# Malicious Hashing: Eve's Variant of SHA-1

Maria Eichlseder, Florian Mendel, Martin Schläffer
Ange Albertini, Jean-Philippe Aumasson

TU Graz corkami KUDELSKI SECURITY



```
>crypto_hash *
test0.jpg 13990732b0d16c3e112f2356bd3d0dad1....
test1.jpg 13990732b0d16c3e112f2356bd3d0dad1....
```

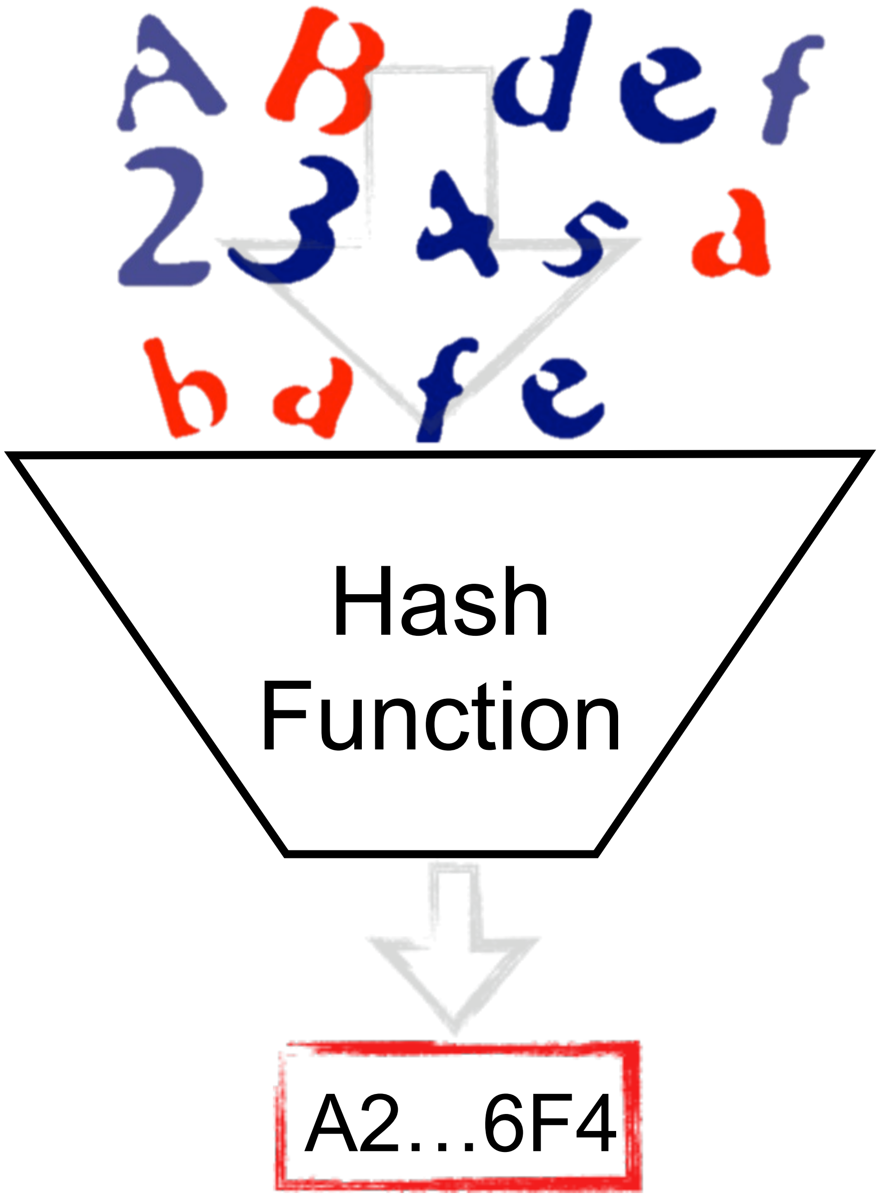
# Outline

Introduction – Hash Functions

Motivation – Backdoors

Malicious Hashing

Malicious SHA-1, Exploitation




- Fast
- Secure

# Applications

- **Short representative** of data
  - Digital signatures
  - Data authentication
- **Randomisation** (PRNG, …)
- **Commitment** schemes
- **One-way function** of string
  - Password protection
  - Micro-payments
  - One-time-signatures

# Security Properties

- **Preimage Resistance**
  Given $h(x)$, difficult to find $x$
- **Second Preimage Resistance**
  Given $x$, $h(x)$, difficult to find $x' \neq x$ such that $h(x')=h(x)$
- **Collision Resistance**
  Difficult to find $x$, $x'$ with $x \neq x'$ such that $h(x)=h(x')$

# Preimage Resistance

- In a **password file** we don't store (usr, pwd)
- For verification it is enough to **store (usr, h(pwd))**
- If an attacker gets the password file, then the **attacker has to find a preimage**

# Second Preimage Resistance

m

Channel A: high capacity but insecure

h(m)

Channel B: low capacity but secure

- An **attacker** can **tamper with Channel A**
- In order to **remain undetected**, the attacker has to **compute a second preimage**

# Collision Resistance

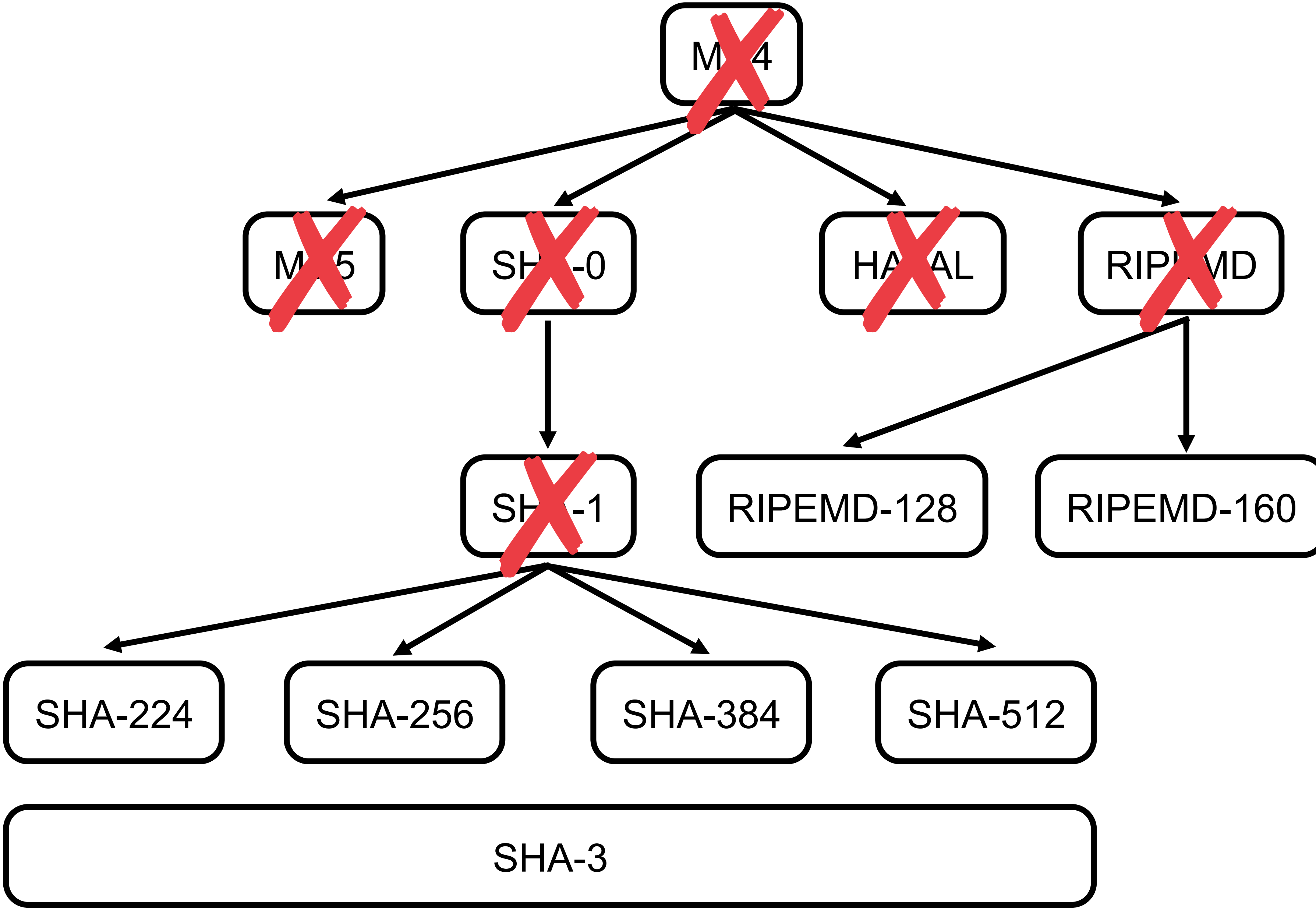
- Malicious **hacker prepares** two versions **(x and x′)** of a driver for an OS company
- The **hacker sends x** to the OS company **for inspection**
- If approved, **OS company** digitally **signs x**
- The hacker can **distribute x′** together with the valid signature
- This works since **h(x) = h(x′)**

# Generic Attacks

- Depend only on the size of the **hash value (n bits)**

- **(Second) Preimage**: guess
  - Expected number of trials: $\mathbf{2^n}$

- **Collision**: birthday attack
  - Expected number of trials: $\mathbf{2^{n/2}}$


RIPEMD-128
RIPEMD-160
SHA-224
SHA-256
SHA-384
SHA-512
SHA-3

# Outline

Introduction – Hash Functions

**Motivation – Backdoors**

Malicious Hashing

Malicious SHA-1, Exploitation

# Who’s interested in crypto backdoors?

# Who's interested in crypto backdoors?

(U) Base resources in this project are used to:

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.
- (U//FOUO) Maintain understanding of commercial business and technology trends.

# Dual EC Speculation

On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng

Dan Shumow

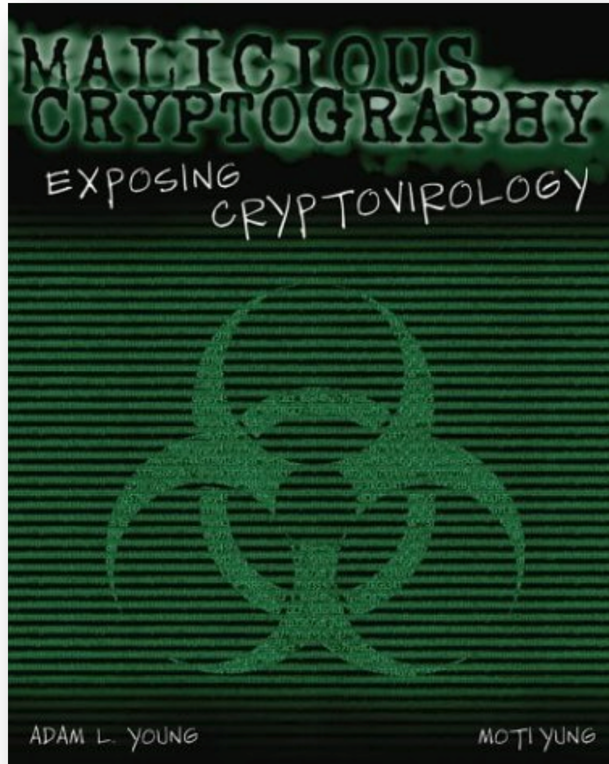
Niels Ferguson

Microsoft

CRYPTO 2007 Rump Session


Clipper (1993)

# Crypto Researchers?




# Young/Yung malicious cipher (2003)

- compresses texts to leak key bits in ciphertexts
- **blackbox** only (internals reveal the backdoor)
- other “cryptovirology” schemes

# Hardware Trojans

## Stealthy Dopant-Level Hardware Trojans

Georg T. Becker[1], Francesco Regazzoni[2], Christof Paar[1,3], and Wayne P. Burleson[1]

CHES 2013

## Trojan Side Channels

### Lightweight Hardware Trojans through Side Channel Engineering

Lang Lin[1] Markus Kasper[2] Tim Güneysu[2]
Christof Paar[1,2] Wayne Burleson[1]

CHES 2009

# Malicious Hashing

Eve's SHA3 candidate: malicious hashing

Jean-Philippe Aumasson

NAGRAVISION
KUDELSKI GROUP

ECRYPT2 Hash Workshop 2011

theoretical framework, but no good example

# What’s a crypto backdoor?

# What’s a crypto backdoor?

## It’s not an implementation backdoor

```
#define TOBYTE(x) (x) & 255
#define SWAP(x,y) do { x^=y; y^=x; x^=y; } while (0)

static unsigned char A[256];
static int i=0, j=0;

unsigned char encrypt_one_byte(unsigned char c) {
    int k;
    i = TOBYTE(i+1);
    j = TOBYTE(j + A[i]);
    SWAP(A[i], A[j]);
    k = TOBYTE(A[i] + A[j]);
    return c ^ A[k];
}
```

RC4 C implementation (Wagner/Biondi)

# What's a crypto backdoor?

a **backdoor** (covert) isn't a **trapdoor** (overt)

RSA has a trapdoor, NSA has backdoors

VSH is a trapdoor hash based on RSA

**VSH, an Efficient and Provable Collision-Resistant Hash Function**

Scott Contini[1], Arjen K. Lenstra[2], and Ron Steinfeld[1]

EUROCRYPT 2006

# Backdoor in a crypto hash?

*“some secret property that allows you to efficiently break the hash”*

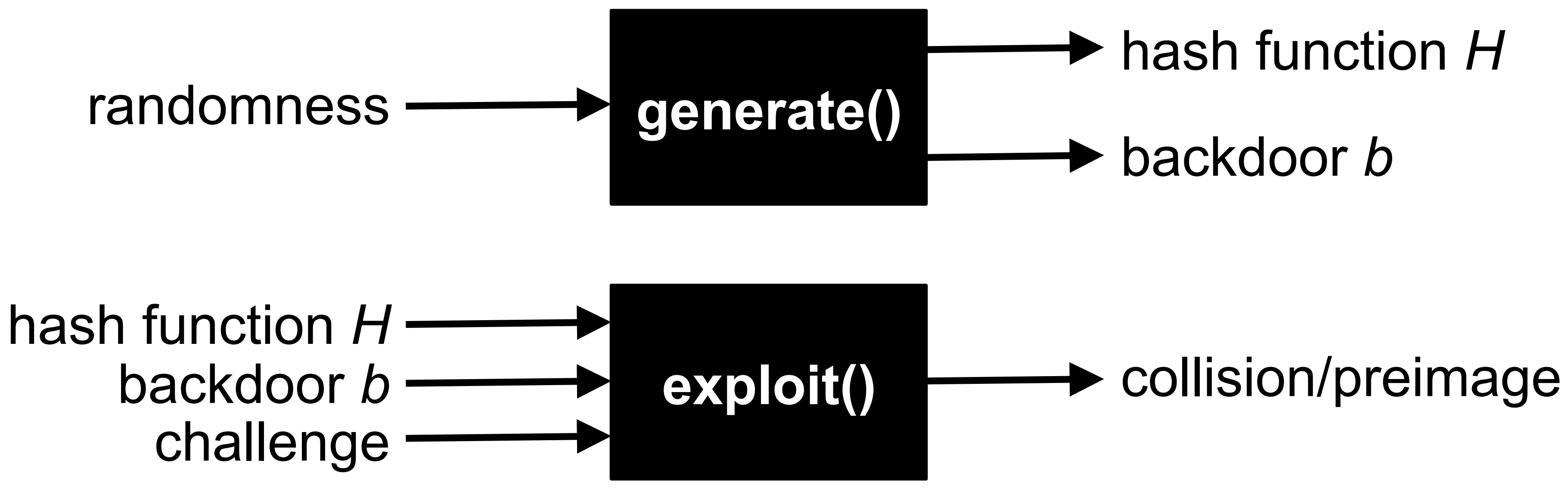

"break" can be about collisions, preimages…
how to model the stealthiness of the backdoor…
exploitation can be deterministic or randomized…

# Role reversal


Eve wants to achieve some security property
Alice and Bob (the users) are the adversaries

# Definitions

malicious hash = pair of algorithms




exploit() either "static" or "dynamic"

# Taxonomy

**static collision backdoor**

returns **constant** $m$ and $m'$ such that $H(m)=H(m')$

**dynamic collision backdoor**

returns **random** $m$ and $m'$ such that $H(m)=H(m')$

**static preimage backdoor**

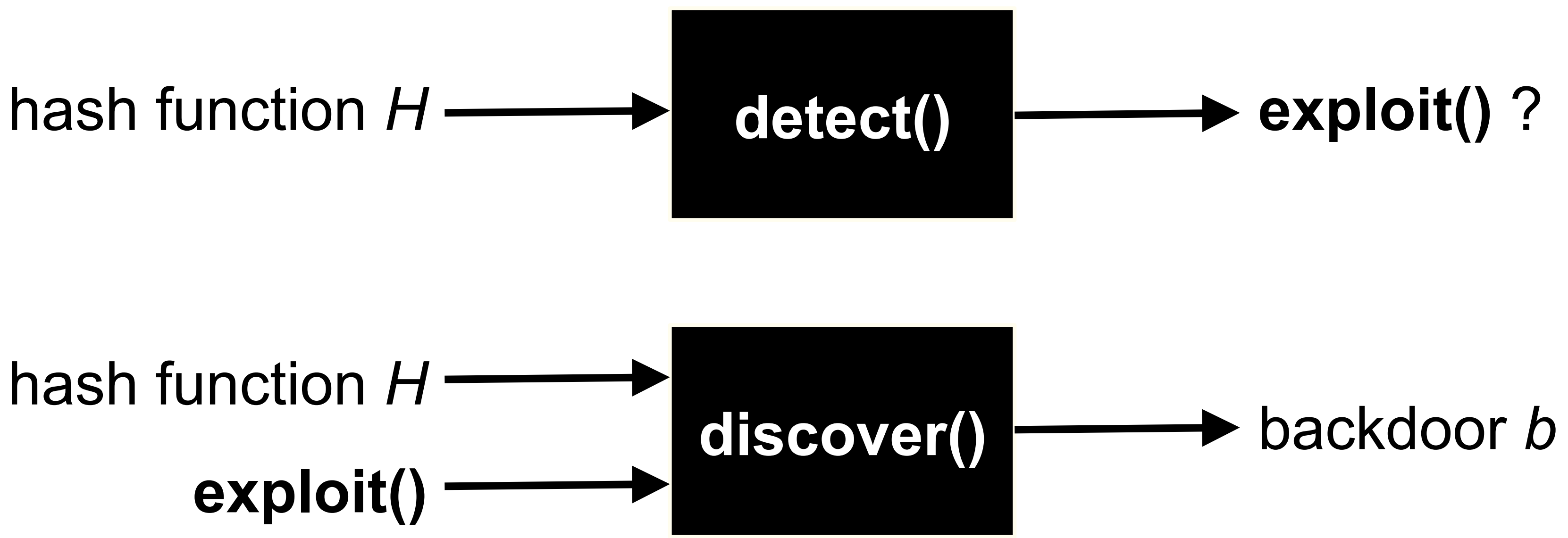
returns $m$ such that $H(m)$ has low entropy

**dynamic preimage backdoor**

given $h$, returns $m$ such that $H(m)=h$

# Stealth Definitions

undetectability vs undiscoverability




detect() may also return levels of suspicion

*H* may be obfuscated...

# Results

**dynamic collision backdoor**

valid structured files with arbitrary payloads

**detectable, but undiscoverable**

and as hard to discover as to break SHA-1

# SHA-1

NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA

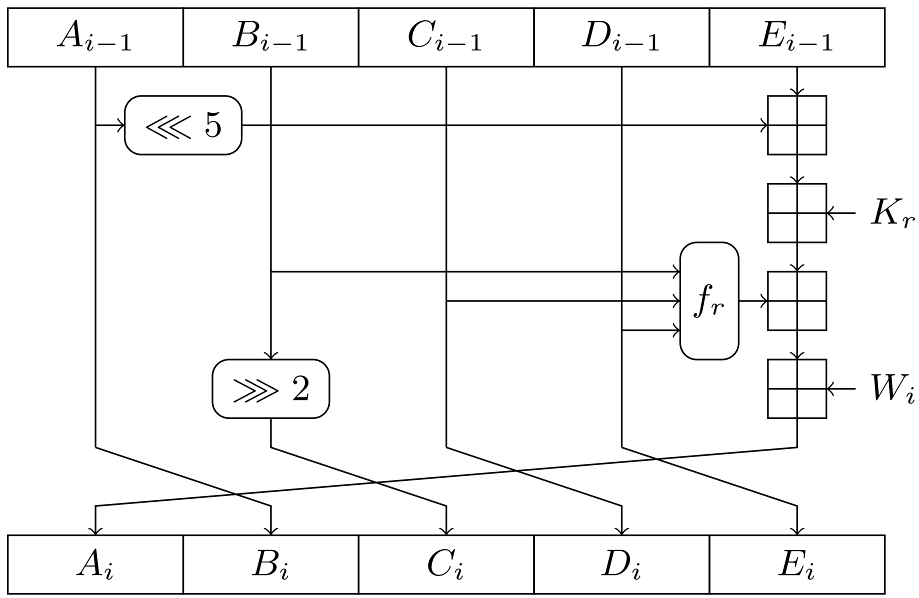
NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

# SHA-1
## everywhere

RSA-OAEP, “RSAwithSHA1”, HMAC, PBKDF2, etc.
⇒ in TLS, SSH, IPsec, etc.

**integrity check**: git, bootloaders, HIDS/FIM, etc.

# SHA-1




| round $r$ | step $i$ | $K_r$ | $f_r$ |
|---|---|---|---|
| 1 | $0 \leq i \leq 19$ | `5a827999` | $f_{\mathrm{IF}}(B, C, D) = B \wedge C \oplus \neg B \wedge D$ |
| 2 | $20 \leq i \leq 39$ | `6ed9eba1` | $f_{\mathrm{XOR}}(B, C, D) = B \oplus C \oplus D$ |
| 3 | $40 \leq i \leq 59$ | `8f1bbcdc` | $f_{\mathrm{MAJ}}(B, C, D) = B \wedge C \oplus B \wedge D \oplus C \wedge D$ |
| 4 | $60 \leq i \leq 79$ | `ca62c1d6` | $f_{\mathrm{XOR}}(B, C, D) = B \oplus C \oplus D$ |

# Collision Attack on SHA-1

**Finding Collisions in the Full SHA-1**

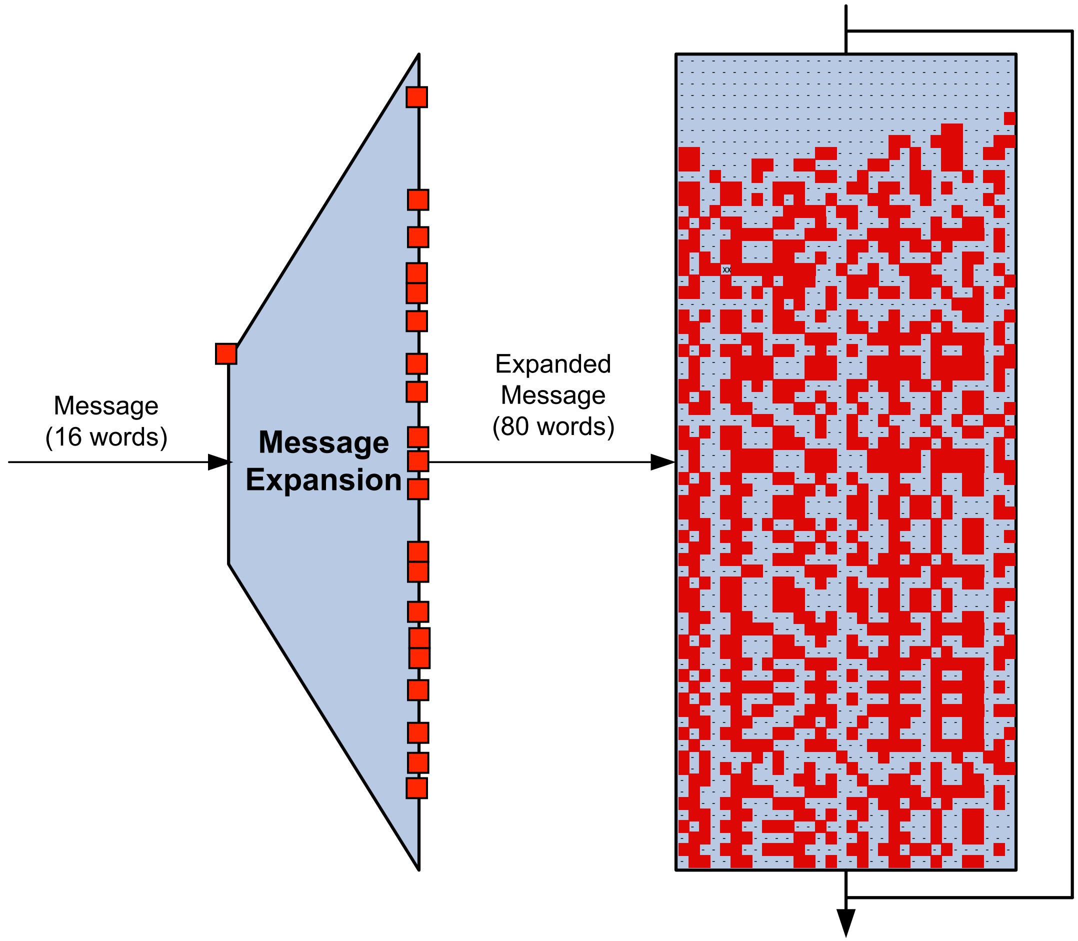
Xiaoyun Wang[1]*, Yiqun Lisa Yin[2], and Hongbo Yu[3]

[1] Shandong University, Jinan 250100, China, xywang@sdu.edu.cn
[2] Independent Security Consultant, Greenwich CT, US, yyin@princeton.edu
[3] Shandong University, Jinan250100, China, yhb@mail.sdu.edu.cn

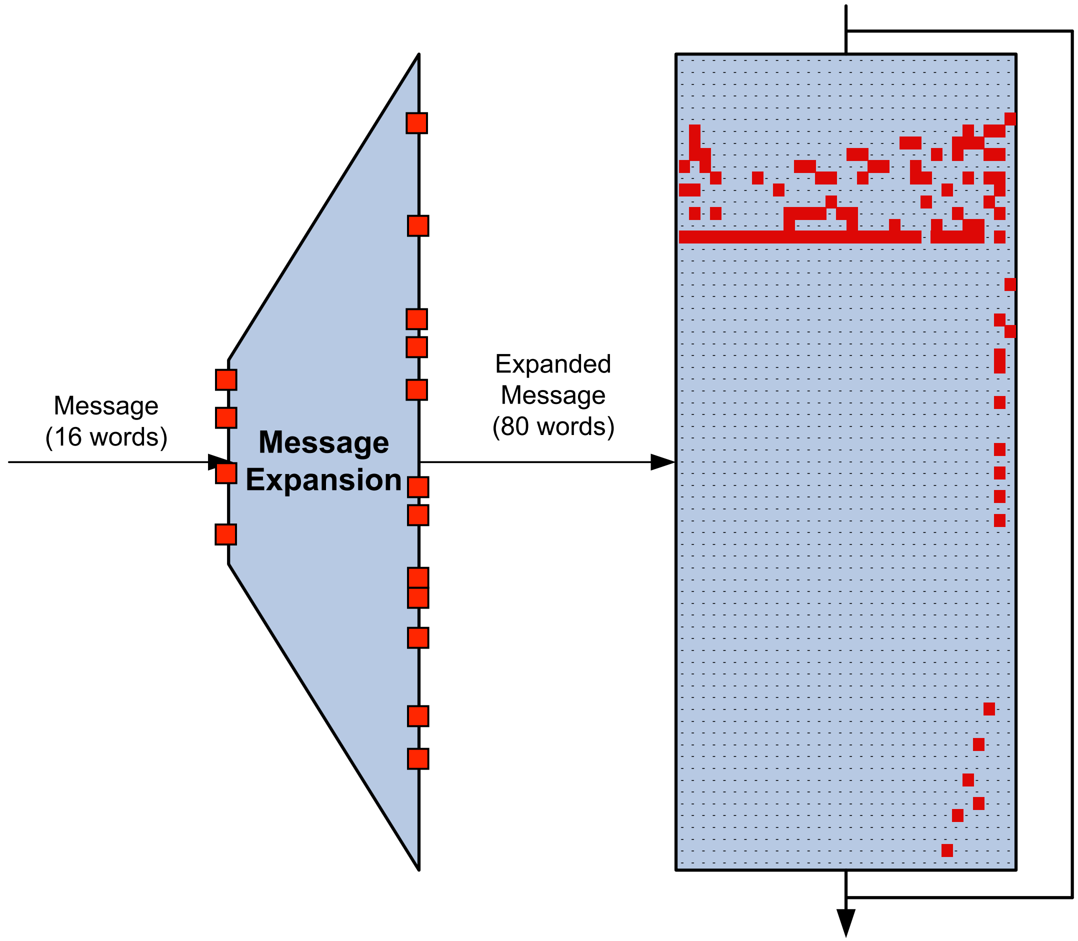
CRYPTO 2005

but no collision published yet
actual complexity unclear ($>2^{60}$)

# Differential cryptanalysis for collisions
# "perturb-and-correct"



# Differential cryptanalysis for collisions “perturb-and-correct”

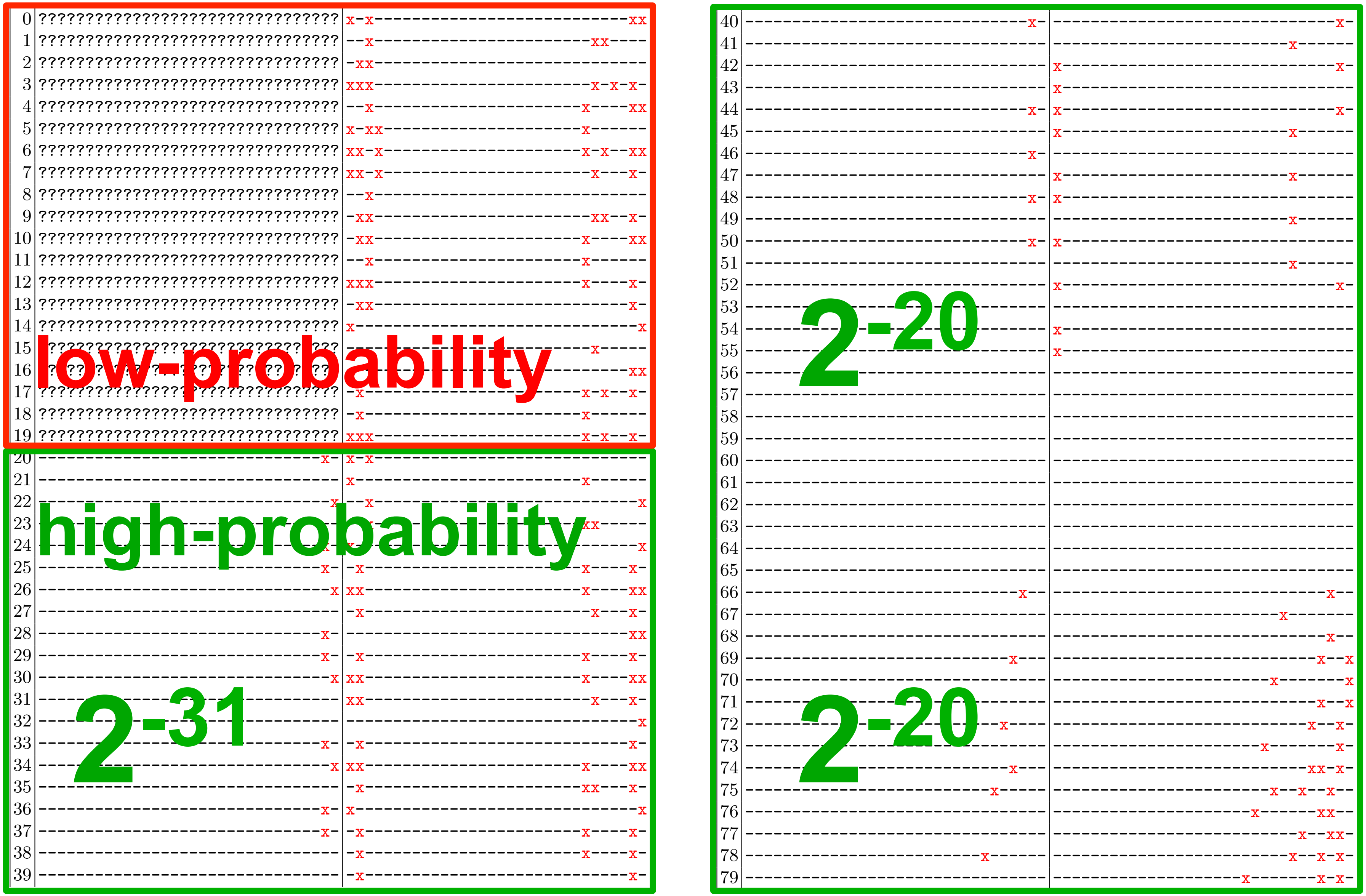


# 2 stages (offline/online)

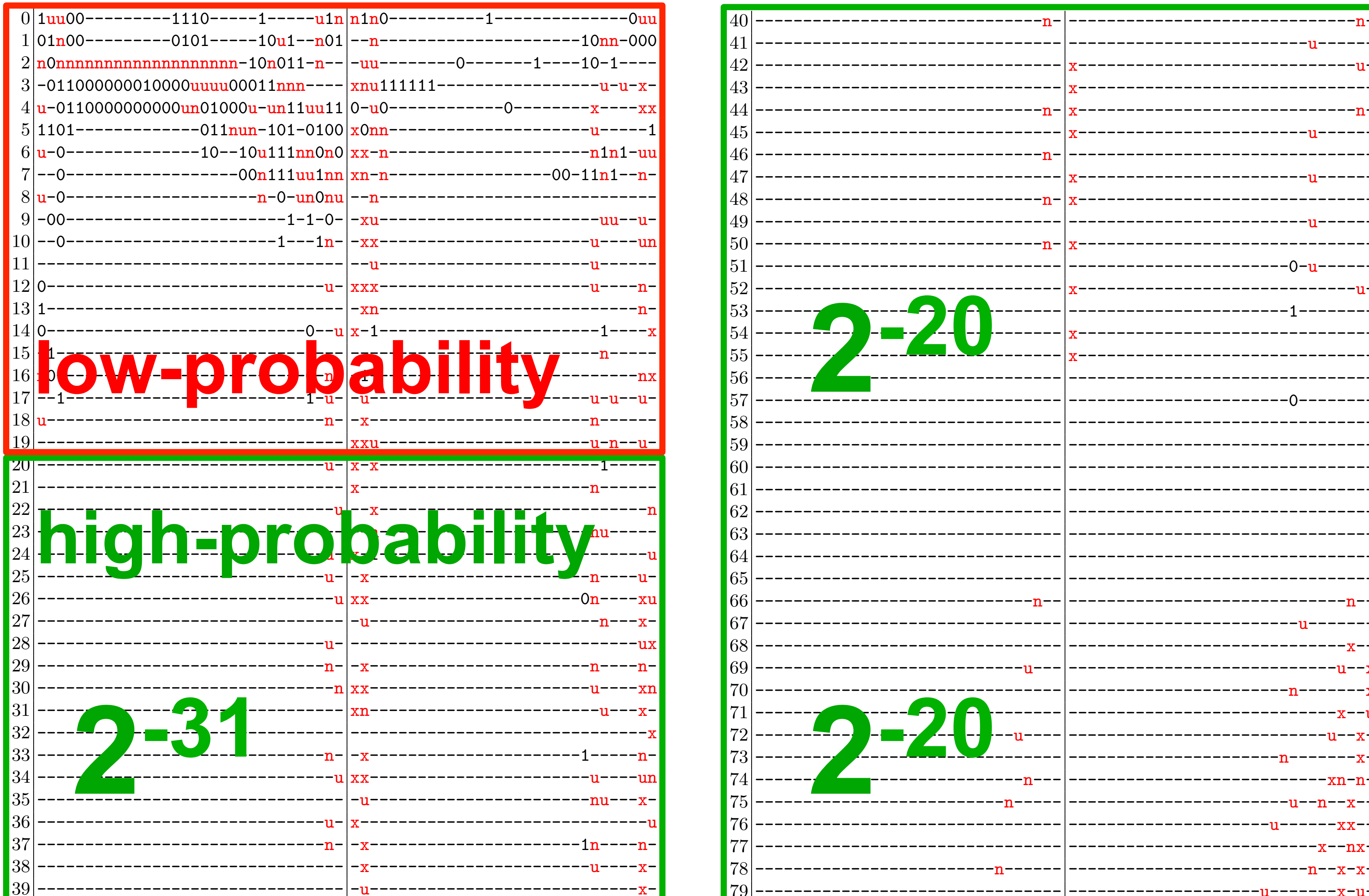
1. find a **good** differential characteristic
   = one of high probability

2. find **conforming messages**
   with message modification techniques

# Find a characteristic: linearization

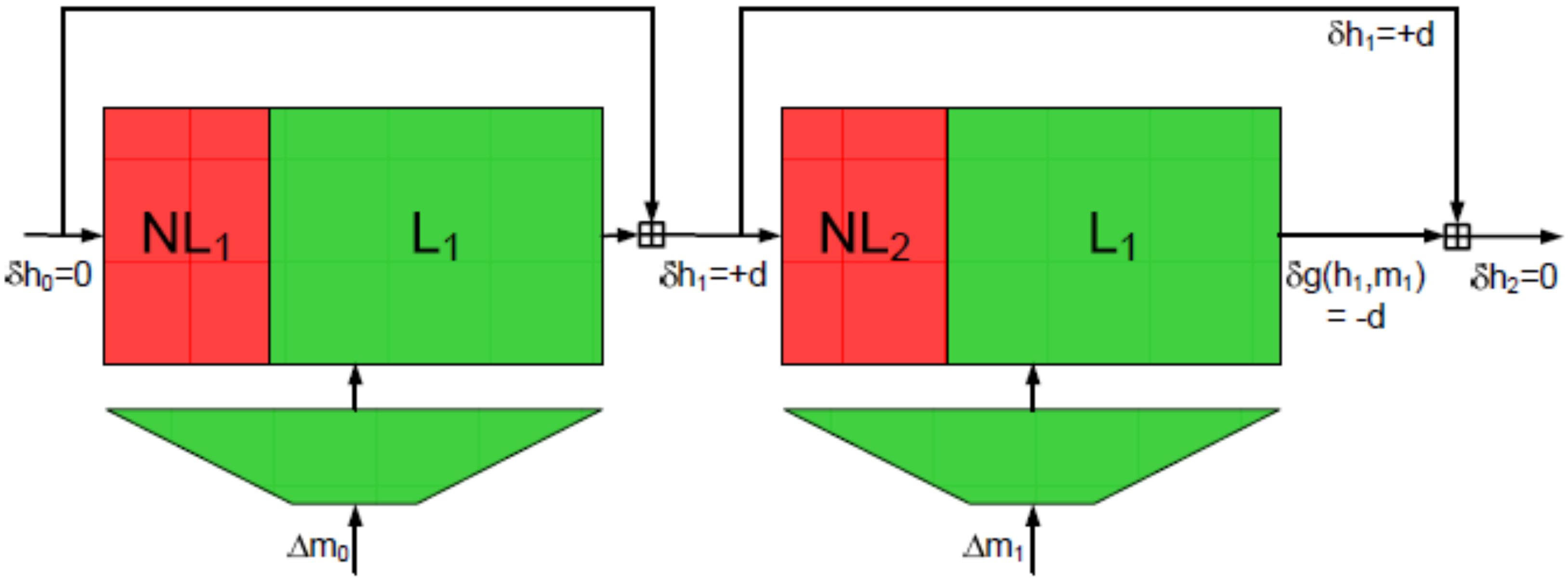


# Find a characteristic: 1st round



# 2-block collision

- near-collision → collision



# Malicious SHA-1

# Find a characteristic: 1-block

| | | |
|---|---|---|
| 0 | 1001111110001101100110001001010n | 111111111101100011111111111000nu |
| 1 | 00u1101001100--------001111u0n00 | 11u0001000000000-----0100unu0n00 |
| 2 | n111n00---1------------uu000un00 | u011n1------------------00001000 |
| 3 | 0uuuu111--0---0---uu---0-0un11nn | nnu-n--------------------nn000u1 |
| 4 | 1n01u1110---u-n-------u0011001n0 | uu1-u------------------00u0011uu |
| 5 | 0011011n1n00--0-un0101-10n1u0n00 | 10u0u--------------------1101u11 |
| 6 | n1n1n1n01000--1-100101-00n000011 | 1u111--------------------u-001u0 |
| 7 | nu1nnnnnnnnnnnnnnnnnnnnnnnn000n1 | 0n10u00------------------n000nn1 |
| 8 | 101111-100110000000010000111nu0u1 | n001u----------------------000u1 |
| 9 | 0-1010101000000000000000001un001 | 10110100-----------------01u1u00 |
| 10 | u1n00------------------------01u | 1011n---------------------0-00n1 |
| 11 | -00-0--------------------1100001 | u0u----------------------n1n0n00 |
| 12 | -0010-----------------------00-1 | u01-n----------------------100n0 |
| 13 | ----1-----------------------1100 | n0100----------------------11011 |
| 14 | -------------------------------n | u1u-u---------------------0000nn |
| 15 | -1[illegible] | n1u-u[illegible]0un10010 |
| 16 | 0[illegible] | [illegible]1000un |
| 17 | ------------------------------u- | nn01---------------------10111u1 |
| 18 | -------------------------------n | nn001--------------------n-010nu |
| 19 | -------------------------------- | un1----------------------un111n1 |

low-probability

| | | |
|---|---|---|
| 20 | -------------------------------- | n11-----------------------0011nu |
| 21 | ------------------------------n- | 0u01---------------------01110n1 |
| 22 | -------------------------------n | 0u0----------------------u1100nu |
| 23 | [illegible] | nn[illegible]un001n1 |
| 24 | [illegible] | n010[illegible]100un |
| 25 | ------------------------------n- | 1n1---------------------011001n0 |
| 26 | -n-----------------------------u | 011----------------------u-110un |
| 27 | -------------------------------- | nn00---------------------nu0u0n0 |
| 28 | ------------------------------u- | nn101---------------------11001u |
| 29 | n-----------------------------n- | uu1n---------------------n0101n0 |
| 30 | -------------------------------- | 1n1u---------------------u1u01u0 |
| 31 | -------------------------------- | uu0u---------------------11101u0 |
| 32 | -------------------------------u | 01n----------------------10110un |
| 33 | -------------------------------- | 01n--------------------10nu00001 |
| 34 | u------------------------------- | 10u----------------------10100nu |
| 35 | ------------------------------n- | nu0--------------------1001n11n1 |
| 36 | ------------------------------n- | 1n0----------------------u0111n0 |
| 37 | -------------------------------- | nun--------------------00u1100u0 |
| 38 | ------------------------------u- | n0u-----------------------110001 |
| 39 | -------------------------------- | 10n0-------------------10n010111 |

high-probability

$2^{-40}$

| | | |
|---|---|---|
| 40 | n------------------------------- | u01----------------------10000n0 |
| 41 | -------------------------------- | 101----------------------01u1001 |
| 42 | u------------------------------u | u10-0---------------------0111un |
| 43 | -------------------------------- | n0u--------------------01nun0010 |
| 44 | -------------------------------- | n1u----------------------10000nu |
| 45 | ------------------------------n- | nu0----------------------00111n1 |
| 46 | ------------------------------u- | uuu----------------------u1111u1 |
| 47 | -------------------------------- | uun--------------------10n0000u0 |
| 48 | ------------------------------u- | n00---------------------10101100 |
| 49 | -------------------------------- | 100--------------------11n010100 |
| 50 | ------------------------------u- | 010--------------------1-0100010 |
| 51 | -------------------------------- | 010--------------------01n100010 |
| 52 | -------------------------------- | u01----------------------11100n0 |
| 53 | -------------------------------- | 101--------------------010110101 |
| 54 | -------------------------------n | n11----------------------101100n |
| 55 | -------------------------------- | u00--------------------110u11000 |
| 56 | ------------------------------u- | 100----------------------00011uu |
| 57 | ------------------------------u- | 1u1--------------------11n1011u0 |
| 58 | -------------------------------u | 1u1----------------------n00101n |
| 59 | -------------------------------- | nu0--------------------10nu001n1 |

$2^{-40}$

| | | |
|---|---|---|
| 60 | -------------------------------- | 101---------------------110000nu |
| 61 | ------------------------------n- | un1--------------------0010000n1 |
| 62 | ------------------------------n- | 1n--1------------------10u0111n1 |
| 63 | -------------------------------- | uu1--------------------11u0111u0 |
| 64 | -------------------------------- | u10-1------------------0100000n0 |
| 65 | -------------------------------- | 0--------------------11110001011 |
| 66 | ------------------------------n- | 111-------------------0-000011n1 |
| 67 | -------------------------------- | u1-------------------0110u100100 |
| 68 | ------------------------------u- | --0--------------------011000100 |
| 69 | -------------------------------- | u1---------------------01n000010 |
| 70 | ------------------------------u- | n-1--------------------111011101 |
| 71 | -------------------------------- | -0-------------------1100n01100- |
| 72 | ------------------------------u- | u----------------------000001110 |
| 73 | -------------------------------- | --1------------------0011n111011 |
| 74 | ------------------------------u- | n-----------------------101111-- |
| 75 | -------------------------------- | ---------------------0000n011101 |
| 76 | -------------------------------- | u----------------------1110001u- |
| 77 | -------------------------------- | -----------------------101000--1 |
| 78 | -------------------------------- | n---------------------000011101- |
| 79 | -------------------------------- | n--------------------111000101-- |

$2^{-15}$

# Find conforming messages

**low-probability** part: “easy”, $K_1$ unchanged
use automated tool to find a conforming message

**round 2**: try all $2^{32}$ $K_2$‘s, repeat $2^8$ times (**cost $2^{40}$**)
consider constant $K_2$ as part of the message!

**round 3**: do the same to find a $K_3$ (**total cost $2^{48}$**)
repeating the $2^{40}$ search of $K_2$ $2^8$ times ....

**round 4**: find $K_4$ in negligible time

*iterate to minimize the differences in the constants...*

# Collision

| | |
|---|---|
| $K_{1...4}$ | 5a827999 4eb9d7f7 bad18e2f d79e5877 |
| IV | 67452301 efcdab89 98badcfe 10325476 c3d2e1f0 |
| $m$ | ffd8ffe1 e2001250 b6cef608 34f4fe83 ffae884f afe56e6f fc50fae6 28c40f81<br>1b1d3283 b48c11bc b1d4b511 a976cb20 a7a929f0 2327f9bb ecde01c0 7dc00852 |
| $m^*$ | ffd8ffe2 c2001224 3ecef608 dcf4fee1 37ae880c 87e56e6b bc50faa4 60c40fc7<br>931d3281 b48c11a8 b9d4b513 0976cb74 2fa929f2 a327f9bb 44de01c3 d5c00832 |
| $\Delta m$ | 00000003 20000074 88000000 e8000062 c8000043 28000004 40000042 48000046<br>88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060 |
| $h(m)$ | 1896b202 394b0aae 54526cfa e72ec5f2 42b1837e |

1-block, vs. 2-block collisions for previous attacks

# But it’s not the real SHA-1!

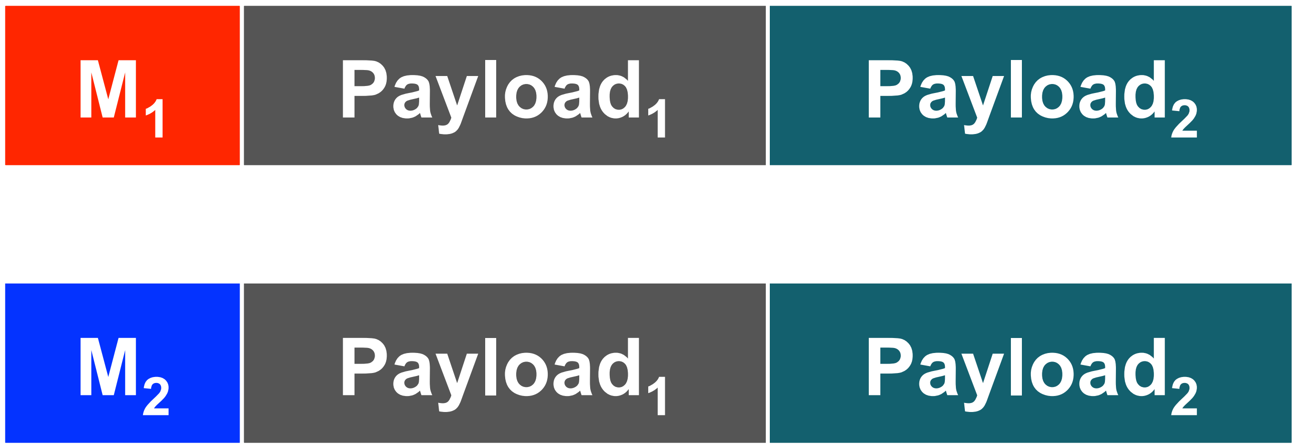
“custom” standards are common in proprietary systems
(encryption appliances, set-top boxes, etc.)

motivations:
customer-specific crypto (customers’ request)
“other reasons”

# How to turn garbage collisions into useful collisions?

(= 2 **valid files** with arbitrary content)

# Basic idea



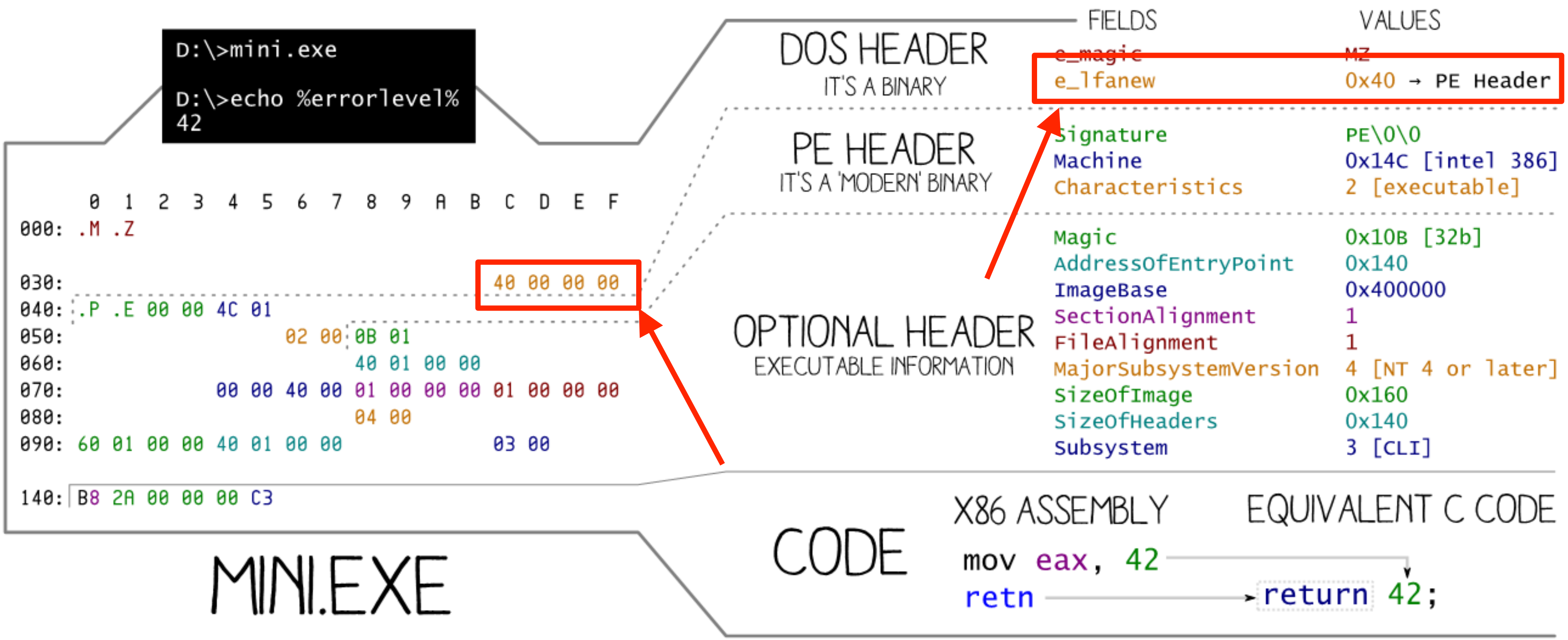

where $H(M_1)=H(M_2)$
and $M_x$ is essentially “process payload x”

# Constraints

differences (only in) the first block

difference in the first four bytes
$\Rightarrow$ 4-byte signatures corrupted

# PE? (Win* executables, etc.)




differences forces EntryPoint to be at > 0x40000000

⇒ 1GiB (not supported by Windows)

PE = fail

ELF, Mach-O = fail

(≥ 4-byte signature at offset 0)

# Shell Scripts?

# Shell Scripts

```
#<garbage, 63 bytes>               //block 1 start
                ..................................
#<garbage with differences>  //block 2 start
                __________________________________
EOL                                      //same payload
<check for block’s content>
```

```
0000000: 231d 1b91 3440 09d8 104d a6d3 54e1 102b  #...4@...M..T..+
0000010: b885 125b 4778 26bd fd37 2bee e650 082c  ...[Gx&..7+..P.,
0000020: 754b 1657 3811 bfd8 a5e0 b244 1a94 512a  uK.W8......D..Q*
0000030: cd36 a204 fee2 8a9f 3255 99aa b47a ed82  .6......2U...z..
0000040: 0a0a 6966 205b 2060 6f64 202d 7420 7831  ..if [ `od -t x1
0000050: 202d 6a33 202d 4e31 202d 416e 2022 247b   -j3 -N1 -An "${
0000060: 307d 2260 202d 6571 2022 3931 2220 5d3b  0}"` -eq "91" ];
0000070: 2074 6865 6e20 0a20 2065 6368 6f20 2220   then .  echo " 
0000080: 2020 2020 2020 2020 285f 5f29 5c6e 2020          (__)\n  
0000090: 2020 2020 2020 2028 6f6f 295c 6e20 202f         (oo)\n  /
00000a0: 2d2d 2d2d 2d2d 2d5c 5c2f 5c6e 202f 207c  -------\\/\n / |
00000b0: 2020 2020 207c 7c5c 6e2a 2020 7c7c 2d2d       ||\n*  ||--
00000c0: 2d2d 7c7c 5c6e 2020 205e 5e20 2020 205e  --||\n   ^^    ^
00000d0: 5e22 3b0a 656c 7365 0a20 2065 6368 6f20  ^";.else.  echo 
00000e0: 2248 656c 6c6f 2057 6f72 6c64 2e22 3b0a  "Hello World.";.
00000f0: 6669 0a                                  fi.

$ sh eve1.sh
         (__)
         (oo)
  /-------\/
 / |     ||
*  ||----||
   ^^    ^^
```

```
0000000: 231d 1b92 1440 09ac 984d a6d3 bce1 1049  #....@...M.....I
0000010: 7085 1218 6f78 26b9 bd37 2bac ae50 086a  p...ox&..7+..P.j
0000020: fd4b 1655 3811 bfcc ade0 b246 ba94 517e  .K.U8......F..Q~
0000030: 4536 a206 7ee2 8a9f 9a55 99a9 1c7a ede2  E6..~....U...z..
0000040: 0a0a 6966 205b 2060 6f64 202d 7420 7831  ..if [ `od -t x1
0000050: 202d 6a33 202d 4e31 202d 416e 2022 247b   -j3 -N1 -An "${
0000060: 307d 2260 202d 6571 2022 3931 2220 5d3b  0}"` -eq "91" ];
0000070: 2074 6865 6e20 0a20 2065 6368 6f20 2220   then .  echo "
0000080: 2020 2020 2020 2020 285f 5f29 5c6e 2020          (__)\n
0000090: 2020 2020 2020 2028 6f6f 295c 6e20 202f         (oo)\n  /
00000a0: 2d2d 2d2d 2d2d 2d5c 5c2f 5c6e 202f 207c  -------\\/\n / |
00000b0: 2020 2020 207c 7c5c 6e2a 2020 7c7c 2d2d       ||\n*  ||--
00000c0: 2d2d 7c7c 5c6e 2020 205e 5e20 2020 205e  --||\n   ^^    ^
00000d0: 5e22 3b0a 656c 7365 0a20 2065 6368 6f20  ^";.else.  echo
00000e0: 2248 656c 6c6f 2057 6f72 6c64 2e22 3b0a  "Hello World.";.
00000f0: 6669 0a                                  fi.

$ sh eve2.sh
Hello World.
```

# COM/MBR

## (DOS executable/Master Boot Record)

no signature!

start with x86 (16 bits) code at offset 0

like shell scripts, skip initial garbage

JMP to distinct addr rather than comments

```
For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT C "."
Drive C is mounted as local directory .\

Z:\>C:

C:\>COM-1.COM
good!

C:\>COM-2.COM
evil!

C:\>_
```

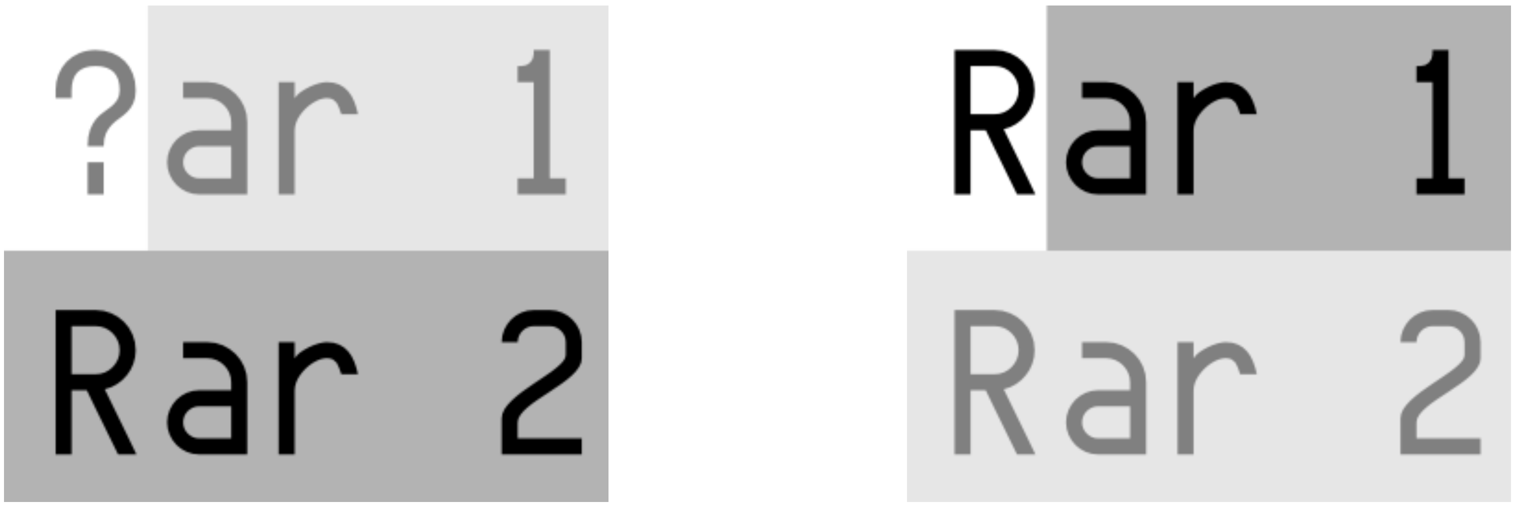
# RAR/7z

scanned forward

≥ 4-byte signature :-(

but signature can start at **any offset :-D**
⇒ payload = 2 concatenated archives

# RAR/7z




killing the 1st signature byte disables the top archive

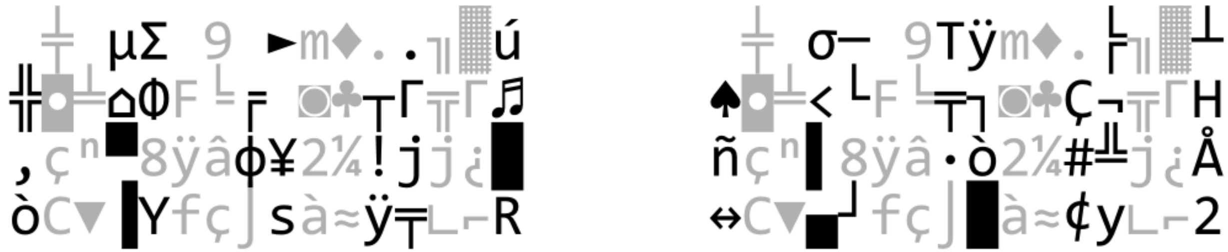
# JPEG

2-byte signature 0xFFD8

sequence of **chunks**

**Idea**

- message 1: first chunk “commented”
- message 2: first chunk processed



JPEG signature

Chunk marker
- ff e5 in block 1
- ff e6 in block 2

Chunk length
- c4 00 in block 1
- e4 00 in block 2

```
00000: ff d8 ff e? ?4 00 39 54 ?? 6d 04 2e ?? b7 b2 ??
       ?? 08 cf ?? ?? 46 d4 ?? ?? 0a 05 ?? ?? cb e2 ??   (contains no 0xff)
       ?? 87 fc ?? 38 98 83 ?? ?? 32 ac ?? ?? 6a a8 ??
       ?? 43 1f ?? ?? 66 87 f5 ?? 85 f7 ?? ?? 1c a9 ??

0c404: ff fe b5 e9       <COMment chunk covering Image 1>

0e404: ff e0             <start of Image 1>

       ...
       ff d9             <end of Image 1> <end of comment>

179ed: ff e0             <start of Image 2>

1b0d7: ff d9             <end of Image 2>
```



```
>crypto_hash *
test0.jpg 13990732b0d16c3e112f2356bd3d0dad1....
test1.jpg 13990732b0d16c3e112f2356bd3d0dad1....
```

# Polyglots

## 2 distinct files, 3 valid file formats!

# Conclusions

# Implications for SHA-1 security?

**None.**

We did not improve attacks on the unmodified SHA-1.

# Did NSA use this trick when designing SHA-1 in 1995?

**Probably not.**

1) cryptanalysis techniques are known since 2004
2) the constants look like NUMSN (√2 √3 √5 √10)
3) remember the SHA-0 fiasco :)

# Can you do the same for SHA-256?

**Not at the moment.**

**Good**: SHA-256 uses distinct constants at each step

**Not good**: The best known attack is on 31/64 steps http://malicioussha1.github.io/