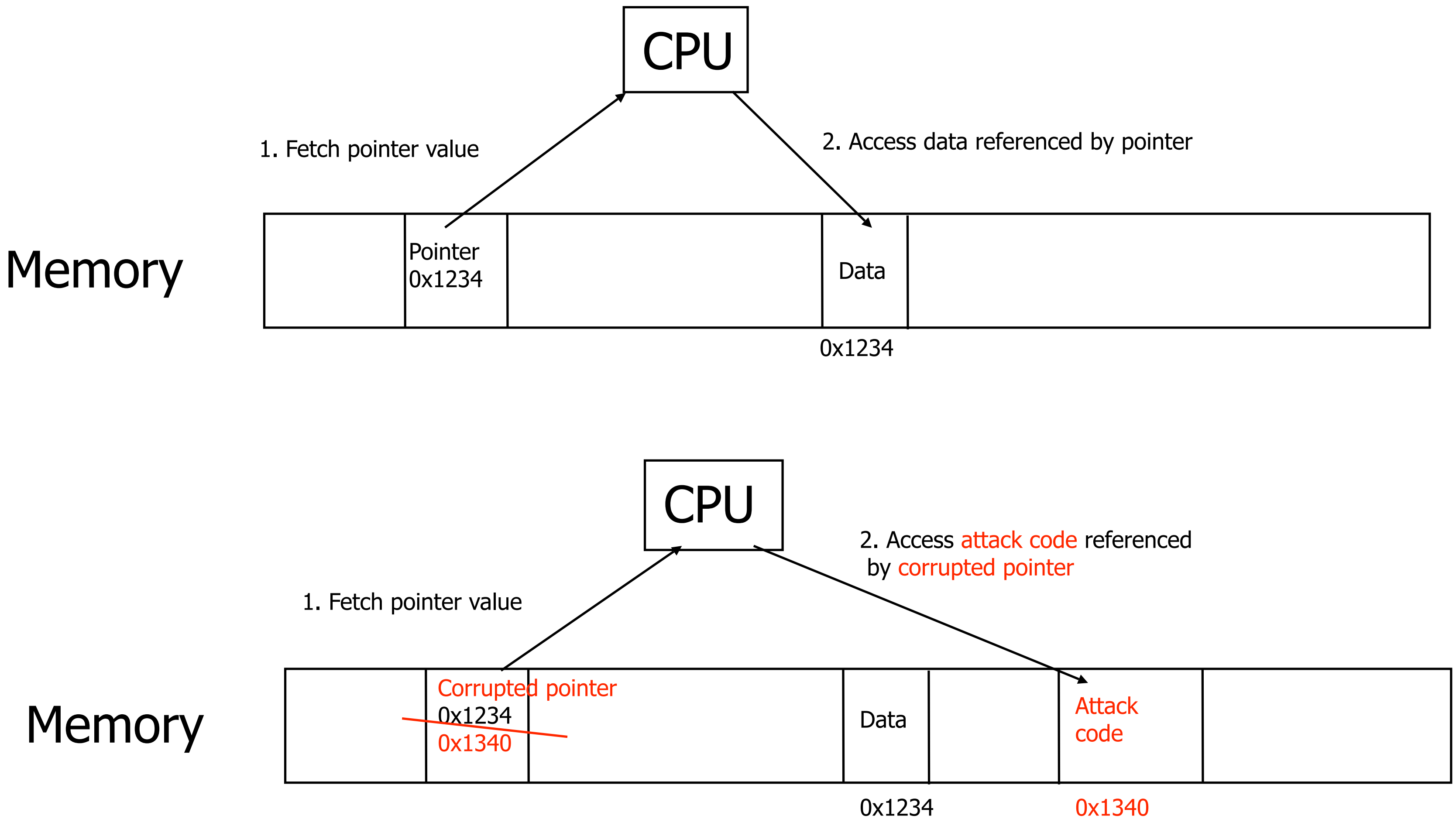
- Idea: overwrite pointer used by some strcpy and make it point to return address (RET) on stack
  - strcpy will write into RET without touching canary!



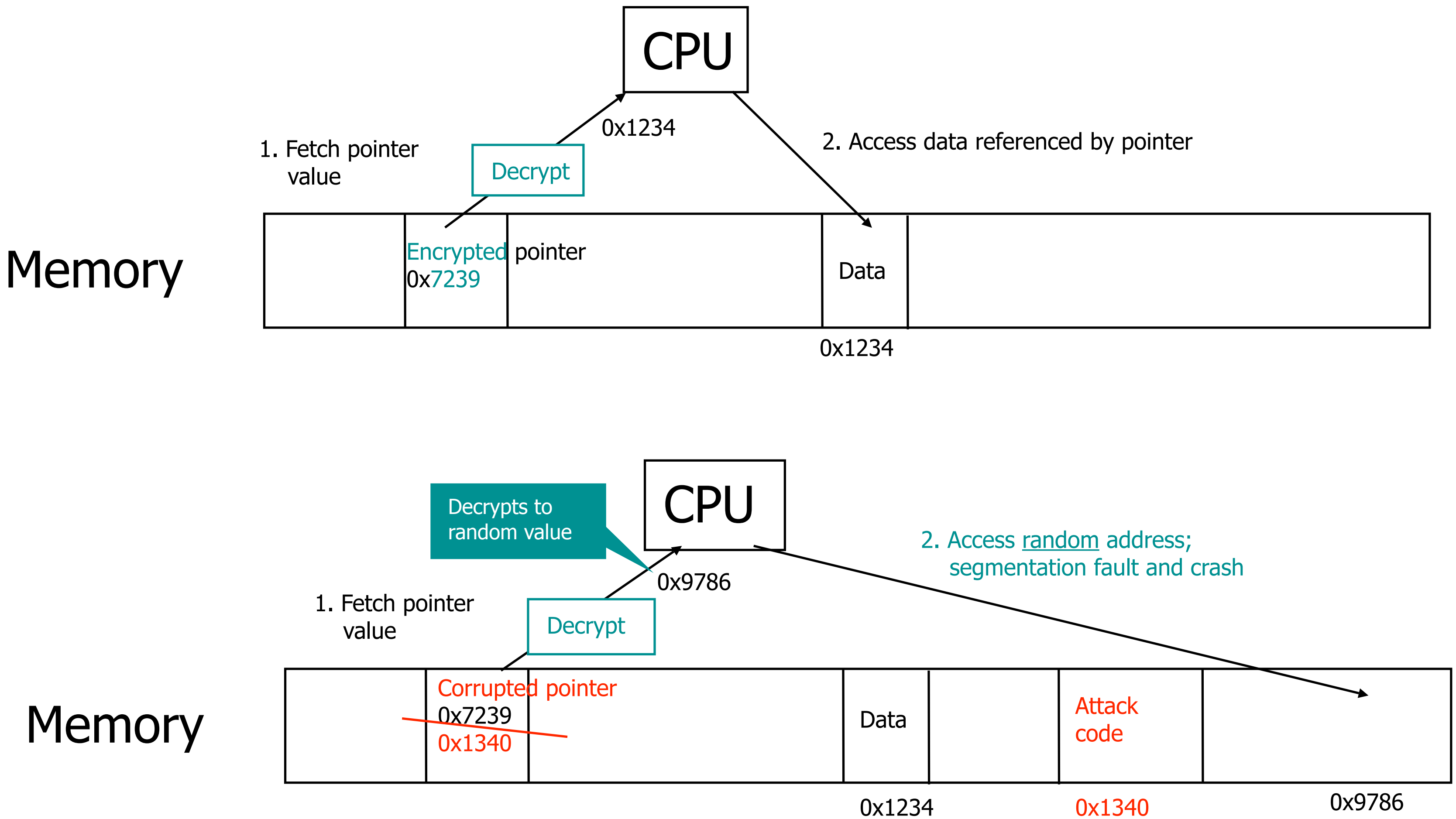
PointGuard

- Attack: overflow a function pointer so that it points to attack code
- Idea: encrypt all pointers while in memory
  - Generate a random key when program is executed
  - Each pointer is XORed with this key when loaded from memory to registers or stored back into memory
    - Pointers cannot be overflown while in registers
- Attacker cannot predict the target program's key
  - Even if pointer is overwritten, after XORing with key it will dereference to a "random" memory address

Normal Pointer Dereference [Cowan]



PointGuard Dereference [Cowan]



Fuzz Testing

- Generate “random” inputs to program
- See if program crashes
  - If crashes, found a bug
  - Bug may be exploitable
- Surprisingly effective

- Now standard part of development lifecycle

- Sometimes conforming to input structures (file formats, etc)

Principles

- Check inputs

Principles

- Least privilege

Principles

- Check all return values

Principles

- Securely clear memory (passwords, keys, etc)

Principles

- Failsafe defaults

Principles

- Defense in depth

Principles

- Reduce size of TCB
- Simplicity
- Modularity

Principles

- Open design? Open source?
- Maybe...

- Linux Kernel Backdoor Attempt: http://www.freedom-to-tinker.com/?p=472

- PGP Corporation: http://www.pgp.com/developers/sourcecode/index.html

Vulnerability Analysis and Disclosure

- What should you think about before analyzing the security of a real system?

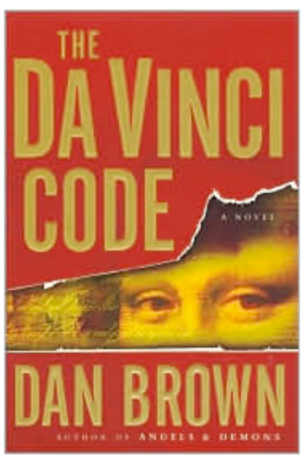
- What do you do if you've found a security problem in a real system?
- Say
  - Electronic voting machine?
  - Airplane?
  - iPhone?
  - IRS website?
  - Medical device?

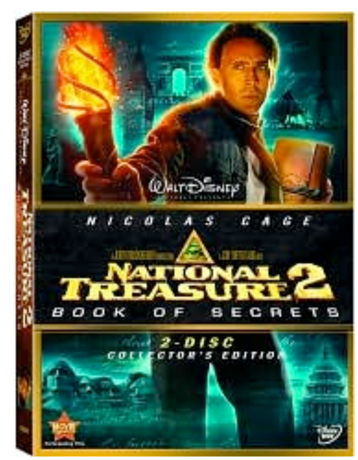
Next

- Cryptography Overview

Cryptography and Security

- Art and science of *protecting* our *information*.
  - Keeping it private, if we want privacy
  - Protecting its integrity, if we want to avoid forgeries.





Images from Wikipedia and Barnes and Noble

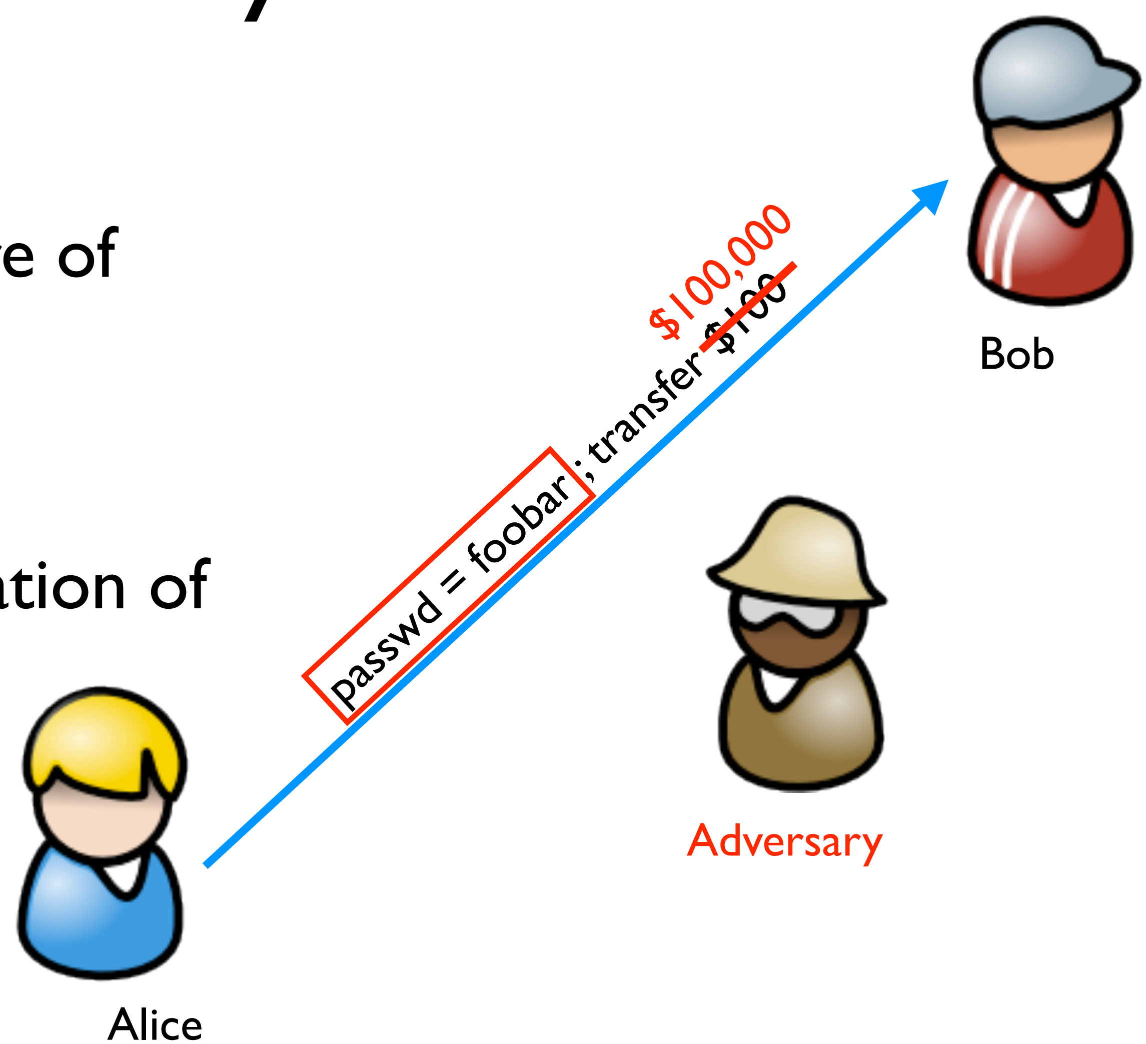
Some thoughts about cryptography

- Cryptography only one small piece of a larger system
- Must protect entire system
  - Physical security
  - Operating system security
  - Network security
  - Users
  - **Cryptography** (following slides)
- "Security only as strong as the weakest link"
  - Need to secure weak links
  - But not always clear what the weakest link is (different adversaries and resources, different adversarial goals)
  - Crypto failures may not be (immediately) detected
- Cryptography helps after you've identified your threat model and goals

Common Communication Security Goals

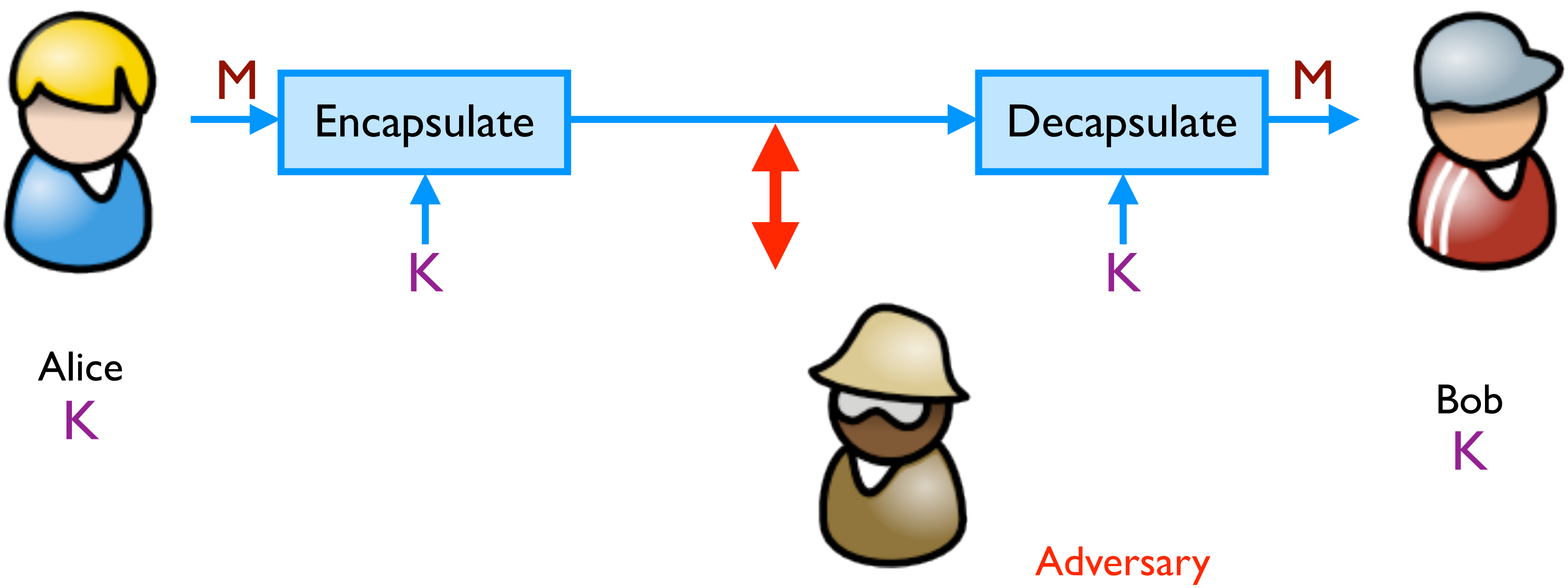
Privacy of data
Prevent exposure of information

Integrity of data
Prevent modification of information



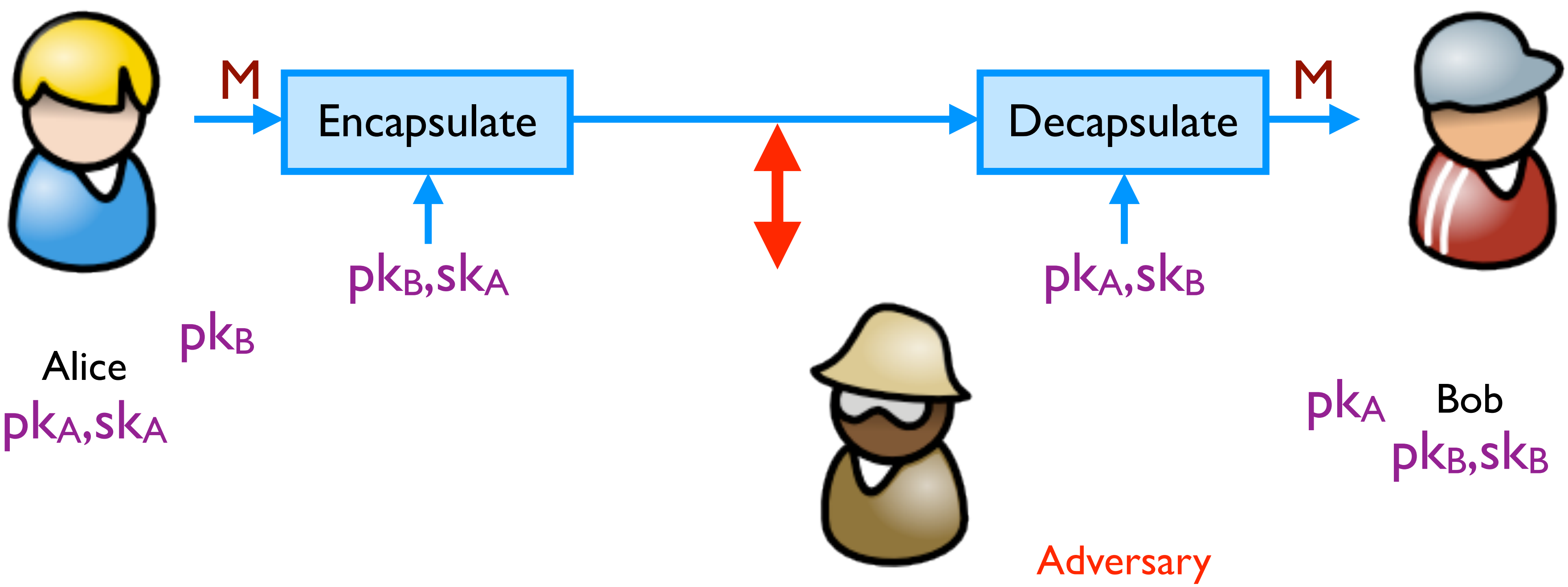
Symmetric Setting

Both communicating parties have access to a shared random string K, called the key.



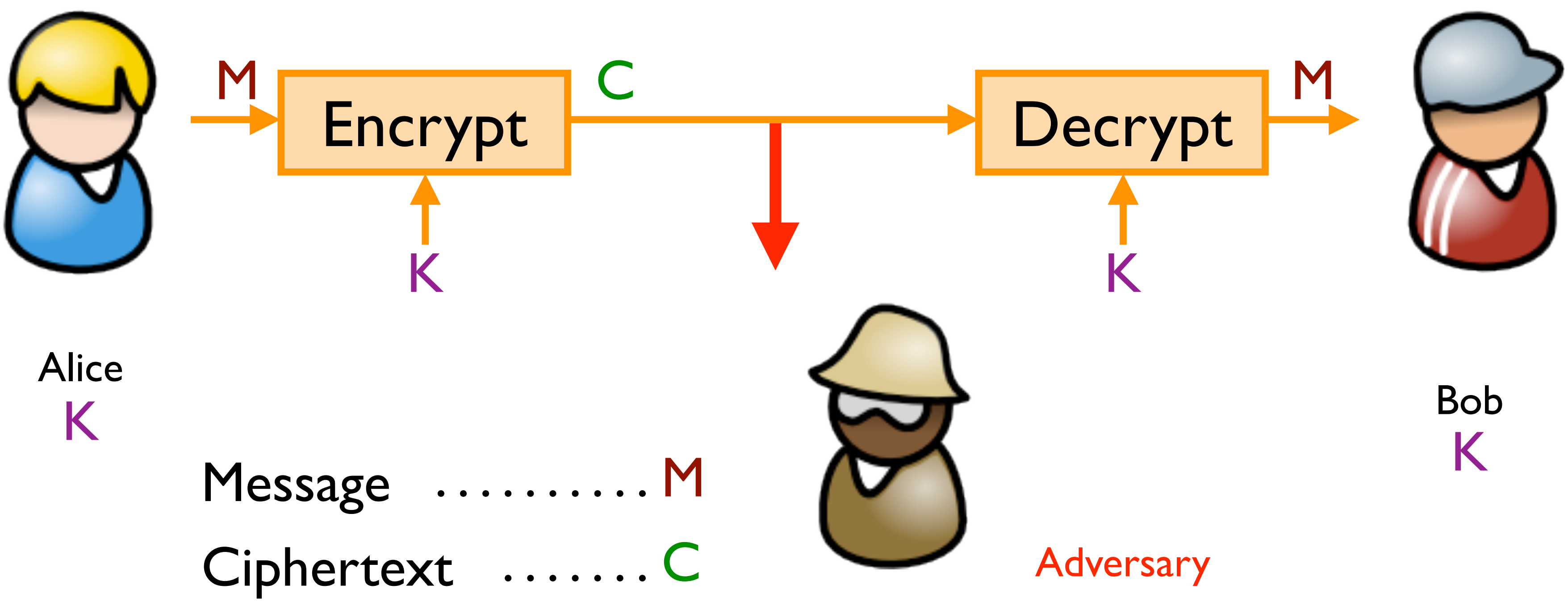
Asymmetric Setting

Each party creates a public key pk and a secret key sk.



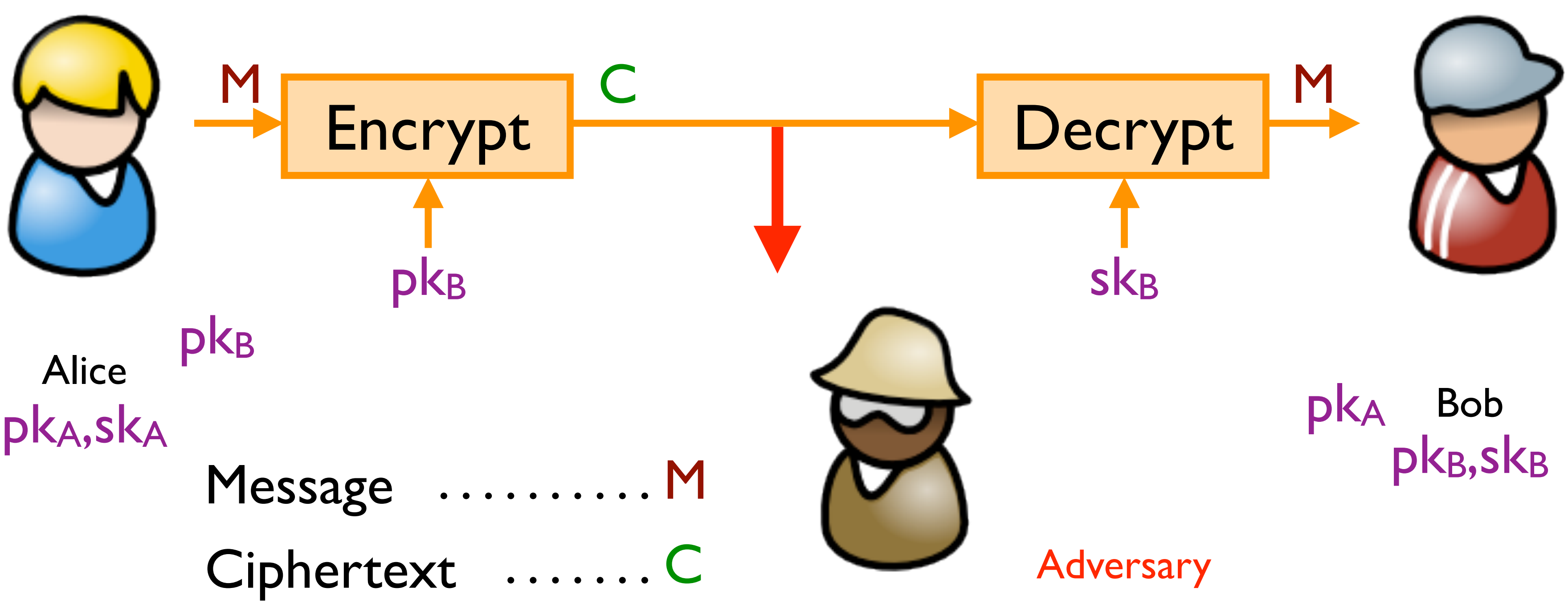
Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.



Achieving Privacy (Asymmetric)

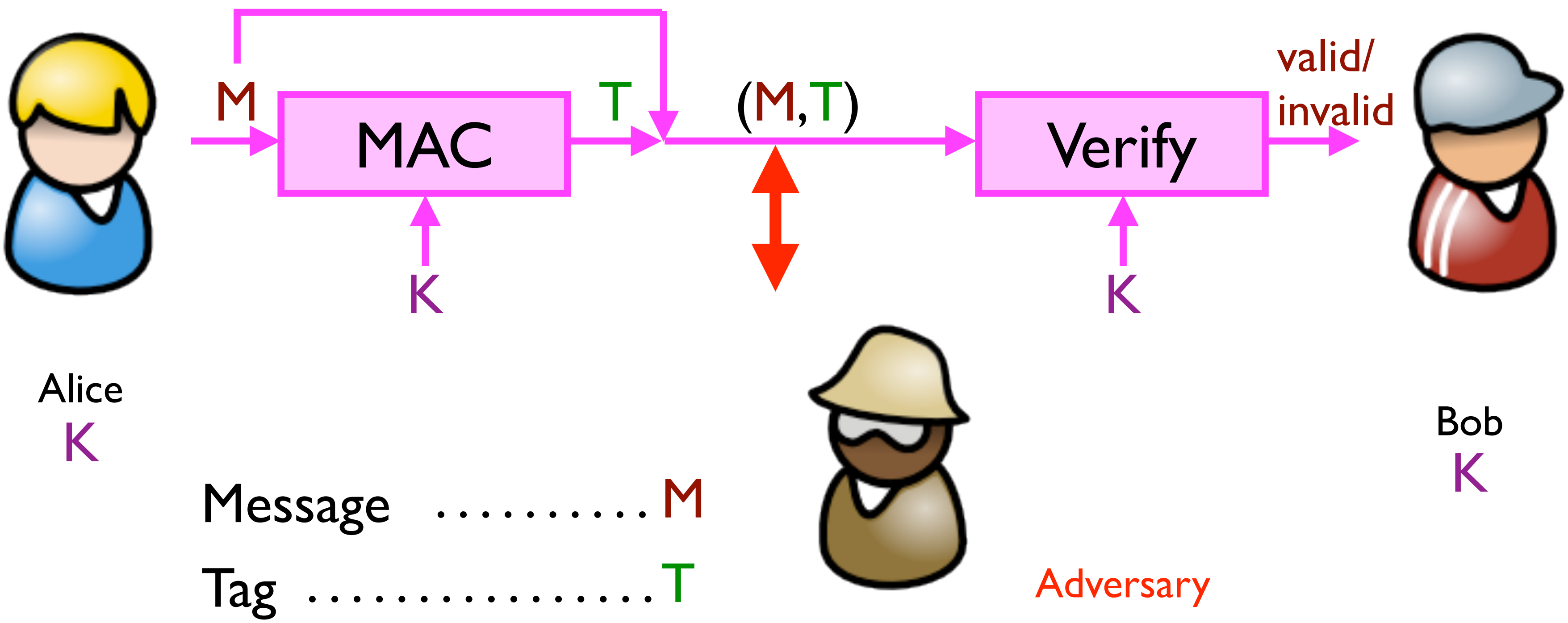
Encryption schemes: A tool for protecting privacy.



Achieving Integrity (Symmetric)

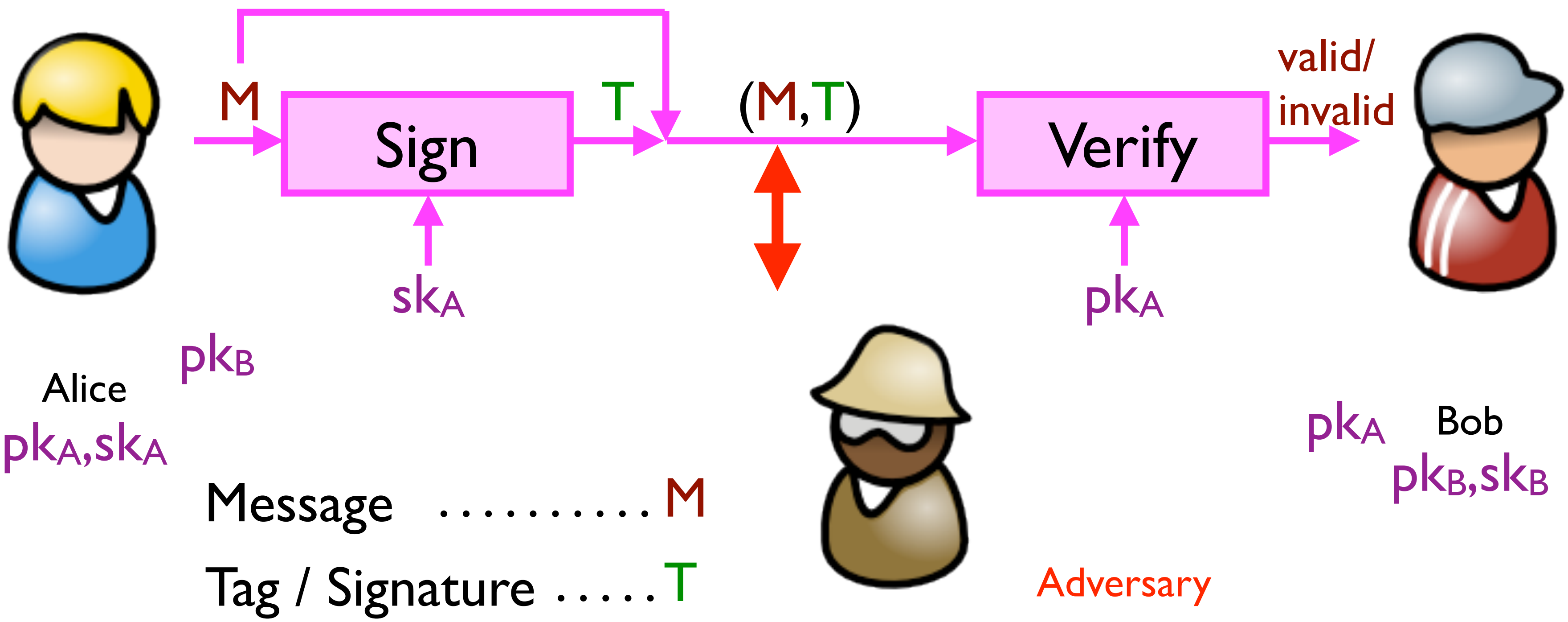
Message authentication schemes: A tool for protecting integrity.

(Also called message authentication codes or MACs.)



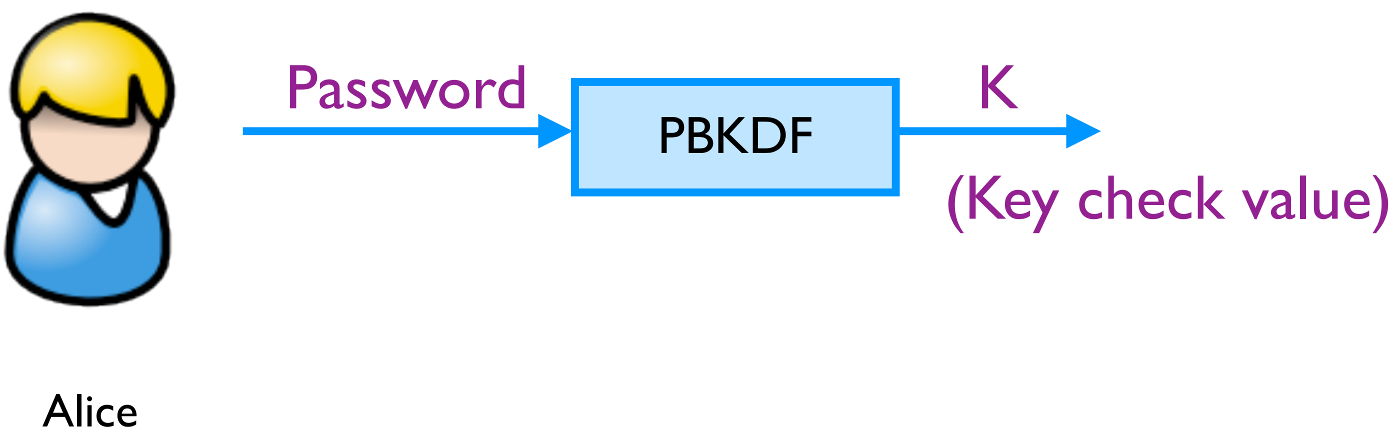
Achieving Integrity (Asymmetric)

Digital signature schemes: A tool for protecting integrity and authenticity.



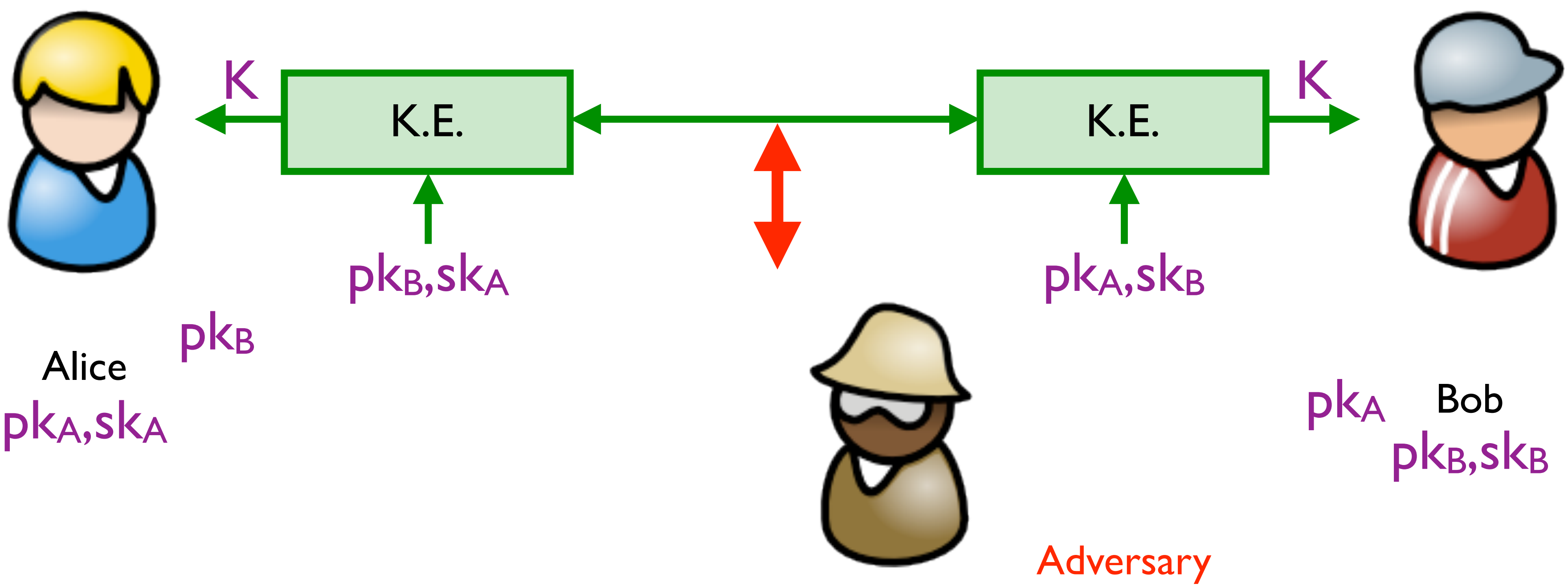
Getting keys: PBKDF

Password-based Key Derivation Functions



Getting keys: Key exchange

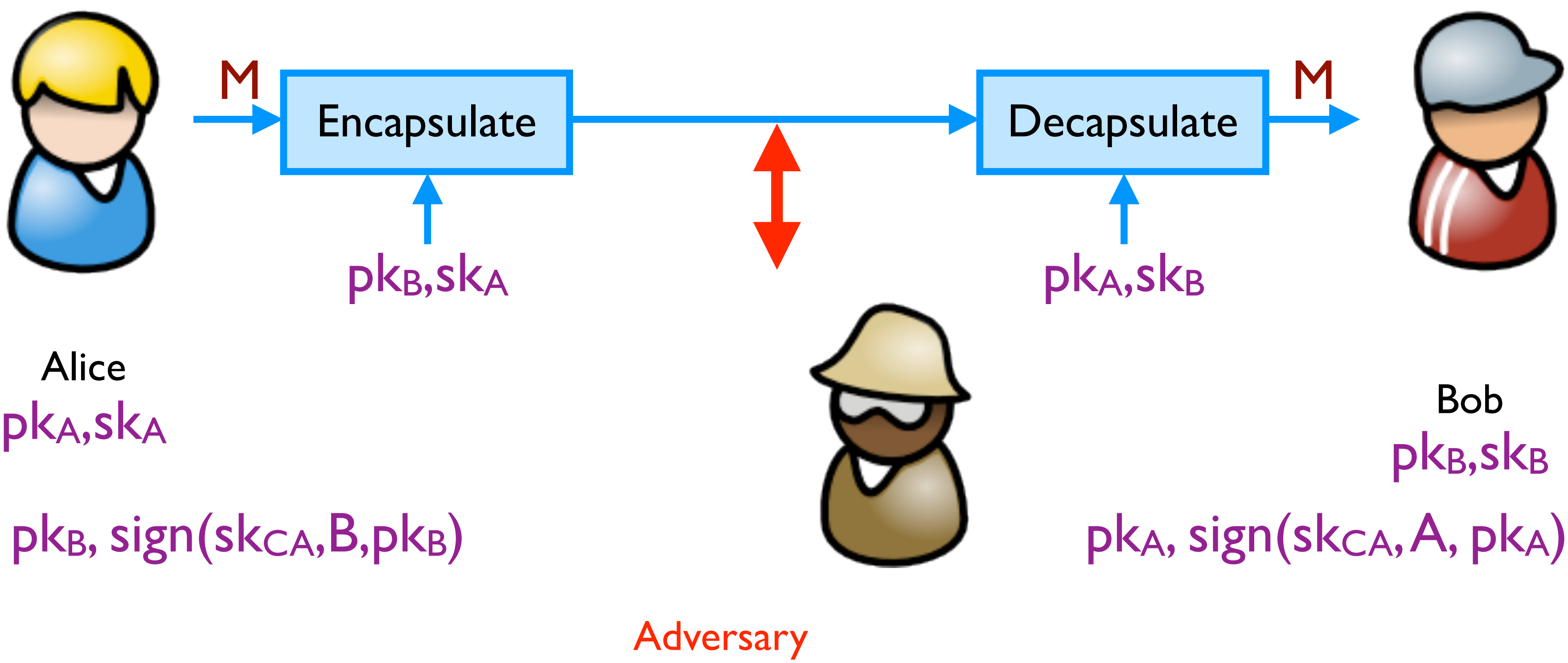
Key exchange protocols: A tool for establishing a share symmetric key



Getting keys: CAs

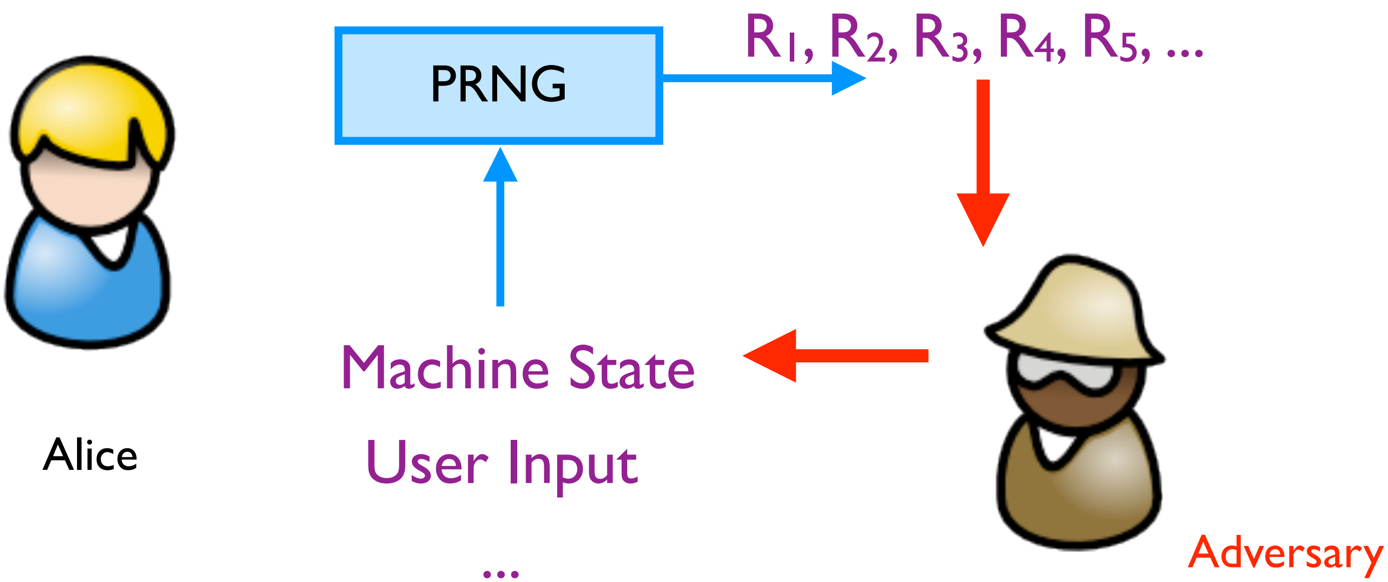
Each party creates a public key pk and a secret key sk.

(Public keys signed by a trusted third party: a certificate authority.)

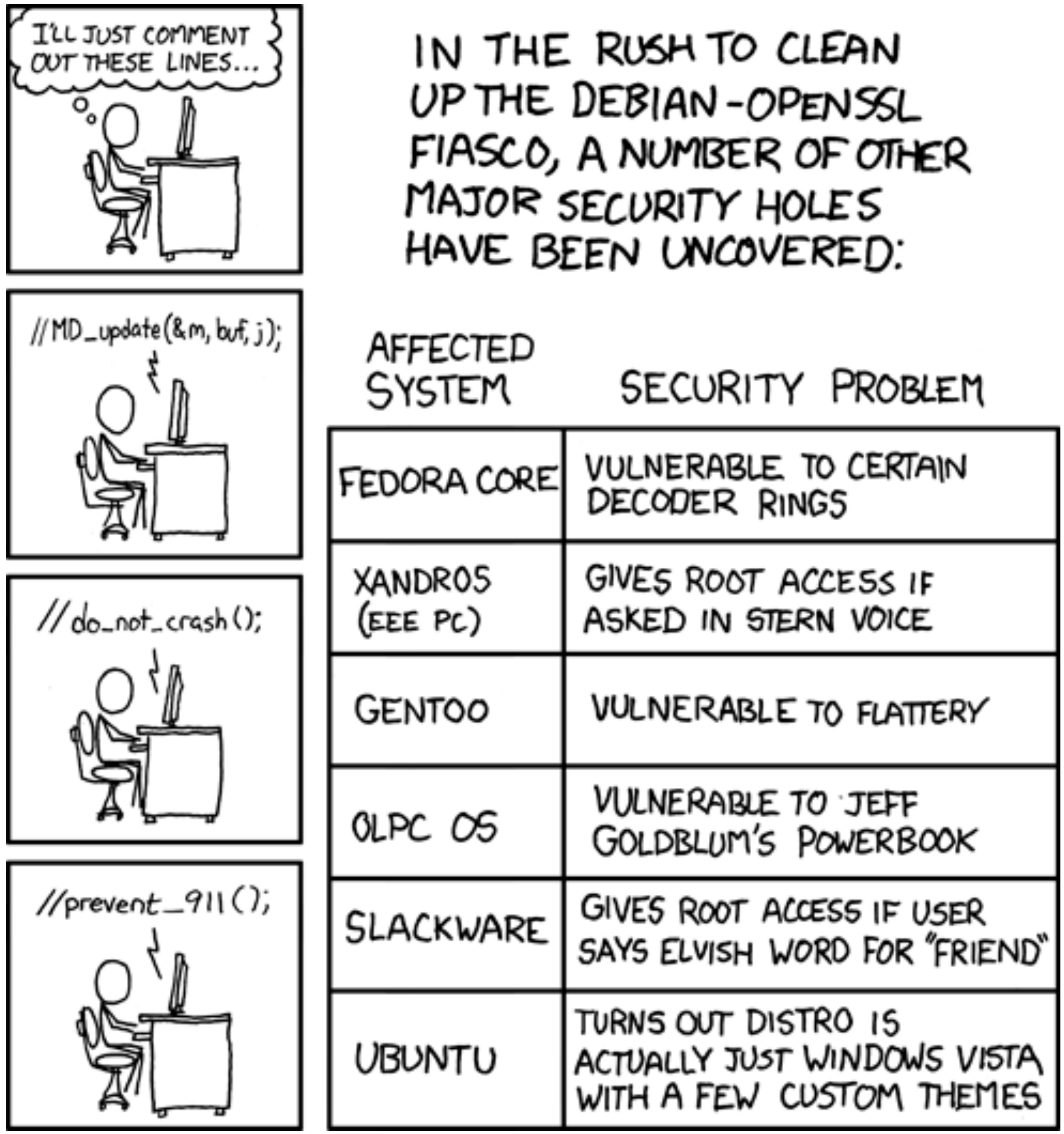


“Random” Numbers

Pseudorandom Number Generators (PRNGs)




DILBERT By Scott Adams
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
scottadams@aol.com
www.dilbert.com
NINE NINE NINE NINE NINE NINE
© 2001 United Feature Syndicate, Inc.
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.


I'LL JUST COMMENT OUT THESE LINES...
//MD_update(&m, buf, j);
//do_not_crash();
//prevent_911();
IN THE RUSH TO CLEAN UP THE DEBIAN-OPENSSL FIASCO, A NUMBER OF OTHER MAJOR SECURITY HOLES HAVE BEEN UNCOVERED:
AFFECTED SYSTEM
SECURITY PROBLEM
FEDORA CORE
VULNERABLE TO CERTAIN DECODER RINGS
XANDROS (EEE PC)
GIVES ROOT ACCESS IF ASKED IN STERN VOICE
GENTOO
VULNERABLE TO FLATTERY
OLPC OS
VULNERABLE TO JEFF GOLDBLUM'S POWERBOOK
SLACKWARE
GIVES ROOT ACCESS IF USER SAYS ELVISH WORD FOR "FRIEND"
UBUNTU
TURNS OUT DISTRO IS ACTUALLY JUST WINDOWS VISTA WITH A FEW CUSTOM THEMES

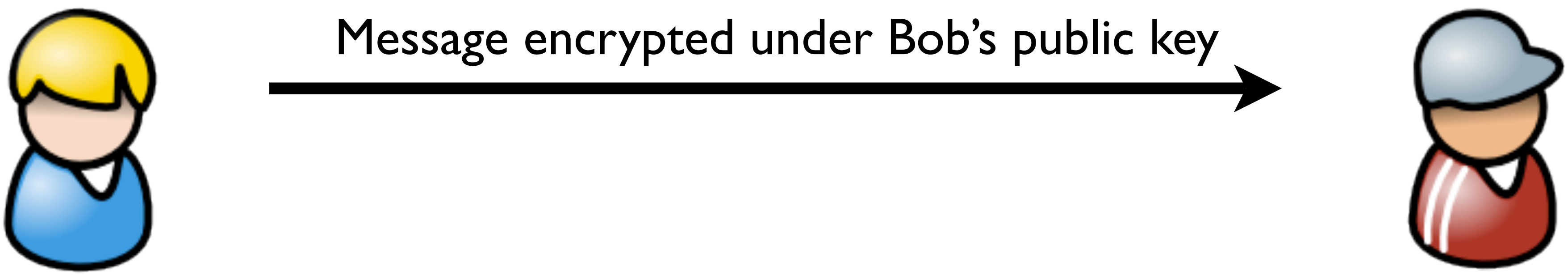
Source: XKCD

Kerckhoff's Principle

- Security of a cryptographic object should depend **only** on the secrecy of the secret (private) key
- Security should not depend on the secrecy of the algorithm itself.
- Why?

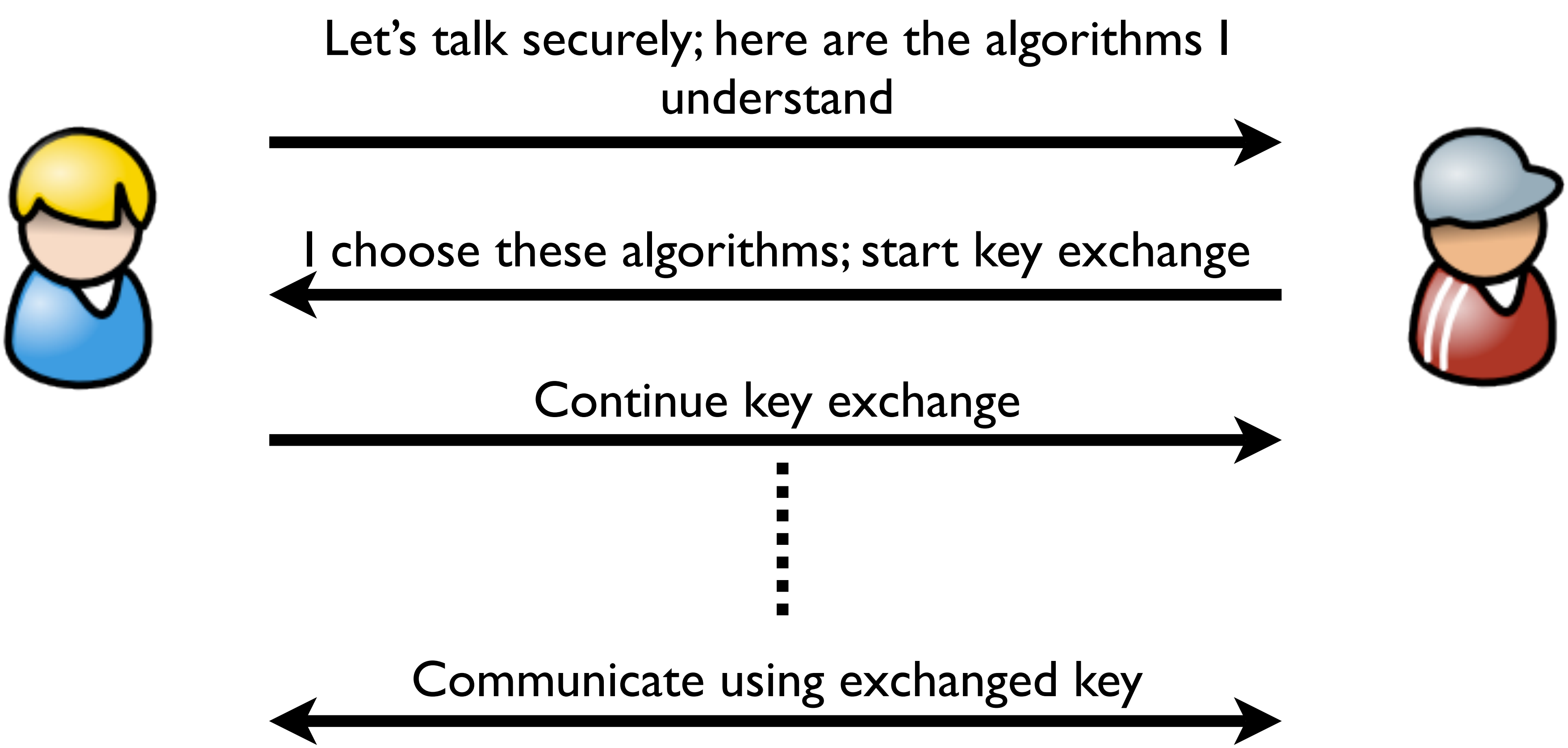
One-way Communications

PGP is a good example



Interactive Communications

In many cases, it's probably a good idea to just use a standard protocol/system like SSH, SSL/TLS, etc...



Let's Dive a Bit Deeper

One-way Communications

(*Informal* example; ignoring, e.g., signatures)

1. Alice gets Bob's public key; Alice *verifies* Bob's public key (e.g., via CA)
2. Alice generates random symmetric keys K1 and K2
3. Alice encrypts the message M the key K1; call result C
4. Alice authenticates (MACs) C with key K2; call the result T
5. Alice encrypts K1 and K2 with Bob's public key; call the result D

6. Send D, C, T

(Assume Bob's private key is encrypted on Bob's disk.)

7. Bob takes his password to derive key K3
8. Bob decrypts his private key with key K3
9. Bob uses private key to decrypt K1 and K2
10. Bob uses K2 to verify MAC tag T
11. Bob uses K1 to decrypt C

Interactive Communications

(*Informal* example; details omitted)

1. Alice and Bob exchange public keys and certificates
2. Alice and Bob use CA's public keys to verify certificates and each other's public keys
3. Alice and Bob take their passwords and derive symmetric keys
4. Alice and Bob use those symmetric keys to decrypt and recover their asymmetric private keys.
5. Alice and Bob use their asymmetric private keys and a *key exchange* algorithm to derive a shared symmetric key

(They key exchange process will require Alice and Bob to generate new pseudorandom numbers)

6. Alice and Bob use shared symmetric key to encrypt and authenticate messages

(Last step will probably also use random numbers; will need to rekey regularly; may need to avoid replay attacks,...)

Next

- Brief History

What cryptosystems have you heard of? (Past or present)

History

- Substitution Ciphers
  - Caesar Cipher
- Transposition Ciphers
- Codebooks
- Machines

- Recommended Reading: **The Codebreakers** by David Kahn and **The Code Book** by Simon Singh.
  - Military uses
  - Rumrunners
  - ....

Classic Encryption

- Goal: To communicate a secret message
- Start with an *algorithm*
- Caesar cipher (substitution cipher):

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

GHIJKLMNOPQRSTUVWXYZABCDEF
```

Then add a secret key

- Both parties know that the secret word is "victory":

  `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

  VICTORYABDEFGHJKLMNPQSUWXZ

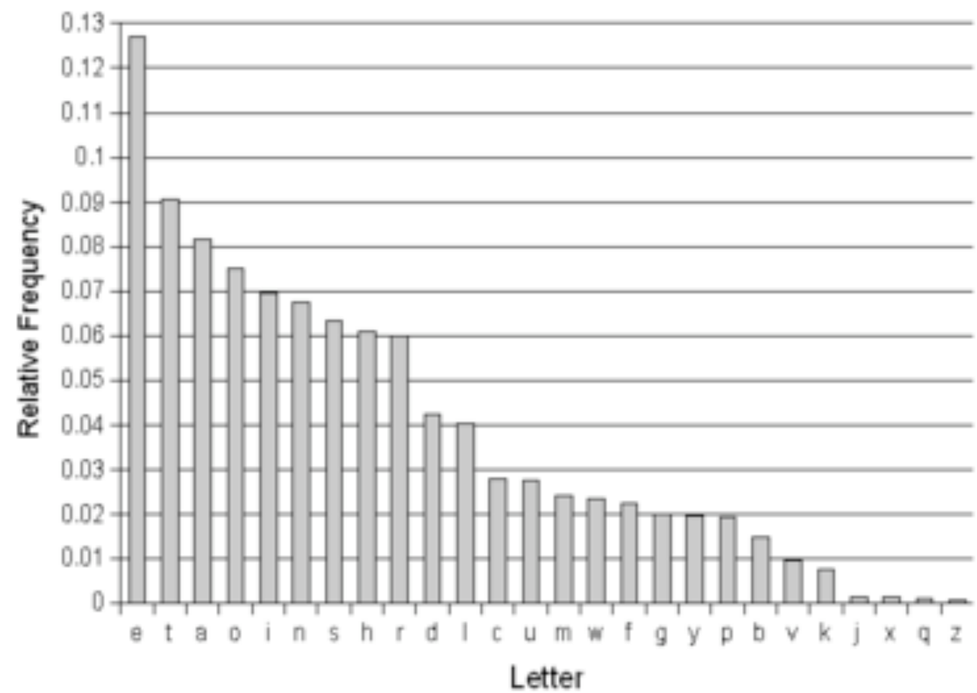
- "state of the art" for thousands of years

Cryptographers vs Cryptanalysts

- A battle that continues today
- Cryptographers try to devise more clever algorithms and keys
- Cryptanalysts search for vulnerabilities
- Early cryptanalysts were linguists:
  - frequency analysis
  - properties of letters

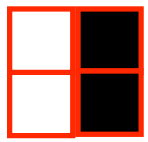
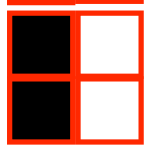
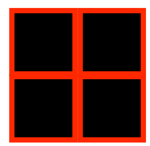
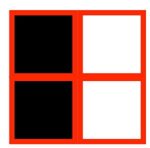
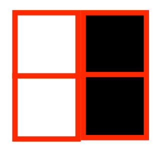
Cryptanalysis and probabilities

Letter	Frequency
a	8.167%
b	1.492%
c	2.782%
d	4.253%
e	12.702%
f	2.228%
g	2.015%
h	6.094%
i	6.966%
j	0.153%
k	0.772%
l	4.025%



From http://en.wikipedia.org/wiki/Letter_frequencies

Diversity in Modern Crypto

- **Visual Cryptography**
- Take a black and white bitmap image
- Encode 0 as: 
- Encode 1 as: 
- 1 xor 0 = 0 xor 1 = 1: 
- 1 xor 1 = 0 xor 0 = 0:  or 
- Nice toolkit online here: http://www.cl.cam.ac.uk/~fms27/vck/

See also http://www.cs.washington.edu/homes/yoshi/cs4hs/cse-vc.html

Key Entry Pad (4-digit PIN)

- This is the key pad on my office safe.
- Inside my safe is a copy of final exam.
- How long would it take a you to break in?

- Answer (combinatorics):
  - 10^4 tries *maximum*.
  - 10^4 / 2 tries on *average*.
- Answer (unit conversion):
  - 3 seconds per try --> 4 hours and 10 minutes on average

Image from profmason.com

Key Entry Pad (4-digit PIN)

Image from profmason.com

- Now assume the safe automatically calls police after 3 failed attempts.
- What is the probability that you will guess the PIN within 3 tries?
- (Assume no repeat tries.)

- Answer (combinatorics):
  - 10000 choose 3 possible choices for the 3 guesses
  - $1 \times (9999 \text{ choose } 2)$ possible choices contain the correct PIN
  - So success probability is 3 / 10000

Key Entry Pad (4-digit PIN)

- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)

- Couldyou do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)

- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)

- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

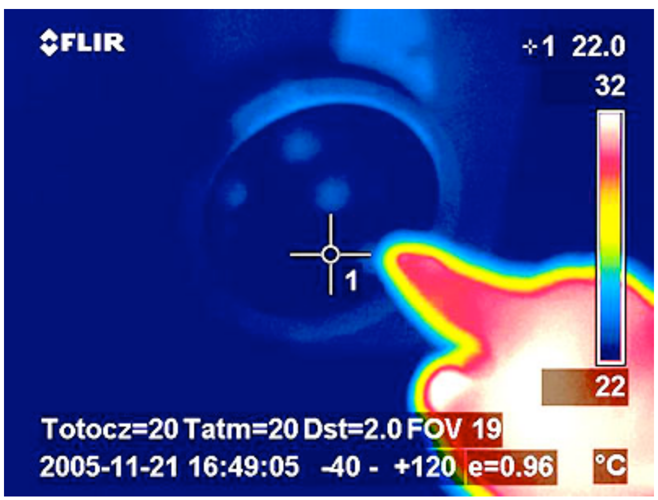
Lesson: Consider the complete system, physical security, etc

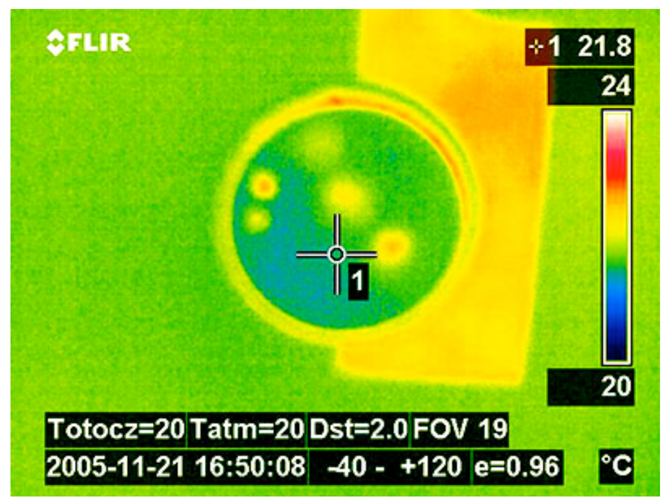
Lesson: Think outside the box

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Thermal Patterns





Images from http://lcamtuf.coredump.cx/tsafe/

General approach for crypto today

- Layered approach:
  - Cryptographic primitives, like block ciphers, stream ciphers, hash functions, and one-way trapdoor permutations
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication
- Public algorithms (Kerckhoff's Principle)
- Security proofs based on assumptions (not this course)