Practical Free-Start Collision Attacks on 76-step SHA-1

Pierre Karpman[⚔][🖋]

[⚔]Inria and École polytechnique, France
[🖋]Nanyang Technological University, Singapore

Joint work with Thomas Peyrin and Marc Stevens

CWI, Amsterdam
2015–07–22

Title deconstruction

$\underbrace{\text{Practical}}_{\text{We can compute it}}$ $\underbrace{\text{Free-Start}}_{\text{Not unlike a false start}}$ $\underbrace{\text{Collision}}_{\text{As in } \mathfrak{f}(x)=\mathfrak{f}(x')}$

$\underbrace{\text{Attacks}}_{\text{We're the baddies}}$ $\underbrace{\text{on 76-step}}_{\text{Not quite the real thing}}$ $\underbrace{\text{SHA-1}}_{\text{Not a cat}}$

Hash functions

Hash function

A (binary) hash function is a mapping $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$

- Many uses in crypto: hash n' sign, MAC constructions...
- It is a keyless primitive
- Sooo, what's a good hash function?

Three security notions (informal)

First preimage resistance

Given t, find m such that $\mathcal{H}(m) = t$
Best generic attack is in $\mathcal{O}(2^n)$

Second preimage resistance

Given m, find $m' \neq m$ such that $\mathcal{H}(m) = \mathcal{H}(m')$
Best generic attack is in $\mathcal{O}(2^n)$

Collision resistance

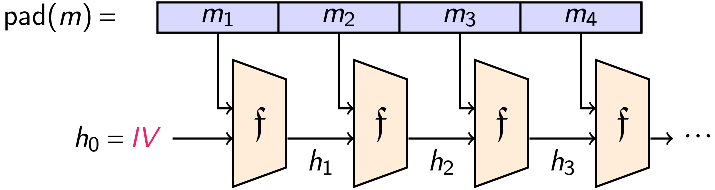
Find $m, m' \neq m$ such that $\mathcal{H}(m) = \mathcal{H}(m')$
Best generic attack is in $\mathcal{O}(2^{\frac{n}{2}})$

Merkle-Damgård construction

A domain of $\{0,1\}^*$ is annoying, so...

1. Start from a compression function $\mathfrak{f} : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$
2. Use a domain extender $\approx$
   $\mathcal{H}(m_1||m_2||\ldots||m_\ell) = \mathfrak{f}(\mathfrak{f}(\ldots\mathfrak{f}(IV, m_1)\ldots), m_\ell)$
3. Reduce the security of $\mathcal{H}$ to the one of $\mathfrak{f}$
   - $\mathsf{A}(\mathcal{H}) \Rightarrow \mathsf{A}(\mathfrak{f})$
   - $\neg\mathsf{A}(\mathfrak{f}) \Rightarrow \neg\mathsf{A}(\mathcal{H})$
   - $(\mathsf{A}(\mathfrak{f}) \Rightarrow\ ???)$
     - Invalidates the security reduction, tho

MD in a picture



Additional security notions for MD

Semi-free-start collisions

The attacker may choose IV, but it must be the same for m and m'

Free-start preimages & collisions

No restrictions on IV whatsoever

Free-start preimages & collisions (variant)

Attack $\mathfrak{f}$ instead of $\mathcal{H}$

What did we do?

- This work: collisions on 76/80 steps of the compression function of SHA-1 (95% of SHA-1)
- And it's practical
- One inexpensive GPU is enough for fast results

The SHA-1 hash function

- Designed by the NSA in 1995 as a quick fix to SHA-0
- Part of the MD4 family
- Hash size is 160 bits ⇒ collision security should be 80 bits
- Message blocks are 512-bit long

SHA-1 round function

It's a 5-branch ARX Feistel

$$A_{i+1} = A_i^{\circlearrowleft 5} + \phi_{i \div 20}(A_{i-1}, A_{i-2}^{\circlearrowright 2}, A_{i-3}^{\circlearrowright 2}) + A_{i-4}^{\circlearrowright 2} + W_i + K_{i \div 20}$$

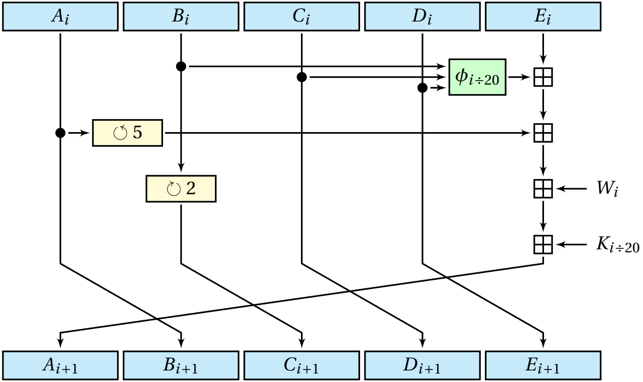
with a linear message expansion:

$$W_{0\ldots15} = M_{0\ldots15},\; W_{i\geq16} = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})^{\circlearrowleft 1}$$

$\leftsquigarrow$ The only difference between SHA-0 and SHA-1

80 steps in total

Round function in a picture



Wang collisions

SHA-1 is not collision-resistant (Wang, Yin, Yu, 2005)

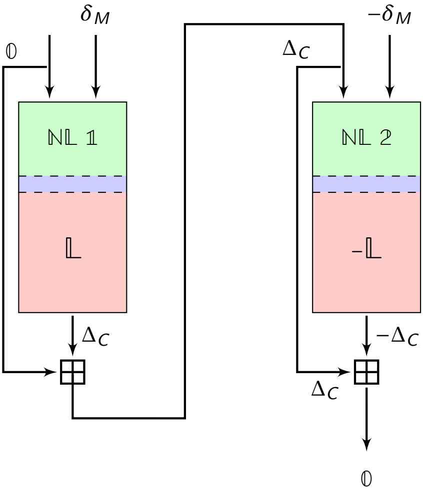
Differential collision attack

- Find a message difference that entails a good *linear* diff. path
- Construct a *non-linear* diff. path to bridge the *IV* with the linear path
- Use *message modification* to speed-up the attack
- Requires a pair of two-block messages

Attack complexity $\equiv 2^{69}$

Eventually improved to $\equiv 2^{61}$ (Stevens, 2013)

Two-block attack in a picture



Preimage detour

SHA-1 is much more resistant to preimage attacks

- No attack on the full function
- Practical attacks up to $\lessapprox$ 30 steps ($\lessapprox$ 37.5% of SHA-1) (De Cannière & Rechberger, 2008)
- Theoretical attacks up to 62 steps (77.5% of SHA-1) (Espitau, Fouque, Karpman, 2015)

Let's break stuff!

Why doing free-start again?

- Main reason is starting from a “middle” state + shift the message
- ⇒ Can use freedom in the message up to a later step
- ⇒ But no control on the *IV* value
- ⇒ Must ensure proper backward propagation

But then we need to...

1. Find a good linear part
2. Construct a good shifted non-linear part
3. Find accelerating techniques

Let's do this for 76 steps!

- Best practical result is on 75 (we wanna beat 'em)
- (First step # with visible result for full SHA-1)

Linear part selection

Criteria:

- High overall probability
- No (or few) differences in last five steps (= differences in IV)
- Few differences in early message words

$\Rightarrow$ Not many candidates

We picked $\mathrm{II}(55,0)$ (Manuel notation, 2011)

Linear path in a picture (part 1/2)

```
                   A                                      W

37: x-------------------------------   ---------------------------x----
38: --------------------------------   --------------------------------
39: x-------------------------------   --x------------------------x----
40: --------------------------------   --x-----------------------------
41: x-------------------------------   ---------------------------x----
42: --------------------------------   -xx-----------------------------
43: -x------------------------------   ---------------------------xx---
44: --------------------------------   xxx-----------------------------
45: --------------------------------   x--x----------------------------
46: x-------------------------------   --xx-----------------------x----
47: --------------------------------   x-xx----------------------------
48: --------------------------------   --x-----------------------------
49: --------------------------------   --x-----------------------------
50: --------------------------------   x-x-----------------------------
51: x-------------------------------   ---------------------------x----
52: --------------------------------   x-------------------------------
53: --------------------------------   --x-----------------------------
54: --------------------------------   --x-----------------------------
55: --------------------------------   --x-----------------------------
56: --------------------------------   x-------------------------------
```

Linear path in a picture (part 2/2)

```
                   A                                  W

57: x-------------------------------   ---------------------------x----
58: --------------------------------   --------------------------------
59: x-------------------------------   --x------------------------x----
60: --------------------------------   x-x-----------------------------
61: --------------------------------   --------------------------------
62: --------------------------------   --x-----------------------------
63: --------------------------------   --x-----------------------------
64: --------------------------------   --------------------------------
65: --------------------------------   --------------------------------
66: --------------------------------   --------------------------------
67: --------------------------------   --------------------------------
68: --------------------------------   --------------------------------
69: --------------------------------   --------------------------------
70: --------------------------------   -------------------------------x
71: -------------------------------x   --------------------------x-----
72: --------------------------------   -------------------------------x
73: --------------------------------   -x----------------------------x-
74: ------------------------------x-   -x-----------------------x-----x
75: -------------------------------x   -x------------------------x---x-
76: --------------------------------
```

Non-linear part construction

- Start with prefix of high backward probability for the first 5 steps
- Use improved JLCA for the rest
- ⇒ Good overall path with “few” conditions (236 up to #36)

Non-linear path in a picture

```
                    A                                  W
-4: --------------------------------
-3: --------------------------------
-2: ------------------------------n-
-1: 0------------------------1-----n
00: 10-0---------------------0-0----   ---------------------------n----
01: -0-1------------------0----n----   -----u---------------------unn--
02: ---1-u-----------1----n----1-u-1   xn--un---------------------n-u--
03: -----u-1----1----n----1-u-0--u-0   ----nu------------------------n-
04: -u-1---11-01n---100u11--u-0--0n1   xu-------------------------u----
05: 1n-0--100-111-u0101u00-1--n-n1-0   x-nn-u---------------------unu--
06: 00-u-u1u1uunnnn001n0u1uu-11-1-0u   --nunn-----------------------n--
07: 1n-u-nu0un10nu00nnun111-0n--0-11   x-nuuu---------------------nu-u-
08: nu-1--nuuuuuuuuuuuuu1unn11-u--n0   --u---------------0--------u----
09: uun0-1-010-1000-10nu-0-100u1-10-   --n--n--1---00--0-1--------unn--
10: u10-1---1000111011001--111--10--   xun-nu---------------------u-n--
11: 1--u1---------------10------1---   ----un------------------------u-
12: n-n--1--------------------------   xn-------------------------n----
13: 0-n-10--------------------u----0   x-uu-u---------------------nuu--
14: --n-------------------------0--1   --u-un-----------------------u--
15: 1u--------------------------1---   x-unnn---------------------nu---
16: n--10---------------------------
```

Accelerating techniques

- Message modification: correct bad instances
- Neutral bits: generate more good instances when one's found
- We choose NBs because:
  - Easy to find
  - Easy to implement
  - Good parallelization potential (more of that later)

Neutral bits (with an offset)

- We start with an offset (remember?)
- ⇒ Use neutral bits with an offset too
- In our attack, offset = 6
  - free message words = W6...21 instead of W0...15
- ⇒ Must also consider backward propagation

Our neutral bits

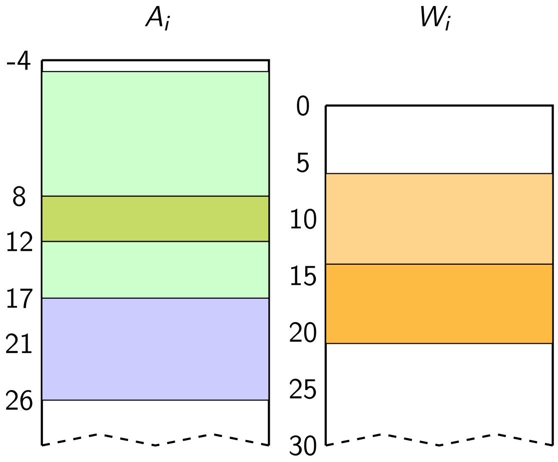
```
A18:
W14  ..................xxxxx.........
W15  ...............x.x..............
A19:
W14  .......................xxxx.....
W15  ..................xxxxxx........
W16  ...............x..x.............
W17  ............x...................
A20:
W15 ........................xxx.....
W16 ....................xxx.........
A21:
W16 .......................xxx......
W17 ..................xxxx..........
W18 ...............xx...............
A23:
W19 .................x.xxxx.........
A24:
W19 ........................xx......
W20 ...................xx...........
W21 ..............xx................
A25:
W19 .......................x........
W20 ..................x......x......
W21 ....................x...........
A26:
W20 ...............................x
```

Let's sum up

- Initialize the state with an offset
- Initialize message words with an offset
- Use neutral bits with an offset
- ⇒ many neutral bits up to late steps (yay)
- ⇒ don't know the IV in advance (duh)

- Linear path ⇒ differences in the IV
- Everything done in one block
- ⇒ Attack on the compression function

Same thing in a picture



Introduction

SHA-1 quickie

History of SHA-1 attacks

Our attack

Implementation

Results

If it's practical you must run it

- Attack expected to be practical, but still expensive
- Why not using GPUs?
- One main challenge: how to deal with the branching?

Target platform

- Nvidia GTX-970
- Recent, high-end, good price/performance
- $13 \times 128 = 1664$ cores @ $\propto 1$ GHz
- High-level programming with CUDA
- Throughput for 32-bit arithmetic: all 1/cycle/core except $\circlearrowleft$
- $\approx$ S\$ 500

Architecture imperatives

- Execution is bundled in warps of 32 threads
- Control-flow divergence is serialized ⇒ minimize branching
- Hide latency by grouping threads into larger blocks
- But careful about register / memory usage

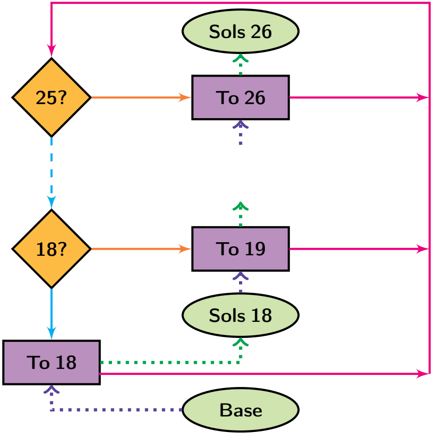
Our snippet-based approach

1. Store partial solutions up to some step in shared buffers
2. Every thread of a block loads one solution
3. ... tries all neutral bits for this step
4. ... stores successful candidates in next step buffer

Our snippet-based approach (cont.)

- Base solutions up to #17 generated on CPU
- Use neutral bits up to #26 on GPU
- Further checks up to #56 on GPU
- Final collision check on CPU

Snippets in a picture



GPU results

- Hardware: one GTX-970 (S$500)
- One partial solution up to #56 per minute on average
- $\Rightarrow$ Expected time to find a collision $\lessapprox$ 5 days
- Complexity $\equiv 2^{50.25}$ SHA-1 compression function

GPU v. CPU

- On one CPU core @ 3.2 GHz, the attack takes $\approx$ 606 days
- $\Rightarrow$ One GPU $\equiv$ 140 cores
- (To compare with $\equiv$ 40 (Grechnikov & Adinetz, 2011))
- For raw SHA-1 computations, ratio is 320
- $\Rightarrow$ Lose only $\times 2.3$ from the branching (not bad)

For more details...

Pierre Karpman, Thomas Peyrin, and Marc Stevens:
Practical Free-Start Collision Attacks on 76-step SHA-1,
CRYPTO 2015

Eprint 2015/530