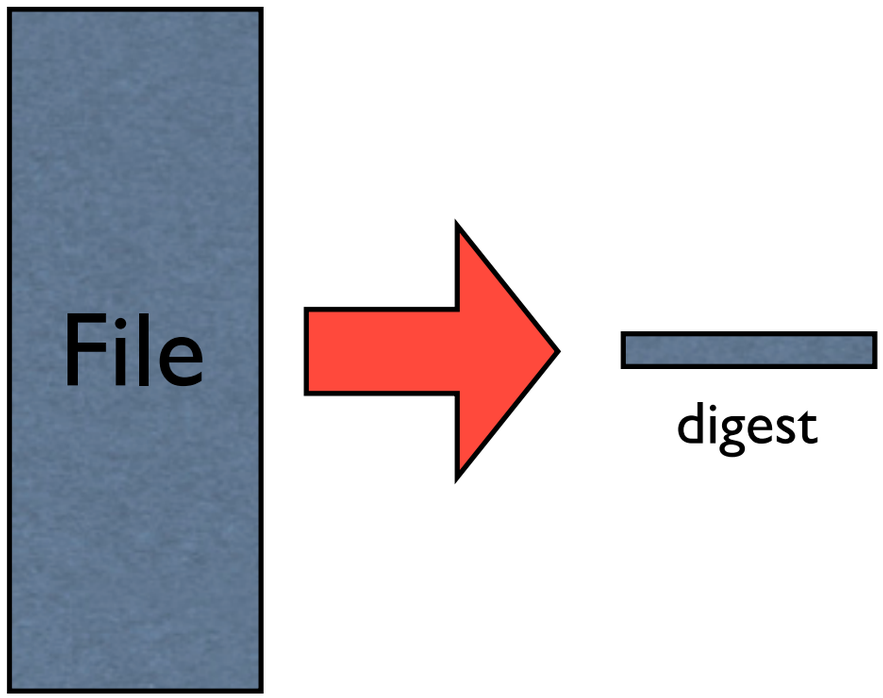
CS G256
L07: Hash Functions
May 25, 2004

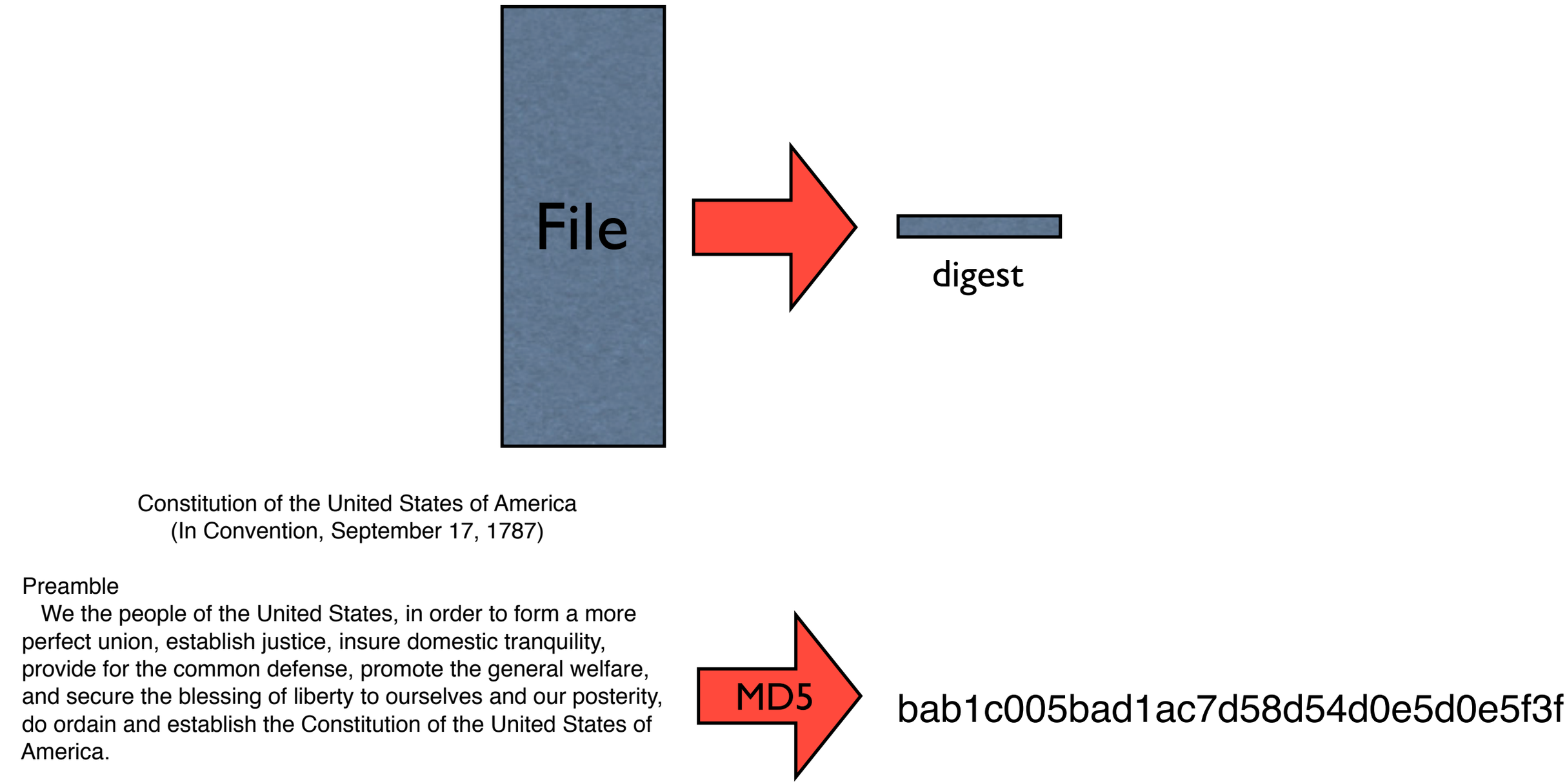
MD5
SHA-1
HMAC

Message Digests

- Message Digests make a "fingerprint" of a file.
- Input: 1-2^{64} bytes
- Output: 128 or 160* bits



Message Digests



Properties of a good Message Digest

Digest = *f* (Input)

- Digest cannot be predicted from the input
- Hard or impossible to find two inputs with the same digest.
- Changing one bit of input changes ~50% of the output bits.

Message Digest Algorithms

- Rivest Functions:
  - MD2 (RFC 1319), MD4 (RFC 1320), MD5 (RFC 1321)
- NIST Functions:
  - SHA, SHA-1, SHA-512, SHA-1024
- Other Functions:
  - Snerfu, N-Hash, RIPE-MD, HAVAL

Comparing Message Digest Functions

Function	Bits	Comments
MD2	128	Slow Secure
MD5	128	Fast Reasonably Secure
SHA-1	160	Slow Secure
SHA-256	256	?
SHA-512	512	?
SHA-1024	1024	?

Message Digest Example

message	MD5(message)
“this is a test”	ff2294133695609 8ae9a564289d1bf1b
“this is c test”	c5e530b91f5f324b 1e64d3ee7a21d573
“this is a test ”	6df4c47dba4b01cc f4b5e0d9a7b8d925

Key Points

- Any change in the input changes the digest:
  - Adding a space
  - Changing a line break
  - Capitalizing a word

“Breaking” a message digest

- Brute-force attack:
  - Search for two messages that have the same digest (they should be many of them)
  - Create a message with a desired message digest
- Algorithm attack
  - Using your knowledge of match, create two document with a given digest.

Documents with Tunable Digests

We the [redacted] of the [redacted], in order to [redacted] a more perfect union, establish justice, insure domestic tranquility, provide for the common defense, promote the general welfare, and secure the blessing of liberty to ourselves and our [redacted], do ordain and establish the Constitution of the United States of America.

4 choices = 2^4 different MD5 codes.

- Still a long way from 2^{128}
- Real problem is finding the match

Just how big is 2^{128}?

2^{128} = 340,282,366,920,938,463,463,374,
607,431,768,211,456

If you could try a billion^2 combinations a second, it would take 10,790 billion years

$(2^{128} / 10^9 / 10^9 / (60*60*24*365) / 10^9)$

Brute force attacks on smaller digests

# bits	example	Time to crack*
3	Check digit	0
8	Check character	0
16	CRC-16	0
32	CRC-32	71 minutes
40		12 days
58	DES MAC	2283 years

* Assuming 1 million tries per second
DES Cracker can do 88 billion keys per second;
with Distributed.NET it could cracking 245 billion keys per sec in 1999

MD5 vs. SHA-1

MD5	SHA-1
Designed by Ron Rivest	Designed by NIST/NSA
128 bit digest	160 bit digest
IETF standard	USG standard
3x faster than SHA-1	3x slower than MD5
May contain flaws	May contain flaws

I prefer MD5, but SHA-1 is “the standard.”

Uses of Digest Functions

- Integrity
  - verifying downloaded code
  - verifying SSL streams
- Authentication
  - verifying a shared secret w/o encryption
- Factoring
  - Use Digest to determine if two files are identical

MD5s for Downloaded Code

ProFTPD Verification

```
ftp> get proftpd-1.2.8.tar.gz
local: proftpd-1.2.8.tar.gz remote: proftpd
-1.2.8.tar.gz
227 Entering Passive Mode (81,223,20,36,149,0).
150 Opening BINARY mode data connection for proftpd
-1.2.8.tar.gz (966281 bytes)
226 Transfer complete.
966281 bytes received in 00:16 (56.25 KB/s)
ftp> quit
221 Goodbye.
[simsong@G2AIR ~] 307 % md5 proftpd-1.2.8.tar.gz
MD5 (proftpd-1.2.8.tar.gz) =
[redacted]
[simsong@G2AIR ~] 308 %
```

match!

MD5 sums and PGP signatures of release files

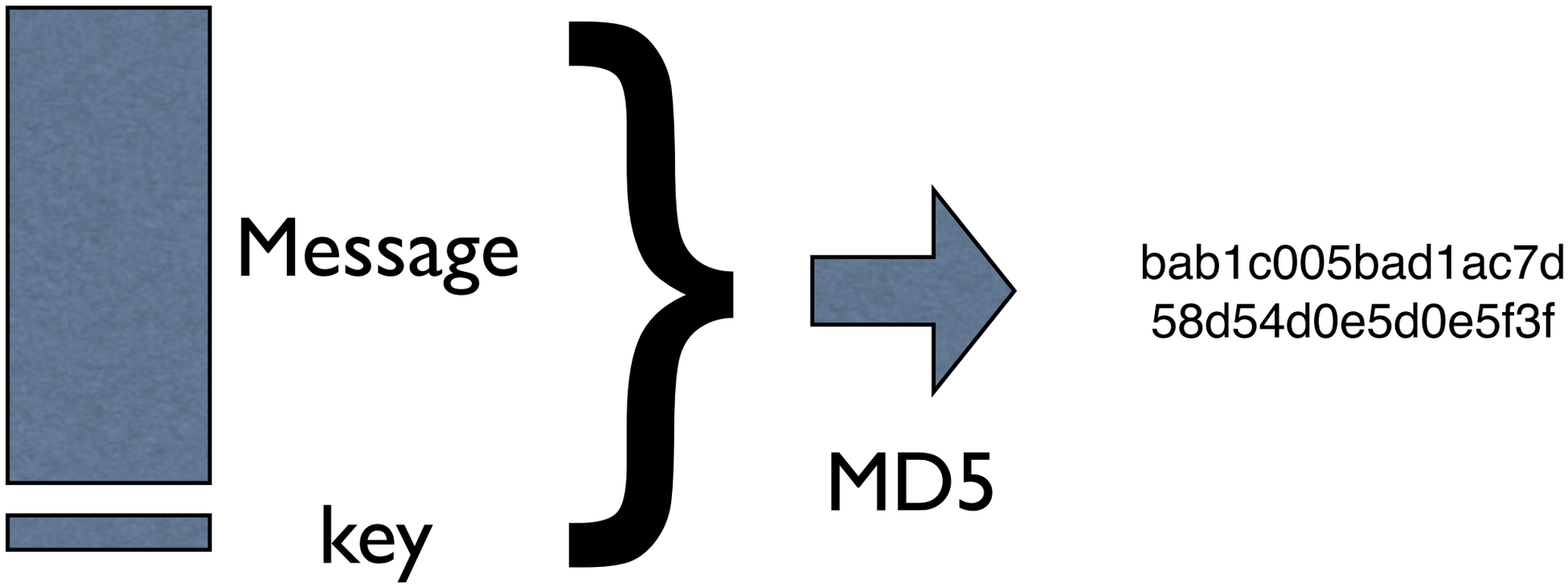
MD5 Digest Hashes

35e669cb085879eea21c6db9e7af2040 proftpd-1.2.8.tar.bz2
[redacted] proftpd-1.2.8.tar.gz

MACs and HMACs

- MAC = "Message Authentication Code"
- HMAC = "Keyed Hashing for Message Authentication" (RFC 2104)
  - http://www.ietf.org/rfc/rfc2404.txt
  - http://www.cs.ucsd.edu/users/mihir/papers/hmac.html

MACs: The Big Idea



RFC 2104: HMAC

$$HMAC(f,K,M) = f(K \oplus 0x5c^{64} \cdot f(K \oplus 0x36^{64} \cdot M))$$

More complicated than concatenating the key and taking the hash, but more secure!

Uses of HMACs

- Data integrity and authentication
  - BGP uses HMAC
    - IPsec Authentication Header and Encapsulating Security Payload use HMAC as a digital "signature."
- Password protocols

MD5 API: Perl

% **man Digest::MD5**

...

```
# Functional style
use Digest::MD5  qw(md5 md5_hex md5_base64);

$digest = md5($data);
$digest = md5_hex($data);
$digest = md5_base64($data);

# OO style
use Digest::MD5;

$ctx = Digest::MD5->new;

$ctx->add($data);
$ctx->addfile(*FILE);

$digest = $ctx->digest;
$digest = $ctx->hexdigest;
$digest = $ctx->b64digest;
```

MD5 HMAC API

NAME

Digest::HMAC_MD5 - Keyed-Hashing for Message Authentication

SYNOPSIS

```
# Functional style
use Digest::HMAC_MD5 qw(hmac_md5 hmac_md5_hex);
$digest = hmac_md5($data, $key);
print hmac_md5_hex($data, $key);

# OO style
use Digest::HMAC_MD5;
$hmac = Digest::HMAC_MD5->new($key);

$hmac->add($data);
$hmac->addfile(*FILE);

$digest = $hmac->digest;
$digest = $hmac->hexdigest;
$digest = $hmac->b64digest;
```

MD5 Representations

M-T^]M-^LM-YM-^O^@M-2^DM-iM-^@ M-^XM-lM-xB~

$digest = $ctx->digest;
$digest = $ctx->hexdigest; → 1B2M2Y8AsgTpgAmY7PhCfg
$digest = $ctx->b64digest;

8ddd8be4b179a529afa5f2ffae4b9858

Using the Perl API

```
#!/usr/bin/perl

use strict;
use Digest::MD5;

foreach $_ (@ARGV){
    open(F,$_) || die "Cannot open $_, $!";
    my $ctx = Digest::MD5->new;
    $ctx->addfile(*F);
    print $_,"\t",$ctx->hexdigest,"\n";
}
```

OpenSSL C API

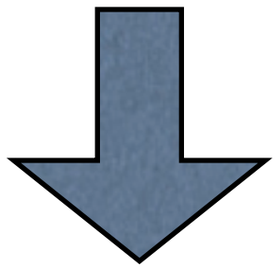
```
#include <openssl/md5.h>

unsigned char *MD5(const unsigned char *d,
                   unsigned long n,
                   unsigned char *md);

void MD5_Init(MD5_CTX *c);
void MD5_Update(MD5_CTX *c, const void *data,
                unsigned long len);
void MD5_Final(unsigned char *md, MD5_CTX *c);
```

void MD5_Init(MD5_CTX *c);

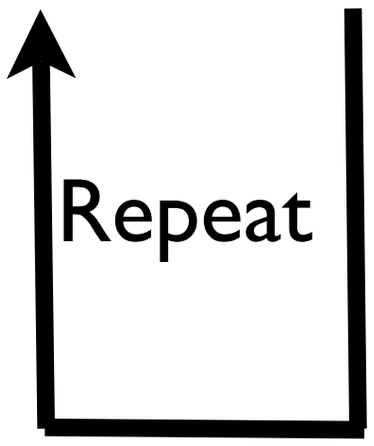
- A Context is used for a process that needs to be repeated over many blocks of data
  - Stream cipher
  - Encryption of many blocks
  - Calculation of a message digest
- Similar to instance variables in object oriented programming.

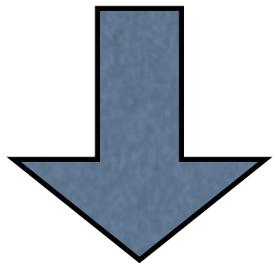
The Context holds state between each block

```
void MD5_Init(MD5_CTX *c);
```



```
void MD5_Update(MD5_CTX *c, const void *data,
                unsigned long len);
```





```
void MD5_Final(unsigned char *md, MD5_CTX *c);
```

`char buf[]` *vs.* `unsigned char buf[]`

- Technically, binary data should be `unsigned char`
- Many C routines take `char`
  - (e.g. read(), write(), etc.)
- Expect frequent casts or compiler warnings

Other uses of MACs

- Hash Trees
- S/KEY
- Password Challenge-Response