MD5

MD5

- **Message Digest 5**
- Strengthened version of MD4
- Significant differences from MD4 are
  - 4 rounds, 64 steps (MD4 has 3 rounds, 48 steps)
  - Unique additive constant each step
  - Round function less symmetric than MD4
  - **Each step adds result of previous step**
  - Order that input words accessed varies more
  - Shift amounts in each round are "optimized"

MD5 Algorithm

- For 32-bit words A,B,C, define

  $F(A,B,C) = (A \wedge B) \vee (\neg A \wedge C)$

  $G(A,B,C) = (A \wedge C) \vee (B \wedge \neg C)$

  $H(A,B,C) = A \oplus B \oplus C$

  $I(A,B,C) = B \oplus (A \vee \neg C)$

- Where $\wedge$, $\vee$, $\neg$, $\oplus$ are AND, OR, NOT, XOR, respectively
- Note that G "less symmetric" than in MD4

MD5 Algorithm

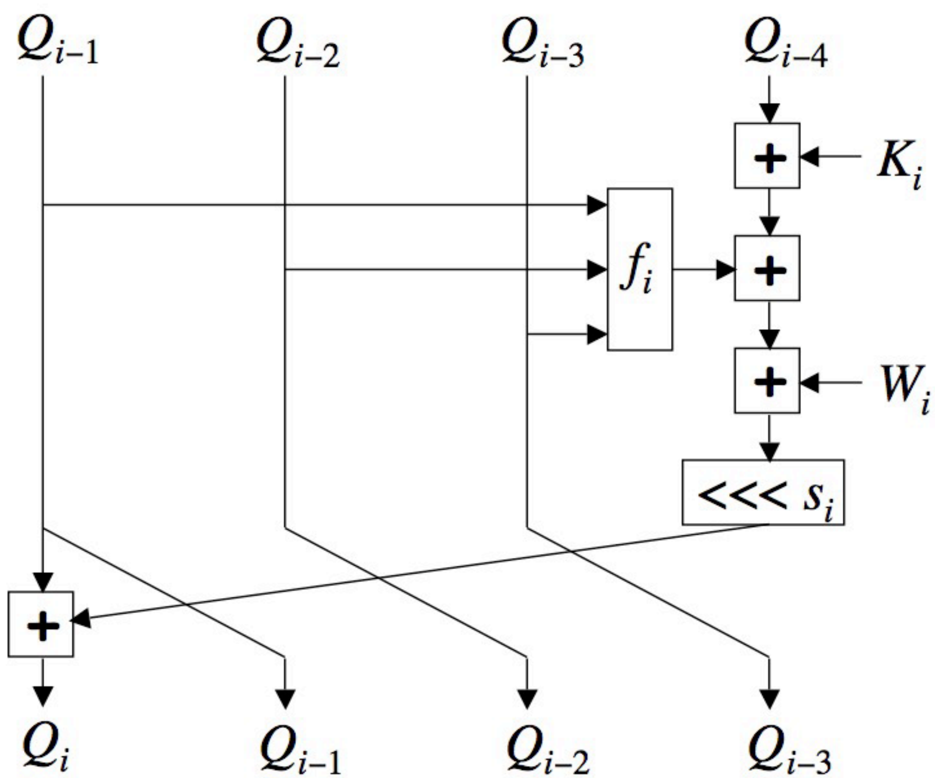
```
// M = (Y_0, Y_1, ..., Y_{N-1}), message to hash, after padding
// Each Y_i is a 32-bit word and N is a multiple of 16
MD5(M)
    // initialize (A, B, C, D) = IV
    (A, B, C, D) = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)
    for i = 0 to N/16 − 1
        // Copy block i into X
        X_j = Y_{16i+j}, for j = 0 to 15
        // Copy X to W
        W_j = X_{σ(j)}, for j = 0 to 63
        // initialize Q
        (Q_{−4}, Q_{−3}, Q_{−2}, Q_{−1}) = (A, D, C, B)
        // Rounds 0, 1, 2 and 3
        Round0(Q, W)
        Round1(Q, W)
        Round2(Q, W)
        Round3(Q, W)
        // Each addition is modulo 2^32
        (A, B, C, D) = (Q_60 + Q_{−4}, Q_63 + Q_{−1}, Q_62 + Q_{−2}, Q_61 + Q_{−3})
    next i
    return A, B, C, D
end MD5
```

MD5 Algorithm

Round0(Q, W)
    // steps 0 through 15
    for $i = 0$ **to** 15
        $Q_i = Q_{i-1} + ((Q_{i-4} + F(Q_{i-1}, Q_{i-2}, Q_{i-3}) + W_i + K_i) \lll s_i)$
    next i
end Round0

- Round 0: Steps 0 thru 15, uses F function
- Round 1: Steps 16 thru 31, uses G function
- Round 2: Steps 32 thru 47, uses H function
- Round 3: Steps 48 thru 63, uses I function

MD5: One Step



- Where $f_i(A,B,C) = \begin{cases} F(A,B,C) & \text{if } 0 \le i \le 15 \\ G(A,B,C) & \text{if } 16 \le i \le 31 \\ H(A,B,C) & \text{if } 32 \le i \le 47 \\ I(A,B,C) & \text{if } 48 \le i \le 63 \end{cases}$

MD5 Notation

- Let $MD5_{i\ldots j}(A,B,C,D,M)$ be steps i thru j
  - "Initial value" (A,B,C,D) at i, message M
- Note that $MD5_{0\ldots 63}(IV,M) \neq h(M)$
  - Due to padding and final transformation
- Let $f(IV,M) = (Q_{60},Q_{63},Q_{62},Q_{61}) + IV$
  - Where "+" is addition mod 2^{32} per 32-bit word
- Then f is the MD5 **compression function**

MD5 Compression Function

- Let $M = (M_0, M_1)$, each M_i is 512 bits
- Then $h(M) = f(f(IV, M_0), M_1)$
  - Assuming M includes padding
- That is, $f(IV, M_0)$ acts as "IV" for M_1
  - Can be extended to any number of M_i
- Merkle-Damgard construction
  - Used in MD4 and many hash functions

MD5 Attack: History

- Dobbertin "almost" able to break MD5 using his MD4 attack (ca 1996)
  - Showed that MD5 might be vulnerable
- In 2004, Wang published one MD5 collision
  - No explanation of method was given
- Based on one collision, Wang's method was reverse engineered by Australian team
  - Ironically, this reverse engineering work has been primary source to improve Wang's attack

MD5 Attack: Overview

- Determine two 1024-bit messages
  - $M' = (M'_0, M'_1)$ and $M = (M_0, M_1)$
- So that MD5 hashes are the same
  - That is, a collision attack
- Attack is efficient
  - Many improvements to Wang's original approach
- Note that
  - Each M_i and M'_i is a 512-bit block
  - Each block is 16 words, 32 bits/word

MD5 Attack: Overview

- Determine two 1024-bit messages
  - $M' = (M'_0, M'_1)$ and $M = (M_0, M_1)$
- So that MD5 hashes are the same
  - That is, a collision attack
- A differential cryptanalysis attack
- Idea is to use first block to generate desired "IV" for 2nd block
  - Can be viewed as a "chosen IV" attack

A Precise Differential

- Most differential attacks use XOR or modular subtraction for difference
- These are not sufficient for MD5
- Wang proposed
  - A "kind of precise differential"
  - More informative than XOR and modular subtraction combined

A Precise Differential

- Consider bytes

  $y' = 00010101$ and $y = 00000101$

  $z' = 00100101$ and $z = 00010101$

- Note that

  $y' - y = z' - z = 00010000 = 2^4$

- Then wrt modular subtraction, these pairs are indistinguishable
- In this case, XOR distinguishes the pairs

  $y' \oplus y = 00010000 \neq z' \oplus z = 00110000$

A Precise Differential

- Modular subtraction and XOR is not enough information!
    - Let $y' = (y'_0,y'_1,\ldots,y'_7)$ and $y = (y_0,y_1,\ldots,y_7)$
- Want to distinguish between, say, $y'_3=0$, $y_3=1$ and $y'_3=1$, $y_3=0$
- Use a **signed difference**, ∇y
    - Denote $y'_i=1$, $y_i=0$ as "+"
    - Denote $y'_i=0$, $y_i=1$ as "–"
    - Denote $y'_i=y_i$ as "."

A Precise Differential

- Consider bytes

  $z' = 10100101$ and $z = 10010101$
- Then ∇z is “. . + - . . . . .”
- Note that both XOR and modular difference can be derived from ∇z
- Also note same ∇ given by pairs

  $x' = 10100101$ and $x = 10010101$

  $y' = 10100101$ and $y = 10010101$

A Precise Differential

- Properties of Wang's signed differential
- More restrictive than XOR or modular difference
  - Provides greater "control" during attack
- But not too restrictive
  - Many pairs satisfy a given ∇ value
- Ideal balance of control and freedom

Wang's Attack

- Next, we outline Wang's attack
  - On part theory and one part computation
  - Overall attack splits into 4 steps
- More details follow
- Then discuss reverse engineering of Wang's attack
- Finally, consider whether attack is a practical concern or not

Wang's Attack

- Somewhat ad hoc
- Consider input and output differences
- Input differences
  - Applies to messages M' and M
  - Use modular difference
- Output differences
  - Applies to intermediate values, Q'_i and Q_i
  - Use Wang's signed difference

Wang vs Dobbertin

- Dobbertin's MD4 attack
  - Input differentials specified
  - Equation solving is main part of attack
- Wang's MD5 attack
  - More of a "pure" differential attack
  - Specify input differences
  - Tabulate output differences
  - Force some output differences to hold
  - Unforced differences satisfied probabilistically

Wang's Attack: Step 1

- **Specify input differential pattern**
  - Must "behave nicely" in later rounds
  - These differentials are given below
  - Modular difference used for inputs
- Only need to specify M
  - Then M′ is determined by differential

Wang's Attack: Step 2

- **Specify output differential pattern**
  - Must "behave nicely" in early rounds
  - That is, easily satisfied in early rounds
  - Restrictive signed difference used
  - Most mysterious part of attack
  - Wang used "intuitive" approach
- Only 1 such pattern known (Wang's)

Wang's Attack: Step 3

- **Derive set of sufficient conditions**
  - Using differential patterns
- If these conditions are all met
  - Differential patterns hold
  - Therefore, we obtain a collision

Wang's Attack: Step 4

- **Computational phase**
- Must find pair of 1024-bit messages that satisfy all conditions in step 3
  - Messages: $M = (M_0,M_1)$ and $M' = (M'_0,M'_1)$
- Deterministically satisfy as many conditions as possible
- Any remaining conditions must be satisfied probabilistically
  - Number of such conditions gis expected work

Wang's Attack: Step 4

- Computational phase:
  a) Generate random 512-bit M_0
  b) Use **single-step modification** to force some conditions in early steps to hold
  c) Use **multi-step modification** to force some conditions in middle steps to hold
  d) Check all remaining conditions—if all hold then have desired M_0, else goto b)
  e) Follow similar procedure to find M_1
  f) Compute M'_0 and M'_1 (easy) and collision!

Wang's Attack: Work Factor

- Work is dominated by finding M_0
- Work determined by number of probabilistic conditions
  - Work is on the order of 2^n where n is number of such conditions
- Wang's original attack: n > 40
  - Hours on a supercomputer
- Best as of today, about n = 32.25
  - Less than 2 minutes on a PC

Wang's Differentials

- Input and output differentials
- Notation: "+" over n for 2^n and "−" over n for -2^n
  - For example: $(\overset{\pm}{31}\,\overset{+}{23}\,\overset{-}{6}) = \pm 2^{31} + 2^{23} - 2^{6}$
- Consider 2-block message: $h(M_0,M_1)$
- Notation: $IV = (A,B,C,D)$
- Denote "IV" for M_1 as IV_1 (and IV'_1 for M'_1)
  - Then $IV_1 = (Q_{60},Q_{63},Q_{62},Q_{61}) + (A,B,C,D)$
  - Where Q_i are outputs when hashing M_0
- Let $h = h(M_0,M_1)$ and $h' = h(M'_0,M'_1)$

Wang's Input Differential

- Required **input differentials**

  $\Delta M_0 = M'_0 - M_0 = (0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0)$

  $\Delta M_1 = M'_1 - M_1 = (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0)$

  - Note: M'_0 and M_0 differ only in words 4, 11 and 14
  - Note: M'_1 and M_1 differ only in words 4, 11 and 14
  - Same differences except in word 11
- Also required that

  $\Delta IV_1 = IV'_1 - IV_1 = (2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31})$
- Goal is to obtain $\Delta h = h' - h = (0,0,0,0)$

Wang's Output Differential

- Required **output differentials**
- Part of ΔM_0 differential table:

j	Output	W_j	ΔW_j	ΔOutput	∇Output
4	Q_4	X_4	2^{31}	$\overset{-}{6}$	....−++++++ ++++++++ ++......
5	Q_5	X_5	0	$\overset{+}{31}\ \overset{+}{23}\ \overset{-}{6}$	+....... +....... ....−......
6	Q_6	X_6	0	$\overset{-}{27}\ \overset{+}{23}\ \overset{-}{6}\ \overset{-}{0}$	++++++−− −....... ....−+++ ++−+++++
7	Q_7	X_7	0	$\overset{-}{23}\ \overset{-}{17}\ \overset{-}{15}\ \overset{+}{0}$	.... −..−+++− +....... ....+
8	Q_8	X_8	0	$\overset{+}{31}\ \overset{-}{6}\ \overset{+}{0}$	−....... ....− ++....+−

  - Q_i are outputs for M_0
  - ΔW_j are input (modular) differences
  - ΔOutput is output modular difference
  - ∇Output is output signed ("precise") difference

Derivation of Differentials?

- Where do differentials come from?
  - "Intuitive", "done by hand", etc.
- Input differences are fairly reasonable
- Output differences are more mysterious
- We briefly consider history of MD5 attacks
- Then reverse engineering of Wang's method
  - None of this is entirely satisfactory…

History of MD5 Attacks

- Dobbertin tried his MD4 approach
  - Modular differences and equation solving
  - No true collision obtained, but did highlight potential weaknesses
- Chabaud and Joux
  - Use XOR differences
  - Approximate nonlinearity by XOR (like in linear cryptanalysis)
  - Had success against SHA-0

History of MD5 Attacks

- Wang's attack
  - Modular differences for inputs
  - Signed differential for outputs
  - Gives more control over outputs and actual step functions, not approximations
  - Also, uses 2 blocks, so second block is essentially "chosen IV" attack
- Wang's magic lies in differential patterns
  - How were these chosen?

Daum’s Insight

- Wang’s attack could be “expected” to work against MD-like hash with 3 rounds
  - Input differential forces last round conditions
  - Single-step modification forces 1st round
  - Multi-step modifications forces 2nd round
- But MD5 has 4 rounds!
- A special property of MD5 is exploited:
  - Output difference of 2^{31} “propagated from step to step with probability 1 in the 3rd round and with probability 1/2” in most of 4th round

Wang's Differentials

- No known method for automatically generating useful MD5 differentials
- Daum: build tree of difference patterns
  - Include both input and output differences
  - Prune low probability paths from tree
  - Connect "inner collisions", etc.
- However, Wang's differentials are only useful ones known today

Reverse Engineering Wang's Attack

- Based on 1 published MD5 collision
- Computed intermediate values
- Examined modular, XOR, signed difference
- Uncovered many aspects of attack
- Resulted in computational improvements
- Overall, an impressive piece of work!

Conditions

- For first round, define

  $T_j = F(Q_{j-1},Q_{j-2},Q_{j-3}) + Q_{j-4} + K_j + W_j$

  $R_j = T_j <<< s_j$

  $Q_j = Q_{j-1} + R_j$
- Initial values: $(Q_{-4},Q_{-3},Q_{-2},Q_{-1})$
- This is equivalent to previous notation

Conditions

- Let Δ be modular difference: $\Delta X = X' - X$
- Then

  $\Delta T_j = \Delta F_{j-1} + \Delta Q_{j-4} + \Delta W_j$
  $\Delta R_j \approx (\Delta T_j) \lll s_j$
  $\Delta Q_j = \Delta Q_{j-1} + \Delta R_j$

- Where $\Delta F_j = F(Q_j, Q_{j-1}, Q_{j-2}) - F(Q'_j, Q'_{j-1}, Q'_{j-2})$
- The ΔR_j equation holds with high probability
- Tabulated ΔQ_j, ΔF_j, ΔT_j, and ΔR_j for all j

Conditions

- Derive conditions on ΔT_j and ΔQ_j that ensure known differential path holds
- Conditions on ΔT_j not used in original attack
  - More efficient recent attacks do use these
- Goal is to deterministically (or with high prob) satisfy as many conditions as possible
  - Reduces number of iterations needed

T Conditions

- Recall

  $\Delta T_j = \Delta F_{j-1} + \Delta Q_{j-4} + \Delta W_j$

  $\Delta R_j \approx (\Delta T_j) \lll s_j$
- Interaction of "Δ" and "$\lll$" is tricky
- Suppose $T' = 2^{20}$ and $T = 2^{19}$ and $s = 10$
- Then

  $(\Delta T) \lll s = (T' - T) \lll s = 2^{29}$ and

  $\Delta(T \lll s) = (T' \lll s) - (T \lll s) = 2^{29}$
- In this example, "Δ" and "$\lll$" commute

T Conditions

- Spse $T' = 2^{22}$, $T = 2^{21} + 2^{20} + 2^{19}$, $s = 10$
- Then

  $(\Delta T) \lll s = (T' - T) \lll s = 2^{29}$

  but

  $(T' \lll s) - (T \lll s) = 2^{29} + 1$
- Here, "Δ" and "$\lll$" do not commute
- Negative numbers can be tricky

T Conditions

- If ΔT and s are specified, conditions on T are implied by $\Delta R = (\Delta T) \lll s$
- Can always force a "wrap around" in ΔR
  - Can be little bit tricky due to non-commuting
- Recall

  $T_j = F(Q_{j-1},Q_{j-2},Q_{j-3}) + Q_{j-4} + K_j + W_j$
- Given M, conditions on T_j can be checked
- Better yet, want to select M so that many of the required T conditions hold

T Conditions: Example

- At step 5 of Wang's collision:

  $\Delta T_5 = 2^{19} + 2^{11}$, $\Delta Q_4 = -2^6$, $\Delta Q_5 = \pm 2^{31} + 2^{23} - 2^6$, $s_5 = 12$
- Since $Q_j = Q_{j-1} + R_j$, it is easy to show that $\Delta R_5 = \Delta Q_5 - \Delta Q_4 = \pm 2^{31} + 2^{23}$
- We also have $\Delta R_5 \approx (\Delta T_5) \lll s_5$
- Implies conditions on **any** ΔT_5 that satisfies Wang's differentials!

T Conditions: Example

- From the previous slide:

  $\Delta R_5 = \pm 2^{31} + 2^{23} = (\Delta T_5) \lll 12$

- Of course, the known ΔT_5 works: $\Delta T_5 = 2^{19} + 2^{11}$
- But, for example, $\Delta T_5 = 2^{20} - 2^{19} + 2^{11}$, does not work, since rotation would "wrap around"
- Implies there can be no 2^{20} term in T_5
  - Complex condition to restrict borrows also needed
- **Bottom line**: Can derive a set of conditions on Ts that ensure Wang's differential path holds

Output Conditions

- Easier to check Q conditions than T
  - The Q are known as "outputs"
  - Actually, intermediate values in algorithm
- Much easier to specify M so that Q conditions hold than T conditions
- In attacks, Q conditions mostly used

Output Conditions

- Use signed differential, ∇X
- For example, if

  $X' = 0x02000020$ and $X = 0x80000000$

  then ∇X is denoted

  "- . . . . . . + ... + ..."
- Also we must analyze round function:

  $F(A,B,C) = (A \wedge B) \vee (\neg A \wedge C)$
- Bits of A choose between bits of B and C

Output Conditions: Example

- At step 4 of Wang's collision:

$\Delta Q_2 = \Delta Q_3 = 0,\ \Delta Q_4 = -2^6,\ \Delta F_4 = 2^{19} + 2^{11}$

	Δ	∇
Q_2	0	`........ ........ ........ ........`
Q_3	0	`........ ........ ........ ........`
Q_4	$\bar{6}$	`........ .-++++++ ++++++++ ++......`
F_4	$\overset{+}{19}\ \overset{+}{11}$	`........ ....+... ....+... ........`

- From ∇Q_4 we have:

$\langle Q_4 = 1 \rangle_9$ and $\langle Q_4 = 0 \rangle_{10\ldots25}$

- Note that $Q'_4 = Q_4$ at all other bits

Output Conditions: Example

- From ∇Q_4 we have:

  $\langle Q_4 = 1 \rangle_9$ and $\langle Q_4 = 0 \rangle_{10\ldots25}$
- Note that $Q'_4 = Q_4$ at all other bits
- Bits 9,10,…,25 are “constant” bits of Q_4
- All others are “non-constant” bits of Q_4
- On constant bits, $Q'_4 = Q_4$ and on non-constant bits, $Q'_4 \neq Q_4$

Output Conditions: Example

- Consider constant bits of Q_4
- Since $F_4 = F(Q_4,Q_3,Q_2)$, from defn of F
  - If $\langle Q_4 = 1\rangle_j$ then $\langle F_4 = Q_3\rangle_j$ and $\langle F'_4 = Q'_3\rangle_j$
  - If $\langle Q_4 = 0\rangle_j$ then $\langle F_4 = Q_2\rangle_j$ and $\langle F'_4 = Q'_2\rangle_j$
- Then $\langle F_4 = F'_4\rangle_i$ for each constant bit j

	Δ	∇
Q_2	0	....
Q_3	0	....
Q_4	$\bar{6}$	....−++++++ ++++++++ ++......
F_4	$\overset{+}{1}9\ \overset{+}{1}1$	....+... ....+... ....

- From table, constant bits of Q_4 are constant bits of F_4 so no conditions on Q_4

Output Conditions: Example

- Consider non-constant bits of Q_4
- Since $F_4 = F(Q_4,Q_3,Q_2)$, from defn of F
  - If $\langle Q_4 = 1\rangle_j$ then $\langle F_4 = Q_3\rangle_j$ and $\langle F'_4 = Q'_2\rangle_j$
  - If $\langle Q_4 = 0\rangle_j$ then $\langle F_4 = Q_2\rangle_j$ and $\langle F'_4 = Q'_3\rangle_j$

	Δ	∇
Q_2	0	....
Q_3	0	....
Q_4	$\bar{6}$	....−++++++ ++++++++ ++......
F_4	$\overset{+}{19}\ \overset{+}{11}$	....+... ....+... ....

- Note that on bits 10,11,13,…,19,21,…,25

  $F_4 = F'_4,\ Q'_4 = 1,\ Q_4 = 0 \Rightarrow F_4 = Q_2,\ F'_4 = Q'_3$
- Since $Q_3 = Q'_3$ we have $\boldsymbol{\langle Q_3 = Q_2\rangle_{10,11,13\ldots19,21,,,25}}$

Output Conditions: Example

- Still need to consider bits 9,12,20
  - See textbook
- From step 4, we derive the following output conditions:

$\langle Q_4 = 0 \rangle_{10,,,25}$, $\langle Q_4 = 1 \rangle_9$

$\langle Q_3 = 1 \rangle_{12,20}$

$\langle Q_2 = 0 \rangle_{12,20}$, $\langle Q_2 = Q_3 \rangle_{10,11,13\ldots19,21,,,25}$

Conditions: Bottom Line

- By reverse engineering one collision…
  - Able to deduce output conditions
- If all of these are satisfied, we will obtain a collision
- This analysis resulted in much more efficient implementations
- All base on one known collision!

Single-Step and Multi-Step Modifications

- Given conditions, how can we use them?
- That is, how can we make them hold?
- Two techniques are used:
- **Single-step modifications**
  - Easy way to force many output conditions
- **Multi-step modifications**
  - Complex way to force a few more conditions

Single-Step Modification

- Select $M_0 = (X_0, X_1, \ldots, X_{15})$ at random
- Note that $W_i = X_i$ for $i = 0, 1, \ldots, 15$
- Also, $IV = (Q_{-4}, Q_{-1}, Q_{-2}, Q_{-3})$
- Compute outputs $Q_0, Q_1, \ldots, Q_{15}$
  - For each Q_i, modify corresponding W_i so that required output conditions hold
  - This is easy—example on next slides

Single-Step Modification

- Suppose Q_0 and Q_1 are done
- Consider Q_2 where
  $Q_2 = Q_1 + (f_1 + Q_{-2} + W_2 + K_2) <<< s_2$
  - Recall that "<<<" is left rotation
  - Recall $f_i = F(Q_i,Q_{i-1},Q_{i-2})$ for $i = 0,1,\ldots,15$
- Required conditions: $\langle Q_2 = 0\rangle_{12,20,25}$
  - This means bits 12, 20 and 25 of Q_2 must be 0 (bits numbered left-to-right from 0 to 31)
  - No restriction on any other bits of Q_2
- We can modify W_2 so condition on Q_2 holds

Single-Step Modification

- For Q_2 we want $\langle Q_2 = 0\rangle_{12,20,25}$
- Compute $Q_2 = Q_1 + (f_1 + Q_{-2} + W_2 + K_2) \lll s_2$
  - Denote bits of Q_2 as $(q_0,q_1,q_2,\ldots,q_{31})$
- Let E_i be 32-bit word with bit i set to 1
  - All other bits of E_i are 0
- Let $D = -q_{12}E_{12} - q_{20}E_{20} - q_{25}E_{25}$
- Let $Q_2 = Q_2 + D$
- Replace W_2 with

  $W_2 = ((Q_2 - Q_1) \ggg s_2) - f_1 - Q_{-2} - K_2$
- Then conditions on Q_2 all hold

Single-Step Mod: Summary

- Modify words of message M_0
  - Alternatively, select $Q_0,Q_1,\ldots,Q_{15}$ so conditions satisfied, then compute corresponding M_0
- All output conditions steps 0 to 15 satisfied
- Suppose c conditions remain unsatisfied
  - Then after 2^c iterations, expect to find M_0 that satisfies all output conditions
- Most output conditions are in first 16 steps
  - Single-step mods provide a shortcut attack
  - But we can do better…

Multi-Step Modification

- Want to force some output conditions beyond step 15 to hold
- Tricky, since we must maintain all conditions satisfied in previous steps
  - And we already modified all input words
- Many multi-step mod techniques
  - We discuss the simplest

Multi-Step Modification

- Let $M_0 = (X_0, X_1, \ldots, X_{15})$ be M_0 after single-step mods
- Want $\langle Q_{16} = 0 \rangle_0$ to hold
- First, single-step modification:

  $D = -q_0 E_0$ and $Q_{16} = Q_{16} + D$ and
  $W_{16} = ((Q_{16} - Q_{15}) \ggg s_{16}) - f_{15} - Q_{12} - K_{16}$
- Note that $W_{16} = X_1$
- And X_1 used to compute Q_i for i=1,2,3,4,5
  - Don't want to change any Q_i in rounds 0 thru 15

Multi-Step Modification

- Compute

  $W_{16} = ((Q_{16} - Q_{15}) >>> s_{16}) - f_{15} - Q_{12} - K_{16}$
- Where $W_{16} = X_1$
- Problem with Q_i for i=1,2,3,4,5
  - No conditions on Q_1, so it's no problem
- Let $Z = Q_0 + (f_0 + Q_{-3} + X_1 + K_1) <<< s_1$
- Then Z is new Q_1, which is OK
- Do "single-step mods" for i=2,3,4,5

Multi-Step Modification

- Have $Z = Q_0 + (f_0 + Q_{-3} + X_1 + K_1) \lll s_1$
- Note that Z is new Q_1
- Do "single-step mods" for i=2,3,4,5

  $X_2 = ((Q_2 - Z) \ggg s_2) - f_1(Z,Q_0,Q_{-1}) - Q_{-2} - K_2$

  $X_3 = ((Q_3 - Q_2) \ggg s_3) - f_2(Q_2,Z,Q_0) - Q_{-1} - K_3$

  $X_4 = ((Q_4 - Q_3) \ggg s_4) - f_3(Q_3,Q_2,Z) - Q_0 - K_4$

  $X_5 = ((Q_5 - Q_4) \ggg s_5) - f_4(Q_4,Q_3,Q_2) - Z - K_5$
- Then all conditions on Q_i, i=0,1,…,15, still hold

Multi-Step Mods: Summary

- Many different multi-step mods
- Ad hoc way to satisfy output conditions
  - Care needed to maintain prior conditions
- Some multi-step mods only hold probabilistically
- Multi-step mods have probably been taken about as far as possible
  - Further improvements, incremental at best
- Best implementation: 2 minutes/collision

Stevens' Implementation

- Best implementation of Wang's attack
- About 2 minutes per collision on PC
- Finding M_0 is most costly (shown here)
- Algorithm for M_1 is similar

```
// Find M_0 = (X_0, X_1, ..., X_15), where "all M_0 conditions" refers to:
//      all Table A-7 conditions,
//      all IV conditions for M_1 (see Table A-8),
//      both <T_21 = 0>_14 and <T_33 = 0>_16
Find M_0
    repeat
        Choose Q_0, Q_2, Q_3, ..., Q_15 satisfying conditions in Table A-6
        Compute X_0, X_6, X_7, ..., X_15
        repeat
            Choose Q_16 satisfying conditions
            Compute X_1 using j = 16
            Compute Q_1 and X_2, X_3, X_4, X_5
            Compute Q_17, Q_18, Q_19, Q_20
        until Q_16, Q_17, ..., Q_20 satisfy conditions in Table A-6
        for (Q_8, Q_9) consistent with X_11
            Compute X_8, X_9, X_10, X_12, X_13
            Compute Q_21, Q_22, ..., Q_63
            if all M_0 conditions are satisfied then
                return M
            end if
        next (Q_8, Q_9)
    until all M_0 conditions are satisfied
end Find M_0
```

A Practical Attack?

- Wang's attack is very restrictive
  - Generates "meaningless" collisions
  - Not feasible for meaningful collision
- Is attack a real-world threat?
- In some cases, meaningless collisions can cause problems
  - We illustrate such a scenario

A Practical Attack

- Consider 2 letters, "written" in postscript:

rec.ps

To Whom it May Concern:

Tom Austin and Ying Zhang have demonstrated decent programming ability. They should do OK in any programming position, provided that the work is not too complex, and that the position does not require any independent thought or initiative.

However, I think they like to steal office supplies, so I would keep a close eye on them. Also, their basic hygiene is somewhat lacking so I would recommend that you have them telecommute.

Sincerely,

Alice

auth.ps

To Bank of America:

Tom Austin and Ying Zhang are authorized access to all of my account information and may make withdrawals or deposits.

Sincerely,

Alice

- Suppose the file rec.ps signed by Alice
  - That is, $S = [h(\text{rec.ps})]_{\text{Alice}}$
- If h(auth.ps) = h(rec.ps), signature broken

A Practical Attack

- Amazingly, h(auth.ps) = h(rec.ps)
- And Wang's attack was used
- How is this possible?
- Postscript has conditional statement: (X)(Y)eq{T_0}{T_1}ifelse
- If X == Y then T_0 is processed; else T_1 is processed

A Practical Attack

- Postscript statement: (X)(Y)eq{T_0}{T_1}ifelse
- How to take advantage of this?
- Add spaces, so that postscript file begins with exactly one 512-bit block
  - Call this block W
  - Last byte of W is "(" in (X)
- Let Z = $\text{MD5}_{0\ldots63}$(IV,W) so that Z is output of compression function applied to W

A Practical Attack

- Let $Z = MD5_{0\ldots63}(IV,W)$
- Use Wang's attack as follows
- Find collision:
  - 1024-bit M and M′ with $M \neq M'$ and $h(M) = h(M')$
  - Where IV is Z instead of standard IV
- Wang's attack easily modified to work for any non-standard IV
- Now what?

A Practical Attack

- Consider …(X)(Y)eq{T_0}{T_1}ifelse
  - Note that "…(" is W
  - Let T_0 = postscript for "rec" letter
  - Let T_1 = postscript for "auth" letter
  - Let L = …(M)(M)eq{T_0}{T_1}ifelse
  - Let L′ = …(M′)(M)eq{T_0}{T_1}ifelse
- Then **h(L) = h(L′)** since
  - h(W,M) = h(W,M′)
  - h(A) = h(B) implies h(A,C) = h(B,C) for any C
- File L displays T_0 and file L′ displays T_1

A Practical Attack

- File L = rec.ps
- First block: W
- X block: M
- Y block: M
- Display "rec"

```
%!PS-Adobe-1.0
%%BoundingBox: 0 0 612 792␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(X)(Y)eq{
/Times-Roman findfont 20 scalefont setfont
25 450 moveto (To Whom it May Concern:) show
25 400 moveto
(Tom Austin and Ying Zhang have demonstrated...
    ⋮
(Sincerely,)
show
25 150 moveto
(Alice)
show
}{/Times-Roman findfont 20 scalefont setfont
25 450 moveto (To Bank of America:) show
25 400 moveto
(Tom Austin and Ying Zhang are authorized access...
    ⋮
(Sincerely,)
show
25 250 moveto
(Alice)
show
}ifelse
showpage
```

A Practical Attack

- File L′ = auth.ps
- First block: W
- X block: M′
- Y block: M
- Display "auth"

```
%!PS-Adobe-1.0
%%BoundingBox: 0 0 612 792␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(X)(Y)eq{
/Times-Roman findfont 20 scalefont setfont
25 450 moveto (To Whom it May Concern:) show
25 400 moveto
(Tom Austin and Ying Zhang have demonstrated...
    ⋮
(Sincerely,)
show
25 150 moveto
(Alice)
show
}{/Times-Roman findfont 20 scalefont setfont
25 450 moveto (To Bank of America:) show
25 400 moveto
(Tom Austin and Ying Zhang are authorized access...
    ⋮
(Sincerely,)
show
25 250 moveto
(Alice)
show
}ifelse
showpage
```

A Practical Attack

- Bottom Line: A meaningless collision is a potential security problem
- Of course, anyone who looks at the file would see that something is wrong
- But, purpose of integrity check is to **automatically** detect problems
  - How to automatically detect such problems?
- This is a serious attack!
  - May also be possible for Word, PDF, etc.

Wang's Attack: Bottom Line

- Extremely clever and technical
- Computational aspects are well-understood
- Theoretical aspects not well-understood
  - Complex, difficult to analyze
  - Not well-explained by inventors
  - Must rely on reverse engineering
- No "meaningful" collisions are possible
- But attack is a practical concern!
- MD5 is broken