what is a hash function?

- \( h : \{0,1\}^* \rightarrow \{0,1\}^n \) (general: \( h : S \rightarrow \{0,1\}^n \) for some set \( S \))
- input: bit string \( m \) of arbitrary length
  - length may be 0
  - in practice a very large bound on the length is imposed, such as \( 2^{64} \approx 2.1 \text{ million TB} \)
  - input often called the message
- output: bit string \( h(m) \) of fixed length \( n \)
  - e.g. \( n = 128, 160, 224, 256, 384, 512 \)
  - compression
  - output often called hash value, message digest, fingerprint
- \( h(m) \) is easy to compute from \( m \)
- no secret information, no key
non-cryptographic hash functions

• hash table
  – index on database keys
  – use: efficient storage and lookup of data

• checksum
  – Example: CRC – Cyclic Redundancy Check
    • CRC32 uses polynomial division with remainder
      – initialize:
        – \( p = 1\ 0000\ 0100\ 1100\ 0001\ 1101\ 1011\ 0111 \)
        – append 32 zeroes to \( m \)
      – repeat until length (counting from first 1-bit) \( \leq 32 \):
        – left-align \( p \) to leftmost nonzero bit of \( m \)
        – XOR \( p \) into \( m \)
      – use: error detection
        • but only of unintended errors!

• non-cryptographic
  – extremely fast
  – not secure at all

October 20, 2011

hash collision

• \( m_1, m_2 \) are a collision for \( h \) if
  \[ h(m_1) = h(m_2) \] while \( m_1 \neq m_2 \)

• there exist a lot of collisions
  – pigeonhole principle
    (a.k.a. Schubladensatz)

I owe you € 100

I owe you € 5000
different documents

identical hash = collision

October 20, 2011
preimage

• given $h_0$, then $m$ is a preimage of $h_0$ if $h(m) = h_0$

second preimage

• given $m_0$, then $m$ is a second preimage of $m_0$ if $h(m) = h(m_0)$ while $m \neq m_0$
cryptographic hash function requirements

- **collision resistance**: it should be computationally infeasible to find a collision $m_1, m_2$ for $h$
  - i.e. $h(m_1) = h(m_2)$
- **preimage resistance**: given $h_0$ it should be computationally infeasible to find a preimage $m$ for $h_0$ under $h$
  - i.e. $h(m) = h_0$
- **second preimage resistance**: given $m_0$ it should be computationally infeasible to find a second preimage $m$ for $m_0$ under $h$
  - i.e. $h(m) = h(m_0)$
- more formal definitions exist, but we’ll keep things practical

other terminology

- **one-way** = preimage + second preimage resistant
  - sometimes only preimage resistant
- **weak collision resistant** = second preimage resistant
- **strong collision resistant** = collision resistant
- **OWHF** – one-way hash function
  - preimage and second preimage resistant
- **CRHF** – collision resistant hash function
  - second preimage resistant and collision resistant
other requirements

- **target collision resistance (TCR)** (Bellare-Rogaway)
  - attacker chooses $m_0$
  - attacker is given random $r$
  - attacker not able to compute $m$ such that $h(r,m) = h(r,m_0)$

- is in between (full) collision resistance and second preimage resistance

- **random oracle property**
  - output of a hash function indistinguishable from random bit string

relations between requirements

- **Theorem**: If $h$ is collision resistant then it is second preimage resistant
  - **Proof**: a second preimage is a collision.

- **Non-theorem**: If $h$ is second preimage resistant then it is preimage resistant
  - **Non-proof**:
    - suppose that for any $h_0$ one can compute a preimage $m$. Then, given $m_0$, one can certainly do that for $h_0 = h(m_0)$.
    - **problem**: to guarantee that $m \neq m_0$

- in practice:
  - collision resistant $\rightarrow$ second preimage resistant $\rightarrow$ preimage resistant
pathologic counterexamples

- if \( g : \{0,1\}^* \rightarrow \{0,1\}^n \) is collision resistant, then take
  \[
  h(m) = \begin{cases} 1 & \text{if } m \text{ has length } n, \\ 0 & \text{otherwise,} \end{cases}
  \]
  then \( h \) is collision resistant but not preimage resistant

- the identity function \( \text{id} : \{0,1\}^n \rightarrow \{0,1\}^n \) is second preimage resistant but not preimage resistant

how are hash functions used?

- asymmetric digital signature
- integrity protection
  - strong checksum
  - for file system integrity (Tripwire) or software downloads
- one-way ‘encryption’
  - for password protection
- MAC – message authentication code
  - symmetric ‘digital signature’
- confirmation of / commitment to knowledge
  - e.g. in hash chain based payment systems (‘hashcash’)
- key derivation
- pseudo-random number generation
- …
trivial (brute force) attacks

- assume: hash function behaves like random function
- preimages and second preimages can be found by random guessing search
  - search space: \( \approx n \) bits, \( \approx 2^n \) hash function calls
- collisions can be found by birthdaying
  - search space: \( \approx \frac{1}{2}n \) bits,
    \( \approx 2^{\frac{1}{2}n} \) hash function calls
- this is a big difference
  - MD5 is a 128 bit hash function
  - (second) preimage random search: \( \approx 2^{128} \approx 3 \times 10^{38} \) MD5 calls
  - collision birthday search: only \( \approx 2^{64} \approx 2 \times 10^{19} \) MD5 calls

rainbow table attack

- assume messages are taken from a fixed set
  - e.g. 8 bit printable ASCII
- define a reduction function \( \text{red} \) that transforms a hash value back into some message
- build hash chains: \( h_{i+1} = h(\text{red}(h_i)) \)
- for each chain only store e.g. every \( k \)th element
- do a one time brute force computation on all possible chains
- storage (the ‘rainbow table’) reduced by factor \( k \)
- to find one preimage only \( k \) hash calls required
- time-memory tradeoff
- used for password recovery
Merkle time-memory tradeoff

- if you have computed $2^t$ hashes, cost to find a second preimage for one of them is only $2^{n-t}$
  - trivial: sort computed hashes and do table lookups

birthday paradox

- birthday paradox
  given a set of $t \geq 10$ elements
  take a sample of size $k$ (drawn with repetition)
  in order to get a probability $\geq \frac{1}{2}$ on a collision
    (i.e. an element drawn at least twice)
  $k$ has to be $> 1.2 \sqrt{t}$
- consequence
  if $F : A \to B$ is a surjective random function
    and $\#A >> \#B$
  then one can expect a collision after about $\sqrt{\#B}$
    random function calls
proof of birthday paradox

• probability that all k elements are distinct is

\[ \prod_{i=0}^{k-1} \frac{t}{t-i} = \prod_{i=0}^{k-1} \left(1 - \frac{i}{t}\right) \leq \prod_{i=0}^{k-1} e^{-\frac{i}{t}} = e^{-\sum_{i=0}^{k-1} \frac{i}{t}} = e^{-\frac{k(k-1)}{2t}} \]

and this is > \(\frac{1}{2}\) when \(k(k-1) > (2 \log 2)t\)

\((\approx k^2)\) \((\approx 1.4t)\)

meaningful birthdaying

• random birthdaying
  – do exhaustive search on \(\frac{1}{2}n\) bits
  – messages will be ‘random’
  – messages will not be ‘meaningful’

• Yuval (1979)
  – start with two meaningful messages \(m_1, m_2\) for which you want to find a collision
  – identify \(\frac{1}{2}n\) independent positions where the messages can be changed at bitlevel without changing the meaning
    • e.g. tab \(\leftrightarrow\) space, space \(\leftrightarrow\) newline, etc.
  – do random search on those positions
implementing birthdaying

- naïve
  - store $2^{n/2}$ possible messages for $m_1$ and $2^{n/2}$ possible messages for $m_2$ and check all $2^n$ pairs
- less naïve
  - store $2^{n/2}$ possible messages for $m_1$ and for each possible $m_2$ check whether its hash is in the list
- smart: Pollard-$\rho$ with Floyd’s cycle finding algorithm
  - computational complexity still $O(2^{n/2})$
  - but only constant small storage required

Pollard-$\rho$ and Floyd cycle finding

- Pollard-$\rho$
  - iterate the hash function:
    $a_0, a_1 = h(a_0), a_2 = h(a_1), a_3 = h(a_2), \ldots$
  - this is ultimately periodic:
    - there are minimal $t, p$ such that $a_{tp} = a_t$
    - theory of random functions: both $t, p$ are of size $2^{n/2}$
- Floyd’s cycle finding algorithm
  - Floyd: start with $(a_1, a_2)$ and compute $(a_2, a_4), (a_3, a_5), (a_4, a_6), \ldots, (a_q, a_{2q})$ until $a_{2q} = a_q$;
    this happens for some $q < t + p$
parallel birthdaying

- birthdaying can easily be parallellized
  - Van Oorschot – Wiener 1999
  - kind of time-memory tradeoff
- define distinguished points by some condition
  - e.g. the first 16 bits must all be 0
- give all processors random \( a_0 \) and let them iterate until a distinguished point \( a_d \) is reached
- centrally store pairs \((a_0, a_d)\) until two \( a_d \)'s collide
  - storage: \( O(\#\text{distinguished points}) \)
- to find the actual collision you only have to recompute the two trails from the two \( a_0 \)'s
- it can be shown that the time needed with \( m \) processors is \( O\left(2^{\frac{n}{2m}}\right)\)
  - though 'total cost' remains \( O(2^{\frac{n}{2}})\)

meet in the middle attack

- assume a hash function design works with intermediate values and allows you to compute backwards halfway
  - given target hash value \( h_0 \)
  - first half: \( IV = h_1(m_1) \)
  - second half: \( h(m_1||m_2) = h_2(IV, m_2) \) where \( h_2 \) is easily invertible in the sense that \( IV = h_2^{-1}(h_0, m_2) \) can be computed for any \( m_2 \)
- then a birthday type attack on (second) preimage resistance is possible
  - birthday for collision \( h_i(m_i) = h_2^{-1}(h_0, m_2) \)
- this reduces the search space from \( 2^n \) to \( 2^{n/2} \)
  - but only for badly designed hash functions
  - note: birthdaying for two functions: iterate them alternatingly
security parameter

• security parameter \( n \): resistant against (brute force / random guessing) attack with search space of size \( 2^n \)
  – complexity of an \( n \)-bit exhaustive search
  – \( n \)-bit security level
• nowadays \( 2^{80} \) computations deemed impractical
  – security parameter 80 seen as sufficient in most cases
• but \( 2^{64} \) computations should be about possible
  – though a.f.a.i.k. nobody has done it yet
  – security parameter 64 now seen as insufficient in most cases
• in the future: security parameter 128 will be required

• for collision resistance hash length should be \( 2n \) to reach security with parameter \( n \)

hash function design - iterated compression

<table>
<thead>
<tr>
<th>data:</th>
<th>block 1</th>
<th>block 2</th>
<th>block 3</th>
<th>......</th>
<th>block n</th>
</tr>
</thead>
</table>

IHV \( \rightarrow \) IHV \( \rightarrow \) IHV \( \rightarrow \) IHV

IHV = hash

October 20, 2011
hash function designs

- other designs exist, e.g. sponge functions
- but we can’t do everything in just 2 hours

Merkle-Damgård construction

- assume that message \( m \) can be split up into blocks \( m_1, \ldots, m_s \) of equal block length \( r \)
  - most popular block length is \( r = 512 \)
- compression function: \( CF : \{0,1\}^n \times \{0,1\}^r \rightarrow \{0,1\}^n \)
- intermediate hash values (length \( n \)) as \( CF \) input and output
- message blocks as second input of \( CF \)
- start with fixed initial \( IHV_0 \) (a.k.a. \( IV = \) initialization vector)
- iterate \( CF : IHV_1 = CF(IHV_0,m_1), IHV_2 = CF(IHV_1,m_2), \ldots, IHV_s = CF(IHV_{s-1},m_s) \),
- take \( h(m) = IHV_s \) as hash value
- advantages:
  - this design makes streaming possible
  - hash function analysis becomes compression function analysis
  - analysis easier because domain of \( CF \) is finite
avoiding meet in the middle attacks

- compression function should not be invertible
- usually done by feed-forward technique
  - use input IHV also at the very end of the compression function

padding

- padding: add dummy bits to satisfy block length requirement
- non-ambiguous padding: add one 1-bit and as many 0-bits as necessary to fill the final block
  - when original message length is a multiple of the block length, apply padding anyway, adding an extra dummy block
  - any other non-ambiguous padding will work as well
Merkle-Damgård strengthening

- let padding leave final 64 bits open
- encode in those 64 bits the original message length
  - that's why messages of length $\geq 2^{64}$ are not supported
- reasons:
  - needed in the proof of the Merkle-Damgård theorem
  - prevents some attacks such as
    - trivial collisions for random $IHV$
    - now $h(IHV_0, m_1 || m_2) = h(IHV_1, m_2)$
- see next slide for more

continued

- fixpoint attack
  fixpoint: $IHV, m$ such that $CF(IHV, m) = IHV$

- long message attack
  message length $s$, so $s$ hashes precomputed, cost $2^n/s$
  Merkle time-memory tradeoff on intermediate hash values to find second preimage for one of the precomputed hashes
compression function collisions

- **collision** for a compression function: \( m_1, m_2, IHV \) such that \( CF(IHV, m_1) = CF(IHV, m_2) \)
- **pseudo-collision** for a compression function: \( m_1, m_2, IHV_1, IHV_2 \) such that \( CF(IHV_1, m_1) = CF(IHV_2, m_2) \)
- Theorem (Merkle-Damgård): If the compression function \( CF \) is pseudo-collision resistant, then a hash function \( h \) derived by Merkle-Damgård iterated compression is collision resistant.
  - Proof: easy, locate the iteration where the collision occurs
- **Note:**
  - a method to find pseudo-collisions does not lead to a method to find collisions for the hash function
  - a method to find collisions for the compression function is almost a method to find collisions for the hash function, we ‘only’ have a wrong \( IHV \)

October 20, 2011

---

the MD4 family of hash functions

- **MD4** (Rivest 1990)
  - **MD5** (Rivest 1992)
  - **HAVAL** (Zheng, Pieprzyk, Seberry 1993)
  - **SHA-0** (NIST 1993)
  - **SHA-1** (NIST 1995)
    - **SHA-224**
    - **SHA-256**
    - **SHA-384**
    - **SHA-512** (NIST 2004)

RIPEMD (RIPE 1992)
- **RIPEMD-128**
- **RIPEMD-160**
- **RIPEMD-256**
- **RIPEMD-320** (Dobbertin, Bosselaers, Preneel 1992)

October 20, 2011
design of MD4 family compression functions

message block
split into words
message expansion
input words for each step
IHV → initial state
each step updates state with an input word
final state ‘added’ to IHV
(feed-forward)

October 20, 2011

---

design details

- MD4, MD5, SHA-0, SHA-1 details:
  - 512-bit message block split into 16 32-bit words
  - state consists of 4 (MD4, MD5) or 5 (SHA-0, SHA-1) 32-bit words
  - MD4: 3 rounds of 16 steps each, so 48 steps, 48 input words
  - MD5: 4 rounds of 16 steps each, so 64 steps, 64 input words
  - SHA-0, SHA-1: 4 rounds of 20 steps each, so 80 steps, 80 input words
  - message expansion and step operations use only very easy to implement operations:
    - bitwise Boolean operations
    - bit shifts and bit rotations
    - addition modulo $2^{32}$
  - proper mixing believed to be cryptographically strong

October 20, 2011
message expansion

- **MD4, MD5** use *roundwise permutation*, for MD5:
  - \( W_0 = M_0, W_1 = M_1, \ldots, W_{15} = M_{15} \)
  - \( W_{16} = M_1, W_{17} = M_6, \ldots, W_{31} = M_{12} \) (jump 5 mod 16)
  - \( W_{32} = M_5, W_{33} = M_8, \ldots, W_{47} = M_2 \) (jump 3 mod 16)
  - \( W_{48} = M_0, W_{49} = M_7, \ldots, W_{63} = M_9 \) (jump 7 mod 16)

- **SHA-0, SHA-1** use *recursivity*
  - \( W_0 = M_0, W_1 = M_1, \ldots, W_{15} = M_{15} \)
  - SHA-0: \( W_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \) for \( i = 17, \ldots, 80 \)
  - problem: \( k \)th bit influenced only by \( k \)th bits of preceding words, so not much diffusion
  - SHA-1: \( W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \ll 1 \)
    (additional rotation by 1 bit, this is the *only* difference between SHA-0 and SHA-1)

step operations in MD4

- in each step only one state word is updated
- the other state words are *rotated* by 1
- state \((A,B,C,D)\) in step \( i \) updated to \((D,A',B,C)\), where
  - \( A' = (A + f_i(B,C,D) + W_i + K_i) \ll s_i \)
  - \( K_i, s_i \) step dependent constants,
  - \( + \) is addition mod \( 2^{32} \),
  - \( f_i \) round dependent boolean functions:
    - \( f_i(x,y,z) = xy \text{ OR } (\neg x)z \) for \( i = 1, \ldots, 16 \),
    - \( f_i(x,y,z) = xy \text{ OR } xz \text{ OR } yz \) for \( i = 17, \ldots, 32 \),
    - \( f_i(x,y,z) = x \text{ XOR } y \text{ XOR } z \) for \( i = 33, \ldots, 48 \),
  - these functions are nonlinear, balanced, and have an *avalanche effect*
step operations in MD5

- very similar to MD4
- state update:
  \[ A' = B + ((A + f_i(B,C,D) + W_i + K_i) \ll s_i) \]
  \( K_i, s_i \) chosen differently (more variation),
  one boolean function changed,
  one more boolean function \( f_i \) needed for 4th round:
  \[ f_i(x,y,z) = xz \text{ OR } y(\neg z) \text{ for } i = 17, \ldots, 32, \]
  \[ f_i(x,y,z) = y \text{ XOR } (y \text{ OR } (\neg z)) \text{ for } i = 49, \ldots, 64, \]

some constants in MD4 and MD5

- initial \( IHV: \)
  - 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476
- MD4: \( K_i = 0 \) (1\text{st} round),
  0x5a827999 (2\text{nd} round, this is \( \sqrt{2} \)),
  0x6ed9eba1 (3\text{rd} round, this is \( \sqrt{3} \))
- MD5: \( K_i = \) first 32 bits of binary value of \( |\sin(i+1)| \)
visualisation of MD5 compression

step operations in SHA-0 and SHA-1

- different constants, boolean functions used in different order
- big-endian byte ordering in stead of little-endian
- state update: from \((A, B, C, D, E)\) to \((E', A, B>>>2, C, D)\)

\[E' = (A<<<5 + f_i(B, C, D) + E + W_i + K_i) <<< s_i\]
visualisation of SHA-1 compression

RIPEMD design

- basic idea of RIPEMD:
  two parallel MD4-like compression functions
  with different constants
  and different message schedules
- RIPEMD-128 and RIPEMD-160
  - final states of two compressions added together
- RIPEMD-256 and RIPEMD-320
  - main difference with RIPEMD-128 resp. RIPEMD-160 is:
    final states of two compressions concatenated
SHA-2 family design

- SHA-224 is SHA-256 with different IV and output truncation to 224 bits
- complexity of step operation increased
  - state of 8 words \( A, B, C, D, E, F, G, H \) updated to \((T_1+T_2, A, B, C, D, E, F, G)\) where
    \[
    T_1 = H + ((E>>>6) \text{ XOR } (E>>>11) \text{ XOR } (E>>>25)) + f(E, F, G) + W_i + K_i
    \]
    \[
    T_2 = ((A>>>2) \text{ XOR } (A>>>13) \text{ XOR } (E>>>22)) + g(A, B, C)
    \]
  - extra rotations should provide much more diffusion
- SHA-384 is SHA-512 with different IV and output truncation to 384 bits
- SHA-512 uses 64-bit words, is very similar to SHA-256

October 20, 2011

42

performance comparison

- for what it’s worth

<table>
<thead>
<tr>
<th>hash function</th>
<th>MB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>217</td>
</tr>
<tr>
<td>SHA-1</td>
<td>68</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>53</td>
</tr>
<tr>
<td>SHA-256</td>
<td>44</td>
</tr>
</tbody>
</table>

43
finding fixpoints

• in the MD4 family finding fixpoints is easy
• given message $m$, the compression function is
  \[ CF(IHV,m) = E(IHV,m) + IHV \]
  (feed-forward technique) where $E(x,m)$ is invertible:
  given $y$ it’s easy to compute $x = E^{-1}(y,m)$ such that
  $y = E(x,m)$
• the fixpoint is $E^{-1}(0,m)$

differential cryptanalysis

• attacking only collision resistance
• two stages:
  – choose differential path
    • until recently done by hand
    • De Cannière-Rechberger (2006): automated for SHA-1
    • Stevens (TU/e, 2006-2009): automated for MD5
    • Stevens (CWI, 2012): new results for SHA-1
  – brute force search for message pair $m, m'$ that ‘follows the path’
differential path

- e.g. in MD5, let \( m, m' \) differ in one word \( M_{14} \) only
  - message expansion: difference propagates only to \( W_{14}, W_{25}, W_{55}, W_{68} \), so collision to be found in steps 15 – 51
- look for inner collisions
  - collision after step 26 propagates to step 35
- look for inner almost-collisions
  - prescribed small bit difference vectors may be found
- distinguish between additive differences (because of additions modulo \( 2^{32} \)) and XOR-differences
- differential path describes conditions on the inputs
- differential path is good if it has only a few conditions

finding the collision

- conditions have a certain probability
- leads to probability for the differential path
  - this should be \( \gg 2^{-64} \)
- brute force search on message pair \( m, m' \)
- all kinds of improvements and tricks are possible
Wang's attack on MD5

- **two-block collision**
  - for any input $IHV_0$, identical for the two messages
    - i.e. $IHV_0 = IHV_0'$, $\Delta IHV_0 = 0$
  - **near-collision** after first block:
    - $IHV_1 = CF(IHV_0, m_1)$, $IHV_1' = CF(IHV_0, m_1')$
      - with $\Delta IHV_1$ having only a few carefully chosen $\pm 1$s
  - full collision after second block:
    - $IHV_2 = CF(IHV_1, m_2)$, $IHV_2' = CF(IHV_1', m_2')$
    - i.e. $IHV_2 = IHV_2'$, $\Delta IHV_2 = 0$

- with $IHV_0$ the standard IV for MD5, and a third block for padding and MD-strengthening, this gives a collision for the full MD5

**example**

$$IHV_0 = \text{FE2BB52F2807AC73BE5191B597442F78}$$

$$m_1$$

CAB9E742C4B626871AB9A524846B05C1
8895FB9365E9A69F480392FF2C3B3F79
8895FB1365E9A69F480392FF2C3B3F79
41AD3406FFADB4034BDF847A4D37014F
41AD3406FFADB4034BDF847A4D37014F
8895FB1365E9A69F480392FF2C3B3F79

$$m_1'$$

CAB9E742C4B626871AB9A524846B05C1
8895FB9365E9A69F480392FF2C3B3F79
8895FB1365E9A69F480392FF2C3B3F79
41AD3406FFADB4034BDF847A4D37014F
41AD3406FFADB4034BDF847A4D37014F
8895FB1365E9A69F480392FF2C3B3F79

$$IHV_1$$

$$IHV_1'$$

$$m_2$$

5AFF7C2E5773689B3319B81564ABE7F5
89CF66C5E4FE790C8E047D36CC77B0AE
89CF66C5E4FE790C8E047D36CC77B0AE
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D

$$m_2'$$

5AFF7C2E5773689B3319B81564ABE7F5
89CF66C5E4FE790C8E047D36CC77B0AE
89CF66C5E4FE790C8E047D36CC77B0AE
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D
5D087F30B560EB8872B34D40678662D

$$IHV_2$$

$$IHV_2'$$

$461A449DCF403F04DDBADC087214F197$
new ideas in Wang’s construction

- two-block collision
- describes precise differential path
  - previous attacks described only partial paths
- set of sufficient conditions
- message modification techniques
  - modify message bits to satisfy conditions
  - speeds up collision search
Recent results on MD5 and SHA-1

- **MD4**: broken
  - Dobbertin 1995, collision found
  - Wang 2004, complexity: only $2^6$
  - Wang 2005, even a preimage attack, complexity: $2^{26}$

- **MD5**: broken
  - Den Boer-Bosselaers 1993: pseudo-collision in the compression function
  - Wang 2004, collision found, complexity: $2^{39}$
  - Klima 2006, Stevens 2006-9, complexity: $2^{16}$ (matter of seconds on a PC)

- **SHA-0**: broken
  - Biham-Chen 2004 and Joux et al. 2004, complexity: $2^{51}$

- **SHA-1**: weakened
  - Wang 2005, complexity: $2^{63}$, no collisions found yet
  - reduced to 64 steps: broken by De Cannière-Rechberger 2006, complexity: $2^{25}$
  - Stevens 2012, complexity: $2^{77}$, no collisions found yet

- **RIPEMD and HAVAL**: some versions affected / broken

---

**Complexiteiten van bekende aanvallen**

<table>
<thead>
<tr>
<th>Jaar</th>
<th>MD5</th>
<th>SHA-1</th>
<th>SHA-2(256)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identical prefix</td>
<td>Chosen prefix</td>
<td>Identical prefix</td>
</tr>
<tr>
<td>2003</td>
<td>64</td>
<td>64</td>
<td>80</td>
</tr>
<tr>
<td>2004</td>
<td>40</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>37</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>32</td>
<td>49</td>
<td>80 - ε</td>
</tr>
<tr>
<td>2007</td>
<td>25</td>
<td>42</td>
<td>61</td>
</tr>
<tr>
<td>2008</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>16</td>
<td>41</td>
<td>???</td>
</tr>
</tbody>
</table>

Cijfers voor optimale snelheid, niet voor optimale blokgrootte
Joux’ multicollision attack

- k-collision: k-tuple \( m_1, \ldots, m_k \) with \( h(m) \) all equal
- Joux (2004): \( 2^t \)-collision costs only \( t \) times as much as 2-collision

\[
\begin{array}{cccc}
IHV_0 & B_1 & IHV_1 & B_2 \\
B'_1 & & B'_2 & \quad & IHV_i & B_i
\end{array}
\]

- this is trivial, but it has interesting consequences

hash function concatenation

- let \( h_1 \) be an \( n_1 \)-bit iterative hash function, and let \( h_2 \) be an \( n_2 \)-bit hash function (not necessarily iterative)
- let \( h \) be the concatenation, i.e. \( h(m) = h_1(m) || h_2(m) \)
- naïve expectation: collision resistance security level of \( h \) is \( \frac{1}{2}(n_1 + n_2) \)-bit
- this is wrong, Joux showed that it is essentially at most \( \frac{1}{2}\max(n_1, n_2) \)-bit
- very simple argument
  - compute \( 2^t \)-collision for \( h_1 \) at cost \( t \cdot 2^{\frac{1}{2}n_1} \)
  - do birthday attack on these \( 2^t \) messages for \( h_2 \) at cost \( 2^t \)
  - collision for \( h_2 \) will be found if \( t > \frac{1}{2}n_2 \)
- total cost is \( O(n_2 \cdot 2^{\frac{1}{2}n_1} + 2^{\frac{1}{2}n_2}) \)
Joux’s preimage attack

- easy exercise: show that a preimage attack on $h = h_1 || h_2$ is possible with a security level of $\max(n_1, n_2)$-bit
- in fact the complexity is $O(n_2 2^{\frac{1}{2}n_1} + 2^{n_1} + 2^{n_2})$
- conclusion: concatenation of iterative hash functions gives almost no extra security above that of the strongest component

Kelsey-Schneier attack

- second preimage: should have cost $2^n$
- can we do better than Merkle time-memory tradeoff?
  - if you have computed $2^t$ hashes, cost to find a second preimage for one of them is only $2^{n-t}$
- Kelsey-Schneier (2006) for iterative hash functions:
  for a message of $2^t$ blocks the cost drops to $t 2^{\frac{1}{2}n+1} + 2^{n-t}$
  for many hash functions even to $3x2^{\frac{1}{2}n+1} + 2^{n-t}$
- uses expandable messages, i.e. multi-collisions of many different lengths
expansible messages

- **generic method**, starting from given $IHV_0$
  
  finding collision between message of length 1 and
  
  message of any given length $\alpha$ takes $\alpha + 2^{\alpha n + 1}$
  
  do this for $\alpha = 1, 2, 4, 8, \ldots, 2^{k-1}$ as follows:

  - choose $2^{\alpha n}$ random blocks and compute their $IHV_\alpha$'s
  - generate $2^{\alpha n}$ random messages $(IHV, m)$, i.e. such that
  
    $IHV = h(IHV, m)$
  
  - repeat the fixpoint as many times as required
  
  - **cost**: about $k 2^{\alpha n + 1}$

  this gives $2^k$ messages, all of different length covering
  
  the range from $k$ to $2^k + k - 1$, that all have the same
  
  final $IHV$ (before padding and MD-strengthening)

- **cost**: about $k 2^{\alpha n + 1}$

  - **remember**: finding fixpoints is easy in the MD4-family

---

**method with fixpoints**

- **better method** for many hash functions
  
  - when fixpoints are easy to compute, expansible
  
  messages can be found faster

  - starting from given $IHV_0$

    choose $2^{\alpha n}$ random blocks and compute their $IHV_\alpha$'s

    generate $2^{\alpha n}$ random fixpoints $(IHV, m)$, i.e. such that

    $IHV = h(IHV, m)$

    there will be a colliding $IHV = IHV_1$

    repeat the fixpoint as many times as required

  - **cost**: about $2^{\alpha n + 1}$

  - **remember**: finding fixpoints is easy in the MD4-family
how to generate second preimages

- given very long message $m$, with $2^t + t + 1$ blocks
- this gives $2^t + t + 1$ intermediate IHVs
- make an expandable message with parameter $t$
- let $IHV_{exp}$ be its output $IHV$
- find a block $b$ that connects $IHV_{exp}$ to one of the message IHVs
  - cost: $2^{n-t+1}$ (second preimage attack with time-memory tradeoff)

continued

- from the expandable message choose the proper message length to fit the length of $m$
- total cost: $t 2^{\frac{1}{2}n+1} + 2^{n-t+1}$
  - with fixpoints even $3x2^{\frac{1}{2}n+1} + 2^{n-t+1}$
- with $t = \frac{1}{2}n$ this gives second preimages at the cost of collisions
- not very realistic: with $t = 32$ for MD5 ($n = 128$) we get second preimages for messages of $2^{32}$ blocks ($= 256$ GB) in $2^{97}$ compression function calls
herding attack

- Kelsey-Kohno 2005
- a.k.a. Nostradamus attack
- commitment to bit string by publishing hash
  - Nostradamus makes claim about predictions
  - does not publish predictions, but only a hash $h_{pred}$
  - when time of predicted event has been reached, Nostradamus publishes document describing actual events, that hashes to $h_{pred}$
- attack: you can commit by a hash to a bit string before you know the string
- this is done by herding

how to herd a hash

- build a tree of depth $k$ and width $2^k$
- start with $2^k$ random IHVs
- find $2^{k-1}$ pairs of them, such that for each pair a collision is found (cost: $2^{\frac{3}{2}(n+k+1)}$)
- repeat $k$ times until one final collision is found
  - total cost: $2^{\frac{3}{2}(n+k)+2}$
• publish the final hash
• when known what string \( m_0 \) to hash, compute its hash \( IHV_{-1} \)
• make a linking block \( b \) to connect \( IHV_{-1} \) to any of the \( 2^k \) initial \( IHVs \)
  – cost: \( 2^{n-k} \) (preimage attack with time-memory tradeoff)
• path \( m_1 \) to final hash already known (in the tree)
• append suffix \( b||m_1 \) to message \( m_0 \)
• use Yuval’s trick to hide suffix in meaningful message

\[
\text{total cost of attack: } 2^{n-k} + 2^{n(k+2)} = 2^{n-k}
\]

October 20, 2011

---

faster herding

• the preimage in the herding attack is not necessary when you commit to one of a set of known messages
  – complexity drops from \( 2^{n-k} \) to \( 2^{n(k+2)} \)
repairing – message preprocessing

- repair proposals to be able to continue using MD5 and SHA-1 without changing implementations
- Szydlo-Yin 2005:
  - message whitening: use only 384 message bits per hash input, and append 128 0-bits in 32-bit words: $M_1, M_2, \ldots, M_{12}, 0,0,0,0$
  - self-interleaving: use only 256 message bits per hash input, doubling each 32-bit word in 32-bit words: $M_1, M_1, M_2, M_2, \ldots, M_8, M_8$
  - make up your own variant
- imposes many more conditions on differential paths that are probably very hard to fulfill

October 20, 2011

repairing – randomized hashing

- Halevi-Krawczyk 2005:
  - randomize input
  - random 512-bit $r$ called salt
  - change hash function $h$ to $h_r$ by
    $$h_r(M_1||\ldots||M_k) = h_r(r||M_1 \text{ XOR } r||\ldots||M_k \text{ XOR } r)$$
  - salt prepended inside so that it’s automatically signed
  - salt $r$ has to be sent / stored with the data
applications of hash collisions

- assumption: attacker can make collisions for arbitrary IHV, but he has no control over how the collisions look like; they're a few random looking 512-bit blocks
- Mikle 2004, Kaminsky 2004: use collision to change program flow
  - files good.exe and bad.exe collide, program looks for specific bit in the colliding blocks that differs in both files, and shows different behaviour
  - can mislead software integrity protection systems, e.g. Tripwire

more applications

- Daum-Lucks 2005: similar idea for PostScript documents
  - file 1:
    - have this signed by trusted party
  - file 2:
    - has identical signature
- relies on superficial inspection by signer and verifier
- fraud easily detected by code inspection of one file only
  - two complete documents in there
  - strange block of random looking data
colliding certificates

- hide collision in public key inside X.509 certificate
  - by Lenstra, Wang, de Weger (Mar. 2005)
- two different certificates with identical CA signature
  - code inspection of only one certificate reveals nothing
    - cryptographic key is random-looking anyway
- drawbacks
  - control over CA needed
  - identical user names limits possible abuse scenarios

chosen-prefix collisions

- latest development on MD5
- Marc Stevens (TU/e MSc student) 2006
  - paper by Marc, Arjen Lenstra and BdW, EuroCrypt 2007
- Marc Stevens (CWI PhD student) 2009
  - paper by Marc, Alex Sotirov, Jacob Appelbaum, David Molnar, Dag Arne Osvik, Arjen Lenstra and BdW, Crypto 2007
  - rogue CA attack
MD5: identical IV attacks

- all attacks following Wang’s method, up to recently
- MD5 collision attacks work for any starting IHV
data before and after the collision can be chosen at will
- but starting IHVs must be identical
data before and after the collision must be identical
- called random collision

MD5: different IV attacks

- new attack
  - Marc Stevens, TU/e
  - Oct. 2006
- MD5 collisions for any starting pair \( \{IHV_1, IHV_2\} \)
data before the collision needs not to be identical
data before the collision can still be chosen at will, for each of the two documents
data after the collision still must be identical
- called chosen-prefix collision
- one example produced so far
how to make chosen-prefix collisions

- random collision (Wang): two MD5 input blocks
  - 1024 bits, looking random
  - nowadays: few seconds on a PC
  - executable can be downloaded (www.win.tue.nl/hashclash)
- chosen-prefix collisions (Stevens): larger number of MD5 input blocks, depending on computation effort
  - our example: 96 bits + 8 MD5 input blocks
  - 4192 bits, still looking random
  - requires massive parallel computation
  - we used a cluster at TU/e and a grid of volunteer home computers (up to 1200 machines) running BOINC
  - peak performance 400 GigaFLOPS
  - took 6 months in total

October 20, 2011

chosen-prefix collision finding method

- chosen prefix pair
  - in our example: each consisting of 4 input blocks, the last one missing 96 bits
  - containing two different certificate owner names
- 96 bits computed by birthdaying method to prepare “smooth” pair of IV’s
  - differing only in 8 triples of bits
  - complexity: $2^{48}$
- fully automated construction of “differential paths” for MD5 compression function
  - each path is able to eliminate one triple of bit differences
  - note: original Wang construction has one manually found differential path

October 20, 2011
chosen-prefix collision in certificate

- allows X.509 certificates with identical signatures but different owner names
  - http://www.win.tue.nl/hashclash/ChosenPrefixCollisions/

- cert 1
  - Alice public key
  - coll.blk. 1
  - CA signature

- cert 2
  - Bob public key
  - coll.blk. 2
  - CA signature

- apparently higher risk
  - still control over CA needed

- drawback: complexity
  - took 6 months to find one example

- this will not be the end…
indeed that was not the end
in 2008 the ethical hackers came by

observation: commercial certification authorities still use MD5

idea: proof of concept of realistic attack as wake up call

→ attack a real, commercial certification authority

purchase a web certificate for a valid web domain
but with a “little spy” built in
prepare a rogue CA certificate with identical MD5 hash
the commercial CA’s signature also holds for the rogue CA certificate

colliding certificates using chosen-prefix collisions, 2008

<table>
<thead>
<tr>
<th>legitimate website certificate</th>
<th>rogue CA certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial number</td>
<td>serial number</td>
</tr>
<tr>
<td>commercial CA name</td>
<td>commercial CA name</td>
</tr>
<tr>
<td>validity period</td>
<td>validity period</td>
</tr>
<tr>
<td>domain name</td>
<td>rogue CA name</td>
</tr>
<tr>
<td>2048 bit RSA public key</td>
<td>1024 bit RSA public key</td>
</tr>
<tr>
<td>v3 extensions Subject = End Entity</td>
<td>v3 extensions Subject = CA</td>
</tr>
<tr>
<td>signature</td>
<td>signature</td>
</tr>
<tr>
<td></td>
<td>tumor</td>
</tr>
</tbody>
</table>
problems to be solved

predict the serial number
predict the time interval of validity
  at the same time
  a few days before
more complicated certificate structure
  “Subject Type” after the public key
small space for the collision blocks
  is possible but much more computations needed
not much time to do computations
  to keep probability of prediction success reasonable

how difficult is predicting?

time interval:
  CA uses automated certification procedure
  certificate issued exactly 6 seconds after click

<table>
<thead>
<tr>
<th>Serial Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>643006</td>
</tr>
<tr>
<td>643007</td>
</tr>
<tr>
<td>643008</td>
</tr>
<tr>
<td>643009</td>
</tr>
<tr>
<td>643010</td>
</tr>
<tr>
<td>643011</td>
</tr>
<tr>
<td>643012</td>
</tr>
<tr>
<td>643013</td>
</tr>
<tr>
<td>643014</td>
</tr>
<tr>
<td>have a guess...</td>
</tr>
</tbody>
</table>

serial number :

Nov 3 07:44:08 2008 GMT   643006
Nov 3 07:45:02 2008 GMT   643007
Nov 3 07:46:02 2008 GMT   643008
Nov 3 07:47:03 2008 GMT   643009
Nov 3 07:48:02 2008 GMT   643010
Nov 3 07:49:02 2008 GMT   643011
Nov 3 07:50:02 2008 GMT   643012
Nov 3 07:51:12 2008 GMT   643013
Nov 3 07:51:29 2008 GMT   643014
Nov 3 07:52:02 2008 GMT   have a guess...
the attack at work

estimated: 800-1000 certificates issued in a weekend
procedure:
1. buy certificate on friday, serial number S-1000
2. predict serial number S voor time T Sunday evening
3. make collision for serial number S and time T: 2 days time
4. short before T buy additional certificates until S-1
5. buy certificate on time T-6
   hope that nobody comes in between and steals our serial number S

cluster of >200
PlayStation3
game consoles
(1 PS3 = 40 PC’s)

complexity: $2^{50}$
memory: 30 GB
→ collision in 1 day
why PlayStation3s?

cell-processor on PlayStation3:
  small instruction set
  8 very fast parallel processors
    identical instruction on different data
  128 bit registers
  ideal for MD5

more modern alternatives:
  cloud (BOINC, Amazon EC2)
  grafical cards (NVidia GTX285)

result

success after 4th attempt (4th weekend)

purchased a few hundred certificates
  (promotion action: 20 for one price)
  total cost: < US$ 1000
other attack ideas for chosen-prefix collisions

- hide collision in image (not macro)
  - inside document (MS Word, Adobe pdf, ...)
- file 1:
  - document 1
  - image
  - coll.blk. 1
  - have this signed by trusted party
- file 2:
  - document 2
  - image
  - coll.blk. 2
  - has identical signature
- code inspection of one document reveals almost nothing
  - collision covers only a few pixels in the image
  - macro features not needed anymore

October 20, 2011

87

code signing example

- Win32 executable still runs normally when random bits attached to it
- assumption (example)
  - Microsoft publishes Word.exe on download site
  - comes with MD5-based signature (Authenticode)
- abuse scenario
  - attacker prepares Worse.exe (doing whatever he wants)
  - attacker computes bitstrings b1 and b2 such that
    \[ \text{MD5(Word.exe | } | b1) = \text{MD5(Worse.exe | } | b2) \]
    - we can do that!
  - attacker gets a Microsoft Authenticode signature on
    Word.exe | | b1 (same functionality as Word.exe)
  - attacker renames Worse.exe | | b2 to Word.exe and publishes
    on Microsoft’s download site

October 20, 2011
faster herding

• chosen-prefix collisions make the herding attack faster
• predict whether Ajax or Feyenoord will win their next match
  – IHV₁ = MD5-CF(IHV₀,"my prediction is: Ajax wins")
  – IHV₂ = MD5-CF(IHV₀,"my prediction is: Feyenoord wins")
  – IHV₃ = MD5-CF(IHV₀,"my prediction is: it’s a draw")
  – produce a chosen-prefix collision \( m₁, m₂ \) for \( IHV₁ \) and \( IHV₂ \):
    \( IHV₄ = MD5-CF(IHV₁,m₁) = MD5-CF(IHV₂,m₂) \)
  – produce a chosen-prefix collision \( m₃, m₄ \) for \( IHV₃ \) and \( IHV₄ \):
    \( IHV₅ = MD5-CF(IHV₃,m₃) = MD5-CF(IHV₄,m₄) \)
  – publish \( IHV₅ \) before the match
  – after the match:
    • if Ajax won, publish: “my prediction is: Ajax wins” \( || m₁ || m₄ \)
    • if Feyenoord won, publish: “my prediction is: Feyenoord wins” \( || m₂ || m₄ \)
    • if it’s a draw, publish: “my prediction is: it’s a draw” \( || m₃ \)
    • (hide suffixes e.g. in image, Yuval’s trick won’t work now)
  – only 2 chosen-prefix collisions required \( \rightarrow \) practical attack!

the “meaningful message” argument

• colliding data cannot be chosen at will, but follow from Wang’s (Stevens’) construction method
  – indistinguishable from random data
  – two colliding data differ in a few bit positions only
  \( \rightarrow \) will most probably not constitute a “meaningful message” as input
• this makes attacks more difficult
  – but not impossible, as we’ve seen
  – meaningful message argument can be weakened by hiding collisions inside the bit level structure of a document
conclusion on collisions

- at this moment, ‘meaningful’ hash collisions are
  - easy to make
  - but also easy to detect
  - still hard to abuse realistically
- with chosen-prefix collisions we come close to realistic attacks
  - especially herding
- to do real harm, second pre-image attack needed
  - real harm is e.g. forging digital signatures
  - this is not possible yet, not even with MD5

provable hash functions

- people don’t like that one can’t prove much about hash functions
- reduction to established ‘hard problem’ such as factoring is seen as an advantage
- Chaum-Van Heijst-Pfitzmann:
  - DLP is a collision problem:
    - a collision $x_1, x_2$ for $F(x) = a^x$ and $G(x) = (a^x b)^x$ solves $a^x = b$
    - let $p = 2q+1$ for $p, q$ prime, and $a, b$ generators in $\mathbb{Z}_{p^*}^*$
    - define hash function
      $h: \{0, \ldots, q-1\} \times \{0, \ldots, q-1\} \rightarrow \{0, \ldots, p-1\}$
      $h(x, y) = a^x b^y \mod p$
    - Theorem: $h$ is collision resistant if and only if DLP in $\mathbb{Z}_{p^*}^*$ is hard
provable hash functions - VSH

- Contini-Lenstra-Steinfeld 2006
- VSH – Very Smooth Hash
- collision resistance provable under assumption that a problem directly related to factoring is hard
- also DLP-variant exists
- much more efficient than Chaum-Van Heijst-Pfitzmann
- but still far from ideal
  - bad performance compared to SHA-256
  - all kinds of multiplicative relations between hash values exist

SHA-3 competition

- NIST started in 2007 an open competition for a new hash function to replace SHA-256 as standard
- more than 50 candidates in 1st round
- now 5 finalists left
- decision in 2012
literature and web resources

- Menezes-Van Oorschot-Vanstone: Handbook of Applied Cryptography, Chapter 9
  - downloadable
  - bit out of date
- Daum-Dobbertin - Chapter 109 of the Handbook of Information Security
  - pretty recent, readable
- NIST website: http://csrc.nist.gov/pki/HashWorkshop
- our website on chosen-prefix collisions: http://www.win.tue.nl/hashclash/