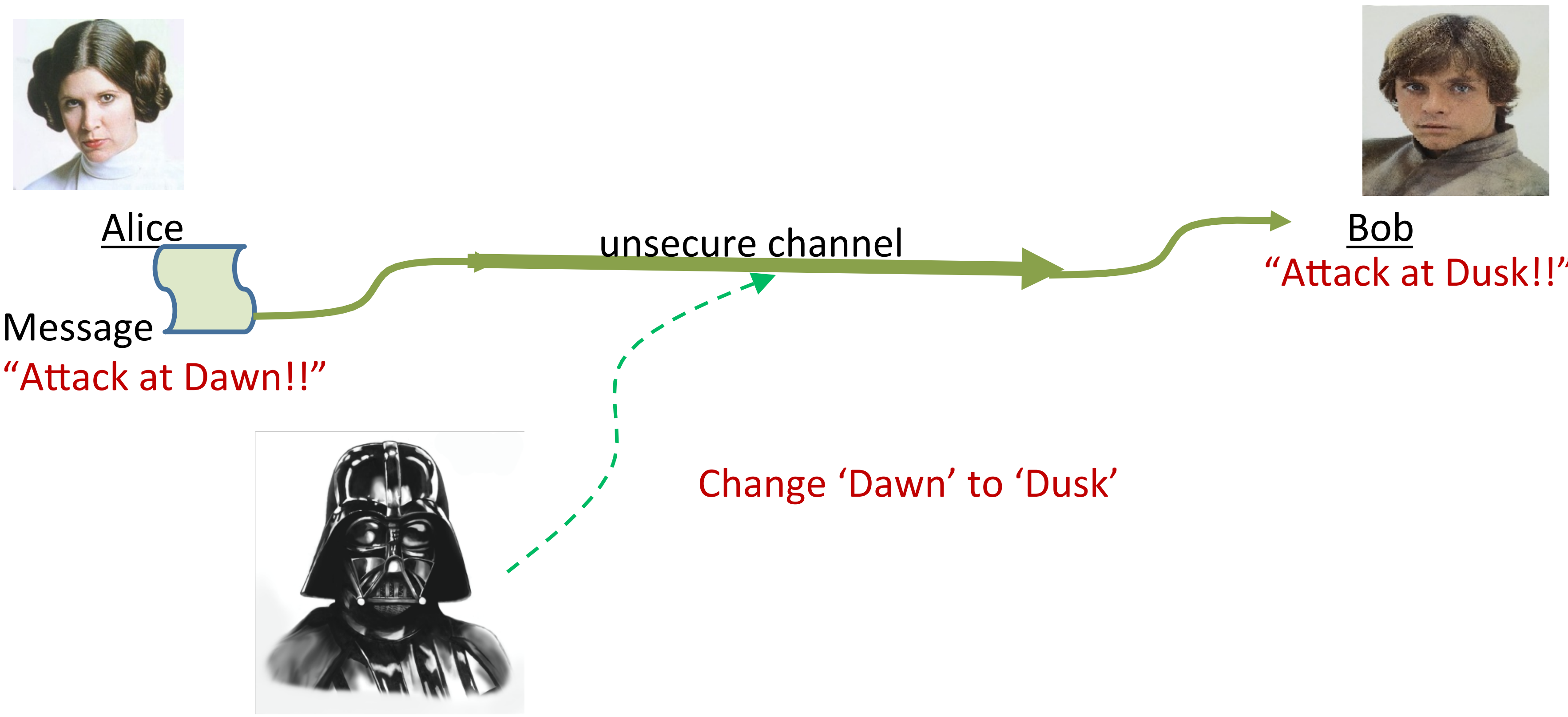
Cryptographic Hash Functions

Chester Rebeiro

IIT Madras

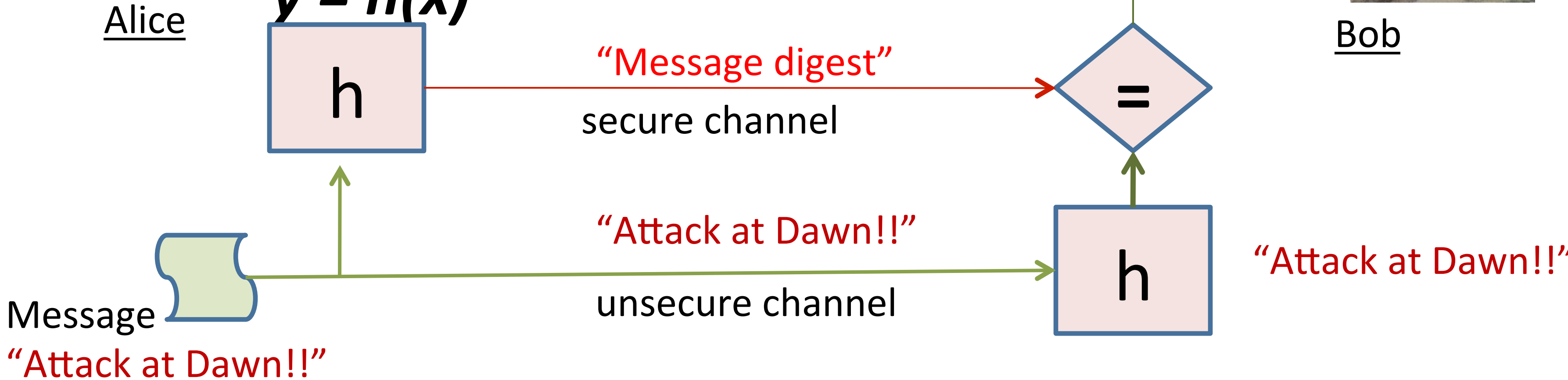
Issues with Integrity



How can Bob ensure that Alice’s message has not been modified?

Note…. We are not concerned with confidentiality here

Hashes



Alice passes the message through a hash function, which produces a fixed length message digest.

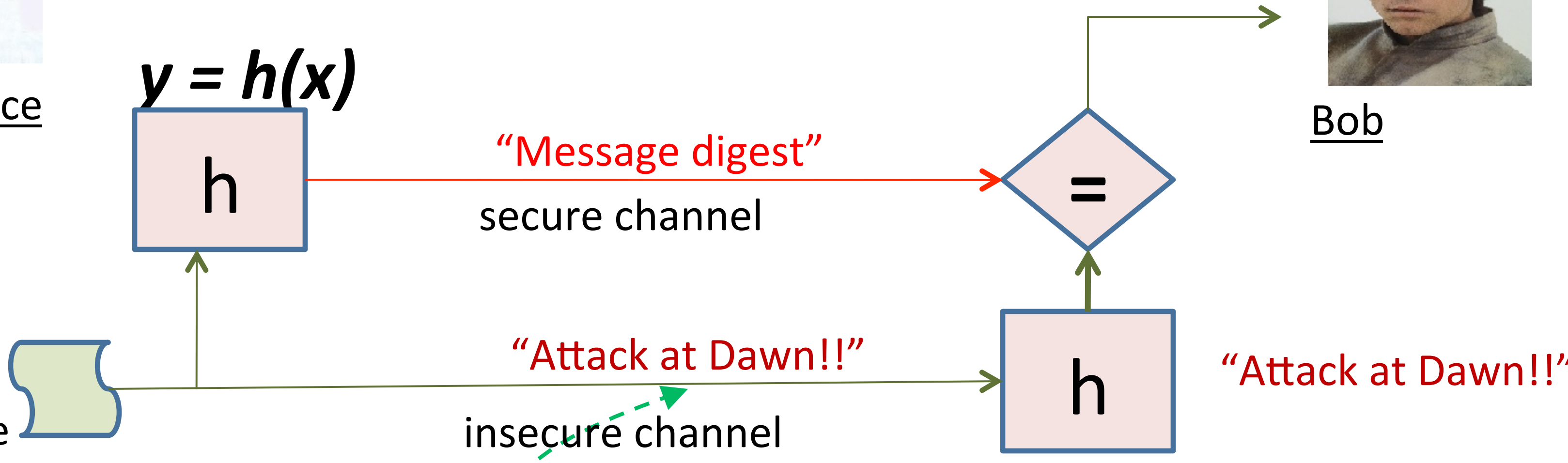
- **The message digest is representative of Alice's message.**
- **Even a small change in the message will result in a completely new message digest**
- **Typically of 160 bits, irrespective of the message size.**

Bob re-computes a message hash and verifies the digest with Alice's message digest.

Integrity with Hashes

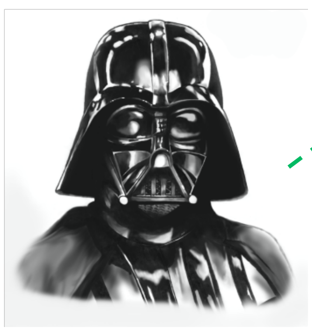
Alice

Bob



Message
"Attack at Dawn!!"

$$y = h(x)$$
$$y = h(x')$$

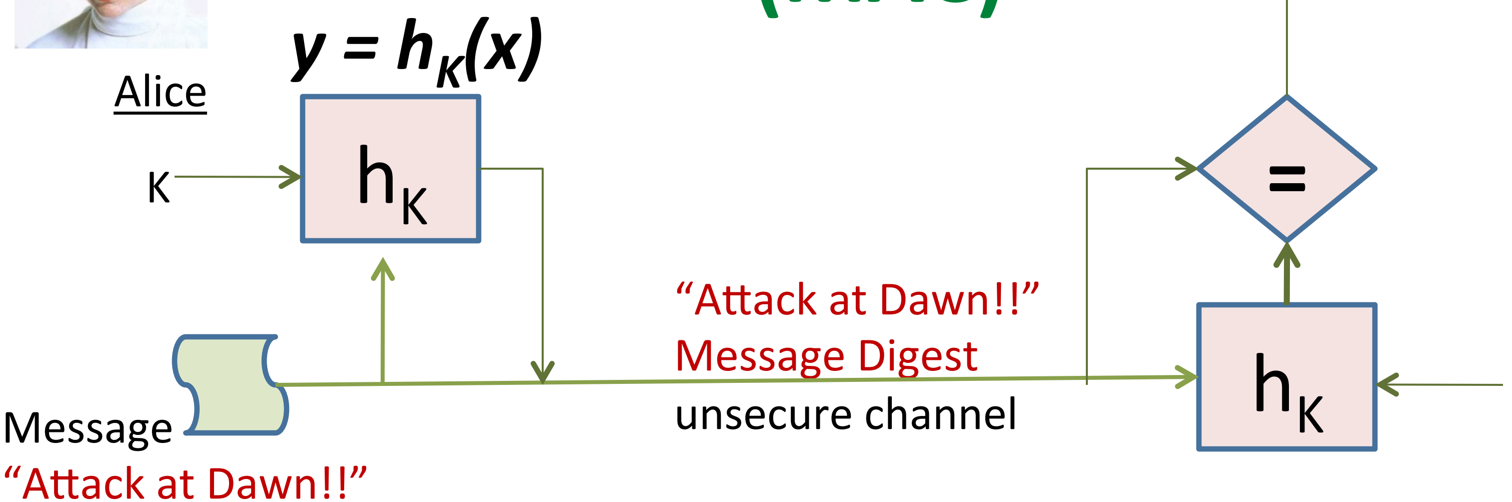


Mallory does not have access to the digest y. Her task (to modify Alice's message) is much more difficult.

If she modifies x to x', the modification can be detected unless h(x) = h(x')

Hash functions are specially designed to resist such collisions

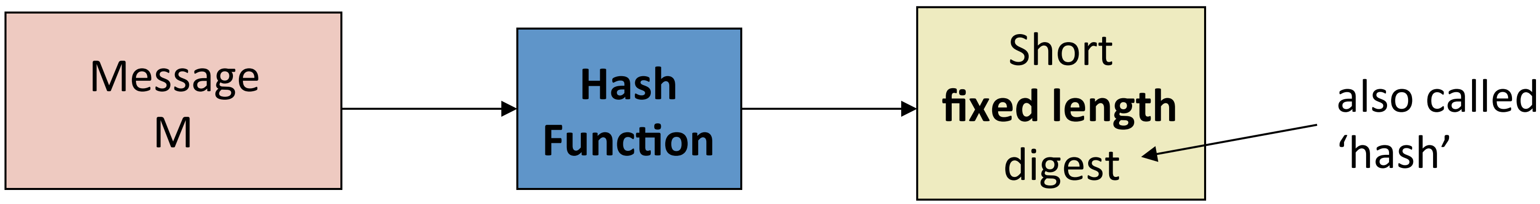
Message Authentication Codes (MAC)



MACs allow the message and the digest to be sent over an insecure channel

However, it requires Alice and Bob to share a common key

Avalanche Effect



Hash functions provide unique digests with high probability.

Even a small change in **M** will result in a new digest

SHA256(“short sentence”)

0x 0acdf28f4e8b00b399d89ca51f07fef34708e729ae15e85429c5b0f403295cc9

SHA256("The quick brown fox jumps over the lazy **dog**")

0x d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592

SHA256("The quick brown fox jumps over the lazy **dog.**")

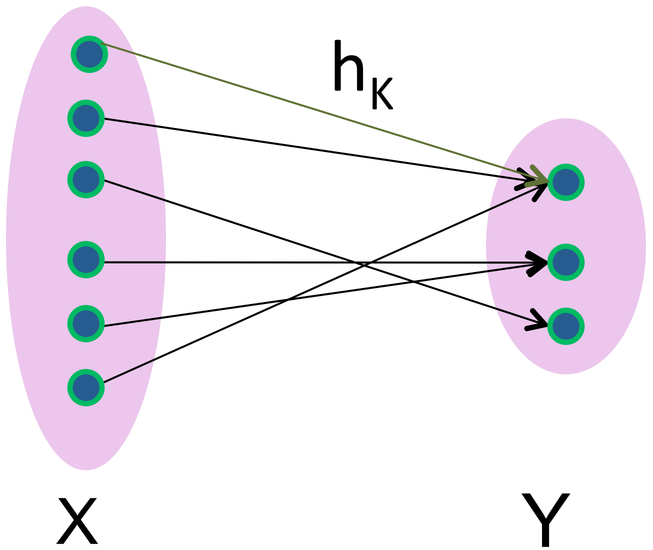
(extra period added)

0x ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c

Hash functions in Security

- Digital signatures
- Random number generation
- Key updates and derivations
- One way functions
- MAC
- Detect malware in code
- User authentication (storing passwords)

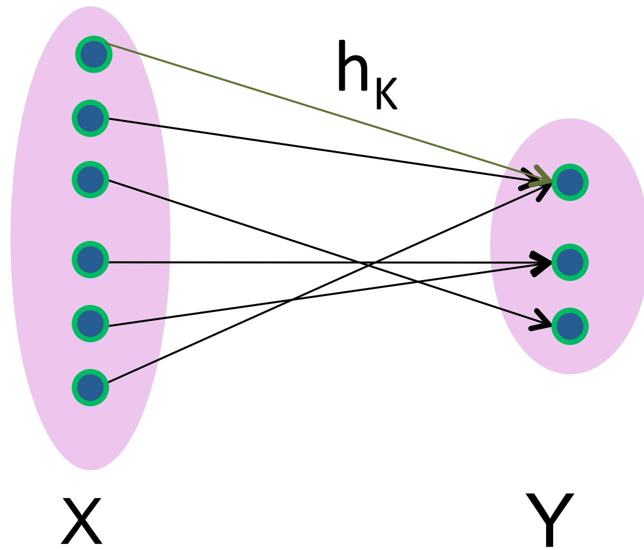
Hash Family



- The hash family is a 4-tuple defined by (X,Y,K,H)
- X is a set of messages
  (may be infinite, we assume the minimum size is at least 2|Y|)
- Y is a finite set of message digests (aka authentication tags)
- K is a finite set of keys
- Each K Ɛ K, defines a keyed hash function h_K Ɛ H

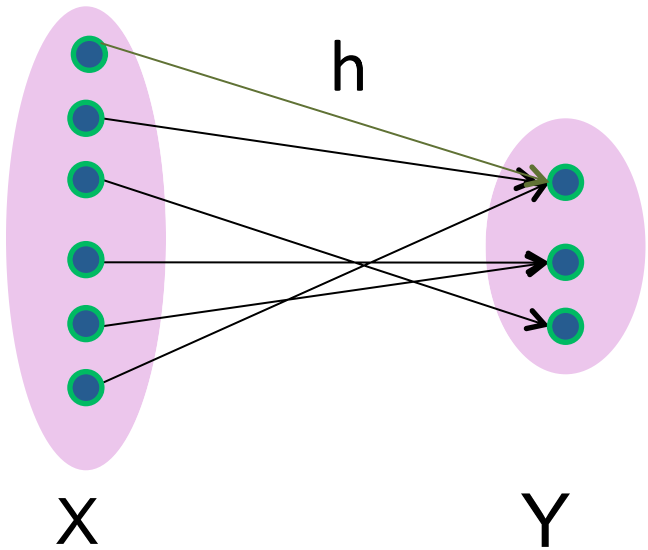
CR

Hash Family : some definitions



- Valid pair under K : (x,y) ε Xxy such that, $x = h_K(y)$
- Size of the hash family:
  is the number of functions possible from set X to set Y
  $|Y| = M$ and $|X| = N$
  then the number of mappings possible is M^N
- The collection of all such mappings are termed (N,M)-hash mapping.

Unkeyed Hash Function



- The hash family is a 4-tuple defined by (X,Y,K,H)
- X is a set of messages
  (may be infinite, we assume the minimum size is at least 2|Y|)
- Y is a finite set of message digests
- In an unkeyed hash function : |K | = 1
- We thus have only one mapping function in the family

Security Aspects of Unkeyed Hash Functions

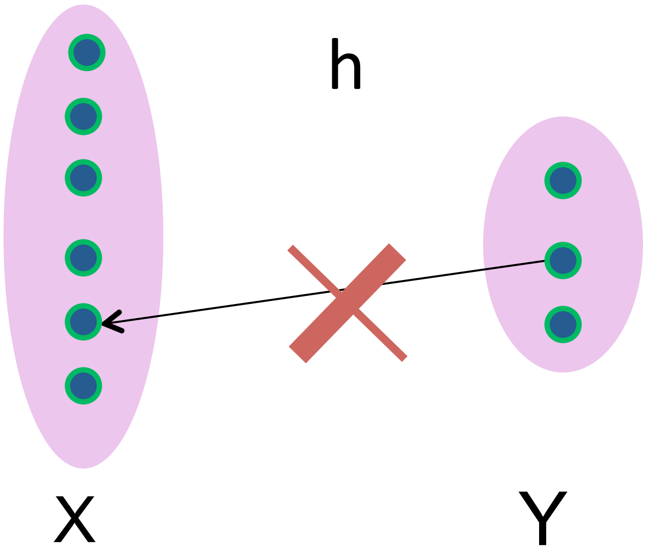
h = X → Y

y = h(x) -----> no shortcuts in computing. The only valid way if computing y is to invoke the hash function h on x

- Three problems that define security of a hash function
  * Preimage Resistance
  * Second Preimage Resistance
  * Collision Resistance

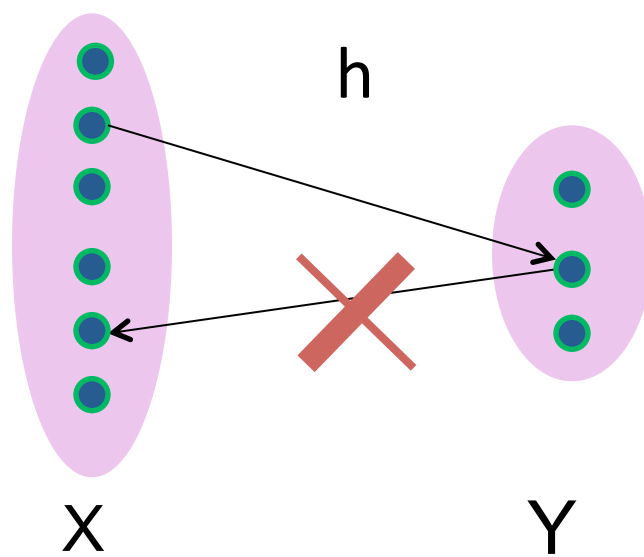
Hash function Requirement 1
Preimage Resistant

- Also know as **one-wayness problem**
- If Mallory happens to know the message digest, she should not be able to determine the message
- Given a hash function h : X →Y and an element y Ɛ Y. Find any x Ɛ X such that, h(x) = y



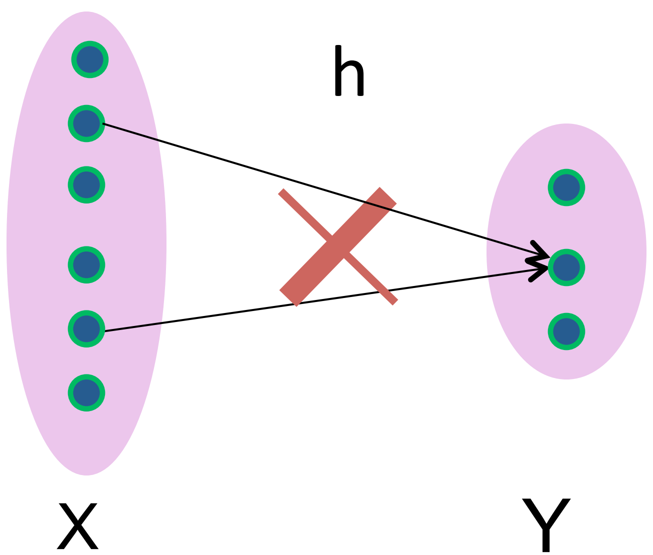
Hash function Requirement 2 (Second Preimage)

- Mallory has x and can compute h(x), she should not be able to find another message x’ which produces the same hash.
  - It would be easy to forge new digital signatures from old signatures if the hash function used weren’t second preimage resistant
- Given a hash function h : X →Y and an element x Ɛ X, find, x’ Ɛ X such that, h(x) = h(x’)



Hash Function Requirement (Collision Resistant)

- Mallory should not be able to find two messages x and x' which produce the same hash
- Given a hash function $h : X \rightarrow Y$ and an element $x \in X$, find, $x, x' \in X$ and $x \neq x'$ such that, $h(x) = h(x')$



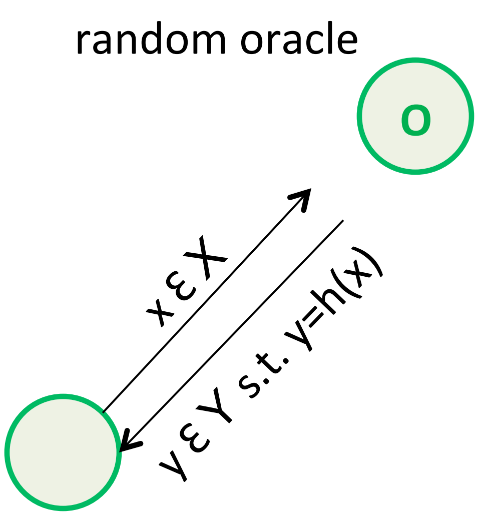
There is no collision Free hash Function but hash functions can be designed so that collisions are difficult to find.

Hash Function Requirement (No shortcuts)

- For a message m, the only way to compute its hash is to evaluate the function h(m)
- This should remain to irrespective of how many hashes we compute
  - Even if we have computed $h(m_1)$, $h(m_2)$, $h(m_3)$, ……., $h(m_{1000})$ There should not be a shortcut to compute $h(m_{1001})$
  - An example where this is not true :
    eg. Consider h(x) = ax mod n
    If $h(x_1)$ and $h(x_2)$ are known, then $h(x_1+x_2)$ can be calculated

The Random Oracle Model (to capture the ideal hash function)

- The ideal hash function should be executed by applying h on the message x.
- The RO model was developed by Bellare and Rogaway for analysis of ideal hash functions



- Let $F^{(X,Y)}$ be the set of all functions mapping X to Y .
- The oracle picks a random function h from $F^{(X,Y)}$. only the Oracle has the capability of executing the hash function.
- All other entities, can invoke the oracle with a message x ε X . The oracle will return y = h(x).

We do not know h. Thus the only way to compute h(x) is to query the oracle.

Independence Property

- Let h be a randomly chosen hash function from the set $F^{(X,Y)}$
- If x_1 Ɛ X and a different x_2 Ɛ X then

$$Pr[h(x_1) = h(x_2)] = 1/M$$

where M = |Y|

this means, the hash digests occur with uniform probability

Complexity of Problems in the RO model

- 3 problems : First pre-image, Second pre-image, Collision resistance
- We study the complexity of breaking these problems
  - Use **Las Vegas randomized algorithms**
    - A Las-Vegas algorithm may succeed or fail
    - If it succeeds, the answer returned is always correct
  - Worst case success probability
  - Average case success probability (e)
    - Probability that the algorithm returns success, averaged over all problem instances is at least e
  - **(e, Q) Las Vegas algorithm:**
    - Is an algorithm which can make Q queries to the random oracle and have an average success probability of e
      e is the average across all M^N hash functions and all possible random choices of x or y.

CR

Las Vegas Algorithm Example

- Find a person who has a birthday today in at-most Q queries

```
BirthdayToday(){
        X = set of Q randomly chosen people
        for x in X{
            if (birthday(x) == today) return x
        }
        return FAILURE;
}
```

CR

Las Vegas Algorithm Example

- Find a person who has a birthday today in at-most Q queries

```
BirthdayToday(){
        X = set of Q randomly chosen people from the universe
        for x in X{
            if (birthday(x) == today) return x
        }
        return FAILURE;
}
```

Is this the average case success?

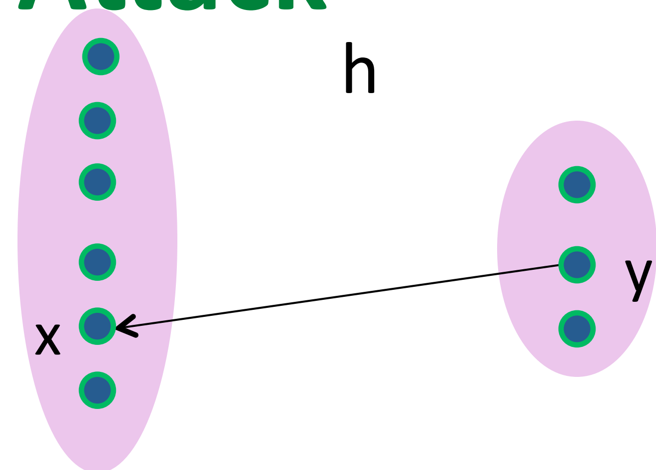
- Let E be the event that a person has a birthday today

$$\Pr \textit{ that a person does not have a birthday today is } \left(1 - \frac{1}{365}\right)$$

$$\Pr[\textit{Success in } Q \textit{ trials}] = 1 - \Pr[\textit{Failure in } Q \textit{ tries}] = 1 - \left(1 - \frac{1}{365}\right)^{Q}$$

First Preimage Attack

Problem : Given a hash y, find an x such that h(x) = y



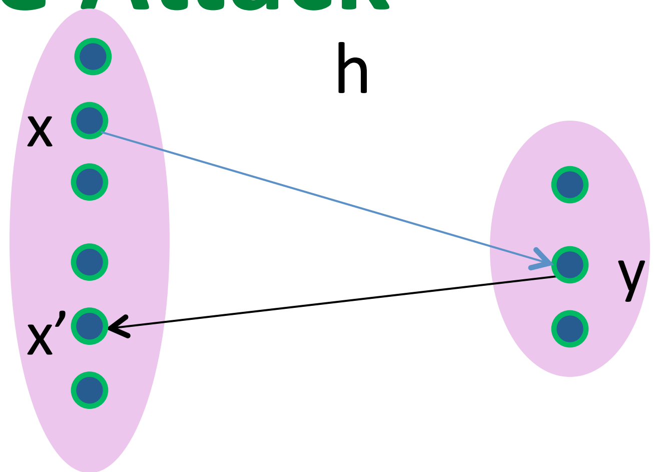
First_PreImage_Attack(h, y, Q){
    choose Q distinct values from $\mathcal{X}$ *(say* $x_1, x_2, \ldots, x_Q$*)*
    for(i=1; i<=Q; ++i){
        if ($h(x_i)$ == y) return x_i
    }
    return FAIL
}

Ideal hash function queried using the RO access

|Y| = M

$$\Pr[Success\ in\ Q\ trials\ on\ average] = 1 - \left(1 - \frac{1}{M}\right)^{Q}$$

Second Preimage Attack



Problem : Given an x, find an x’ (≠x) such that h(x’) = h(x)

Extra Oracle query

```
Second_PreImage_Attack(h, x, Q){
    choose Q-1 distinct values from X (say x_1, x_2, ...., x_{Q-1})
    y = h(x)
    for(i=1; i<=Q-1; ++i){
        if (h(x_i) == y) return x_i
    }
    return FAIL
}
```

$$\Pr[Success\, in\, Q\, trials\, on\, average] = 1 - \left(1 - \frac{1}{M}\right)^{Q-1}$$

Finding Collisions

```
Find_Collisions(h, Q){
    choose Q distinct values from X (say x_1, x_2, …., x_Q)
    for(i=1; i<=Q; ++i) y_i = h(x_i)
    if there exists (y_j == y_k) for j ≠k then return (x_j, x_k)
    return FAIL
}
```

$$Success\Pr obability(\varepsilon)\, is\, \varepsilon = 1 - \prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right)$$

CR

Birthday Paradox

- Find the probability that at-least two people in a room have the same birthday

$$Event\ A: atleast\ two\ people\ in\ the\ room\ have\ the\ same\ birthday$$

$$Event\ A': no\ two\ people\ in\ the\ room\ have\ the\ same\ birthday$$

$$\Pr[A] = 1 - \Pr[A']$$

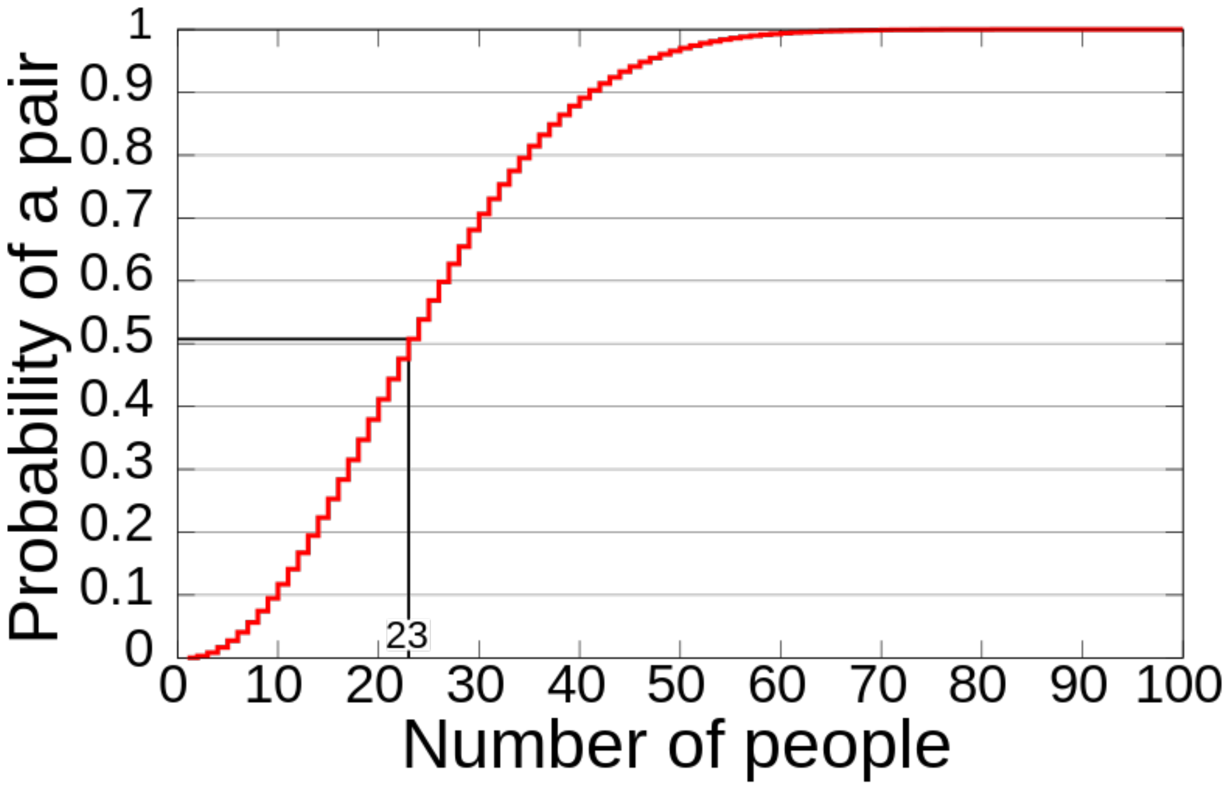
$$\Pr[A'] = 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \left(1 - \frac{3}{365}\right) \cdots\cdots \left(1 - \frac{Q-1}{365}\right)$$

$$= \prod_{i=1}^{Q-1}\left(1 - \frac{i}{365}\right)$$

$$\Pr[A] = 1 - \prod_{i=1}^{Q-1}\left(1 - \frac{i}{365}\right)$$

Birthday Paradox

- If there are 23 people in a room, then the probability that two birthdays collide is 1/2



Collisions in Birthdays to Collisions in Hash Functions

```
Find_Collisions(h, Q){
    choose Q distinct values from X (say x_1, x_2, ...., x_Q)
    for(i=1; i<=Q; ++i) y_i = h(x_i)
    if there exists (y_j == y_k) for j ≠k then return (x_j, x_k)
    return FAIL
}
```

$$Success\Pr obability(\varepsilon) is\ \varepsilon = 1 - \prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right)$$

|Y| = M

Relationship between Q, M, and success

$$Q \approx \sqrt{2M \ln \frac{1}{1-\varepsilon}}$$

$$If\ \varepsilon = 0.5\ then\ Q \approx 1.17\sqrt{M}$$

Q always proportional to square root of M.

Ɛ only affects the constant factor

Birthday Attacks and Message Digests

$$Q \approx 1.17\sqrt{M}$$

- If the size of a message digest is 40 bits
- $M = 2^{40}$
- A birthday attack would require 2^{20} queries

- Thus to achieve 128 bit security against collision attacks, hashes of length at-least 256 is required

Comparing Security Criteria

- Finding collisions is easier than solving pre-image or second preimage
- Do reductions exist between the three problems?

collision resistance → second preimage

- We can reduce collision resistance to second preimage problem

collision resitance → 2^{nd} preimage

  - i.e. If we have an algorithm to attack the 2^{nd} preimage problem, then we can solve the collision problem

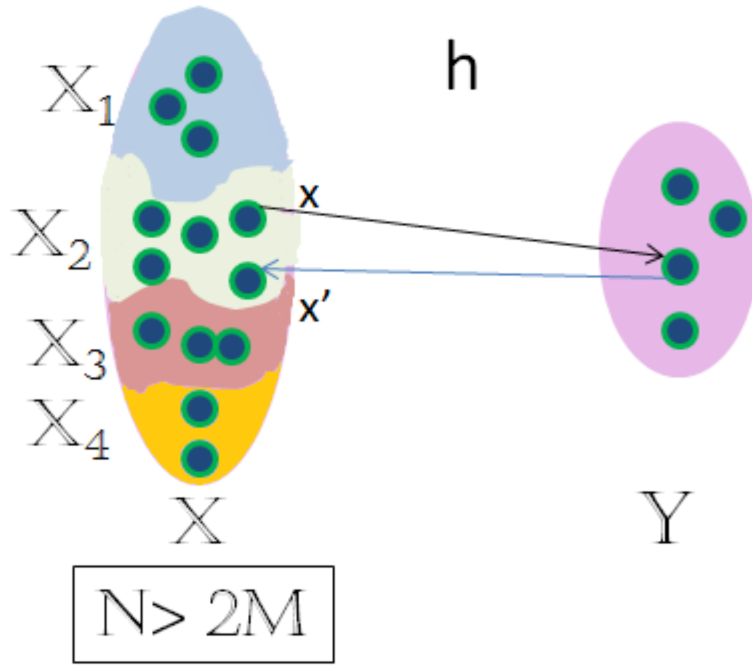
```
findCollisions1(h, Q){
    choose x randomly from X
    if(Second_PreImage_Attack(h, x, Q) == x')
        return (x,x')
    else
        return FAIL
}
```

CR

collision resistance → preimage

```
Find_Collisions2(h, Q){
    choose x randomly from X
    y = h(x)
    x' = PreImage_Attack(h, y, Q-1)
    if (x ≠ x')
        return (x,x')
    else
        return FAIL
}
```

$X = X_1 \cup X_2 \cup X_3 \cup X_4$



X_i is an equivalence class.
Each y corresponds to a partition.
The number of partitions formed is |Y|

Assume Preimage_Attack always finds the pre-image of y in Q-1 queries to the Oracle, **then, Find_Collisions2 is a (1/2, Q) Las Vegas algorithm**

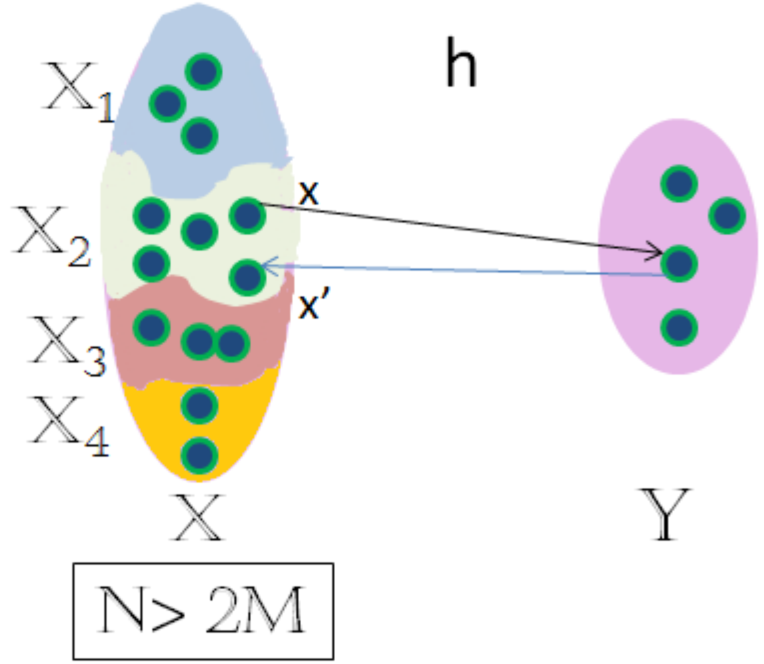
Proof

$y \in Y$ *partitions X as follows.*

$X_y = \{x \in X \mid s.t.\ h(x) = y\}$

Number of partitions of X is $|Y| = M$

$(assume\ |X| \le \frac{M}{2})$



$$\Pr[success] = \Pr[x \ne x'] = \frac{1}{N}\sum_y \sum_{X_y}\left(1 - \frac{1}{|X_y|}\right)$$

$$= \frac{1}{N}\sum_y |X_y|\left(1 - \frac{1}{|X_y|}\right)$$

$$= \frac{1}{N}\sum_y (|X_y| - 1) \quad = \frac{1}{N}(N - M)$$

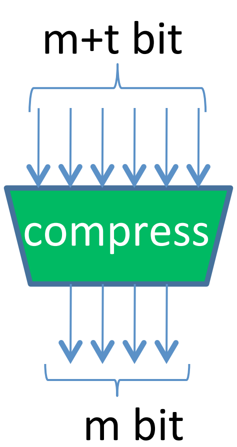
$$\ge \left(\frac{N - N/2}{N}\right) \qquad (use\ N \ge 2M)$$

$$= \frac{1}{2}$$

Iterated Hash Functions

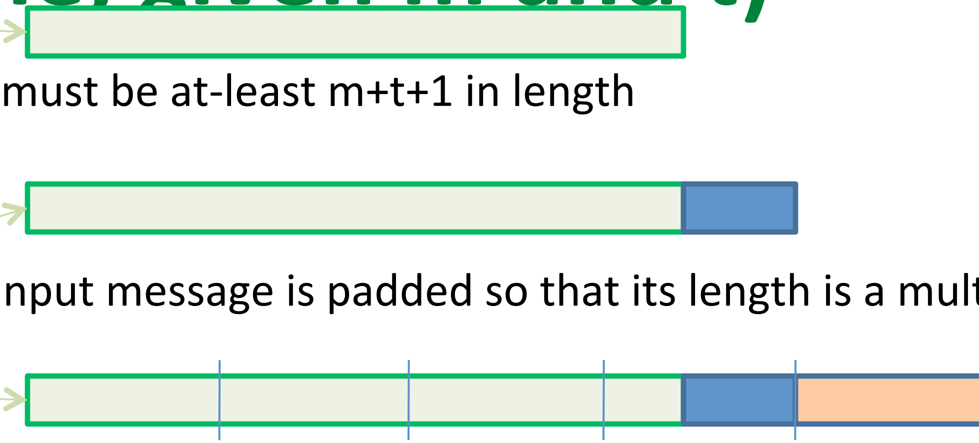
- So far, we've looked at hash functions where the message was picked from a finite set X
- What if the message is of an infinite size?
  - We use an iterated hash function
- The core in an iterated hash function is a function called compress
  - Compress, hashes from m+t bit to m bit

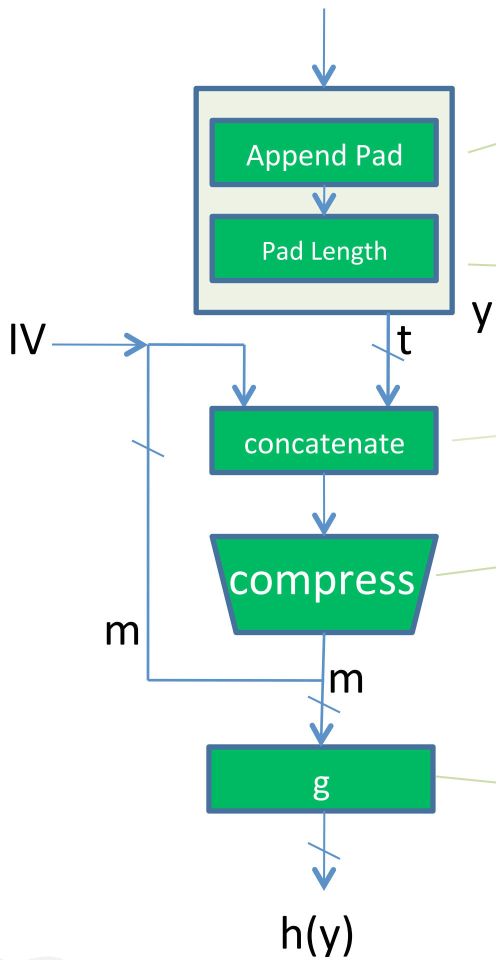
$$compress : \{0,1\}^{m+t} \longrightarrow \{0,1\}^{m}$$
$$t \geq 1$$



Iterated Hash Function (Principle, given m and t)

input message (x)
(may be of any length)

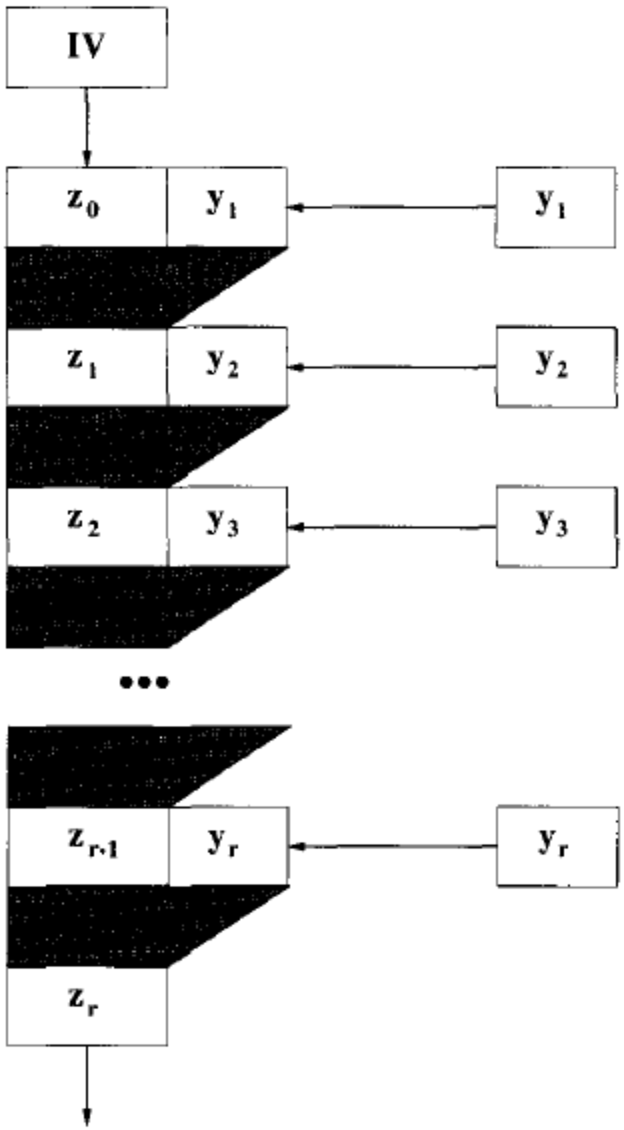




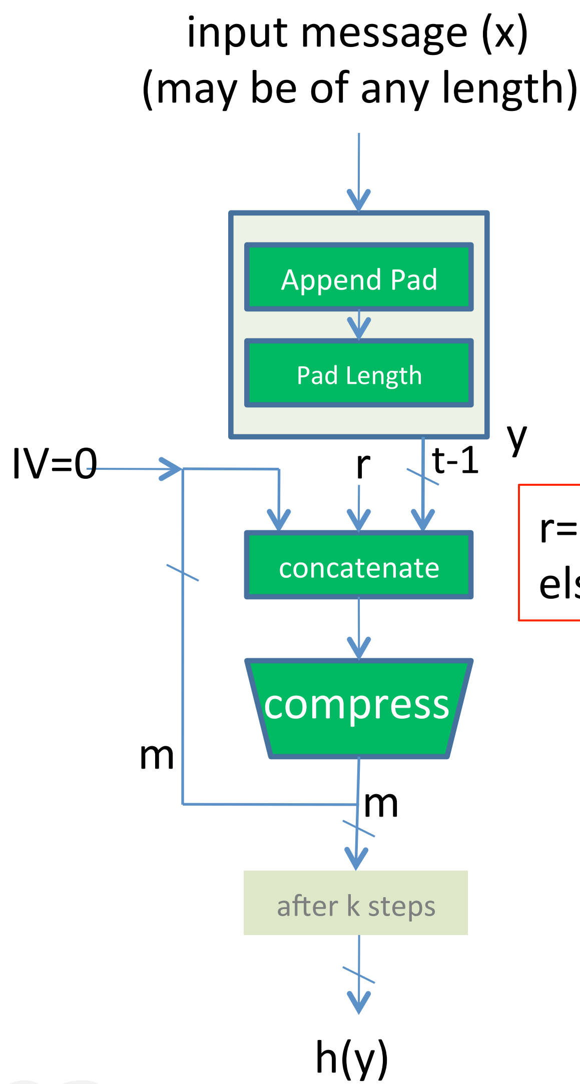
- must be at-least m+t+1 in length
- Input message is padded so that its length is a multiple of t
- Number of bits in the pad appended
- Concatinate previous m bit output with next t bit block (IV used only during initialization)
- The compress function is invoked iteratively for each t bit block in the message. For the first operation, an initialization vector is used
- After all t bit blocks are processed, there is a post processing step, and finally the hash is obtained. This step is optional.

Iterated Hash Function (Principle)

- Another perspective



Merkle-Damgard Iterated Hash Function



r=0 for the first iteration
else r=1

$$h : \{0,1\}^{m+t} \rightarrow \{0,1\}^{m}$$

$$X = \bigcup_{i=m+t+1}^{\infty} \{0,1\}^{i}$$

Itrated hash function construction
That uses a compress function h

If h is collision resistant then the Merkle Damgard construction is collision resistant

Merkle-Damgard Iterated Hash Function

Algorithm : MERKLE-DAMGÅRD(x)

external compress
comment: compress : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$, where $t \geq 2$

$n \leftarrow |x|$

Message length

$k \leftarrow \lceil n/(t-1) \rceil$

$d \leftarrow k(t-1) - n$

k :Num of blocks of in x. Each block has length t-1
Note that t cannot be = 1

for $i \leftarrow 1$ **to** $k-1$
  do $y_i \leftarrow x_i$

$y_k \leftarrow x_k \,||\, 0^d$

Apply padding

$y_{k+1} \leftarrow$ the binary representation of d

Append d

$z_1 \leftarrow 0^{m+1} \,||\, y_1$

IV is 0^m

Amount of padding required to make message a multiple of t-1

$g_1 \leftarrow$ **compress**(z_1)

for $i \leftarrow 1$ **to** k

$$\textbf{do} \begin{cases} z_{i+1} \leftarrow g_i \,||\, 1 \,||\, y_{i+1} \\ g_{i+1} \leftarrow \textbf{compress}(z_{i+1}) \end{cases}$$

$h(x) \leftarrow g_{k+1}$

return $(h(x))$

On Merkle-Damgard Construction

Theorem: If the compress function is collision resistant then the Merkle-Damgard construction is collision resistant

Proof: We show the contra-positive…

If the Merkle-Damgard construction results in a collision then the compress function is NOT collision resistant

Merkle-Damgard Construction is Collision Resistant (Proof)

- Assume we have two message x and x' which result in the same hash.
- Proof proceeds by considering 2 cases:

(1) $\boxed{|x| \neq |x'| \bmod (t-1)}$

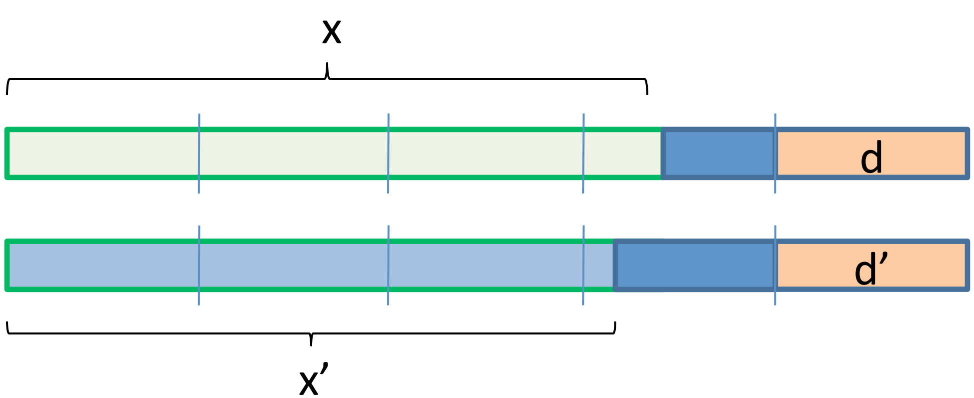
(2) $\boxed{|x| = |x'| \bmod (t-1)}$

(2a) $\boxed{|x| = |x'|}$

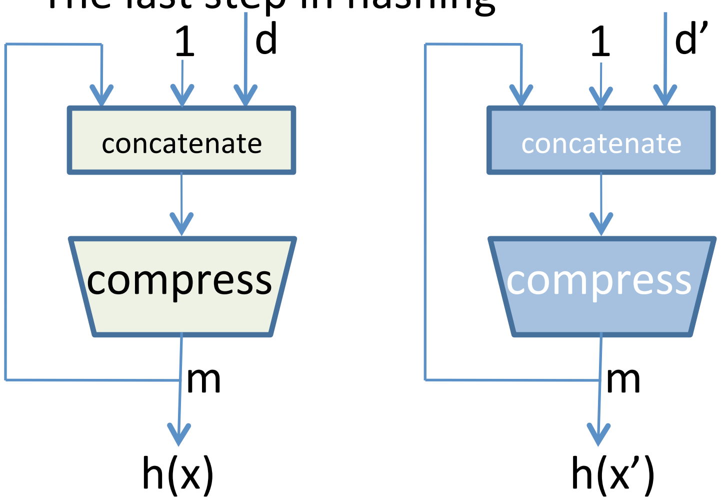
(2b) $\boxed{|x| \neq |x'|}$

Case 1 $|x| \neq |x'| \bmod (t-1)$

- This means that the padding (resp. d and d’) applied to x and x’ is different (i.e. d ≠ d’)



The last step in hashing



If h(x) = h(x’) then
compress(xx||1||d) = compress(xx||1||d’)

Since d≠ d’, we have a collision in compress.

Case 1 formally : $|x| \neq |x'| \bmod (t-1)$

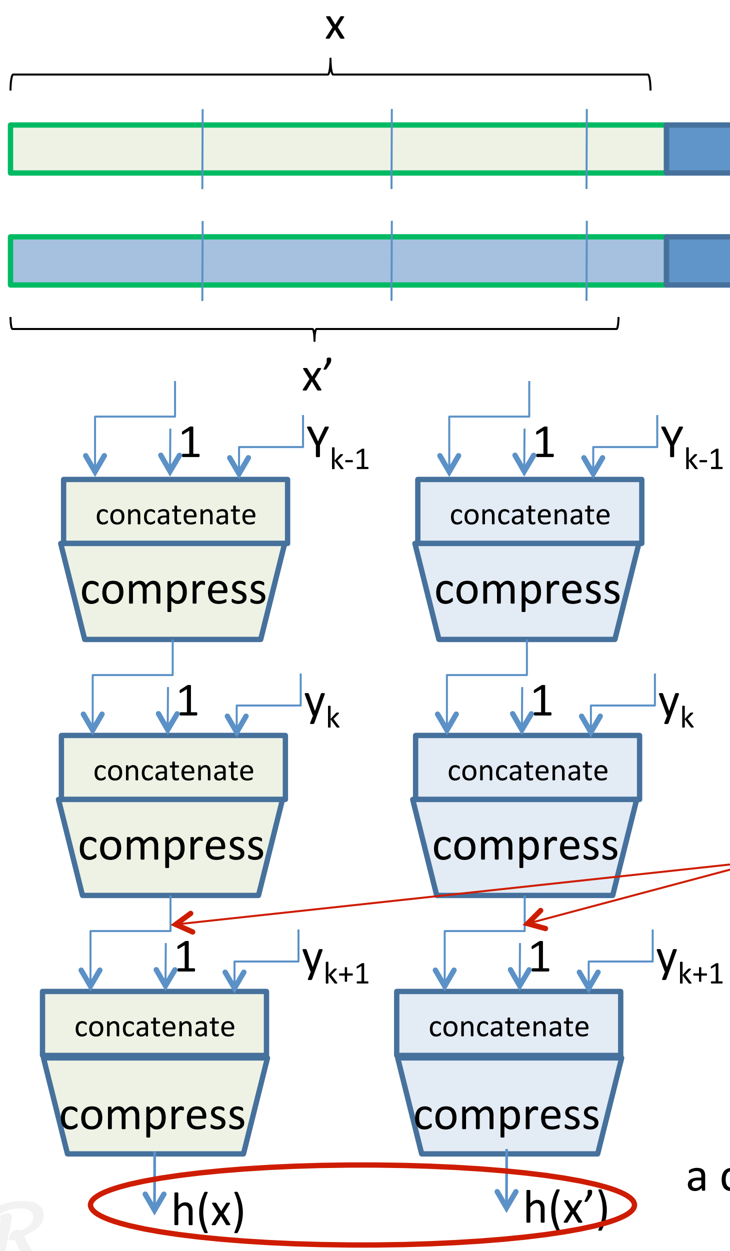
case 1: $|x| \not\equiv |x'| \pmod{t-1}$.

Here $d \neq d'$ and $y_{k+1} \neq y'_{\ell+1}$. We have

$$\begin{aligned}
\textbf{compress}(g_k \,\|\, 1 \,\|\, y_{k+1}) &= g_{k+1} \\
&= h(x) \\
&= h(x') \\
&= g'_{\ell+1} \\
&= \textbf{compress}(g'_\ell \,\|\, 1 \,\|\, y'_{\ell+1}),
\end{aligned}$$

which is a collision for **compress** because $y_{k+1} \neq y'_{\ell+1}$.

Case 2a : $|x| = |x'| \bmod (t-1) \;\; and \;\; |x| = |x'|$

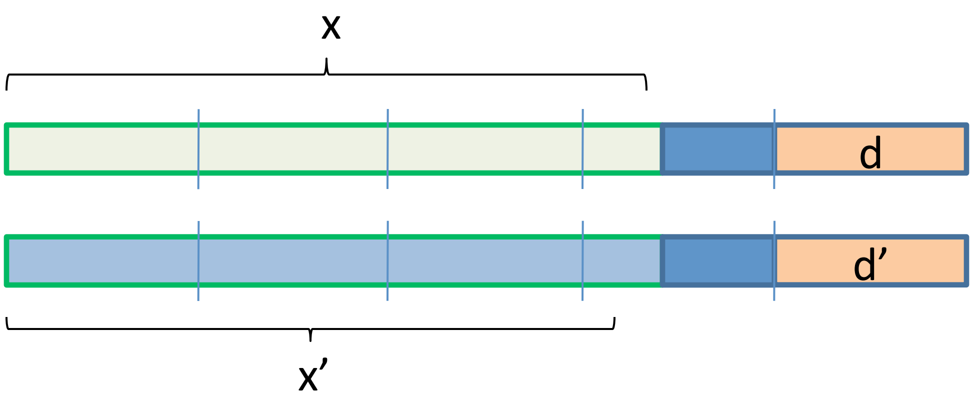


In this case, padding in x and x’ are the same. Hence d = d’.
… can’t use the old trick ☹

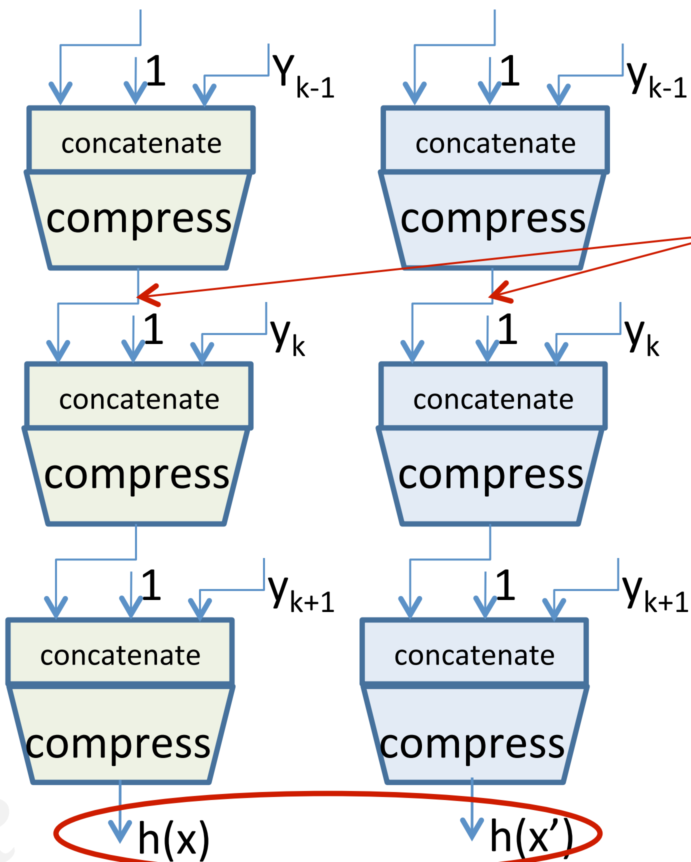
These may or may not collide.
If they collide, we are done : we have shown a collision in compress. If they don’t collide we look at the previous iteration

a collision here

Case 2a : $|x| = |x'| \bmod (t-1) \;\; and \;\; |x| \neq |x'|$



In this case, padding in x and x' are the same. Hence d = d'.
… can't use the old trick ☹



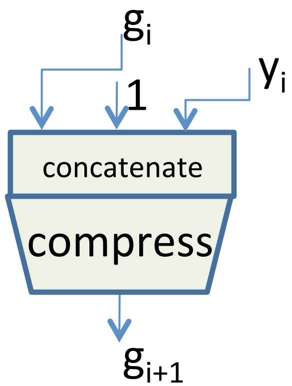
These may or may not collide.
If they collide, we are done :
We have shown a collision in compress.
If they don't collide we look at the previous iteration

We continue this back tracking, until we find a collision. We will definitely find a collision at some point because x ≠ x'.

Case 2a formally : $|x| = |x'| \bmod (t-1)$ and $|x| = |x'|$

Here we have $k = \ell$ and $y_{k+1} = y'_{k+1}$. We begin as in case 1:

$$\begin{aligned}
\mathbf{compress}(g_k \parallel 1 \parallel y_{k+1}) &= g_{k+1} \\
&= h(x) \\
&= h(x') \\
&= g'_{k+1} \\
&= \mathbf{compress}(g'_k \parallel 1 \parallel y'_{k+1}).
\end{aligned}$$



If $g_k \neq g'_k$, then we find a collision for **compress**, so assume $g_k = g'_k$. Then we have

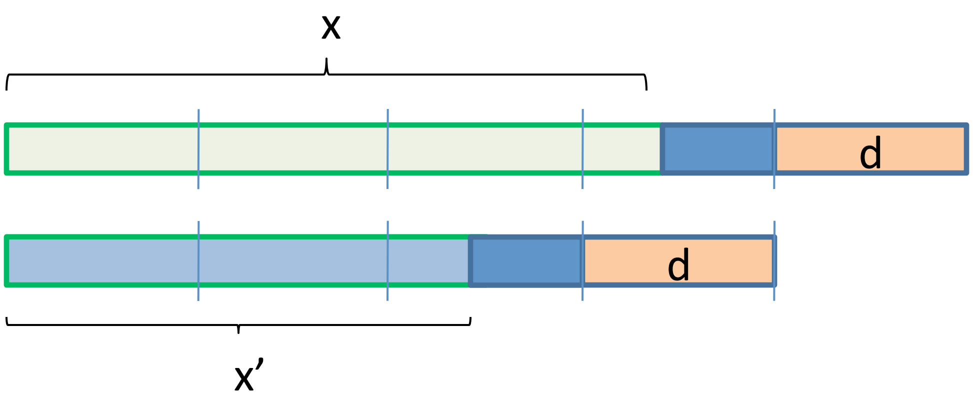
$$\begin{aligned}
\mathbf{compress}(g_{k-1} \parallel 1 \parallel y_k) &= g_k \\
&= g'_k \\
&= \mathbf{compress}(g'_{k-1} \parallel 1 \parallel y'_k).
\end{aligned}$$

Either we find a collision for **compress**, or $g_{k-1} = g'_{k-1}$ and $y_k = y'_k$. Assuming we do not find a collision, we continue working backwards, until finally we obtain

$$\begin{aligned}
\mathbf{compress}(0^{m+1} \parallel y_1) &= g_1 \\
&= g'_1 \\
&= \mathbf{compress}(0^{m+1} \parallel y'_1).
\end{aligned}$$

but $y_1 = y_1'$ implies $x = x'$.
which is a contradiction.

Case 2b : $|x| = |x'| \bmod (t-1)$ and $|x| \neq |x'|$

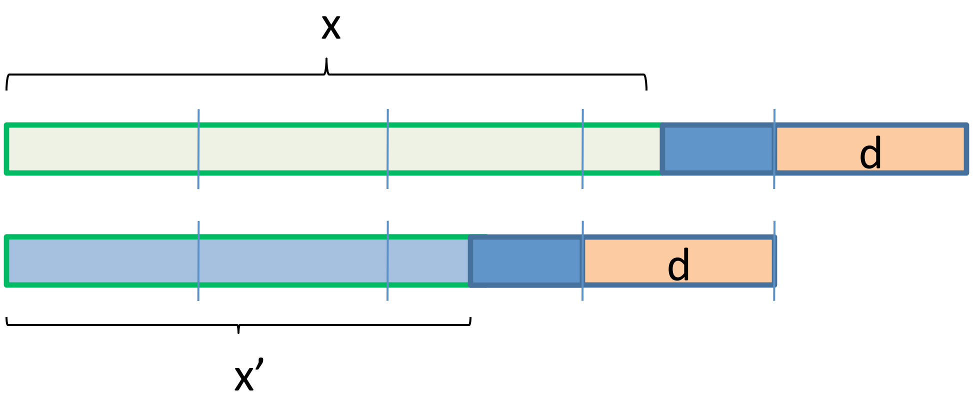


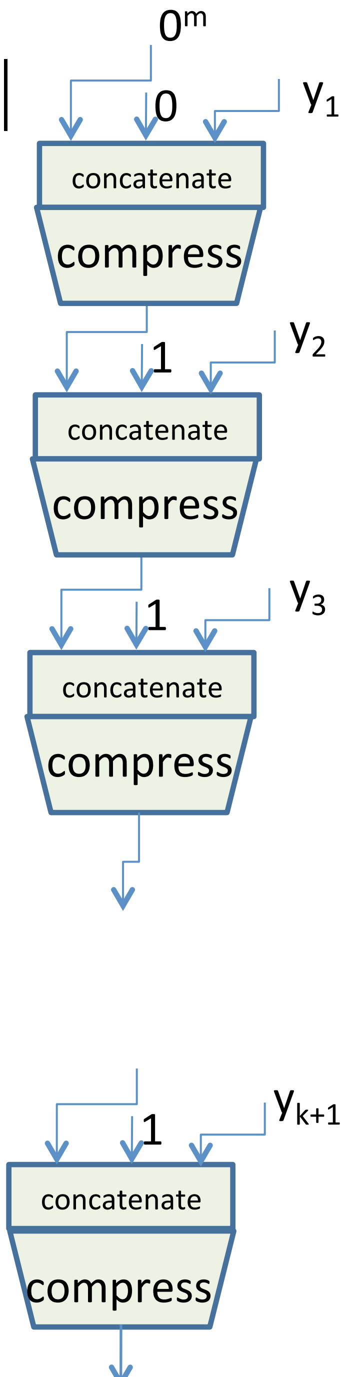
Note here that d=d' even though lengths of the messages are not the same.

In most cases, the proof would proceed similar to case 2a.

But there is a cornercase.

Case 2b : $|x| = |x'| \bmod (t-1) \;\; and \;\; |x| \neq |x'|$





- The corner case: x = (x''|x')
  back tracking in such as case will not help find a collision
- Handling this case:

  the inserted bit r
  (r=0 for the 1st round, else r=1)

Case 2b formally : $|x| = |x'| \bmod (t-1) \;\; and \;\; |x| \neq |x'|$

case 2b: $|x| \neq |x'|$.

Without loss of generality, assume $|x'| > |x|$, so $\ell > k$. This case proceeds in a similar fashion as case 2a. Assuming we find no collisions for **compress**, we eventually reach the situation where

$$\begin{aligned} \mathbf{compress}(0^{m+1} \parallel y_1) &= g_1 \\ &= g'_{\ell-k+1} \\ &= \mathbf{compress}(g'_{\ell-k} \parallel 1 \parallel y'_{\ell-k+1}). \end{aligned}$$

But the $(m+1)$st bit of

$$0^{m+1} \parallel y_1$$

is a 0 and the $(m+1)$st bit of

$$g'_{\ell-k} \parallel 1 \parallel y'_{\ell-k+1}$$

is a 1. So we find a collision for **compress**.

Merkle-Damgard-2
(for the case when t=1)

Algorithm : MERKLE-DAMGÅRD2(x)

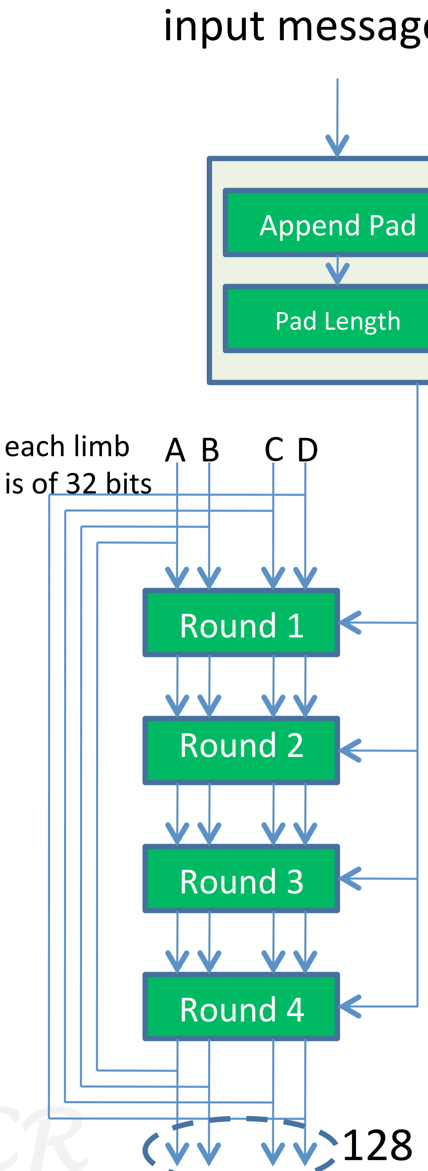
external **compress**
comment: **compress** : $\{0,1\}^{m+1} \rightarrow \{0,1\}^m$

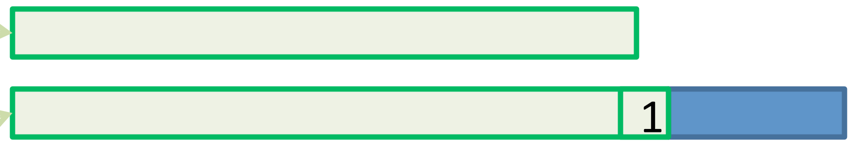
$n \leftarrow |x|$
$y \leftarrow 11 \,\|\, f(x_1) \,\|\, f(x_2) \,\|\, \cdots \,\|\, f(x_n)$
denote $y = y_1 \,\|\, y_2 \,\|\, \cdots \,\|\, y_k$, where $y_i \in \{0,1\}, 1 \leq i \leq k$
$g_1 \leftarrow$ **compress**$(0^m \,\|\, y_1)$
for $i \leftarrow 1$ **to** $k-1$
  do $g_{i+1} \leftarrow$ **compress**$(g_i \,\|\, y_{i+1})$
return (g_k)

Hash Functions in Practice

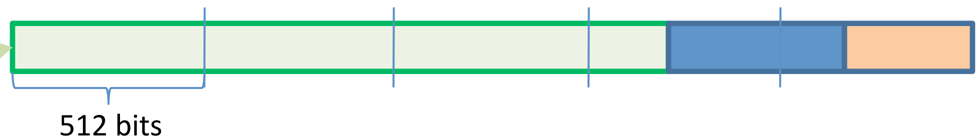
- MD5
- NIST specified “secure hash algorithm”
  - SHA0 : published in 1993. 160 bit hash.
    - There were unpublished weaknesses in this algorithm
    - The first published weakness was in 1998, where a collision attack was discovered with complexity 2^{61}
  - SHA1 : published in 1995. 160 bit hash.
    - SHA0 replaced with SHA1 which resolved several of the weaknesses
    - SHA1 used in several applications until 2005, when an algorithm to find collisions with a complexity of 2^{69} was developed
    - In 2010, SHA1 was no longer supported. All applications that used SHA1 needed to be migrated to SHA2
  - SHA2 : published in 2001. Supports 6 functions: 224, 256, 384, 512, and two truncated versions of 512 bit hashes
    - No collision attacks on SHA2 as yet. The best attack so far assumes reduced rounds of the algorithm (46 rounds)
  - SHA3 : published in 2015. Also known as Kecchak

MD5

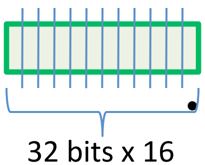




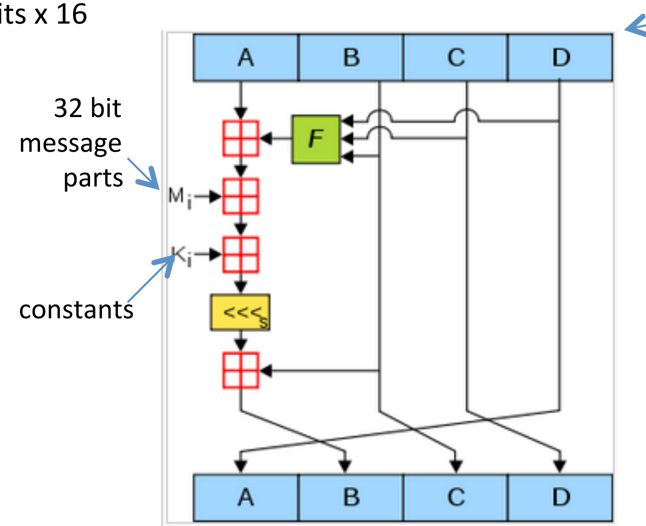
- Appended with 1 and then 0s so that length is a multiple of 512 – 64 = 448



- Message length appended (in 64 bits) and split into blocks of 512 bits



- Each round has 16 similar operations of this **modified Feistel form**



round operations

round 1 $F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$

round 2 $G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$

round 3 $H(B,C,D) = B \oplus C \oplus D$

round 4 $I(B,C,D) = C \oplus (B \vee \neg D)$

Collisions in MD5 (Timeline)

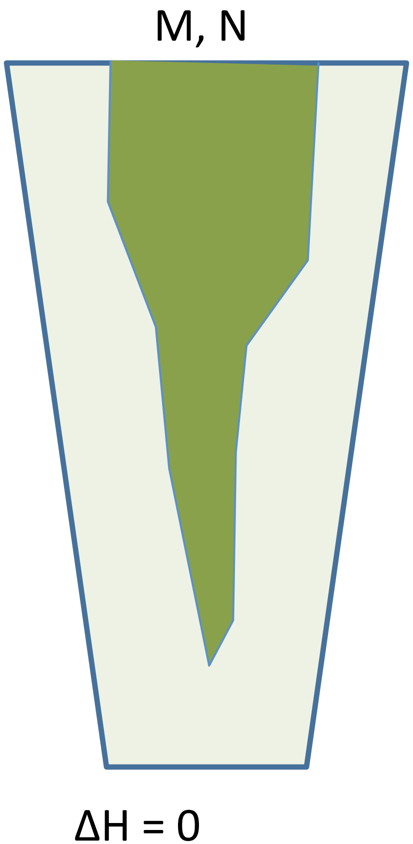
- A birthday attack on MD5 has complexity of 2^{64}
- Small enough to brute force collision search
- 1996, collisions on the inner functions of MD5 found
- 2004, collisions demonstrated practically
- 2007, chosen-prefix collisions demonstrated

  Given two different prefixes p1, p2 find two appendages m1 and m2 such that hash(p1 || m1) = hash(p2 || m2)

- 2008, rogue SSL certificates generated
- 2012, MD5 collisions used in cyberwarfare
  - **Flame malware** uses an MD5 prefix collision to fake a Microsoft digital code signature

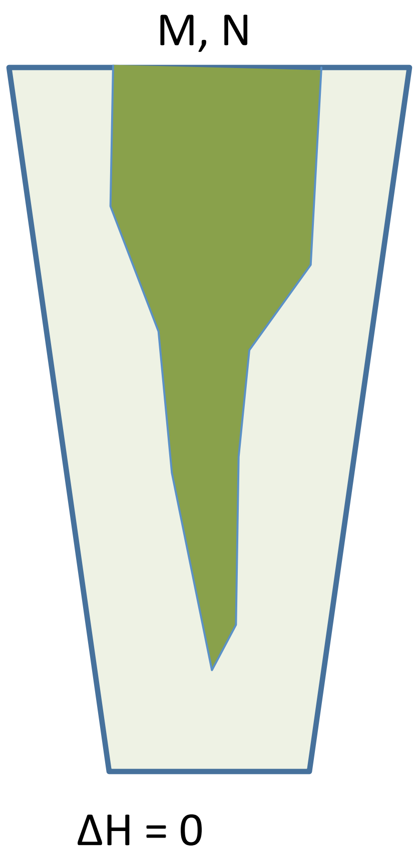
MD5 Collisions demos : http://www.mscs.dal.ca/~selinger/md5collision/

Collision attack on MD5 like hash functions



- Analyze differential trails
- A bit different from block ciphers
  - No secret key involved
  - We can choose M and N as we want
- We have a valid attack if probability of trail is $P > 2^{-N/2}$

Collision attack on MD5 like hash functions



Wang and Yu made it possible to find two pairs of blocks (m_i, m_{i+1}) and (n_i, n_{i+1}) such that

$F(F(s, m_i), m_{i+1}) = F(F(s, n_i), n_{i+1})$

Where s is some state of the hash function (can be anything)

The method makes it possible to construct two strings

$m_0, m_1, m_2, \ldots.. m_i, m_{i+1}, \ldots\ldots.. m_k,$

$m_0, m_1, m_2, \ldots.. n_i, n_{i+1}, \ldots\ldots.. m_k,$

which have the same MD5 hash.

Example of an MD5 collision

Block 1

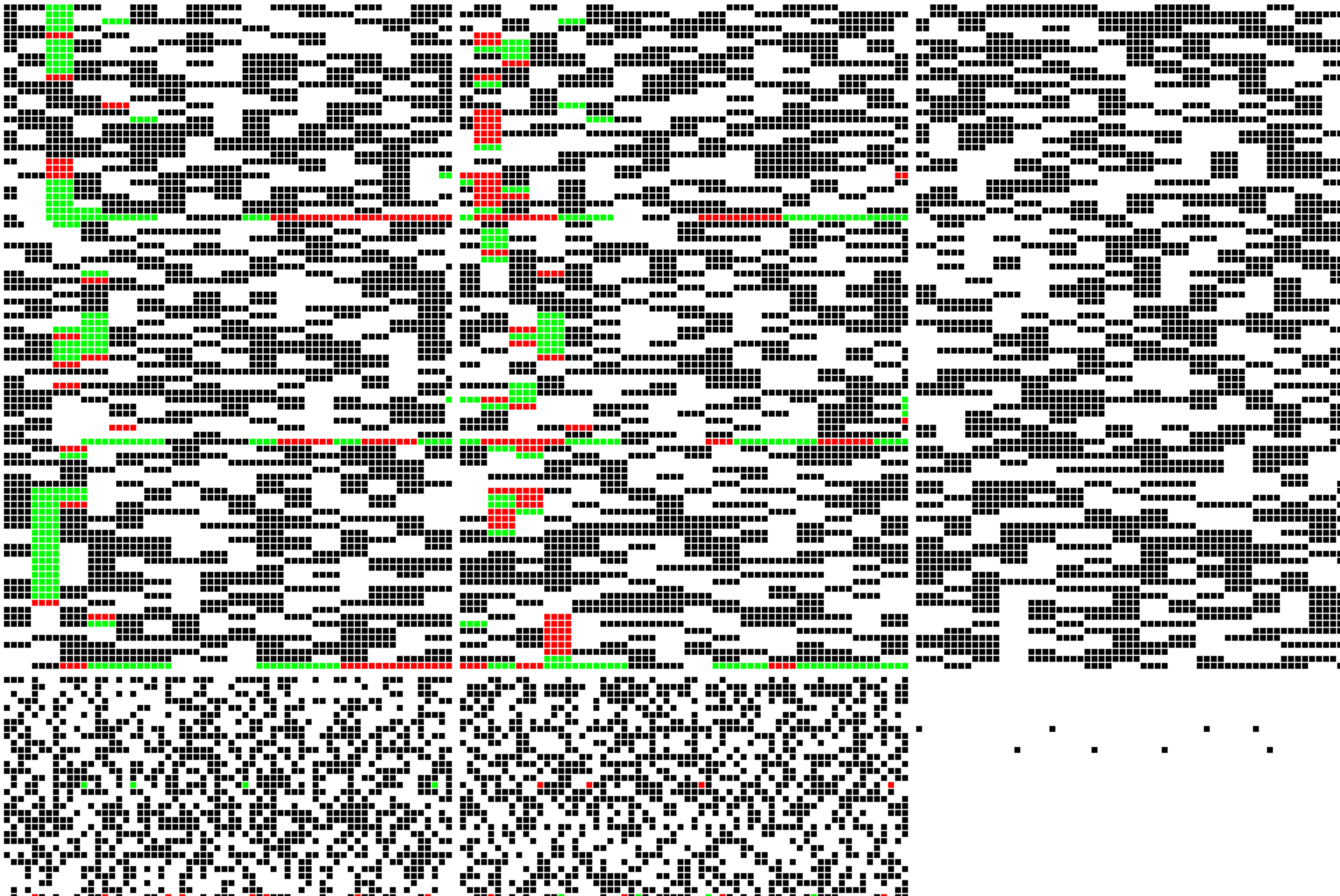
```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

Block 2

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70
```

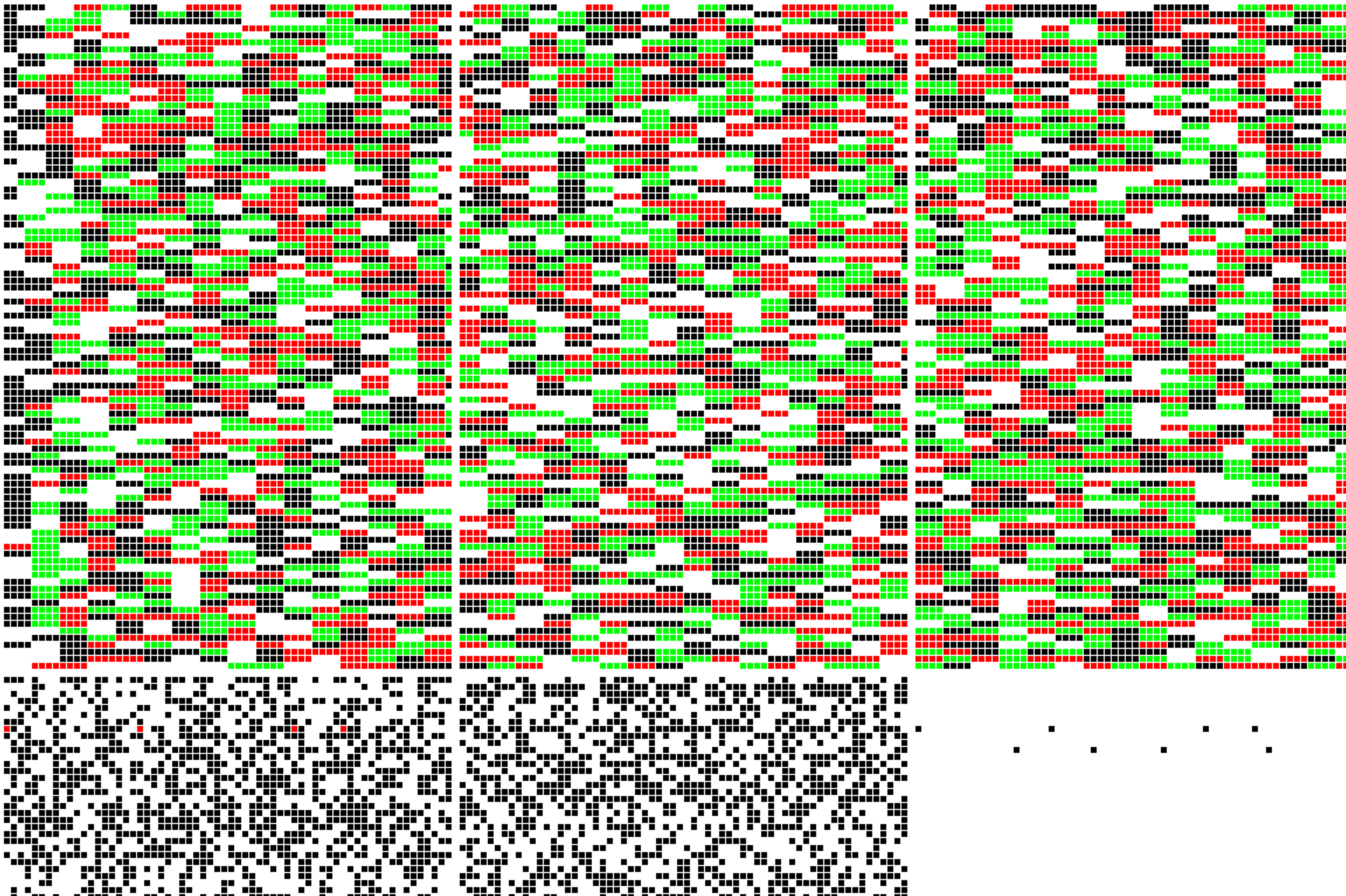
MD5 hash 79054025255fb1a26e4bc422aef54eb4

A Visualization of the Collision



http://www.links.org/?p=6

A Visualization
(Difference in just one MSB of the two blocks)



SHA1

input message (x)
(may be of any length less than 2^{64})

Algorithm : SHA-1-PAD(x)

comment: $|x| \leq 2^{64} - 1$

$d \leftarrow (447 - |x|) \bmod 512$

$\ell \leftarrow$ the binary representation of $|x|$, where $|\ell| = 64$

$y \leftarrow x \,||\, 1 \,||\, 0^d \,||\, \ell$

global $K_0, \ldots, K_{79}$

$y \leftarrow$ SHA-1-PAD(x)

denote $y = M_1 \,||\, M_2 \,||\, \cdots \,||\, M_n$, where each M_i is a 512-bit block

IV

$H_0 \leftarrow$ 67452301

$H_1 \leftarrow$ EFCDAB89

$H_2 \leftarrow$ 98BADCFE

$H_3 \leftarrow$ 10325476

$H_4 \leftarrow$ C3D2E1F0

for $i \leftarrow 1$ **to** n

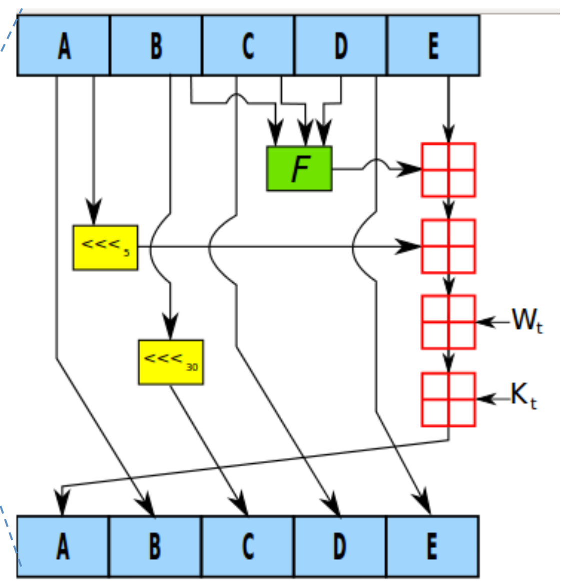
do
- denote $M_i = W_0 \,||\, W_1 \,||\, \cdots \,||\, W_{15}$, where each W_i is a word
- **for** $t \leftarrow 16$ **to** 79
  - **do** $W_t \leftarrow \mathbf{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
- $A \leftarrow H_0$
- $B \leftarrow H_1$
- $C \leftarrow H_2$
- $D \leftarrow H_3$
- $E \leftarrow H_4$
- **for** $t \leftarrow 0$ **to** 79
  - **do**
    - $temp \leftarrow \mathbf{ROTL}^5(A) + f_t(B, C, D) + E + W_t + K_t$
    - $E \leftarrow D$
    - $D \leftarrow C$
    - $C \leftarrow \mathbf{ROTL}^{30}(B)$
    - $B \leftarrow A$
    - $A \leftarrow temp$
- $H_0 \leftarrow H_0 + A$
- $H_1 \leftarrow H_1 + B$
- $H_2 \leftarrow H_2 + C$
- $H_3 \leftarrow H_3 + D$
- $H_4 \leftarrow H_4 + E$

return $(H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4)$

each word is 32 bits (512/16=32)

expand to 79 words

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq t \leq 79. \end{cases}$$



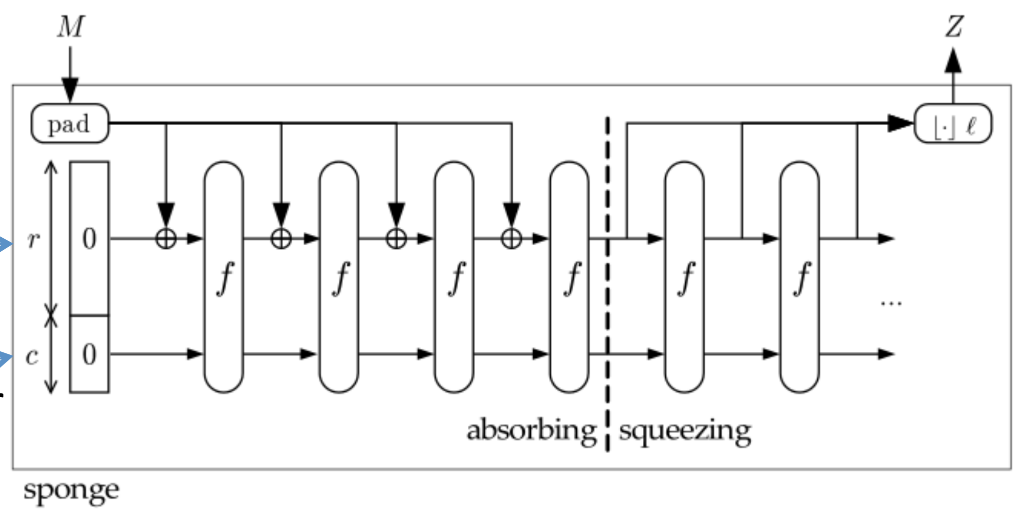
32*5=160 bit hash output

Kacchak and the SHA3

- Uses a sponge construction
  - Achieves variable length hash functions

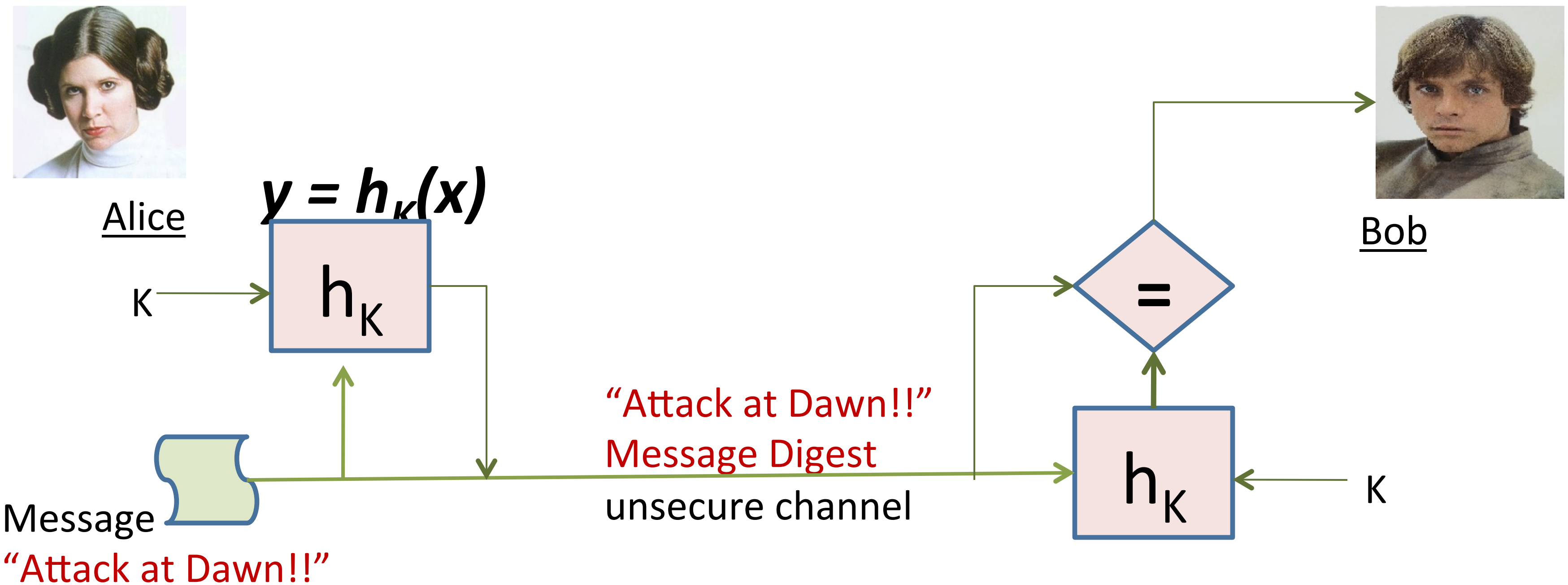
bit rate

security parameter



Success of an attack against Kecchak < $N^2/2^{c+1}$
where N is number of calls to f

Message Authentication Codes (Keyed Hash Functions)



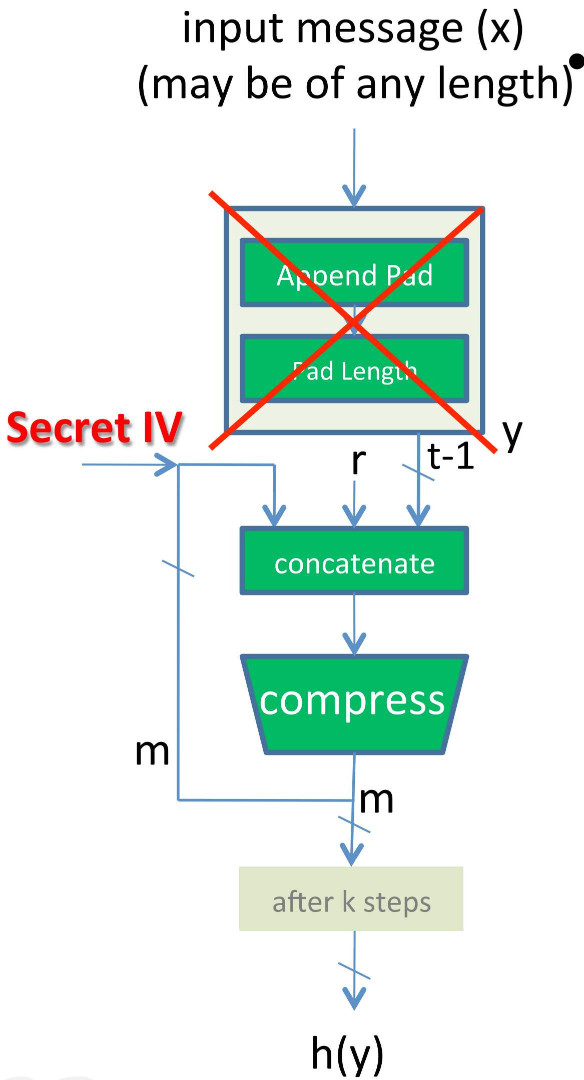
Provides Integrity and Authenticity
Integrity : Messages are not tampered
Authenticity : Bob can verify that the message came from Alice
(Does not provide non-repudiation)

How to construct MACs? recall … shortcuts

- For a message m, the only way to compute its hash is to evaluate the function $h_K(m)$
- This should remain to irrespective of how many hashes we compute
  - Even if we have computed $h_K(m_1)$, $h_K(m_2)$, $h_K(m_3)$, ……., $h_K(m_{1000})$
    It should be difficult to compute $h_K(x)$ without knowing the value of K

CR

Constructing a MAC (Naïve Attempt)

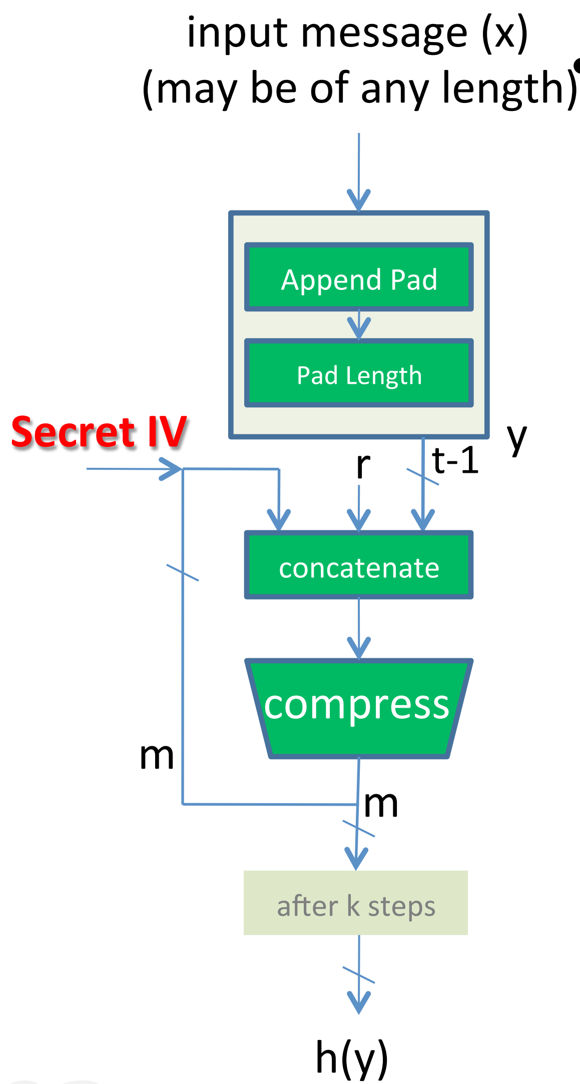


- Won't work if no preprocessing step
  - attackers could append messages and get the same hash

    $x \rightarrow h_K(x)$,

    $x \,||\, x' \rightarrow \text{compress}(h_K(x) \,||\, x')$

Constructing a MAC (Naïve Attempt)



- Won't work if preprocessing step present

$$\text{suppose } y = x \,\|\, pad(x) \qquad \text{where } |y| = rt$$

$$\text{consider } x' = x \,\|\, pad(x) \,\|\, w \qquad \text{where } |w| = t$$

$$y' = x' \,\|\, pad(x') = x \,\|\, pad(x) \,\|\, w \,\|\, pad(x')$$

$$\text{where } |y'| = r't \quad \text{for some integer } r' > r$$

$$\text{Let } z_r = h_K(x)$$

$$z_{r+1} \leftarrow compress(h_K(x) \,\|\, y_{r+1})$$

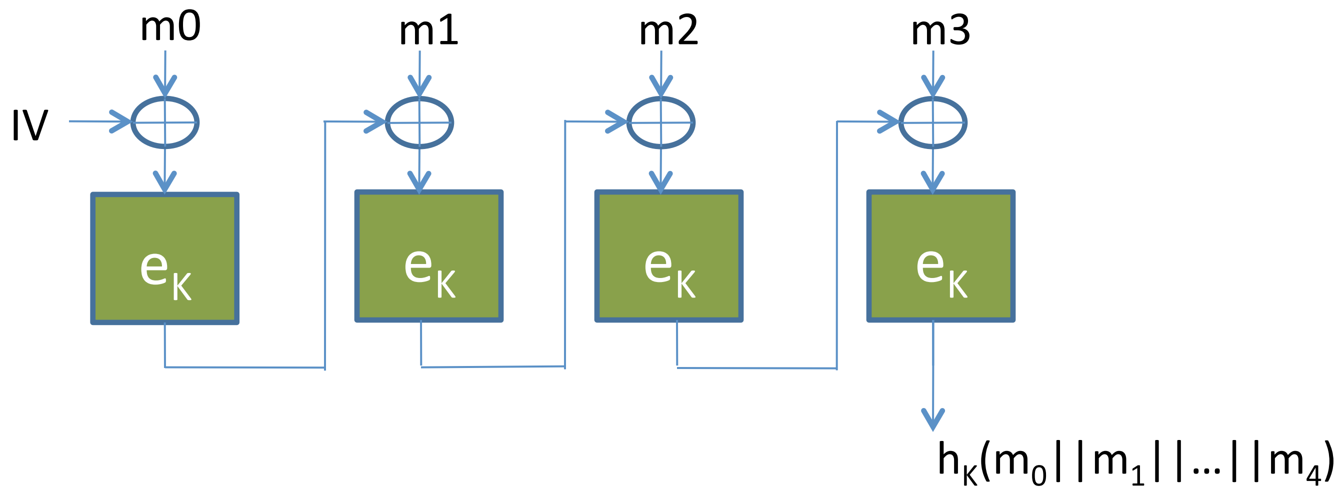
$$z_{r+2} \leftarrow compress(z_{r+1} \,\|\, y_{r+2})$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

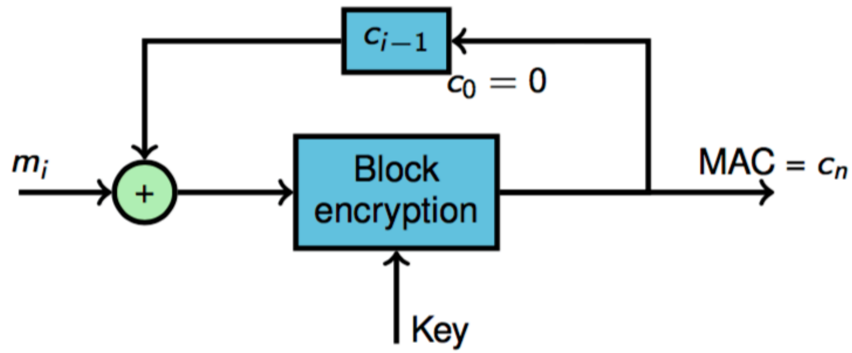
$$z_{r'} \leftarrow compress(z_{r'-1} \,\|\, y_{r'})$$

$$thus \quad h_K(x') = z_{r'}$$

CBC-MAC



Birthday Attack on CBC MAC



By Birthday paradox, in 2^{64} steps (assuming a 128 bit cipher), a collision will arise. Let's assume that the collision occurs in the a-th and b-th step.

$$c_a = c_b$$

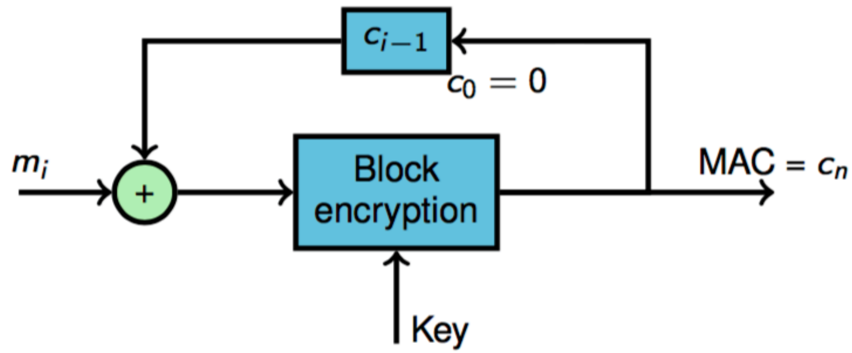
$$E_k(m_a \oplus c_{a-1}) = E_k(m_b \oplus c_{b-1})$$

$$thus$$

$$m_a \oplus c_{a-1} = m_b \oplus c_{b-1}$$

$$m_a \oplus m_b = c_{a-1} \oplus c_{b-1}$$

Birthday Attack on CBC MAC



By Birthday paradox, in 2^{64} steps (assuming a 128 bit cipher), a collision will arise.
Let's assume that the collision occurs in the a-th and b-th step.

$$c_a = c_b$$
$$E_k(m_a \oplus c_{a-1}) = E_k(m_b \oplus c_{b-1})$$
$$thus$$
$$m_a \oplus c_{a-1} = m_b \oplus c_{b-1}$$
$$m_a \oplus m_b = c_{a-1} \oplus c_{b-1}$$

$$M_1 = m_1 \| m_2 \| \ldots \| m_i \| \ldots \| m_n$$
$$M_2 = m_1 \| m_2 \| \ldots \| (m_i \oplus c_{a-1} \oplus c_{a-2}) \| \ldots \| m_n$$

HMAC

- FIPS standard for MAC
- Based on unkeyed hash function (SHA-1)

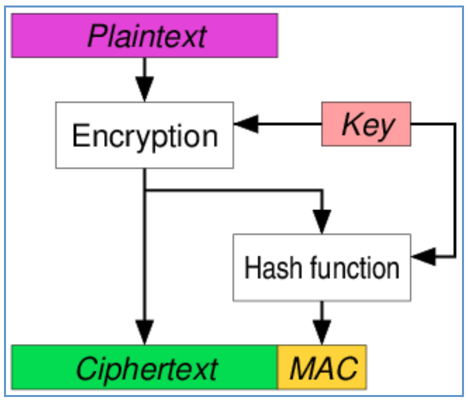
$$HMAC_k(x) = SHA1((K \oplus opad) \| SHA1(K \oplus ipad) \| x))$$

Ipad and opad are predefined constants

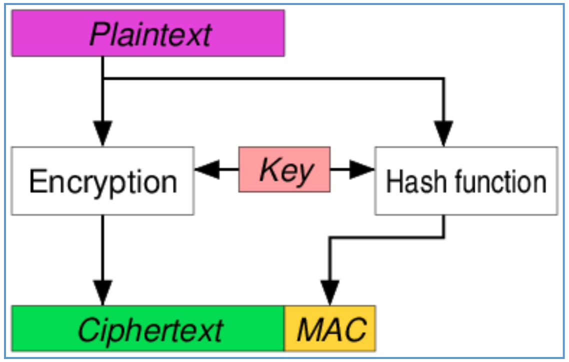
CR

Authenticated Encryption

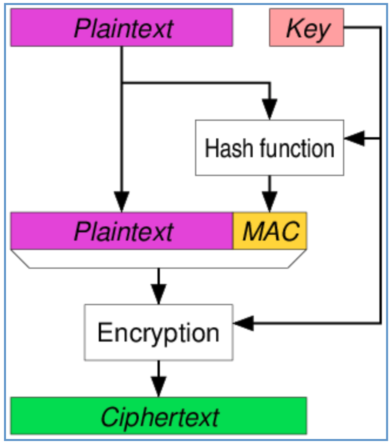
- Achieves Confidentiality, Integrity, and Authentication



EtM
(encrypt then MAC)



E&M



MtE
(MAC then Encrypt)

Using CBC-MAC for Authenticated Encryption

1. Consider $p = (p_0, p_1, p_2, p_3)$ is a message Alice sends to Bob
    1. She encrypts it with CBC as follows
       $c_0 = E_k(p_0)$; $c_1 = E_k(p_1 + c_0)$; $c_2 = E_k(p_2 + c_1)$; $c_3 = E_k(p_3 + c_2)$
    2. She computes **mac** = $\text{CBC-MAC}_k(p)$
       She transmits (**c**, **mac**) to Bob : where **c** = (c_0, c_1, c_2, c_3)
2. Mallory modifies one or more of the ciphertexts (c_0, c_1, c_2) to (c_0', c_1', c_2')
3. Bob will
    1. Decrypt (c_0', c_1', c_2') to (p_0', p_1', p_2')
    2. And use it compute the MAC **mac'**

We show that **mac' =** $\mathbf{c_3}$ irrespective of how Mallory modifies the ciphertext

Using CBC-MAC for Authenticated Encryption

Alice's side (encryption)

$$c_0 = E_k(p_0)$$
$$c_1 = E_k(p_1 \oplus c_0)$$
$$c_2 = E_k(p_2 \oplus c_1)$$
$$c_3 = E_k(p_3 \oplus c_2)$$

Bob's side (decryption)

$$p'_0 = D_k(c'_0)$$
$$p'_1 = D_k(c'_1) \oplus c'_0$$
$$p'_2 = D_k(c'_2) \oplus c'_1$$
$$p'_3 = D_k(c_3) \oplus c'_2$$

$$(assume\, IV = 0)$$

$$\begin{aligned} mac' &= CBCMAC(p') \\ &= E_k(p'_3 \oplus E_k(p'_2 \oplus E_k(p'_1 \oplus E_k(p'_0)))) \\ &= E_k(p_3 \oplus c'_2) \\ &= E_k(D_k(c_3) \oplus c'_2 \oplus c'_2) \\ &= E_k(D_k(c_3)) \\ &= c_3 \end{aligned}$$

Without modifying the final ciphertext, Mallory can change any other ciphertext as she pleases. The CBC-MAC will not be altered.

Moral of the story: Never use CBC-MAC with CBC encryption!!

Counter Mode + CBC-MAC for Authenticated Encryption

Consider p = (p_0, p_1, p_2, p_3) is a message Alice sends to Bob

1. She encrypts p with counter mode as follows
   $c_0 = p_0 + E_k(ctr)$; $c_1 = p_1 + E_k(ctr + 1)$;
   $c_2 = p_2 + E_k(ctr + 2)$; $c_3 = p_3 + E_k(ctr + 3)$

2. She computes **mac** = CBC-MAC$_k$(p)
   She transmits (**c**, **mac**) to Bob : where **c** = (c_0, c_1, c_2, c_3)