WILLIAM & MARY

CHARTERED 1693

CSCI 667: Concepts of Computer Security

Lecture 4.b

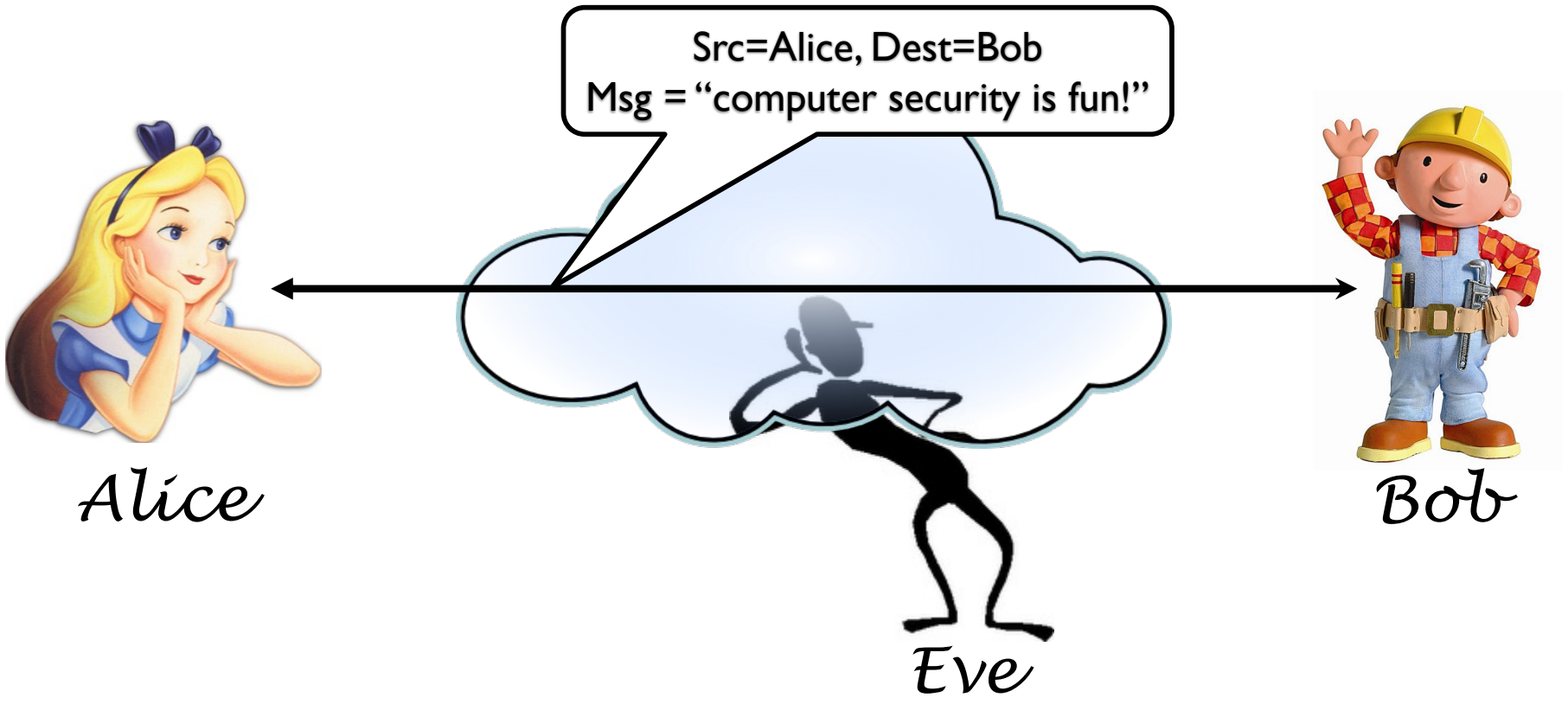
Prof. Adwait Nadkarni

Derived from slides by William Enck, Micah Sherr, Patrick McDaniel, and Peng Ning

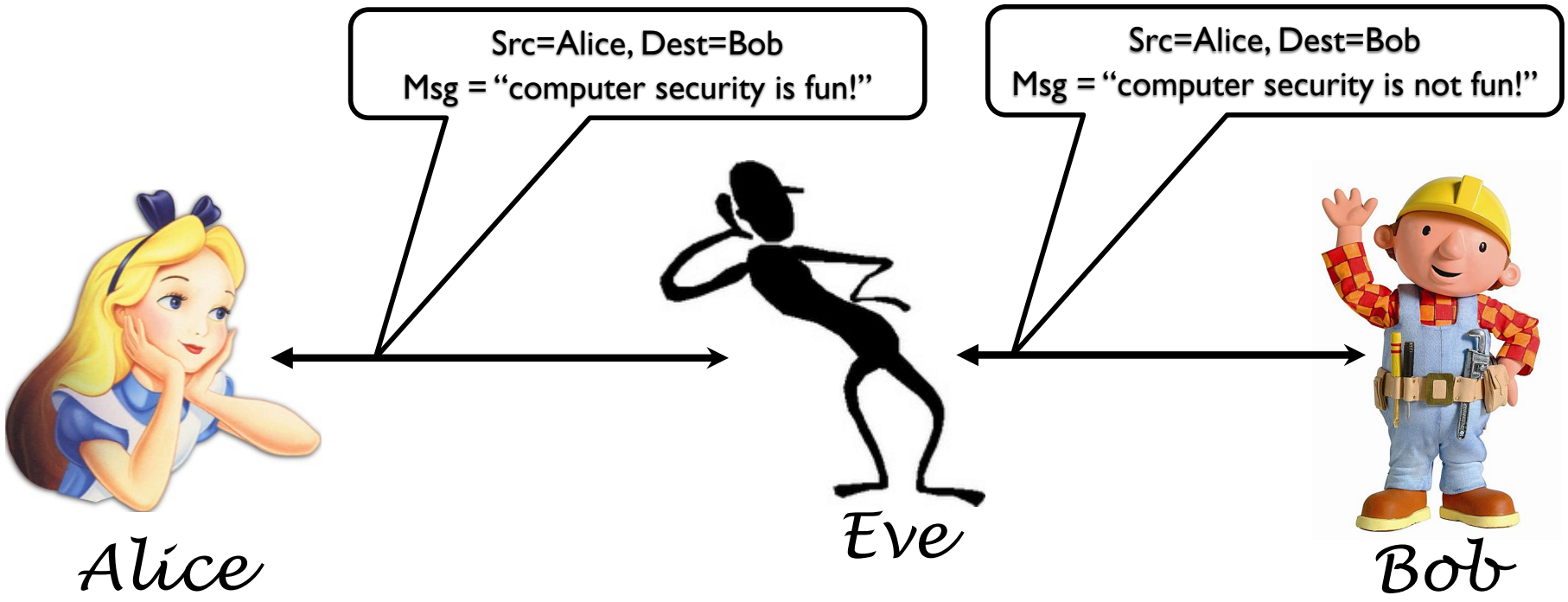
What encryption does and does not

- Does:
  - confidentiality
- Doesn't do:
  - data integrity
  - source authentication
- **Need:** ensure that data is not altered and is from an authenticated source

Principals


Src=Alice, Dest=Bob
Msg = "computer security is fun!"
Alice
Eve
Bob

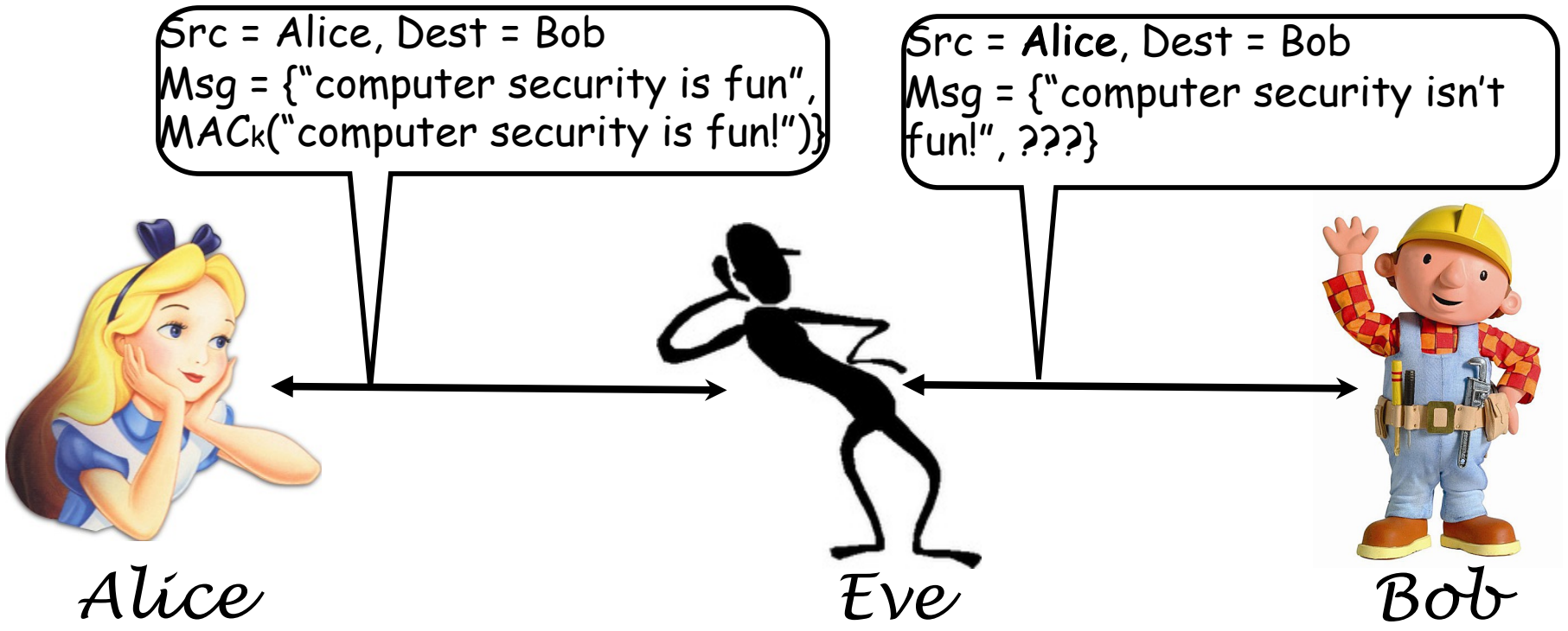
Man-in-the-Middle (MitM) attack



Message Authentication Codes (MACs)

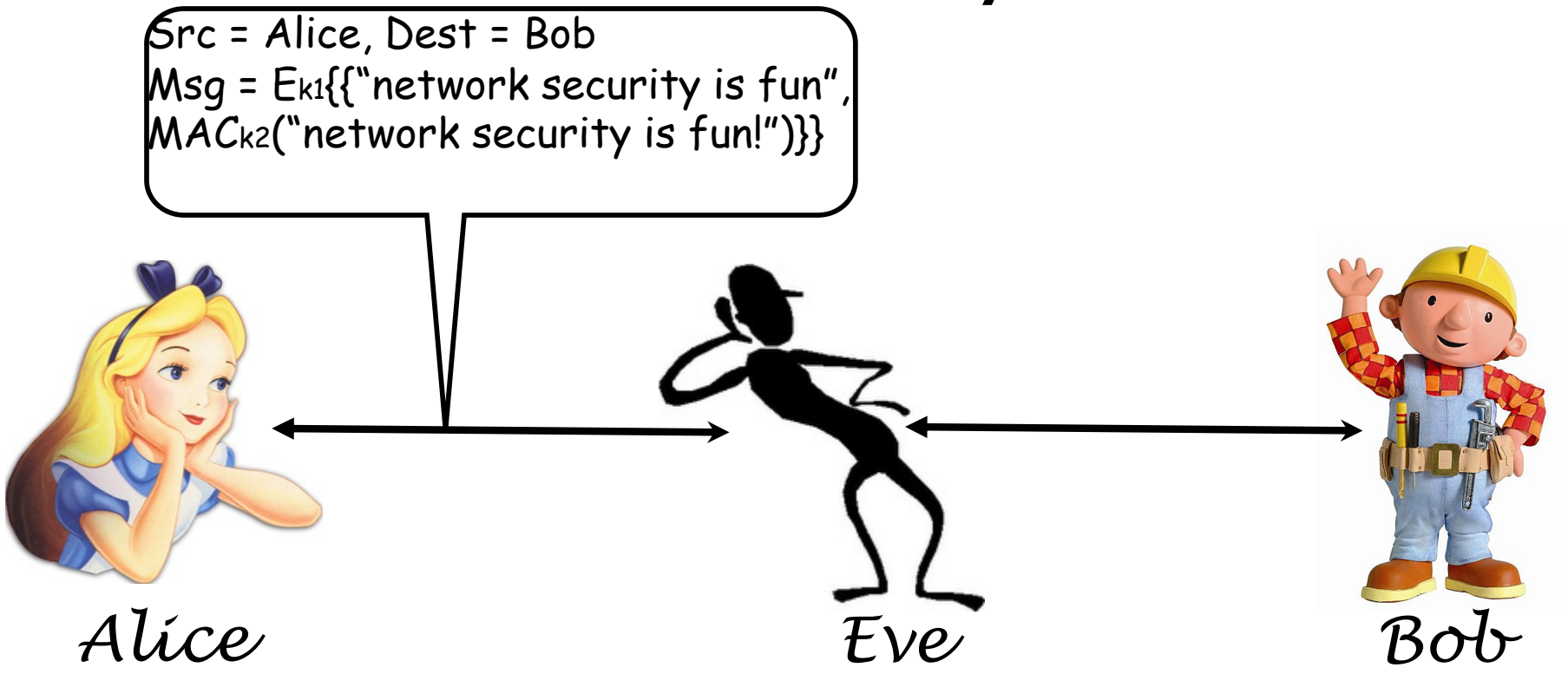
- MACs provide message **integrity** and **authenticity**
- $MAC_K(M)$ – use symmetric encryption to produce **short sequence of bits** that depends on both the message (M) and the key (K)
- MACs should be resistant to **existential forgery**: Eve should not be able to produce a valid MAC for a message M' without knowing K
- To provide confidentiality, authenticity, and integrity of a message, Alice sends
  - $E_K(M, MAC_K(M))$ where $E_K(X)$ is the encryption of X using key K
- Proves that M was encrypted (confidentiality and integrity) by someone who knew K (authenticity)

Message Authenticity



Without knowledge of *k*, Eve can't compute a valid MAC for her forged message!

Encryption and Message Authenticity



Without knowing *k1*,
Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message!

Cryptographic Hash Functions

- **Hash function** `h`: deterministic one-way function that takes as input an arbitrary message `M` (sometimes called a *preimage*) and returns as output `h(M)`, a small fixed length *hash* (sometimes called a *digest*)
- Hash functions should have the following two properties:
  - *compression*: reduces arbitrary length string to fixed length hash
  - *ease of computation*: given message `M`, `h(M)` is easy to compute

Hash functions are usually fairly inexpensive

(i.e., compared with public key cryptography)

```
adwait$ openssl speed sha
To get the most accurate results, try to run this
program when this computer is idle.
Doing sha1 for 3s on 16 size blocks: 9255072 sha1's in 2.97s
Doing sha1 for 3s on 64 size blocks: 6687775 sha1's in 2.97s
Doing sha1 for 3s on 256 size blocks: 3570692 sha1's in 2.98s
Doing sha1 for 3s on 1024 size blocks: 1234275 sha1's in 2.97s
Doing sha1 for 3s on 8192 size blocks: 174704 sha1's in 2.97s
Doing sha256 for 3s on 16 size blocks: 6374888 sha256's in 2.98s
Doing sha256 for 3s on 64 size blocks: 3926000 sha256's in 2.98s
Doing sha256 for 3s on 256 size blocks: 1697500 sha256's in 2.98s
Doing sha256 for 3s on 1024 size blocks: 532592 sha256's in 2.97s
Doing sha256 for 3s on 8192 size blocks: 72132 sha256's in 2.97s
Doing sha512 for 3s on 16 size blocks: 4913872 sha512's in 2.97s
Doing sha512 for 3s on 64 size blocks: 4915170 sha512's in 2.97s
Doing sha512 for 3s on 256 size blocks: 2160195 sha512's in 2.97s
Doing sha512 for 3s on 1024 size blocks: 795869 sha512's in 2.97s
Doing sha512 for 3s on 8192 size blocks: 113596 sha512's in 2.97s
OpenSSL 0.9.8zh 14 Jan 2016
built on: Jan 23 2017
options:bn(64,64) md2(int) rc4(ptr,char) des(idx,cisc,16,int) aes(partial) blowfish(idx)
compiler: -arch x86_64 -fmessage-length=0 -pipe -Wno-trigraphs -fpascal-strings -fasm-blocks -O3 -
D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIAN -DMD32_REG_T=int -DOPENSSL_NO_IDEA -DOPENSSL_PIC
DOPENSSL_THREADS -DZLIB -mmacosx-version-min=10.6
available timing options: TIMEB USE_TOD HZ=100 [sysconf value]
timing function used: getrusage
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
sha1             49891.95k   144024.59k   307178.70k   425012.39k   482007.81k
sha256           34281.92k    84424.15k   146042.36k   183727.34k   198842.41k
sha512           26445.57k   105956.90k   186126.06k   274305.03k   313698.39k
```

Why might hashes be useful?

- **Message authentication codes (MACs):**
  - e.g.: $MAC_K(M) = h(K|M)$
    (but don't do this, use HMAC instead)
- **Modification detection codes:**
  - detect modification of data
  - any change in data will cause change in hash

Prof. Pedantic proposes the following hash function, arguing that it offers both compression and ease of computation.

- h(M) = 0 if the number of 0s in M is divisible by 3
- h(M) = 1 otherwise

Why is this a lousy crypto hash function?

Cryptographic Hash Functions

- Properties of good cryptographic hash functions:
  - **preimage resistance:** given digest y, computationally infeasible to find preimage x' such that h(x')=y
    (also called "one-way property")
  - **2nd-preimage resistance:** given preimage x, computationally infeasible to find preimage x' such that h(x)=h(x')
    (also called "weak collision resistance")
  - **collision resistance:** computationally infeasible to find preimages i,j such that h(i)=h(j)
    (also called "strong collision resistance")

Birthday Attack

- **Birthday Paradox:** chances that 2+ people share birthday in group of 23 is > 50%.
- General formulation
  - function f() whose output is uniformly distributed over H possible outputs
  - Number of experiments Q(H) until we find a collision is approximately:

$$Q(H) \approx \sqrt{\frac{\pi}{2}H}$$

  - E.g.,

$$Q(365) \approx \sqrt{\frac{\pi}{2}365} = 23.94$$

- Why is this relevant to hash sizes?

See: https://betterexplained.com/articles/understanding-the-birthday-paradox/

Practical Implications

- Choosing two messages that have the same hash h(x) = h(x') is more practical than you might think.
- Example attack: secretary is asked to write a "bad" letter, but wants to replace with a "good" letter.
  - Boss signs the letter after reading

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}

Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the}

Northern {European / Europe} {area / division} . He {will take / has taken} over {the / --}

responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches}

in the {area / region} . Please {afford / give} him {every / all the} help he {may need / needs}

to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the

market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the

{latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum}

of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document}

as proof of identity. An order with his signature, which is {appended / attached}

{authorizes / allows} you to charge the cost to this company at the {above / head office}

address. We {fully / --} expect that our {level / volume} of orders will increase in

the {following / next} year and {trust / hope} that the new appointment will {be / prove}

{advantageous / an advantage} to both our companies.

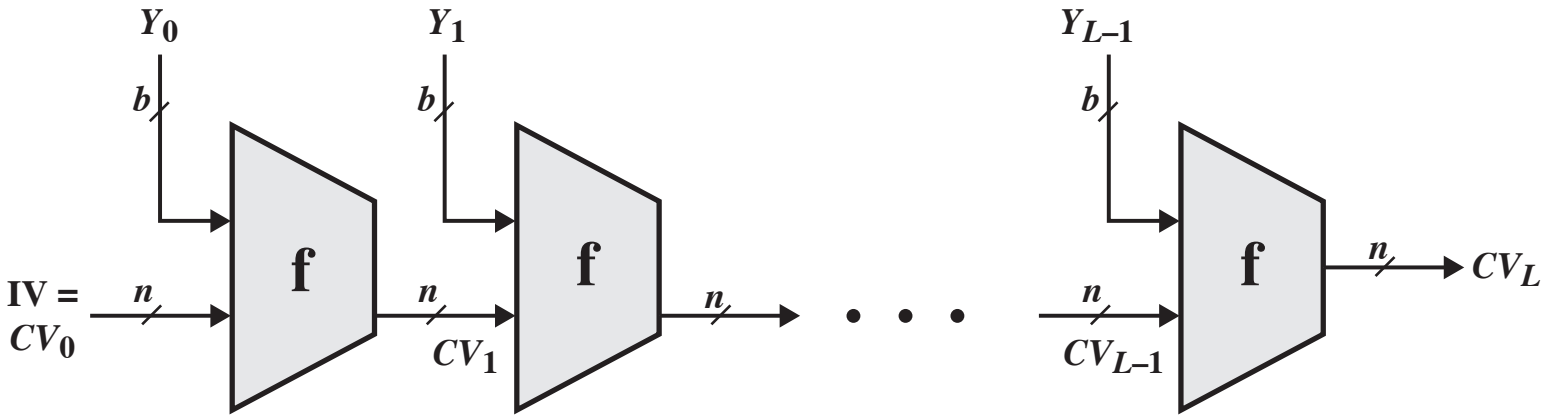
Figure 11.7 A Letter in 2^{37} Variations

(from Stallings, Crypto and Net Security)

Some common cryptographic hash functions

- MD5 (128-bit digest) [don't use this]
- SHA-1 (160-bit digest) [don't use this]
- SHA-256 (256-bit digest) [stop using this*]
- SHA-512 (512-bit digest) [stop using this*]
- SHA-3 [recent competition winner]

General Structure of Hash



IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm

L = number of input blocks
n = length of hash code
b = length of input block

(from Stallings, Crypto and Net Security)

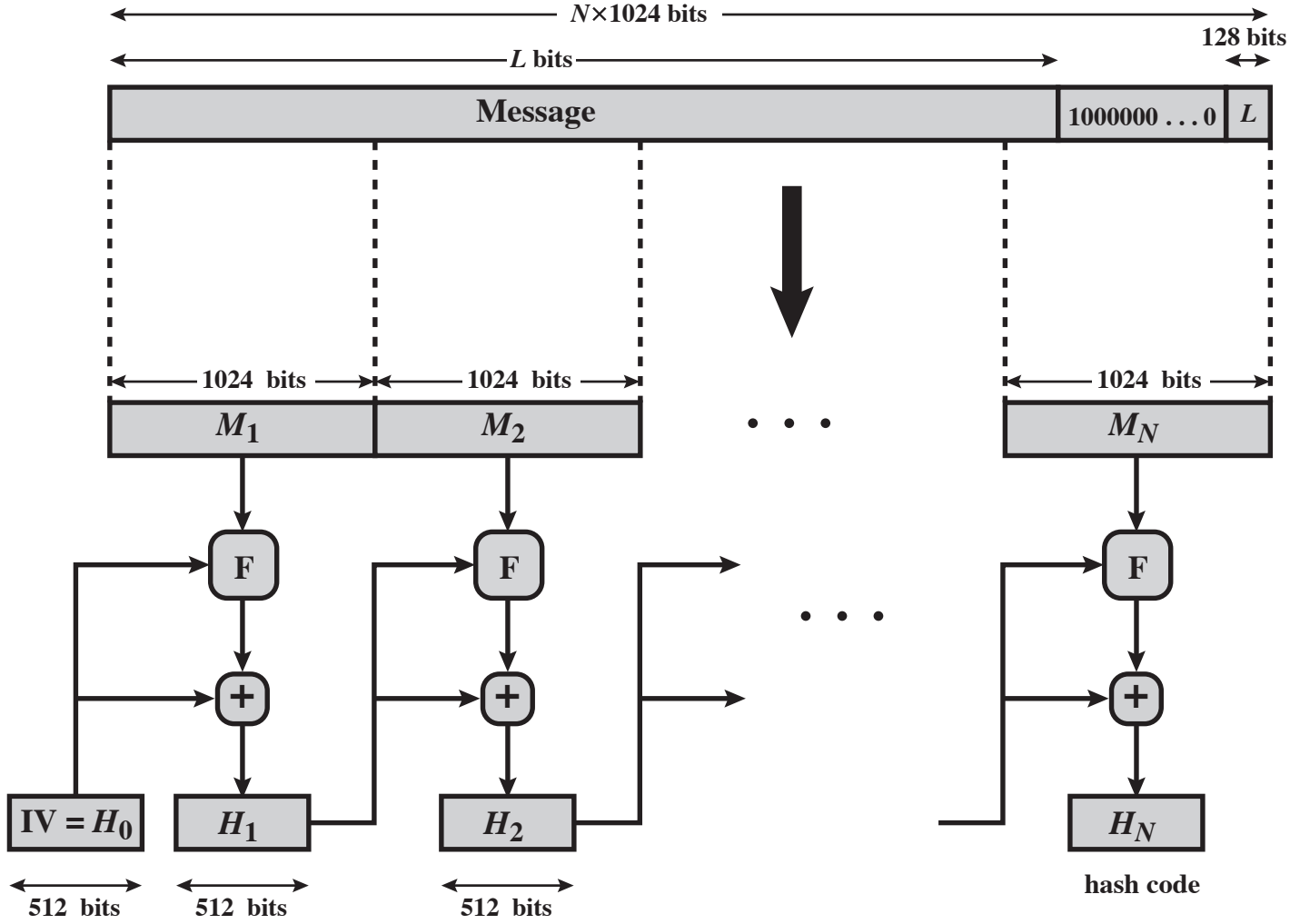
Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Note: All sizes are measured in bits.

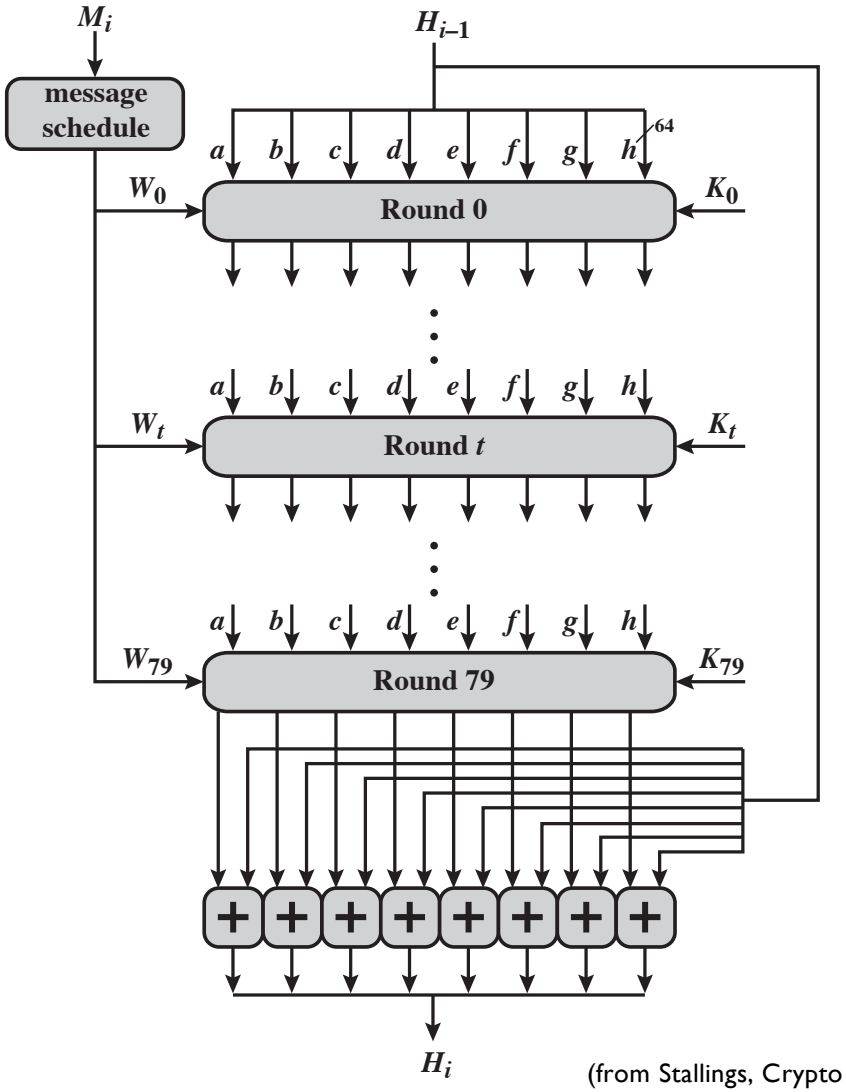
(from Stallings, Crypto and Net Security)

SHA-512



(from Stallings, Crypto and Net Security)

SHA-512 Function



(from Stallings, Crypto and Net Security)

Message Extension Attack

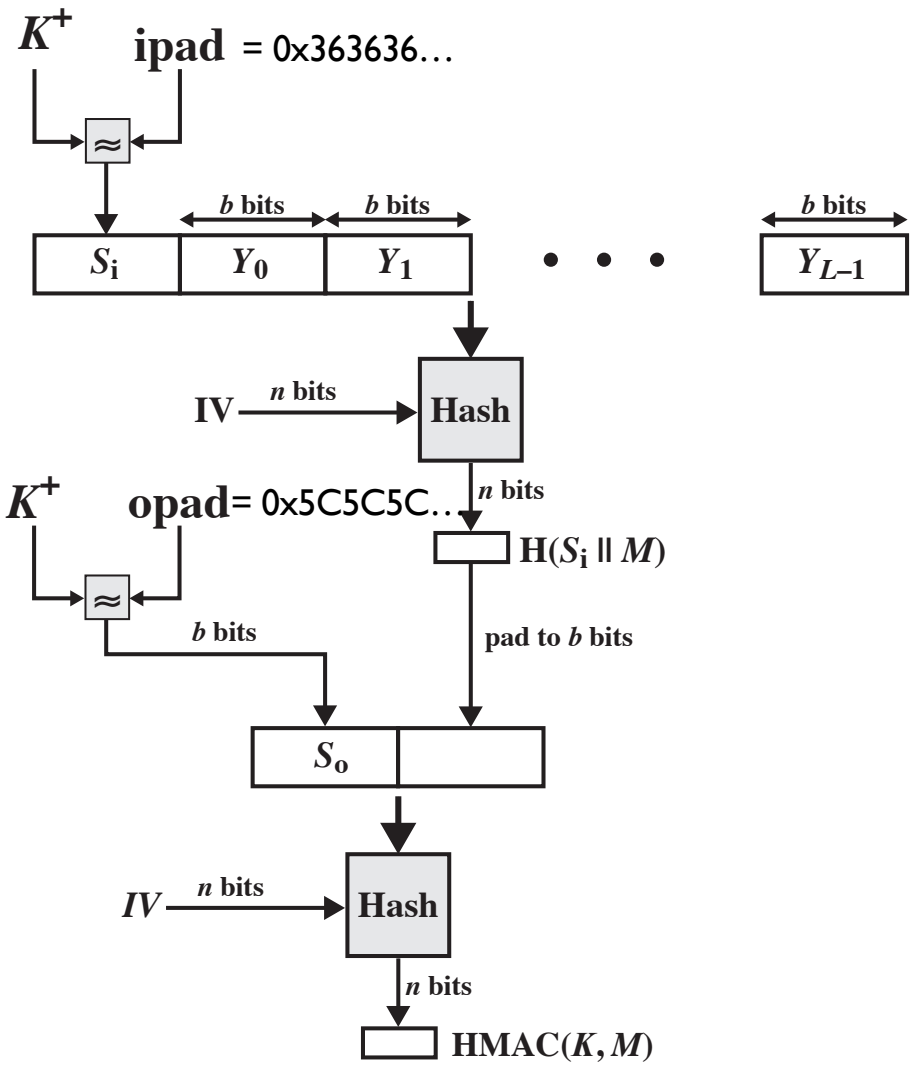
- Why is $MAC_k(M) = H(k|M)$ bad?
- How can Eve append M' to M?
  - Goal: compute H(k|M|M') without knowing k
- Solution: Use H(k|M) as IV for next f iteration in H()

A Better MAC

- Objectives
  - Use available hash functions without modification
  - Easily replace embedded hash function as more secure ones are found
  - Preserve original performance of hash function
  - Easy to use

HMAC

- HMAC(k,M) =
  H(k⊕opad || H(k⊕ipad || M))
  - Attacker cannot extend MAC as before
  - Prove it to yourself



(from Stallings, Crypto and Net Security)

Using hashes as authenticators

- Consider the following scenario
  - Prof. Smart E. Pants has not decided if she will cancel the next lecture.
  - When she does decide, she communicates to Bob the student through Mallory, her evil TA.
  - She does not care if Bob shows up to a cancelled class, but she does not want students to not show up if the class hasn't been cancelled
  - Prof. Pants does not trust Mallory to deliver the message.
- Prof. Smart E. Pants and Bob use the following protocol:
  - Prof. Pants invents a secret t
  - Prof. Pants gives Bob h(t), where h() is a crypto hash function
  - If she cancels class, she gives t to Mallory to give to Bob
  - If does not cancel class, she does nothing
  - If Bob receives the token t, he knows that Prof. Pants sent it

Hash Authenticators

- Why is this protocol secure?
  - t acts as an authenticated value (authenticator) because Mallory could not have produced t without inverting h()
  - Note: Mallory can convince Bob that class is occurring when it is not by simply not delivering t (but we assume Bob is smart enough to come to that conclusion when the room is empty)
- Note that it is important that Bob gets the original value h(t) from Alice directly (was provably authentic)

Hash chain

- Now, consider the case where Alice wants to do the same protocol, only for all 26 classes (the semester)
- Alice and Bob use the following protocol:

1. Alice invents a secret t
2. Alice gives Bob $H^{26}(t)$, where $H^{26}()$ is 26 repeated uses of H().
3. If she cancels class on day d, she gives $H^{(26-D)}(t)$ to Mallory, e.g.,
   - If cancels on day 1, she gives Mallory $H^{25}(t)$
   - If cancels on day 2, she gives Mallory $H^{24}(t)$
   - …….
   - If cancels on day 25, she gives Mallory $H^{1}(t)$
   - If cancels on day 26, she gives Mallory t
4. If Alice does not cancel class, she does nothing

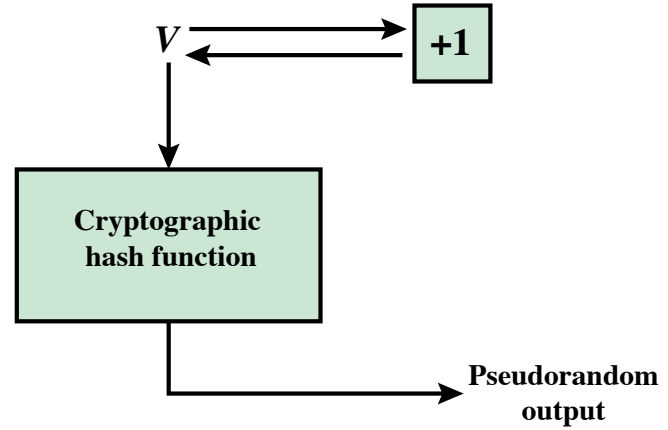
– If Bob receives the token t, he knows that Alice sent it

Hash Chain (cont.)

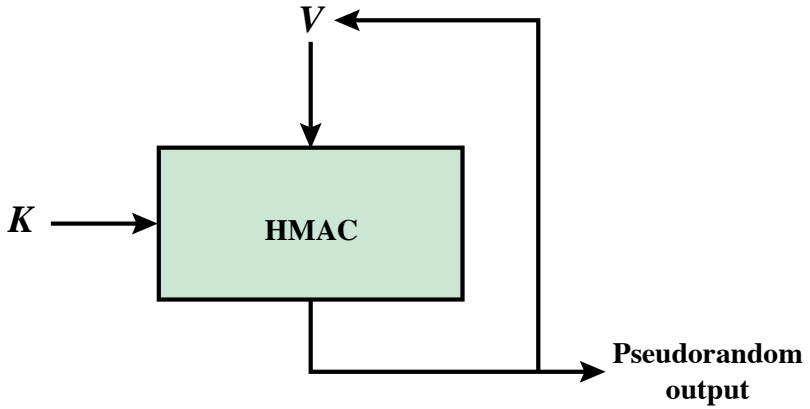
- Why is this protocol secure?
  - On day d, $H^{(26-d)}(t)$ acts as an authenticated value (authenticator) because Mallory could not create t without inverting H() because for any $H^k(t)$ she has $k>(26-d)$
  - That is, Mallory potentially has access to the hash values for all days prior to today, but that provides no information on today's value, as they are all post-images of today's value
  - Note: Mallory can again convince Bob that class is occurring by not delivering $H^{(26-d)}(t)$
  - Chain of hash values are ordered authenticators
- Important that Bob got the original value $H^{26}(t)$ from Alice directly (was provably authentic)

PRNG using Hashes

- Hash functions can also be used to build pseudorandom number generator (PRNGs) for generating a stream of pseudo random bits
- Two uses:
  - Create random values: Seed should only be known to the user
  - Stream encryption: Seed known to sender and receiver



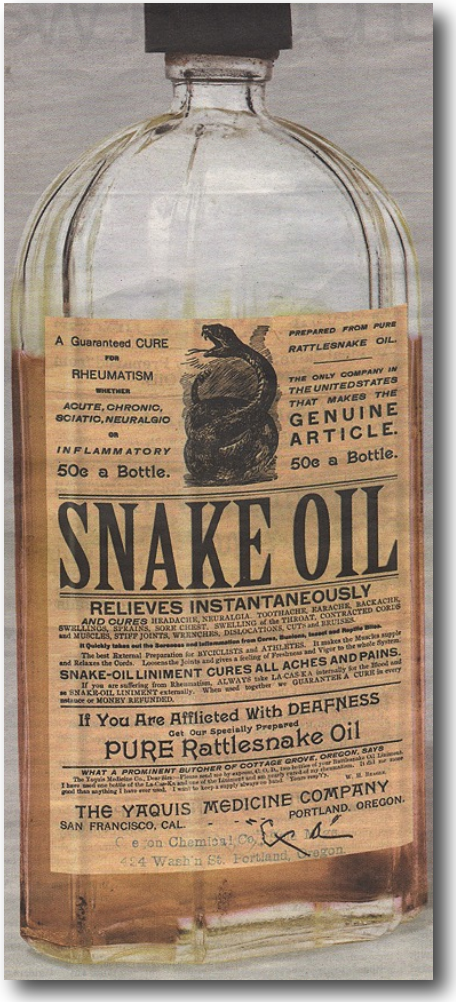
(a) PRNG using cryptographic hash function



(b) PRNG using HMAC

(from Stallings, Crypto and Net Security)

Basic truths of cryptography



- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
  - Vetted through crypto community
  - Avoid any "proprietary" encryption
  - Claims of "new technology" or "perfect security" are almost assuredly **snake oil**

Building systems with cryptography

- Use quality libraries
  - SSLeay, cryptolib, openssl
  - Find out what cryptographers think of a package before using it
- Code review like crazy
- Educate yourself on how to use library
  - Understand caveats by original designer and programmer

Cipher.getInstance("AES") defaults to ECB mode!

Common pitfalls

- Generating randomness
- Storage of secret keys
- Virtual memory (pages secrets onto disk)
- Protocol interactions
- Poor user interface
- Poor choice of parameters or modes