Speeding up detection of SHA-1 collision attacks using unavoidable conditions

Marc Stevens - CWI

Dan Shumow – Microsoft Research

Usenix Security ‘17

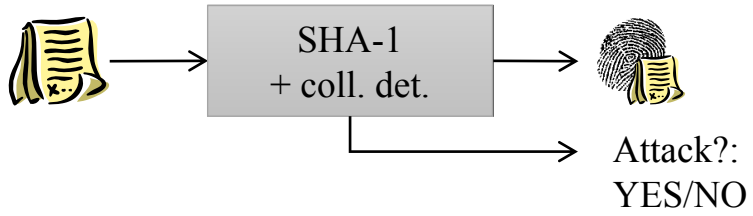
Cryptanalytic attack mitigation

- Finding collisions for SHA-1 is now practical [SBKAM2017]
- One SHA-1 collision can be reused against many targets

- Best practice: migrate away from broken crypto [duh]
- Migration is difficult in practice.
  - Interoperability
  - Backwards compatibility
  - How to handle old signatures
- SHA-1 collisions are a potential threat for many applications
  - Signatures, GPG, Deduplication (SVN), Git, Content-addressed-storage, …

MD5 & SHA-1 collision detection

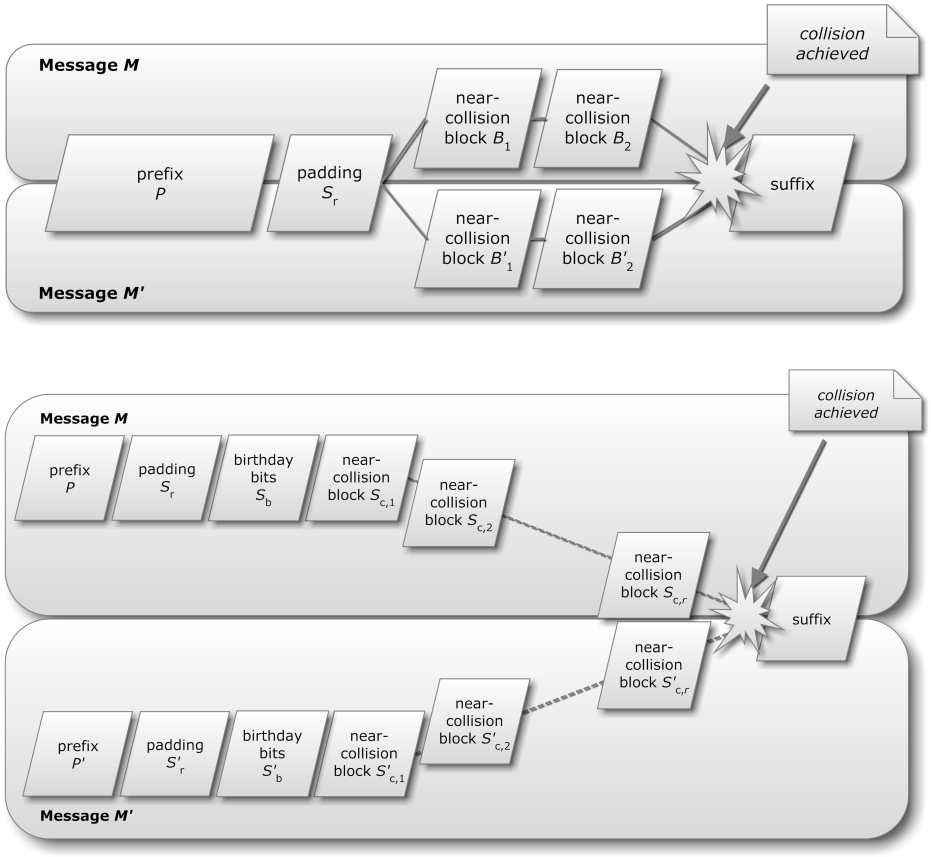
- Temporary solution: counter-cryptanalysis [S13].
- Real-time detection of cryptanalytic collision attacks.
- Repair security with drop-in strengthened MD5 & SHA-1.



- Detects unavoidable anomalies present due to crucial attack properties
  - Internal zero difference at some point.
  - Few feasible message differences.
- Exposed unknown MD5 collision attack in supermalware Flame [S13,FS15].

MD5 & SHA-1 collision attacks

- Collision attacks use *compression function attacks.*
- Identical prefix collision.
- Chosen prefix collision.



MD5 & SHA-1 collision detection

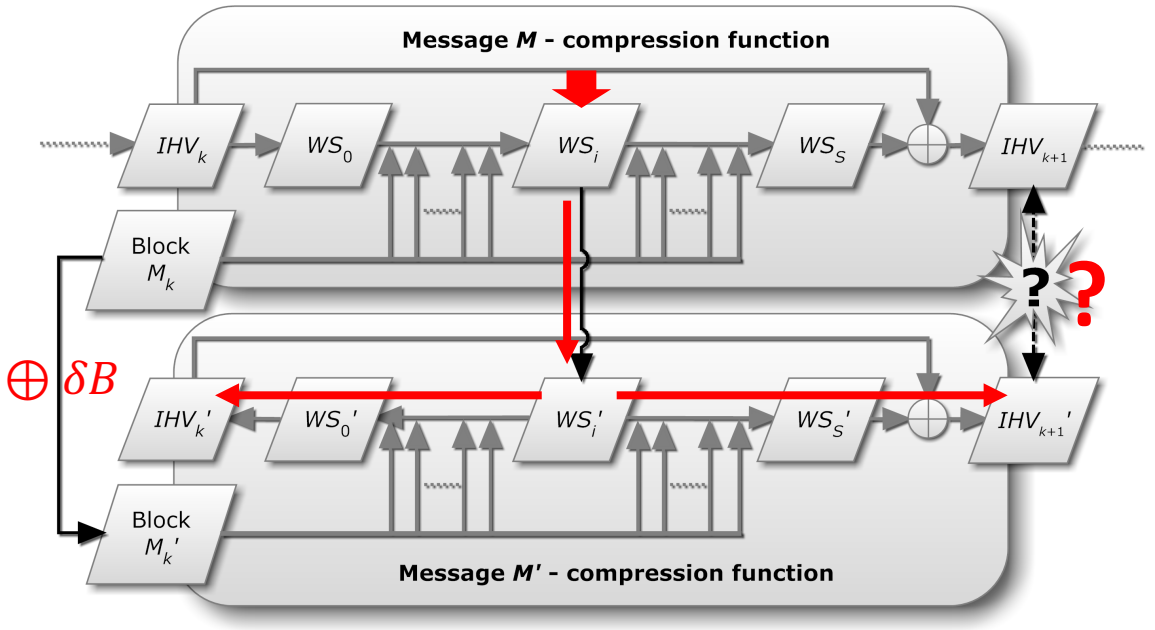
Detect collision block

Attack class defined by:

- i Step with zero working state difference.
- δB Message input difference.

1. Copy i-th working state WS_i.
2. Apply δB to message input.
3. Compute virtual second block.
4. Check for identical output.

Detects all variant attacks that use δB and have zero difference at step i.



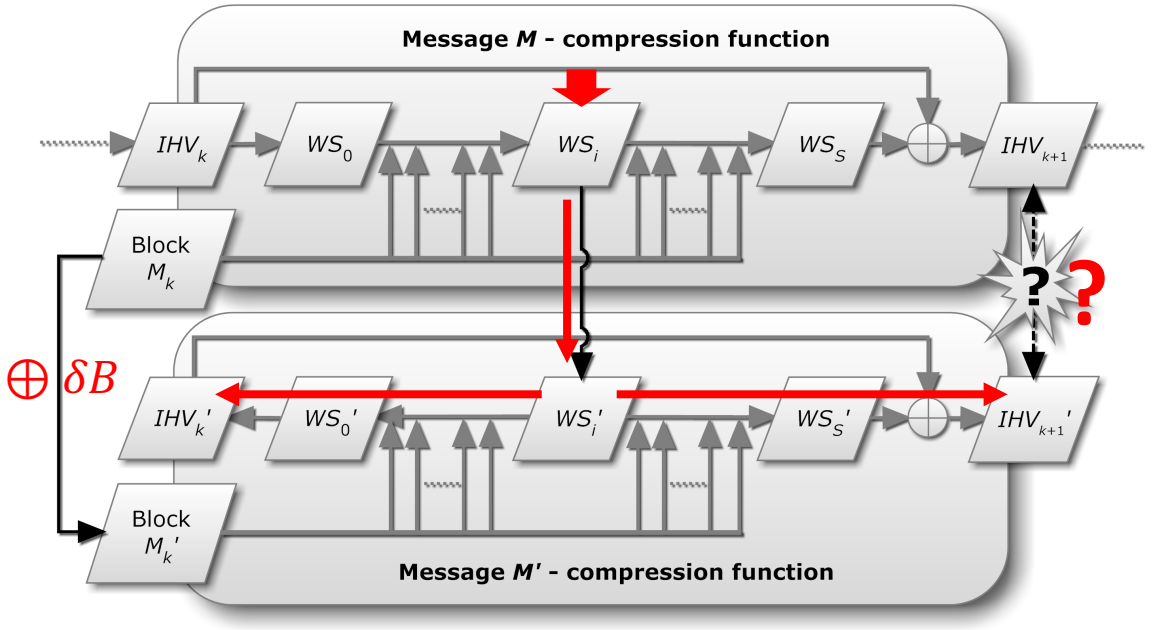
MD5 & SHA-1 collision detection

Full collision detection:

For each input block M_k:

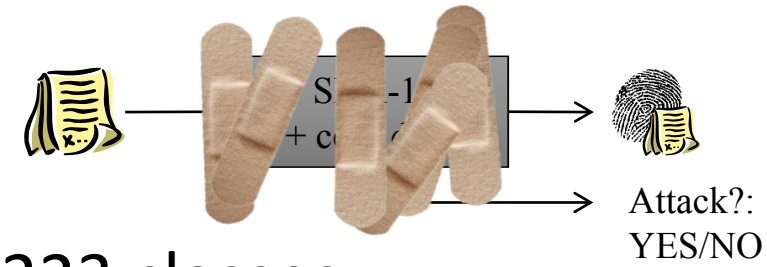
For each attack class $(i, \delta B)$:

Detect collision in M_k with $(i, \delta B)$.



MD5 & SHA-1 collision detection

- Collision detection has high cost:
  Every attack class costs an additional hash operation.
- SHA-1 is weak: ≥14 classes



- MD5 is very weak: ≥223 classes



MD5 & SHA-1 collision detection

Strong guarantees

1. False positives occur with negligible probability.
   Conjectured: $\approx 2^{-128}$ (MD5), $\approx 2^{-160}$ (SHA-1)

2. No false negatives:
   - Assuming list of feasible attack classes is ‘sufficiently complete.’
   - Lists based on thorough analyses in literature.
     MD5 [dBB93,WY05,SLdW07,XLF08,VJBT08,SSA+09,XF10,…]
     SHA1 [WYY05,BC04,PRR05,RO05,MP05,JP05,MPRR06,YIN+08,Man11,S13,…]

Improved collision detection

- Currently each attack class costs 1 full compression.
- Speed up with *unavoidable bit conditions (UBC)*:

  Conditions on mesage bits that are necessary for an attack class.
- Verify unavoidable conditions quickly, most of the time skip full work.
- Find UBCs by analyzing all feasible attack variants to avoid introducing false negatives.

SHA-1 collision detection library

- Collision detection performance:
  - Original paper: MD5: 224 x (223 attack classes)
  - Original paper: SHA-1: 15 x (14 attack classes)
  - This paper: SHA-1: <1.7 x (32 attack classes)
- Implementation of our improved algorithm available:

  https://github.com/cr-marcstevens/sha1collisiondetection
- Data + code generation tools + verification tools available:

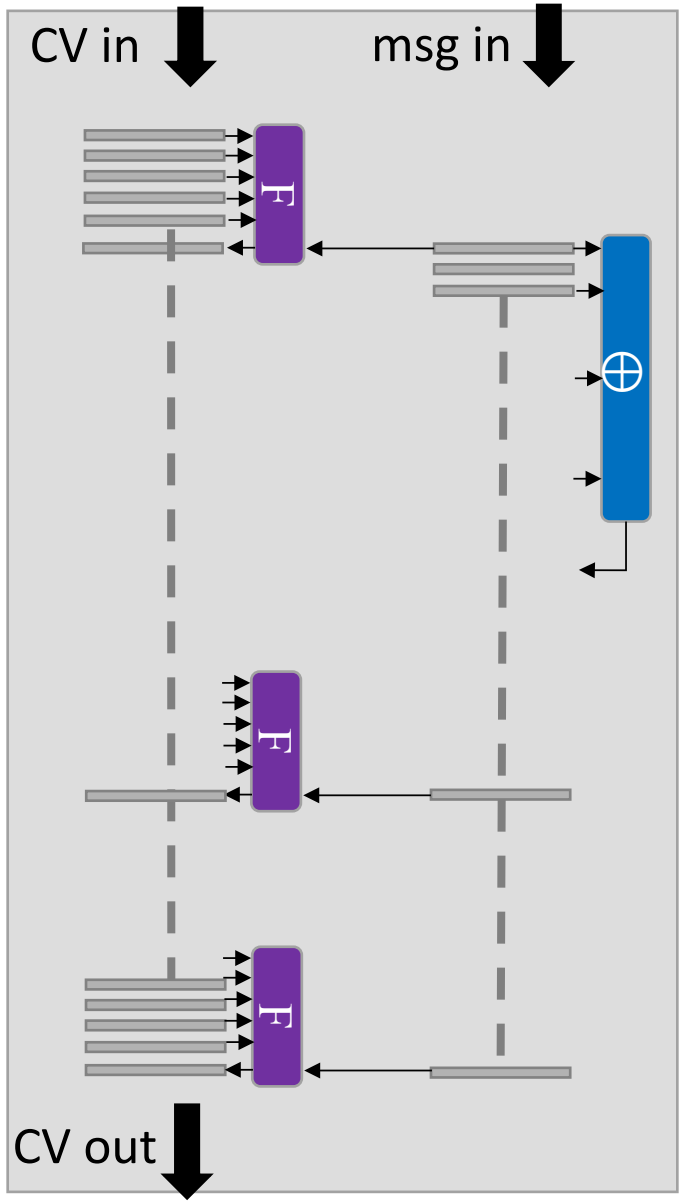
  https://github.com/cr-marcstevens/sha1collisiondetection-tools
- Deployed:
  - Git+Github (Released in Git 2.13)
  - Gmail+Google Drive
  - Microsoft OneDrive



SHA-1 differential cryptanalysis

Compression function

- Processes 512-bit chunk of input.
- Updates 160-bit chaining value (CV).
- Linearly expands input to words $W_0, \ldots, W_{79}$.
- Non-linearly mixes W_i into state in 80 rounds.

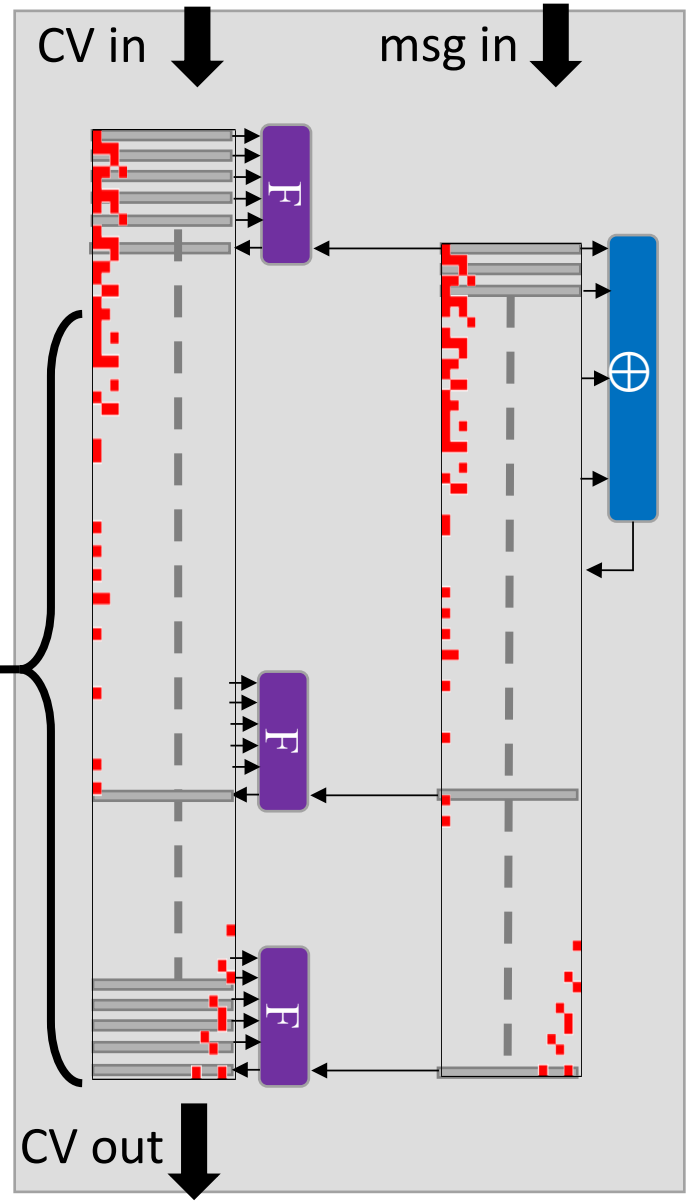


SHA-1 compression function

SHA-1 differential cryptanalysis

Differential Path

- Analyze two different instances.
- Describes exact differences in state & msg-input.
- Last 60 steps determine most of attack's complexity.
- Derive system of equations & solve.

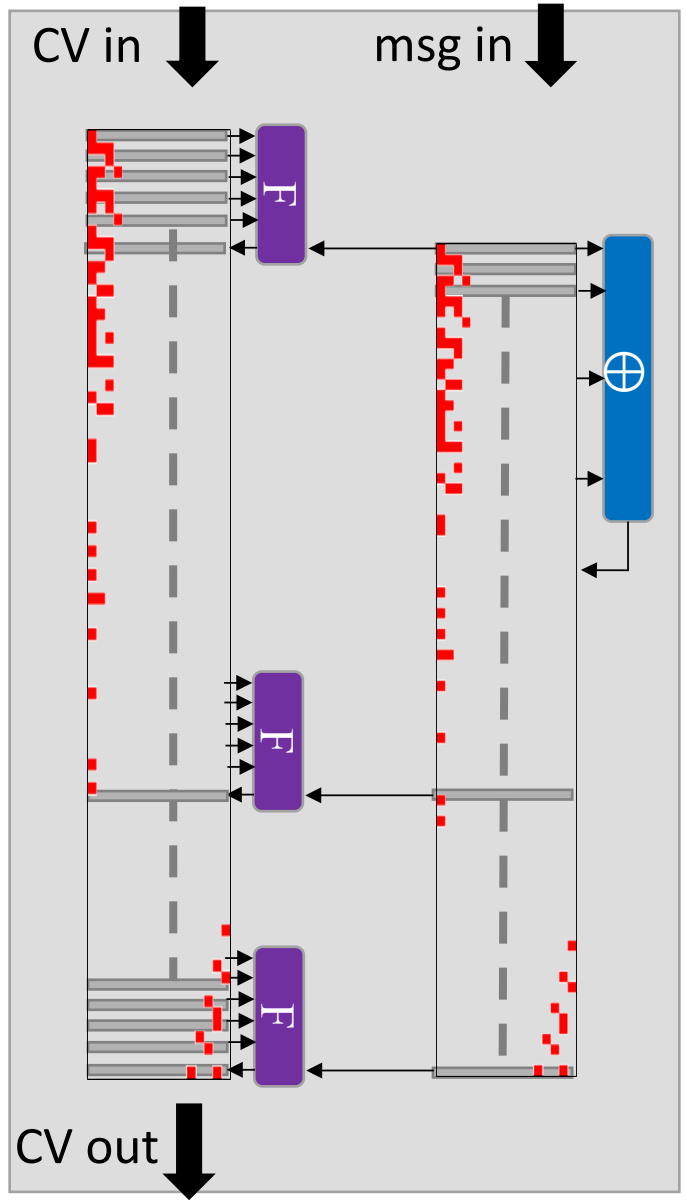


SHA-1 compression function

SHA-1 differential cryptanalysis

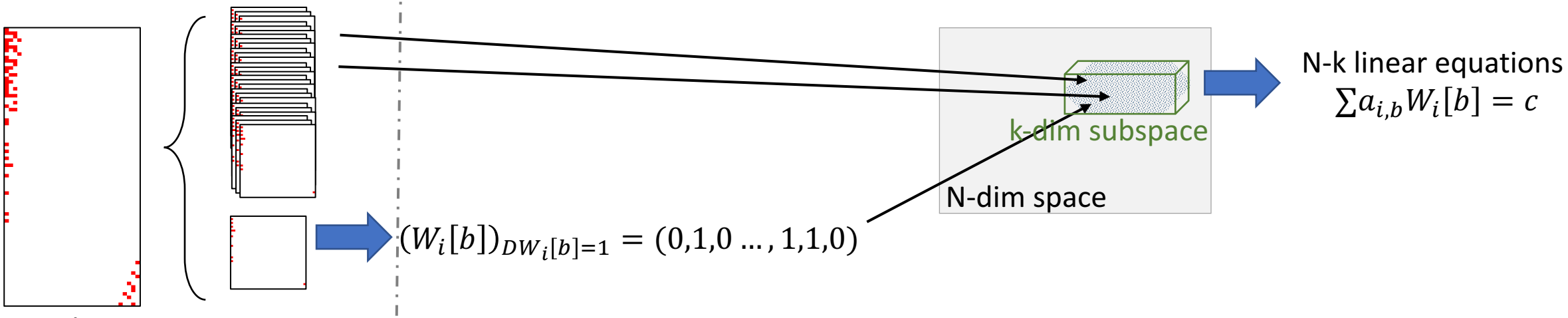
Disturbance vector (DV).

- Differential path DesignPlan™
- Describes combination of *local collisions.*
- Only known way compatible with msg expansion.
- Only two known feasible classes: I(K,b), II(K,b)
  - Zero difference at step K+9.
  - Determines message word XOR differences.



SHA-1 compression function

Computing Unavoidable Bit Conditions



Disturbance Vector

Enumerate <u>all</u> possible diffpath rounds 35-64

Translate into N-dim vector equation

$\mathcal{P}$ possible then $-\mathcal{P}$ also possible

$\Rightarrow$ vector w and $\overline{w}$ present

Determine smallest enveloping affine subspace $\mathcal{S}$

Compute N-k linear equations

Fast to check unavoidable bit conditions for possible attack

<u>Hard part</u>

Efficient exhaustive analysis using *JLCA* [S13,KPS15,SKP16,SBKAM17]

<u>Easy part</u>

Basic linear algebra

SHA-1 unavoidable bit conditions

- Selected the 32 DVs leading to lowest complexity attacks.
- Each DV has 7 to 15 UBC equations.
- 373 total *overlapping* equations.

$II(46,2)$:

$$W_{61}[2] \oplus W_{62}[7] = 1$$
$$W_{47}[1] \oplus W_{51}[1] = 1$$
$$W_{48}[6] \oplus W_{51}[1] = 0$$
$$W_{50}[6] \oplus W_{51}[1] = 0$$
$$W_{41}[1] \oplus W_{43}[1] = 1$$
$$W_{41}[1] \oplus W_{42}[6] = 1$$
$$W_{36}[1] \oplus W_{37}[6] = 1$$

Where $W_j[i]$ denotes the i^{th} bit of the j^{th} block of expanded input.

Exploiting overlapping UBCs

Example

$I(48,0)$:

$W_{35}[4] \oplus W_{39}[29] = 0$
$W_{63}[0] \oplus W_{64}[5] = 1$
$W_{42}[4] \oplus W_{52}[29] = 1$
$W_{42}[4] \oplus W_{51}[29] = 1$
$W_{42}[4] \oplus W_{49}[29] = 0$
$W_{42}[4] \oplus W_{48}[4] = 1$
$W_{42}[4] \oplus W_{47}[29] = 1$
$W_{42}[4] \oplus W_{46}[4] = 0$
$W_{42}[4] \oplus W_{44}[29] = 0$
~~$W_{42}[4] \oplus W_{45}[29] = 0$~~
$W_{42}[4] \oplus W_{44}[4] = 1$
$W_{38}[4] \oplus W_{42}[29] = 0$
$W_{38}[4] \oplus W_{40}[29] = 0$
$W_{38}[4] \oplus W_{41}[29] = 0$
$W_{44}[29] \oplus W_{45}[29] = 0$

$$\frac{\begin{array}{l} W_{42}[4] \oplus W_{44}[29] = 0 \\ W_{42}[4] \oplus W_{45}[29] = 0 \end{array}}{W_{44}[29] \oplus W_{45}[29] = 0}$$

$$\frac{\begin{array}{l} W_{41}[4] \oplus W_{44}[29] = 0 \\ W_{41}[4] \oplus W_{45}[29] = 0 \end{array}}{W_{44}[29] \oplus W_{45}[29] = 0}$$

$II(45,0)$:

$W_{36}[4] \oplus W_{40}[29] = 0$
$W_{41}[4] \oplus W_{43}[29] = 0$
$W_{41}[4] \oplus W_{44}[29] = 0$
~~$W_{41}[4] \oplus W_{45}[29] = 0$~~
$W_{52}[29] \oplus W_{53}[29] = 0$
$W_{50}[29] \oplus W_{53}[29] = 1$
$W_{60}[0] \oplus W_{61}[5] = 1$
$W_{63}[1] \oplus W_{64}[29] = 1$
$W_{47}[4] \oplus W_{53}[29] = 1$
$W_{49}[29] \oplus W_{53}[29] = 1$
$W_{49}[4] \oplus W_{53}[29] = 0$
$W_{44}[29] \oplus W_{45}[29] = 0$

Exploiting overlapping UBCs

Observations:

- The set of equations used to check the UBC for a given DV is not unique.
- DVs may have overlapping UBCs.

Question: Can we use these observations reduce the total number of equations checked?

Greedy selection algorithm:

1. Enumerate all potential UBC checking equations.
2. Count how many DVs each equation corresponds to and order by this count.
3. Select equation, prioritizing highest counts.
4. Repeat selection until the collection of equations contains a set of equations that completely checks the UBC for each DV.

Solution: Reduces 373 UBC equations over 32 DVs to 156 unique equations, each corresponding to anywhere from 1 to 7 DVs.
(all of the form: $W_i[a] \oplus W_j[b] = c$)

UBC check algorithm

- In our reduced set of equations, we must also keep track of all the corresponding DVs in a 32 bit mask.
- If a UBC equation fails to hold, we use this to mask out potentially satisfied UBCs.
- After checking all equations we are left with a 32bit word where a 1 bit corresponds to a DV with satisfied UBC.
- The C-code for this optimized check is automatically generated by the implementation of our greedy selection algorithm.

SHA1 Collision Detection Implementation: Testing and Verification

- Optimized UBC checking code compared against unoptimized UBC checking code.
  - Randomized testing: Hash many random blocks and compare the results of optimized and unoptimized code.
- Known Answer Tests:
  - Colliding SHA-1 PDFs.
  - Reduced-round SHA-1 collisions.
  - Random files (negative test.)

UBC check expected performance

- Given the probability that an UBC with m equations is satisfied is 2^{-m}. We estimate that given our set of 32 DVs on average we only check 0.049 DVs per block, or one full check every 20.2 blocks.
- We expect that our implementation will run about:

$$1 + u + 0.049\times \text{ SHA-1}$$

  Where u is the cost of UBC check relative to a SHA-1 compression.
- Experiments show that our UBC check function costs about 0.46 to 0.76 the cost of a SHA-1 compression.
- We expect our implementation to cost $1.51\times$ to $1.81\times$ SHA-1.

SHA1 Collision Detection Implementation: Performance

	SHA1	SHA1DC no UBC Check	SHA1DC UBC Check
gcc x86-64	148.14	3.75 (39.50x)	92.82 (1.60x)
clang x86-64	226.60	7.58 (29.88x)	136.33 (1.66x)
msvc x86-64	115.80	2.69 (42.98x)	72.23 (1.60x)
msvc x86-32	83.42	2.06 (40.58x)	58.14 (1.43x)
gcc arm	26.11	0.81 (32.04x)	16.30 (1.60x)

Count of 2KB message hashed per millisecond and slowdown over base operation.

- Performance of SHA-1 implementation tuned against Git's previous block-sha implementation.
- The performance improvement given by UBC checks made the difference between the code being deployed, such as in Git.
- Pure C / No Assembler

SHA1 Collision Detection Library

- Implementation of improved algorithm available:
  https://github.com/cr-marcstevens/sha1collisiondetection

- Deployed:
  - Git/Github (released in Git 2.13)
  - Gmail/Google Drive
  - Microsoft OneDrive

- Data + code generation tools + verification tools available :
  https://github.com/cr-marcstevens/sha1collisiondetection-tools

Ongoing Work

- Constant-time/execution implementation:
  - Performance improvement using SIMD: SSE128, AVX256, AVX512, NEON.
- Assembler implementation

Thank You