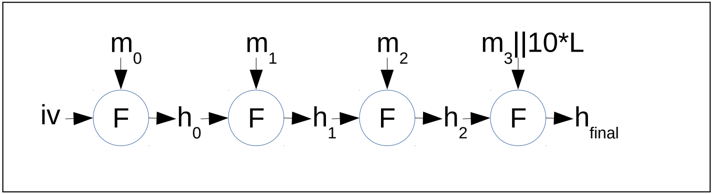
Some Attacks on Merkle-Damgård Hashes

John Kelsey, NIST and KU Leuven

May 8, 2018



Overview

- Cryptographic Hash Functions
- Thinking About Collisions
- Merkle-Damgård hashing
- Joux Multicollisions[2004]
- Long-Message Second Preimage Attacks[1999,2004]
- Herding and the Nostradamus Attack[2005]

Why Talk About These Results?

- These are very visual results–looking at the diagram often explains the idea.
- The results are pretty accessible.
- Help you think about what's going on inside hashing constructions.

Part I: Preliminaries/Review

- Hash function basics
- Thinking about collisions
- Merkle-Damgård hash functions

Cryptographic Hash Functions

- Today, they're the workhorse of crypto.
- Originally: Needed for digital signatures
  - You can't sign 100 MB message–need to sign something short.
  - "Message fingerprint" or "message digest"
  - Need a way to condense long message to short string.
- We need a stand-in for the original message.
- Take a long, variable-length message...
- ...and map it to a short string (say, 128, 256, or 512 bits).

Properties

What do we need from a hash function?

- Collision resistance
- Preimage resistance
- Second preimage resistance
- Many other properties may be important for other applications

Note: cryptographic hash functions are designed to behave randomly.

Collision Resistance

The core property we need.

- Can't find $X \neq Y$ such that $\texttt{HASH}(X) = \texttt{HASH}(Y)$
- Note, there must be *huge* numbers of collisions...
  - How many million-bit strings are there?
  - Way more than number of 256-bit strings.
- ...but it's very hard to find them.
- Ideally, best way to find collisions is trying lots of messages
- ...until a pair of outputs happen to collide by chance.

Preimage and Second Preimage Resistance

What other properties do we need from a hash function?

- Preimage resistance
  - Given H, can't find X such that $H = \texttt{HASH}(X)$
- Second preimage resistance
  - Given X, can't find Y such that $\texttt{HASH}(X) = \texttt{HASH}(Y)$.
  - Like finding a collision, but harder–you already have a target message.

Generic Attacks

For **any** *hash function, we have these generic attacks:*

- Collision with $2^{n/2}$ tries.
- Preimages and second preimages with 2^n tries.

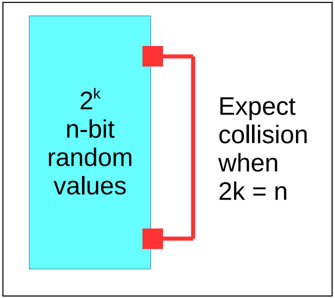
If hash function behaves randomly, these are the best we can do.

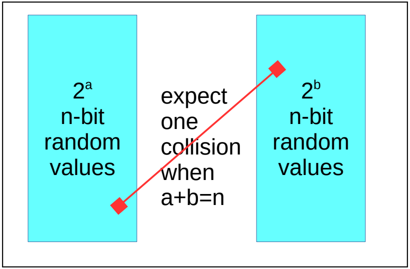
Other Properties

Where else are hashes used?

- Over time, hash functions became workhorses, used in many places:
  - Message authentication (HMAC)
  - Key derivation functions
  - Password hashing
  - Cryptographic PRNGs (HashDRBG, FIPS186 PRNG)
  - Hashing data for commitments
  - Proofs of work
- These applications often require other properties.

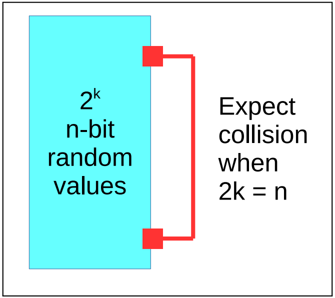
Thinking About Collisions





Collisions in a List

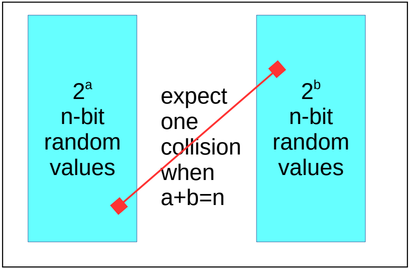
Suppose we have a list of 2^k random n-bit numbers.
How many collisions can we expect?



- $\binom{2^k}{2} \approx 2^{2k-1}$ pairs of random values.
- Each pair has probability 2^{-n} to collide.
- So we expect about 2^{2k-n-1} collisions.

Matching Between Two Lists

Suppose we have two lists of random n-bit numbers.
How many collisions can we expect?



- 2^{a+b} pairs of random values.
- Each pair has probability 2^{-n} to collide.
- So we expect about 2^{a+b-n} collisions.

Merkle-Damgård

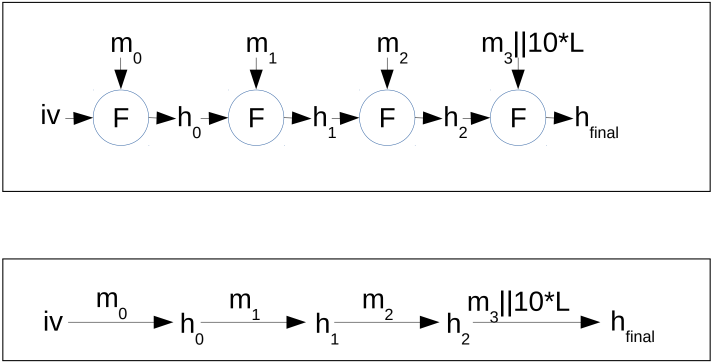
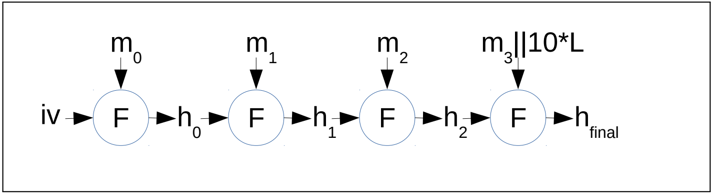


Figure: Two Different Ways to Represent Merkle-Damgård Hashing

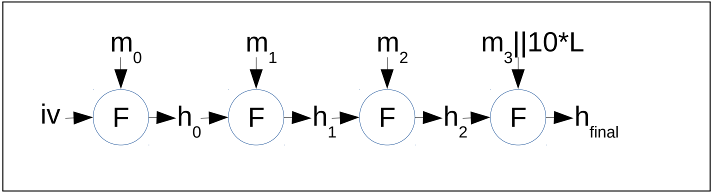
How to Make a Good Hash Function?

- We needed to be able to build good hash functions
  - Collision resistance, second preimage resistance, preimage resistance
- About the only thing anyone knew how to build were block ciphers.
- Merkle and Damgård independently worked out a strategy
- ...that was wildly successful.



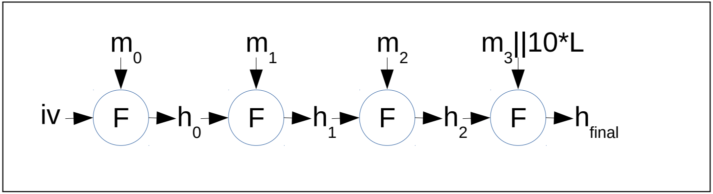
Merkle-Damgård Hashes (1)

Big idea: Make a good fixed-length hash function, then build a variable-length hash from it.



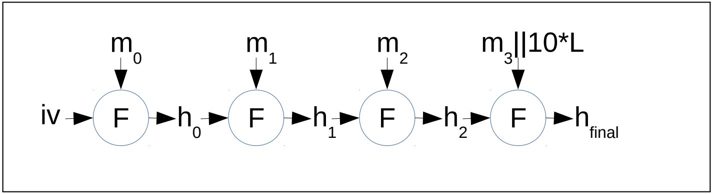
- We need a fixed-length *compression function*, $F(h, m)$
  - h_{in} = hash chaining value, n bits. (Example $n = 256$)
  - h_{out} = hash chaining value, n bits.
  - m = message block, w bits. (Example $w = 512$)
- Pad the message, break into w-bit chunks, and process sequentially.

Merkle-Damgård Hashes (2)



1. Pad message to integer multiple of w bits:
   - 10* padding
   - ...plus length of unpadded message (Merkle-Damgård strengthening)
2. Break padded message into blocks $m_{0,1,2,\ldots,k-1}$.
3. $h_{-1} =$ some fixed initial value, iv.
4. $h_i \leftarrow F(h_{i-1}, m_i)$ for $i = 0, 1, 2, \ldots, k-1$.
5. Final h_i is $\texttt{HASH}(M)$

This strategy was *wildly successful*!

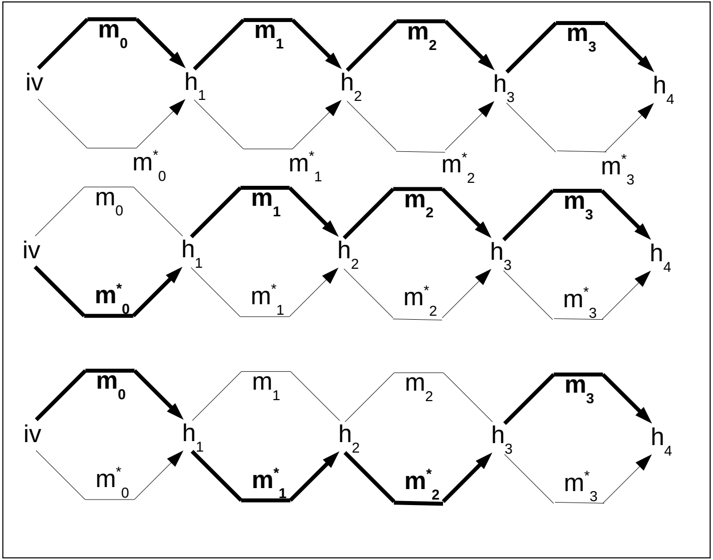


- Merkle-Damgård construction lets you worry about security of *compression function*
- ...let construction take care of whole *hash function*.
- Almost all hashes for next 20+ years used Merkle-Damgård construction!
    - MD4, MD5
    - SHA0, SHA1, SHA256, SHA512
    - RIPE-MD, RIPE-MD160, Haval
    - Snefru,Tiger, Whirlpool

Part II: Surprising Properties of Merkle-Damgård Hashes

- Joux multicollisions
- Long-message second preimage attacks
- Herding attacks

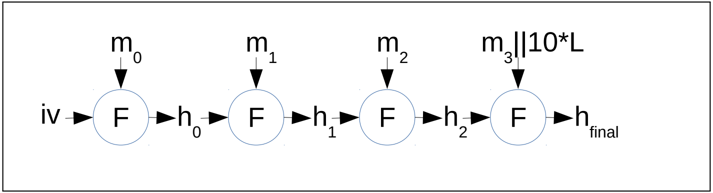
Joux Multicollisions



Joux' Multicollision Result

- In 2004, Joux published a new attack on Merkle-Damgård hashes.
- ...showing that we hadn't really understood them despite 20+ years of work.
- He showed that:
  - Finding 2^{30} values with the same hash for an Merkle-Damgård hash...
  - ...takes only about 30 times the work of finding one collision!
  - Concatenating two Merkle-Damgård hashes doesn't give much extra security.
- Joux's work was the basis for the other results I'll talk about today.

A Property of Merkle-Damgård Hashes



- h_k contains everything `HASH` will ever know about $m_{0,1,2,\ldots,k-1}$
- This is necessary for `HASH` to be efficient
  - `HASH` needs to process the data in one pass.
- But it has some surprising consequences....

Notation

This is an equivalent way to show Merkle-Damgård hashing.

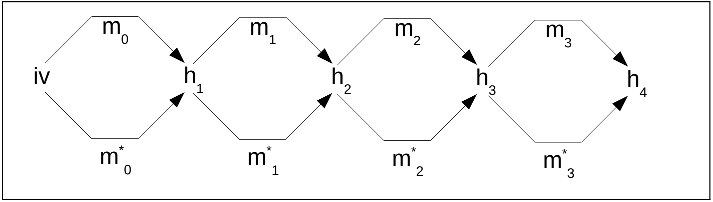
$$iv \xrightarrow{m_0} h_0 \xrightarrow{m_1} h_1 \xrightarrow{m_2} h_2 \xrightarrow{m_3||10*L} h_{final}$$

Figure: A Different Way to Represent Merkle-Damgård Hashing

- The nodes are hash chaining values
- The edges are message blocks
- This is useful for thinking about Joux Multicollisions

Constructing a Joux Multicollision

We can concatenate collisions!

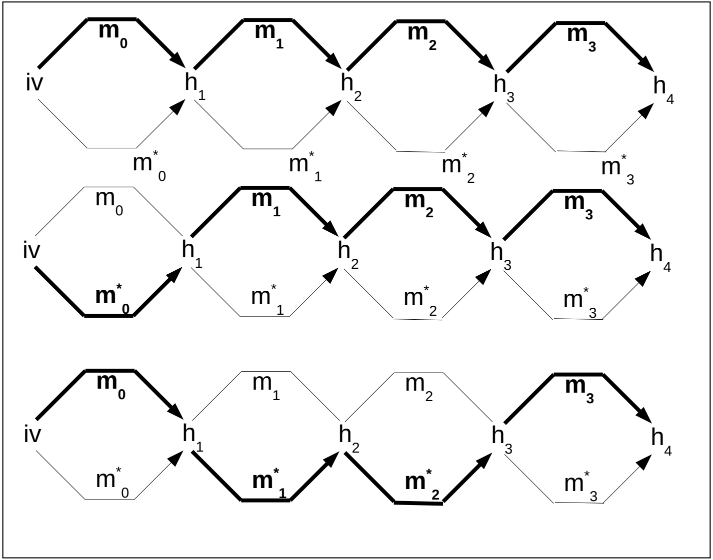


1. Find colliding pair from iv: $(m_0, m_0^*) \rightarrow h_1$.
2. Find colliding pair from h_0: $(m_1, m_1^*) \rightarrow h_2$.
3. Find colliding pair from h_1: $(m_2, m_2^*) \rightarrow h_3$.
4. Find colliding pair from h_2: $(m_3, m_3^*) \rightarrow h_4$.

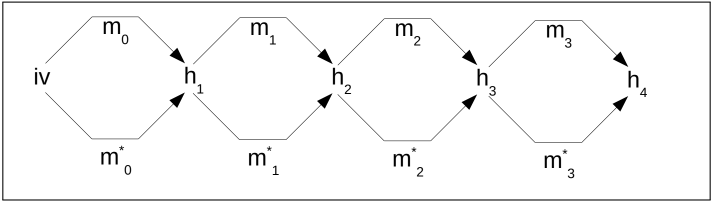
Four collision searches, work $\approx 4 \times 2^{n/2}$

How many different values have we found that all hash to h_4?

Each Path = Different Message (All with Same Hash)



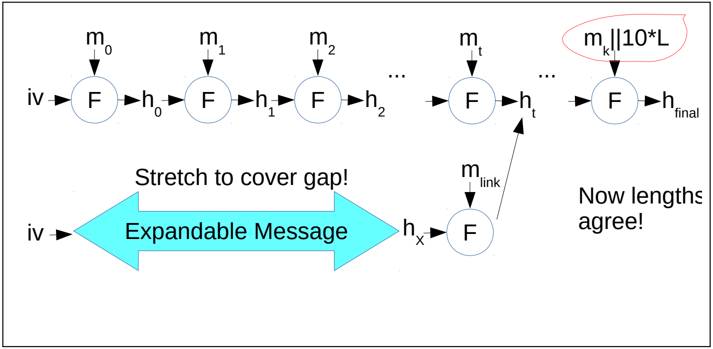
Joux Multicollisions: Work



- k collision-searches $\rightarrow 2^k$ values all with same hash

  k choices in the path $= 2^k$ total paths.
- A 2^k-multicollision
- An ideal hash function would not have this property.
  - It should be incredibly hard to find a 2^{32}-way multicollision.
- This was a huge surprise...but it was only the beginning!

The Long-Message Second Preimage Attack



The Long-Message Second Preimage Attack: Setting

$$iv \xrightarrow{m_0} h_0 \xrightarrow{m_1} h_1 \xrightarrow{m_2} h_2 \xrightarrow{m_3} h_3 \xrightarrow{m_4} h_4 \cdots \xrightarrow{m_{k-1}||10^*L} h_{target}$$

1. We are given a very long *target message*, M_{target}.

   $k = 2^{\ell}$ blocks long.

   Example: 2^{55}-block (about 2^{64} bit) message for SHA1.

2. We want to find a new message M_{second} such that:

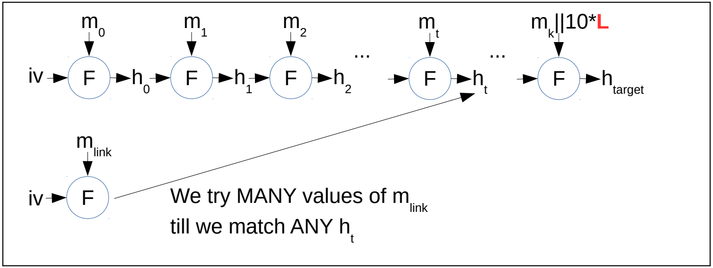
$$M_{\text{second}} \neq M_{\text{target}}$$
$$\texttt{HASH}(M_{\text{second}}) = \texttt{HASH}(M_{\text{target}})$$

.

3. This is *expected* cost about 2^n work.

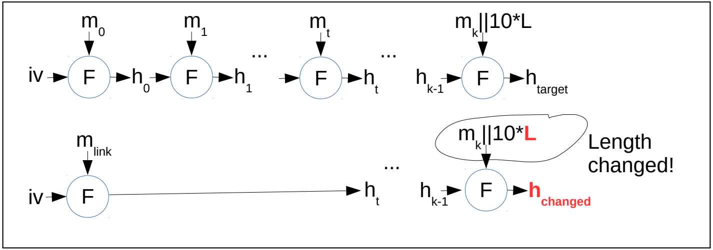
   Just like a preimage attack.

An Attack that ALMOST Works



1. Try *lots* of values for m_{link}.
2. After $2^{n-\ell}$ tries, expect to hit *some* intermediate hash.

Blocked By the Length in the Padding!



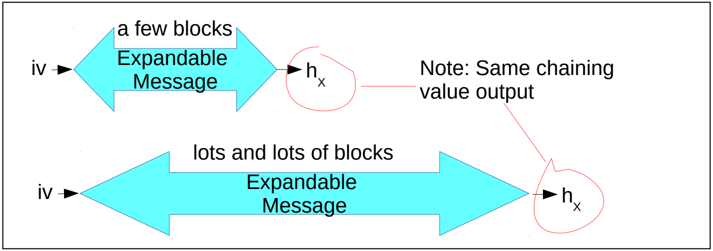
3 ...but our new message is the wrong length!

Everything is fine until the final compression function...

...then L changes, and so does h_{final}.

Winternitz had proposed this attack on some earlier hash constructions.

What We Need: An Expandable Message

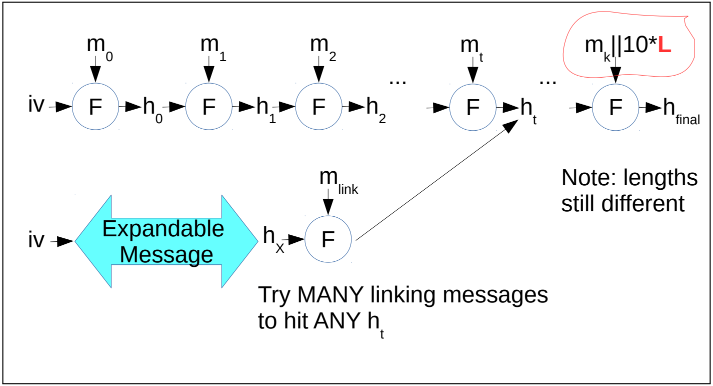


We need a new tool–an *expandable message*.

- Set of messages that can take on wide range of possible lengths...
- ...but always has the same intermediate hash at the end

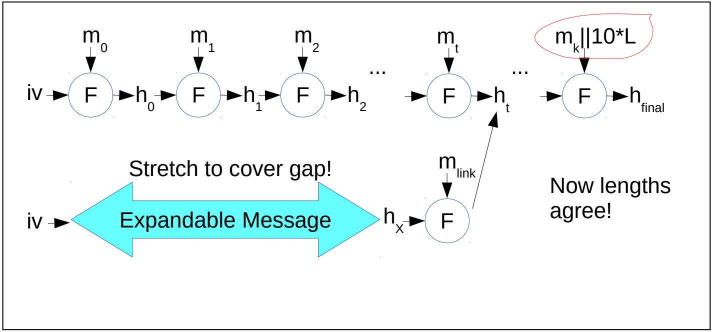
  Note: this is an intermediate hash, so Merkle-Damgård strengthening hasn't touched it yet.
- We can *stretch* this message to many different lengths.

How Would an Expandable Message Help?



- As before, we compute our linking message...

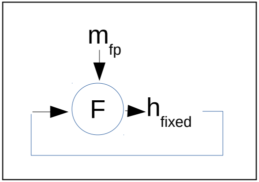
Make the Lengths Agree!



- But now we can make the lengths agree
- ...bypassing the length in the final block's padding!

So if we could find expandable messages, we could find second preimages on long messages.

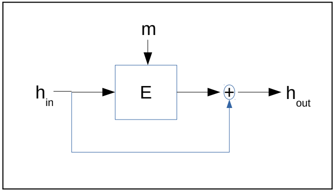
Detour: Fixed Points



- A *fixed point* is a value for which some function gives its input as its output.
- In this case, there's some h_{fixed}, m_{fp} such that

$$h_{fixed} = F(h_{fixed}, m_{fp})$$

Common Way of Making Compression Functions: Davies-Meyer



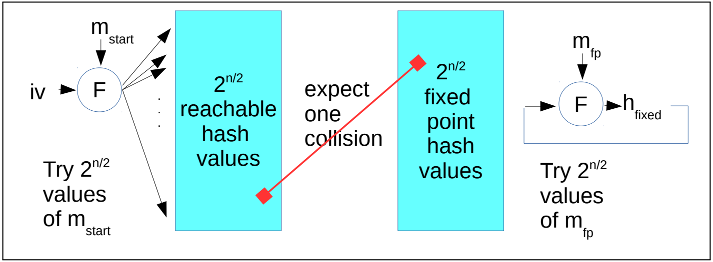
This **should** be hard, but....

$$F(h, m) = E_m(h) \oplus h$$

To find a fixed point, choose *any* m and compute

$$\begin{aligned} h &= D_m(0) \\ E_m(h) &= 0 \\ F(h, m) &= E_m(h) \oplus h \\ &= 0 \oplus h \\ &= h \end{aligned}$$

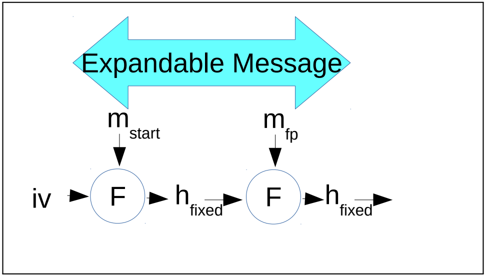
Expandable Message from Fixed Points [Dean 99]



1. Generate $2^{n/2}$ random fixed point hashes.
2. Generate $2^{n/2}$ random starting messages.
3. Expect one collision.
4. Expandable message $= m_{\text{start}} \mid\mid m_{\text{fp}}$
5. Expected work to construct: $2^{n/2+1}$.

Dean discovered this in 1999, in his PhD thesis–but nobody knew about it!
(We rediscovered it in 2004!)

The Expandable Message



- The minimum length is two message blocks.
- It can expand to *any* length.

Stretching the Expandable Message

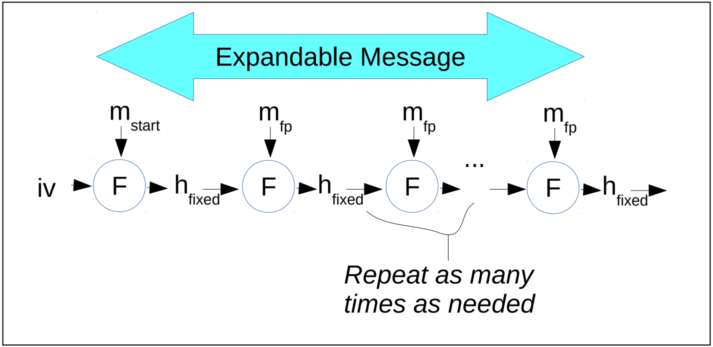
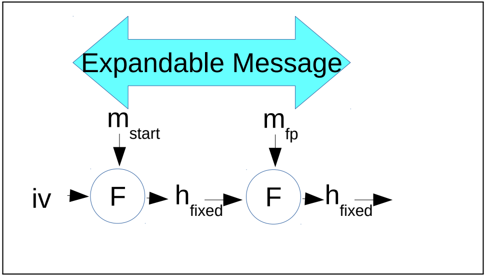


Figure: Stretching Expandable Message By Repeating m_{fp}

- Once we have expandable message, it's trivial to stretch it...
- ...just repeat m_{fp} as many times as needed.

Expandable Messages from Fixed Points: Work



- Depends on compression function–not all Merkle-Damgård hashes have easy-to-find fixed points.
- ...but this works for MD5, SHA1, SHA2
- Work to construct: $2^{n/2+1}$

Expandable Messages From Joux Multicollisions

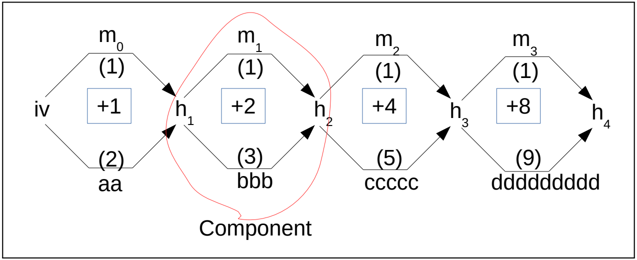


Figure: Expandable Messages from Joux Multicollisions

- We discovered these in 2004.

  (Lucky for us, or Dean would have totally scooped us!)
- These *always* work for *any* Merkle-Damgård hash.
- Consists of many *components* (collisions)
- Each component:
  - Costs $2^{n/2}$ to build.
  - Doubles number of possible lengths of expanded message.

How It Works: Minimum Length

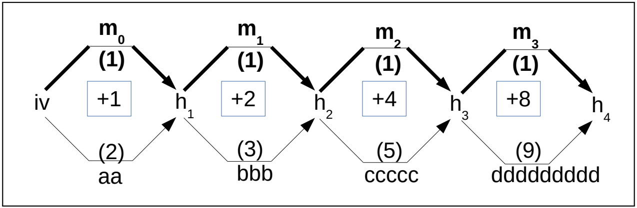


Figure: Expandable Message at Shortest Length: 4 Blocks

- We choose a length by choosing a path through the multicollision.
- Each component has two paths that differ in length by a power of 2.
- Result: With k components, length from k to $k + 2^k$ blocks.

How It Works: Choosing a Length

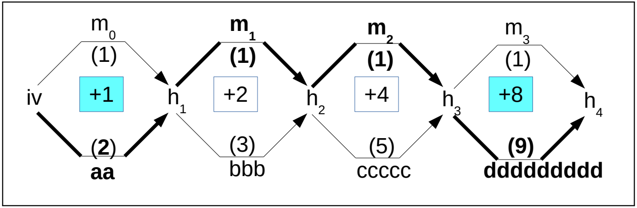
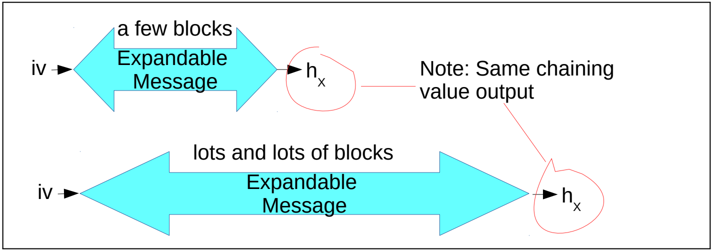


Figure: Message Expanded to 13 Blocks

- By choosing a different path, we can add blocks to the length of the message.
- In this case, we chose a length of 13 blocks.

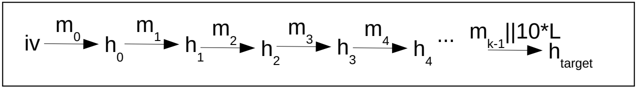
Now We Have Expandable Messages



- Fixed-point expandable messages
  - Cheaper to build, but don't always work.
- Joux-multicollision based expandable messages.
  - More expensive to build, but work for all Merkle-Damgård hashes.

...so we can carry out long-message second preimage attacks!

The Long-Message Second Preimage Attack



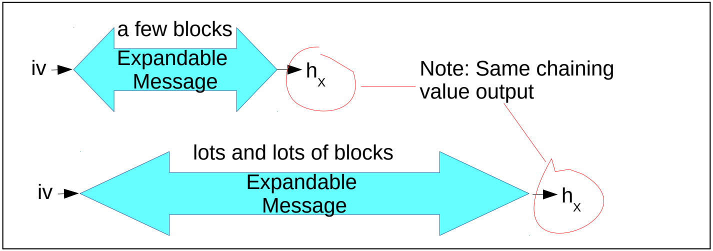
Given: Target message M_{target} of $k = 2^{\ell}$ blocks.

Steps:

1. Construct expandable message with length up to k blocks.
2. Find linking message to *any* intermediate hash for M_{target}.
3. Expand message to cover skipped-over message blocks.

Total cost $=$ expandable message $+$ linking message.

Step One: Build Expandable Message

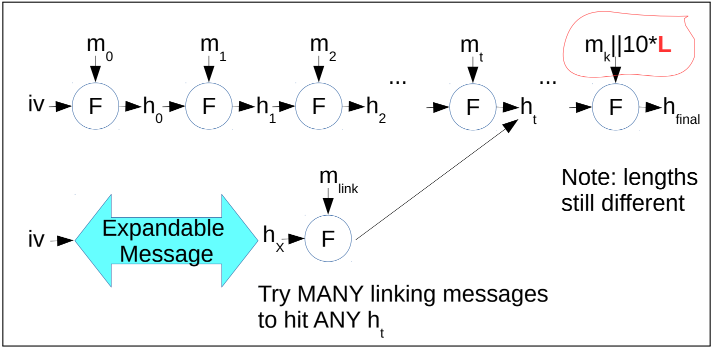


Reminder: M_{target} *is* 2^{ℓ} *blocks long*

- For fixed-point expandable messages, $2^{n/2+1}$.
- For multicollision expandable messages, $\ell \times 2^{n/2+1}$

This is almost never the expensive part of the attack.

Step Two: Find Linking Message

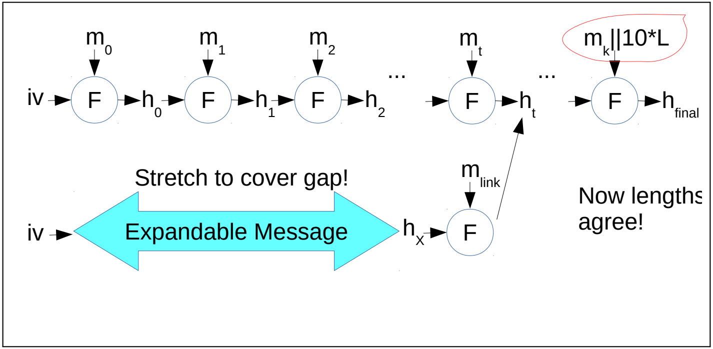


Reminder: M_{target} *is* 2^{ℓ} *blocks long*

- There are about 2^{ℓ} intermediate hash values to hit.
- For n-bit hash output, expect $2^{n-\ell}$ tries to get a match.

This is almost always the expensive part of the attack.

Step Three: Stretch Expandable Message to Fix Length



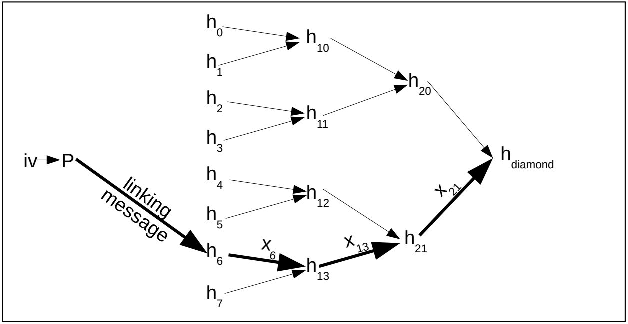
- This costs almost nothing for either type of expandable message.
- *Result:* Second message with same hash output as M_{target}.
  ...and same *length* as M_{target}.

Total Cost

- Merkle-Damgård hashes have maximum lengths they will support.
  - MD5, SHA1, SHA256: About 2^{55} blocks.
  - SHA512: About 2^{107} blocks.
- Attack gets cheaper (but less practical) for longer messages.
- Second preimage attack on SHA1 with 2^{55}-block message:

$$\begin{aligned}\text{total cost} &= \text{expandable message} + \text{linking message} \\ &= 2^{81} + 2^{160-55} \\ &= 2^{81} + 2^{105} \\ &\approx \boxed{2^{105}}\end{aligned}$$

Herding Hash Functions and the Nostradamus Attack



Using Hash Functions to Commit to a Result

- Suppose I claim I can tell the future...
- ...say, I clam I can predict presidential elections or the stock market.
- How can I prove my prophetic abilities without disclosing my predictions ahead of time?
- I could publish a `HASH` of my predictions.

Using Hash Functions to Commit to a Result(2)

So how does this work?

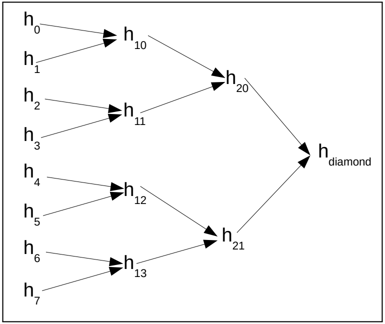
- I make my predictions
  - Using statistical models, prediction markets, dartboards, and crystal balls.
- I write them into a document, P.
- I hash the document, $H \leftarrow \texttt{HASH}(P)$.
- I publish H so that I can prove I'm a real prophet.
- ...After my predictions have come to pass, I reveal P.

Should You Believe I Can Tell the Future?

Suppose I go through this protocol using a somewhat-weak hash function like MD5.

- Is this evidence I can tell the future?
- What property of the hash function are you relying on?
- It's not exactly collision-resistance, but maybe not quite preimage resistance either....

The Diamond Structure: A Merkle-Tree Computed by Finding Collisions.

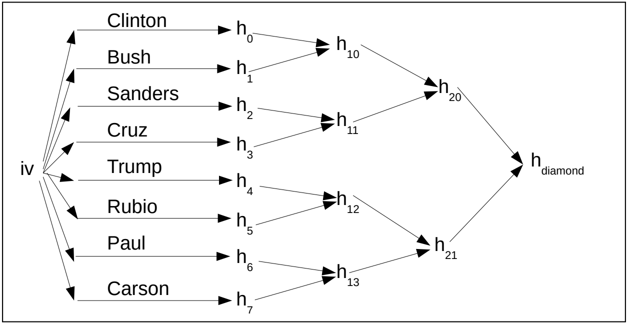


- Starting from 2^k random hash values, build a hash tree.
- ...by finding collisions.
- Result: A diamond structure that routes 2^k input hash chaining values into one output hash.

Note: Edges have multiple message blocks; nodes are hash chaining values.

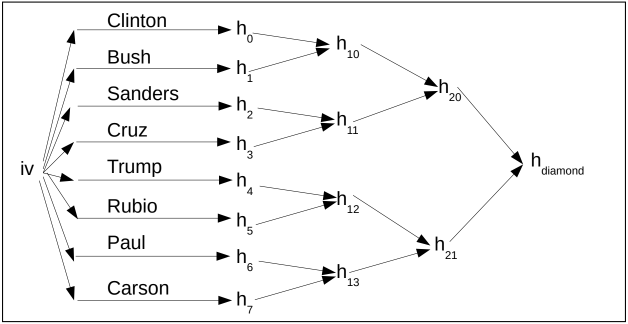
Precomputing the Diamond

I claim to predict the outcome of the 2016 presidential election.



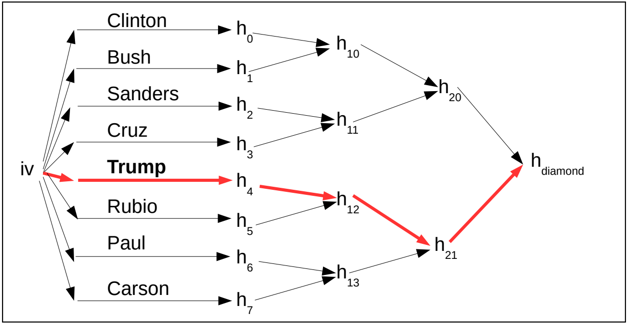
- I precompute messages predicting each of eight likely winners.
- Starting from *iv*, I generate eight prediction strings that are all the same length.
- Each arrow has multiple message blocks of boilerplate.
- I hash them into a *diamond structure*.
- I publish h_{diamond}.

Cost to Precompute a Diamond



- For 2^k precomputed prediction strings...
- Naive approach: $2^k - 1$ collision searches.
- ...better approach for big k.
- I can reveal any of my precomputed choices after the election.
- But I have no more flexibility than that.
  - Once h_{diamond} is published, I'm stuck with my predictions.

Routing the Diamond



- When I want to “reveal” my prediction, I follow the edges of the tree.

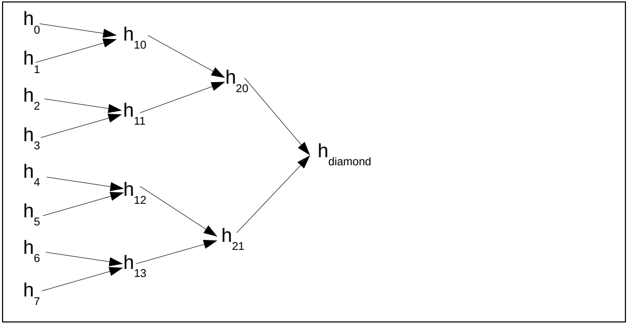
  This costs nothing.
- Each edge has some message blocks that are appended to my prediction string.
- At the end, can choose any of my precomputed predictions to reveal!

Herding Hash Functions

First, commit to a hash output, h_{diamond}.
Then, hash *any* prefix P to h_{diamond}.

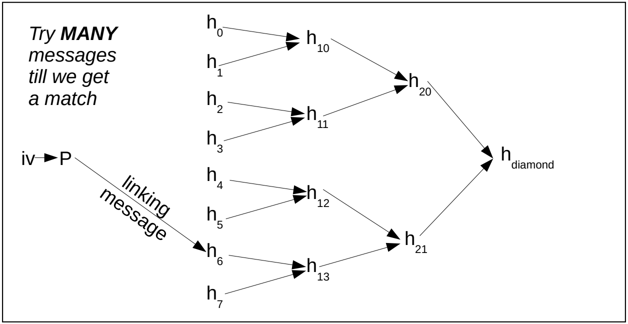
1. Build a random diamond structure with 2^k starting hash values.
2. Commit to h_{diamond}.
3. Decide what prediction P I want to have made.
4. Find a linking message from P to one of the starting hash values.
5. Route through the diamond to h_{diamond}.

Building the Random Diamond Structure



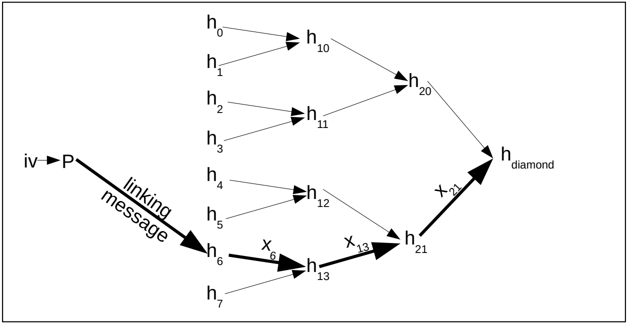
1. 2^k target values.
2. We need big k to make finding the linking message workable.
3. Intuition: We don't care which values hash together.
   - Compute $2^{n/2}$ messages from each random target hash value.
   - Expect to find enough collisions to get down to next layer of tree.
   - Repeat process until number of intermediate hashes is small enough to do naive algorithm.
   - Expected work is about $2^{(n+k)/2+2}$.

Finding a Linking Message



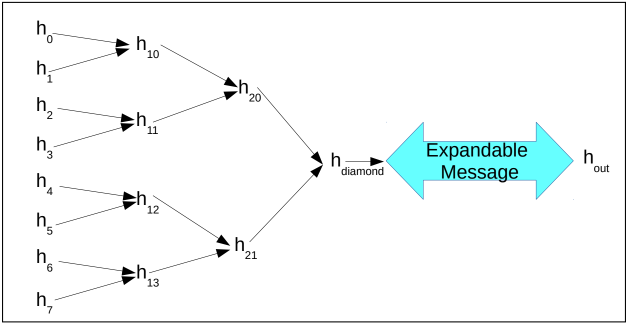
- With 2^k target values, we need about 2^{n-k} work to find a linking message.

Routing Through the Diamond



- Once we've found the linking message, we can route any prefix of the expected length to h_{diamond}.
- Total work: $2^{n/2+k/2+2}$ to make the diamond structure, and 2^{n-k} to find a linking message.

Refinements: Adding an Expandable Message



Adding an expandable message to the end of the diamond structure has two benefits:

- We now have some flexibility in length of P.
- We can hit *any* of the $2^{k+1} - 1$ total intermediate hashes in the diamond structure.
  - Makes it about twice as fast to find the linking message.

Wrapup

- I've talked about some fun attacks on Merkle-Damgård hashes.
- I hope I've also got you thinking about how hashing constructions work internally.
- We spent 20+ years thinking we understood hash functions...
- ...only to discover big surprises.
  - Multicollisions are way cheaper than anyone expected.
  - Second preimages are way cheaper than anyone expected.
  - You can break hash-based commitments without preimage attacks.
- There are many more results along these lines, and maybe you can discover some.

Questions

- Questions?