Message Shuffling to Prevent Hash Extension Attacks

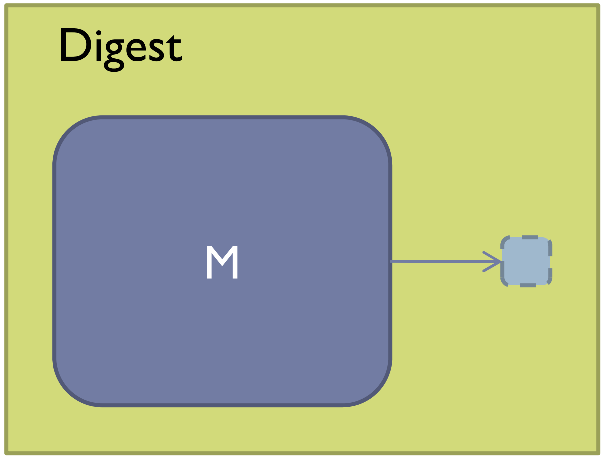
Elizabeth Reid, Chris Wilkens, Christina Wright

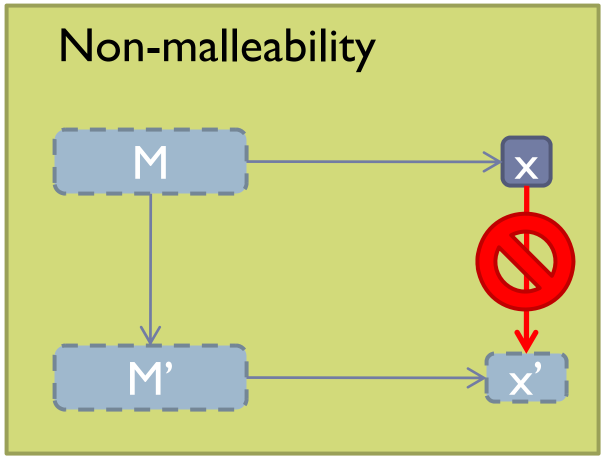
Outline

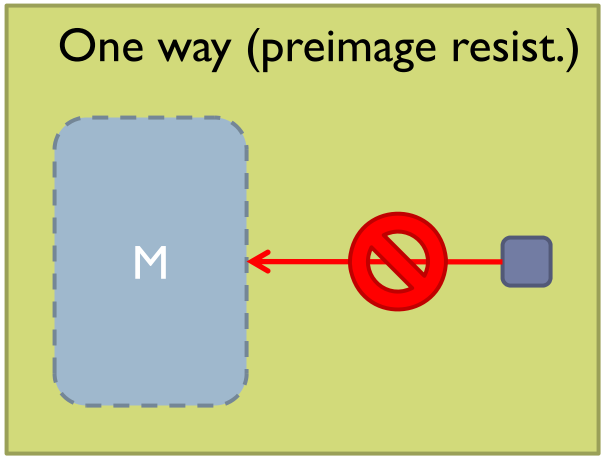
- Hash Review
  - Properties
  - Implementation & Issues
- Our Solution
- Proof of Security
- Implementation & Results

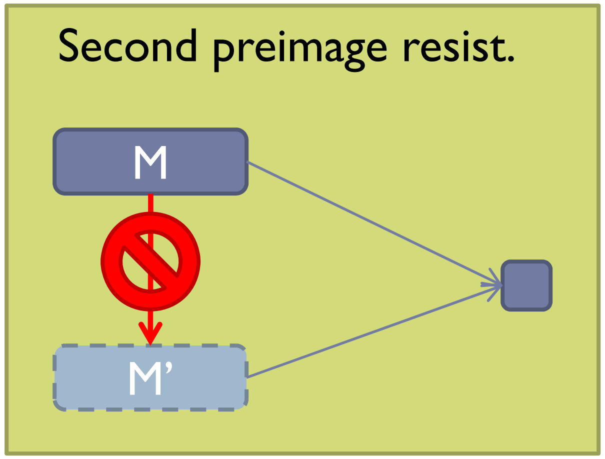
Hash Review

Desirable Properties



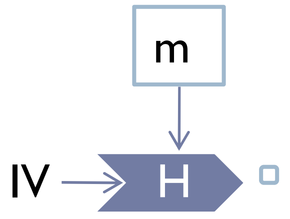




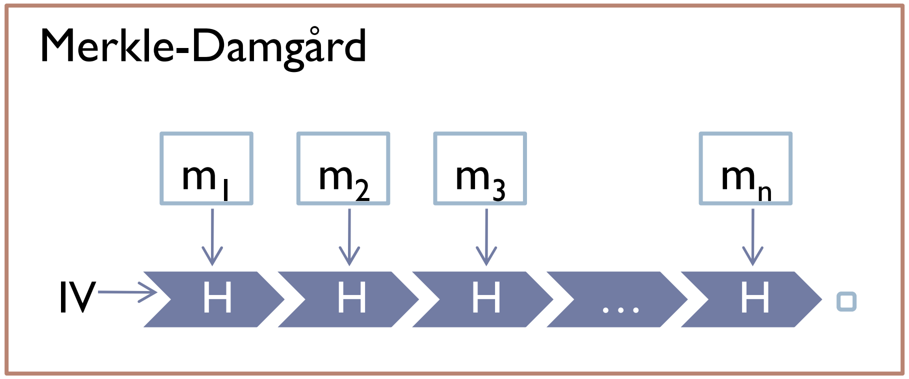


Implementation

- Compression Function
  - Fixed length input
  - Ideal hash properties
    - Collision resistance
    - Psuedo-random function
    - Random oracle

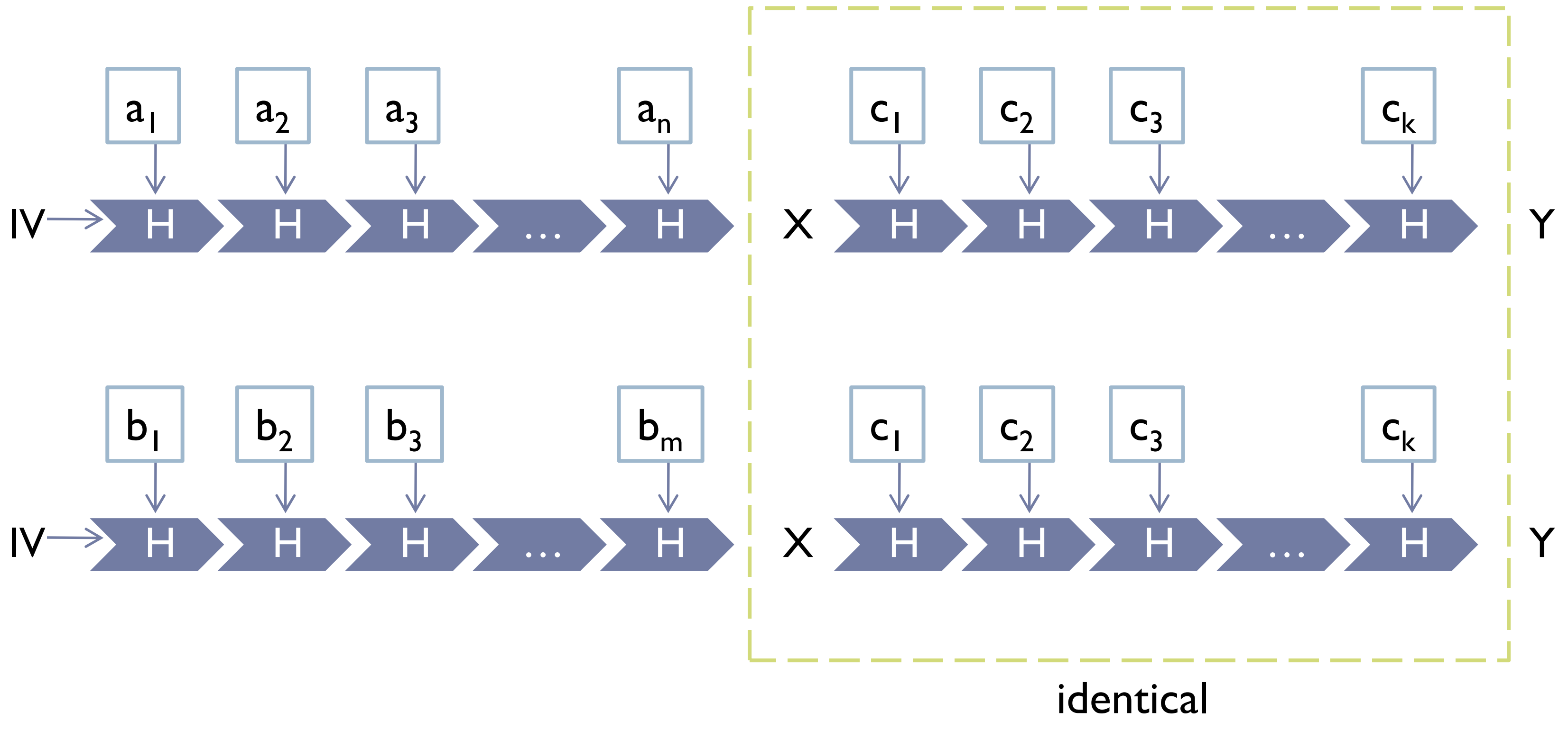


- Hash Domain Extension
  - Arbitrary length input
  - Preserves hash properties



Merkle-Damgård Extension Attack

- $H(A) = H(B) \rightarrow H(A||C) = H(B||C)$



Existing Solutions to Extension Attack

- Double Hashing
  - $h_1(h_1(M)||M)$
  - Requires reading data twice
- Prefix-free
  - Restrict input messages

Our Idea

Our Contribution

GOALS

- Prevent extension attacks
- Improve collision resistance
  - Particularly multicollisions
- Only read message once

ACHIEVEMENTS

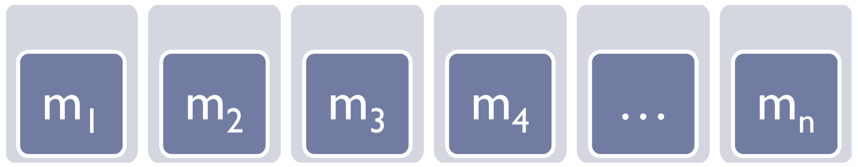
- Proved secure against extension attacks
- Hypothesized increased collision resistance
- Practical speed and space requirements

High Level Idea

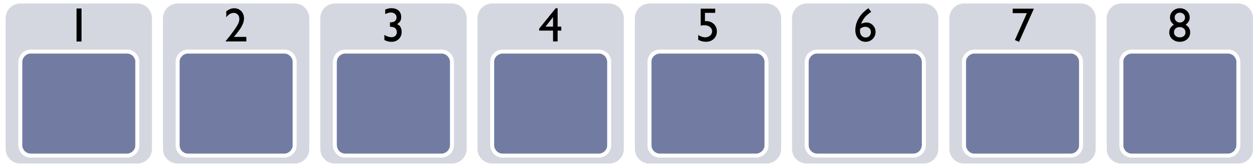
$$\begin{tikzcd}
\boxed{M = M_1 M_2 M_3 \ldots M_n} \arrow[d, "h_1"'] \\
\boxed{\Gamma = e_1 M_{\alpha 1} \,||\, e_2 M_{\alpha 2} \,||\, e_3 M_{\alpha 3} \,||\, \ldots \,||\, e_n M_{\alpha n}} \arrow[d, "h_2"', "\text{NORMAL HASH}"] \\
\boxed{h_{MIX}(M)}
\end{tikzcd}$$

How it works
Components

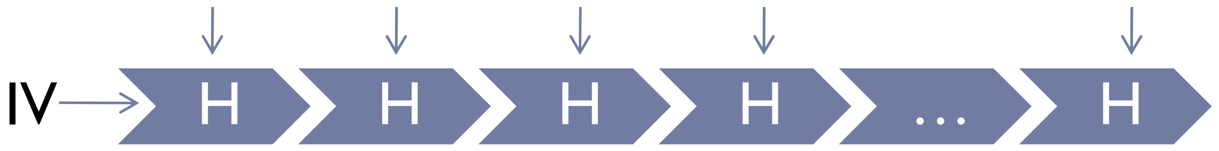
Feeder



Mixer



Hasher

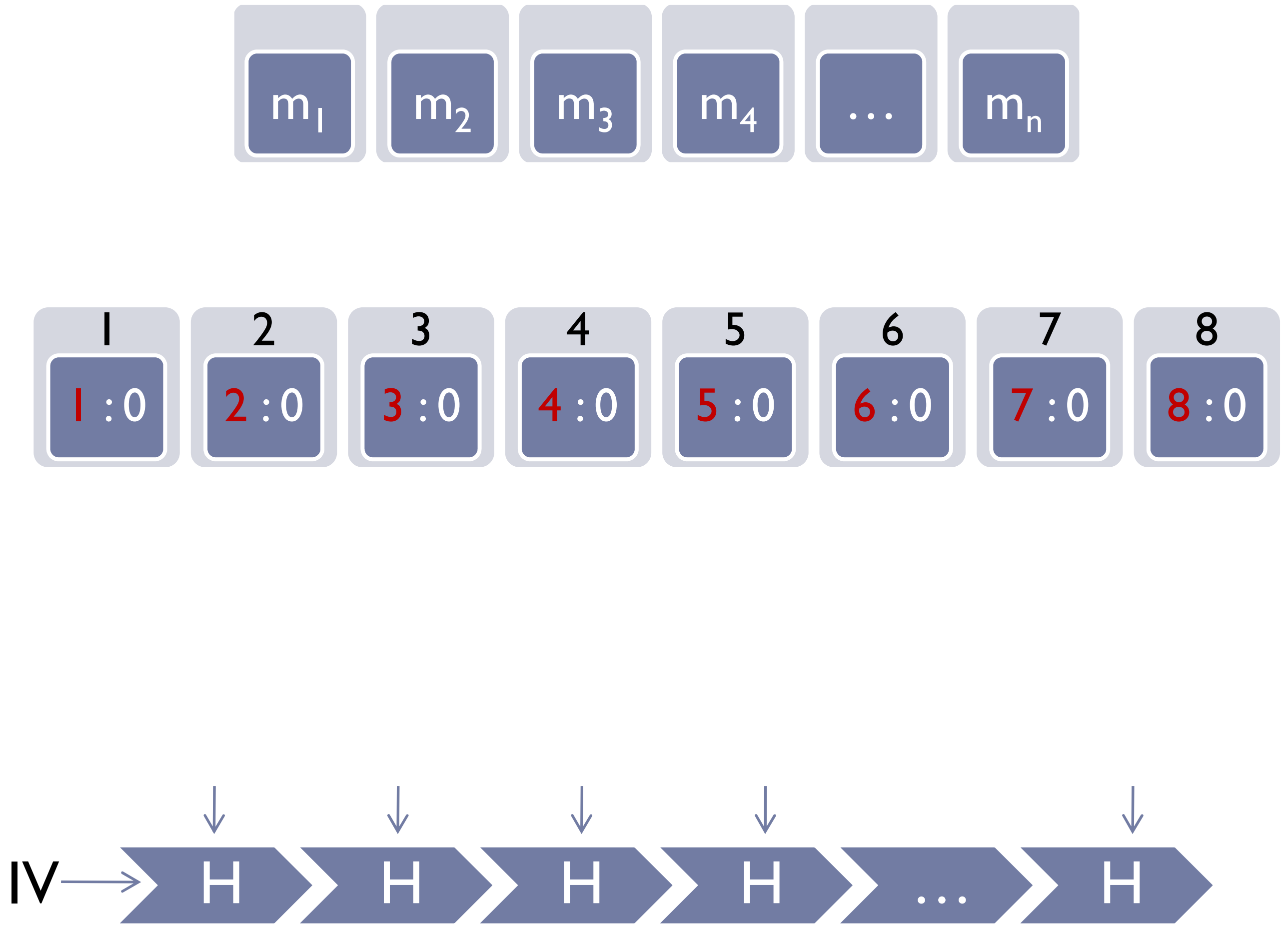


How it works
Initialize

Feeder

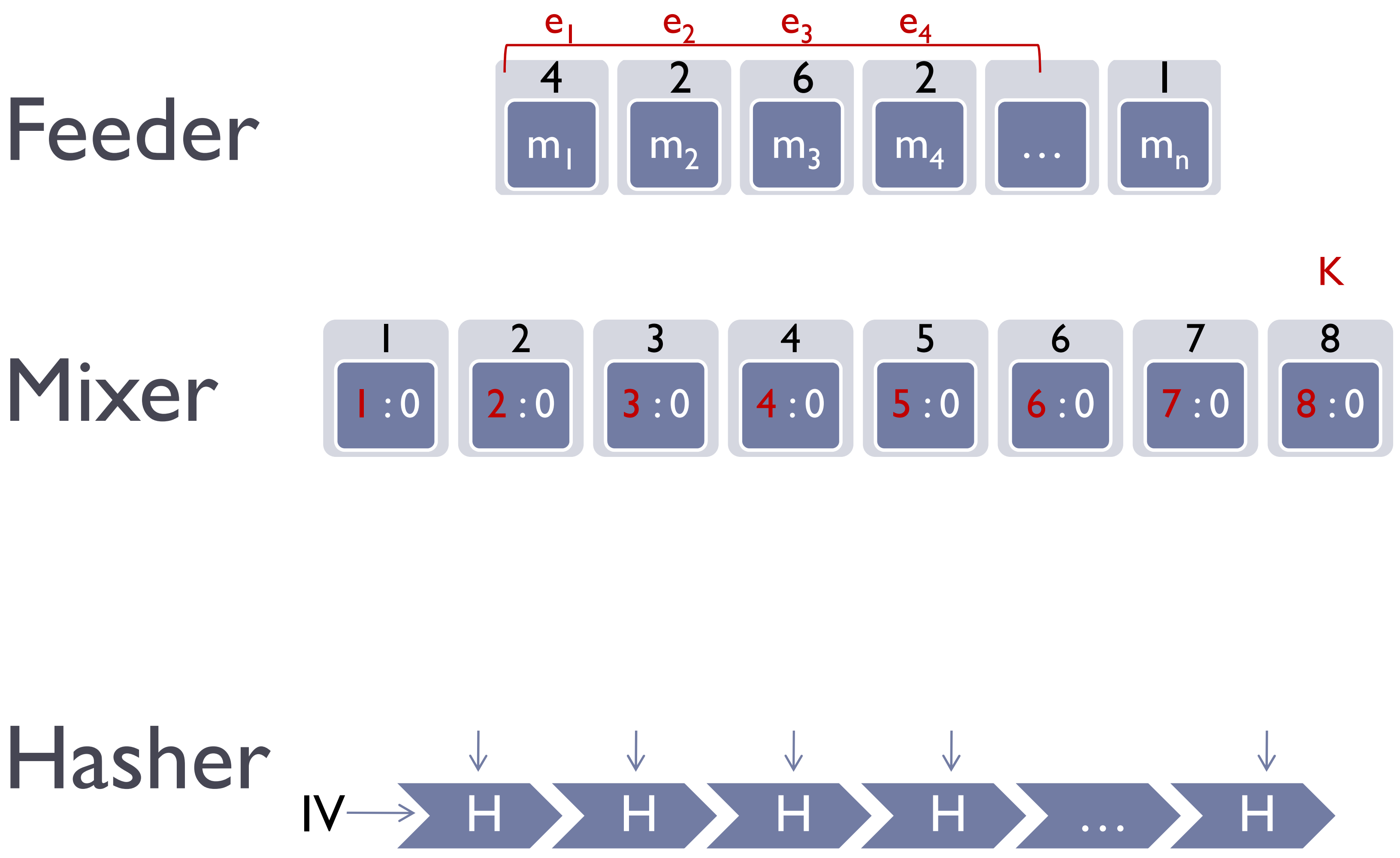
Mixer

Hasher

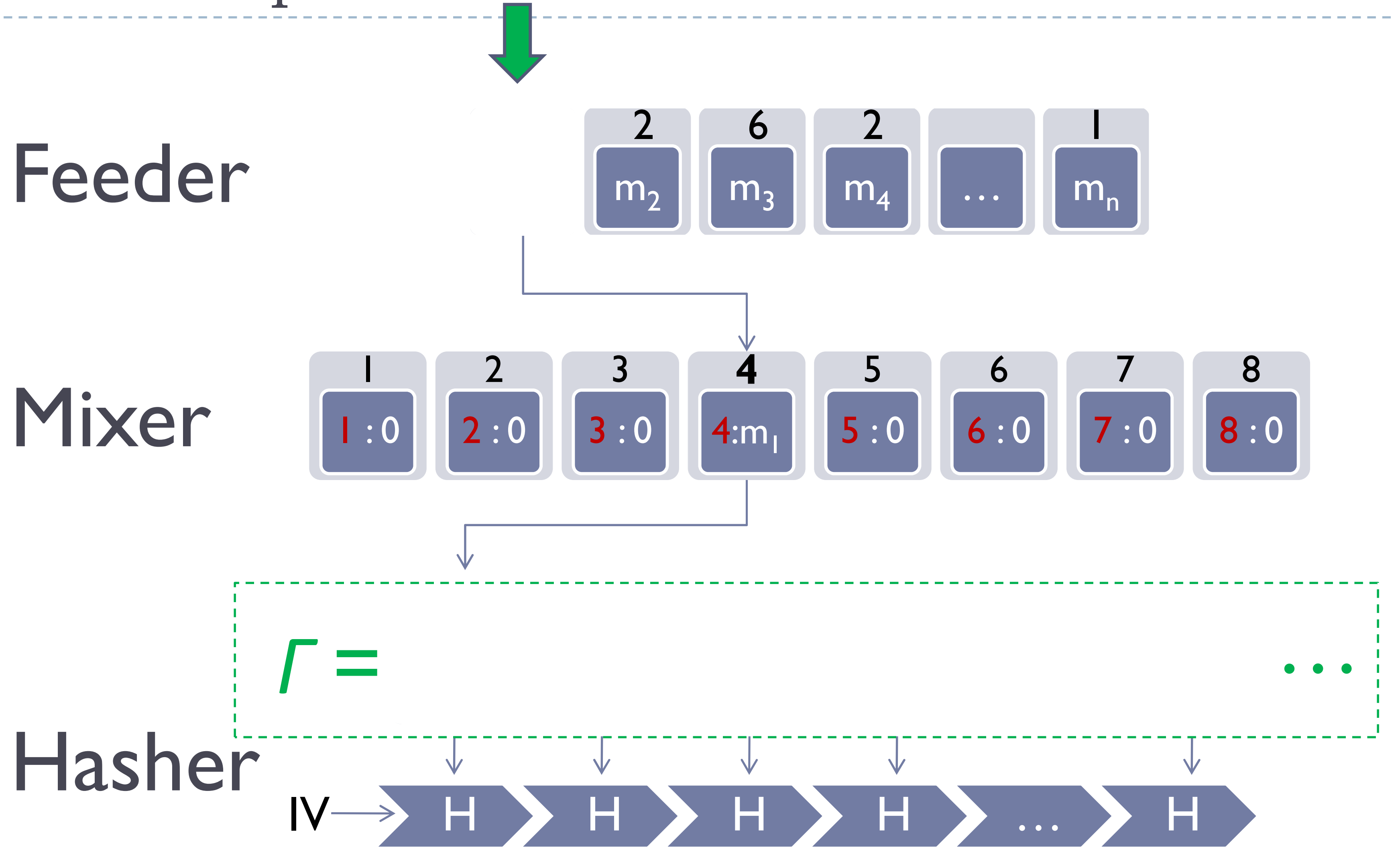


How it works
Determine e values

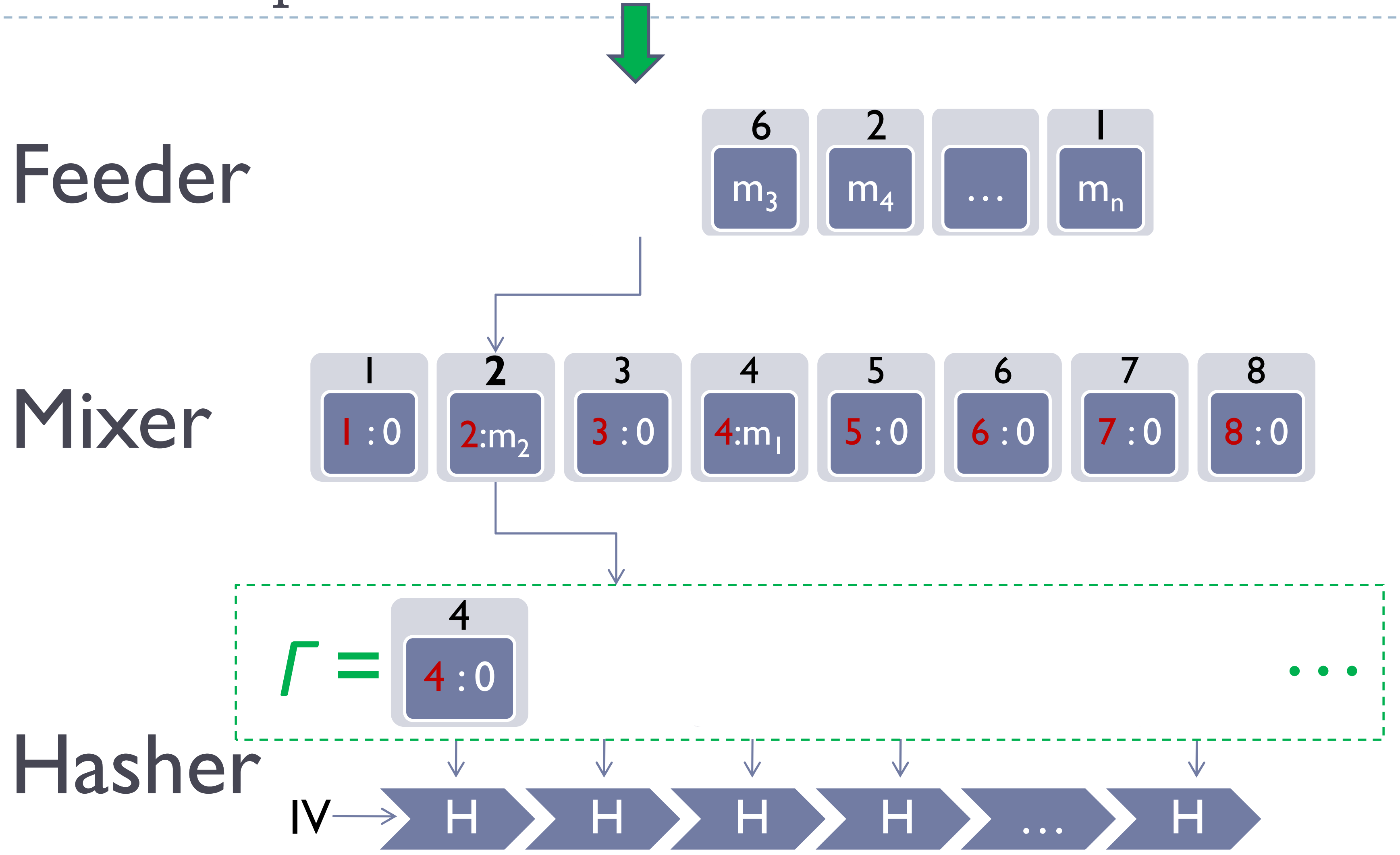
$h_1(m_1\ m_2 \ldots m_k) = e_1 \| e_2 \| \ldots \| e_K$



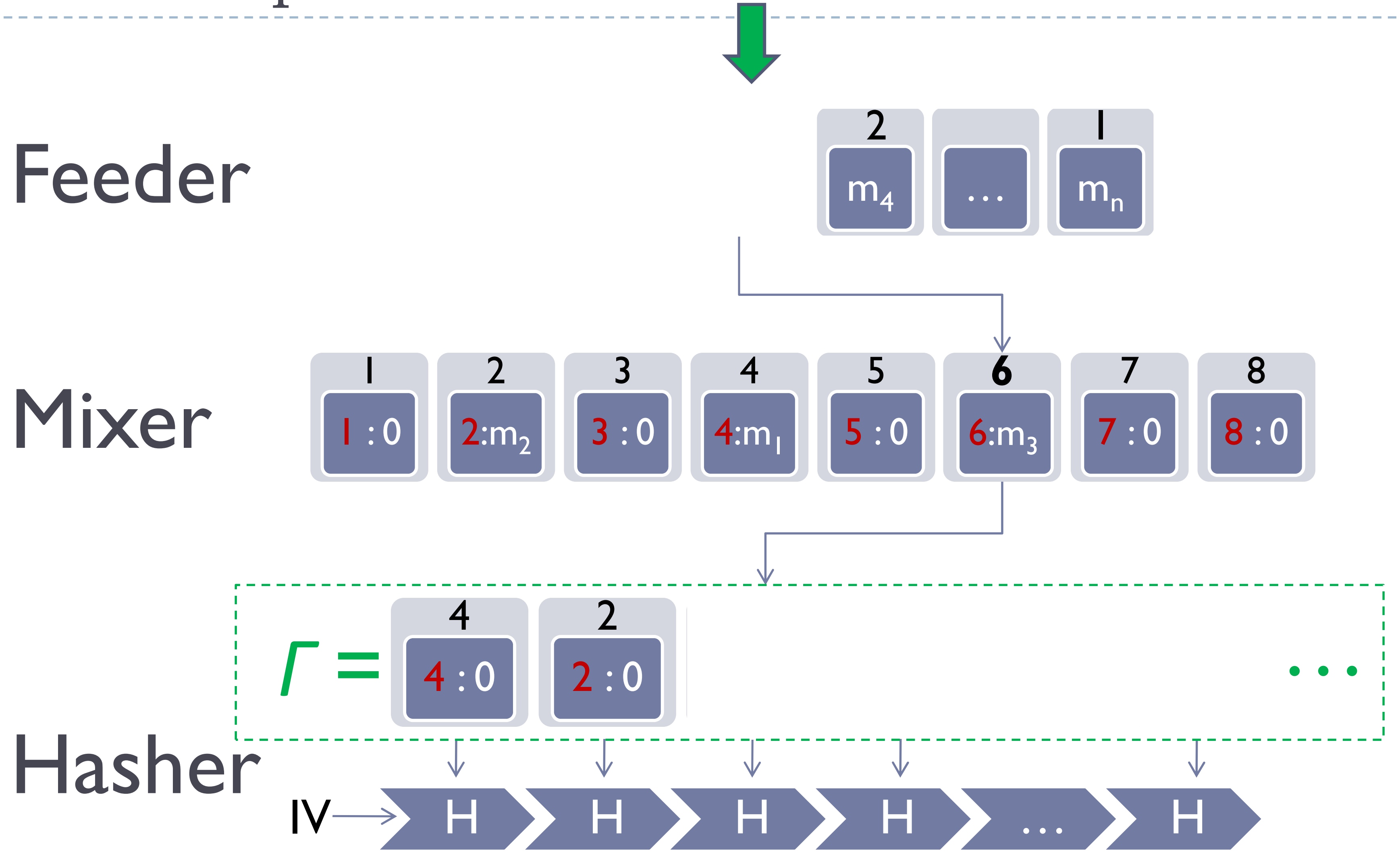
How it works
Step 1



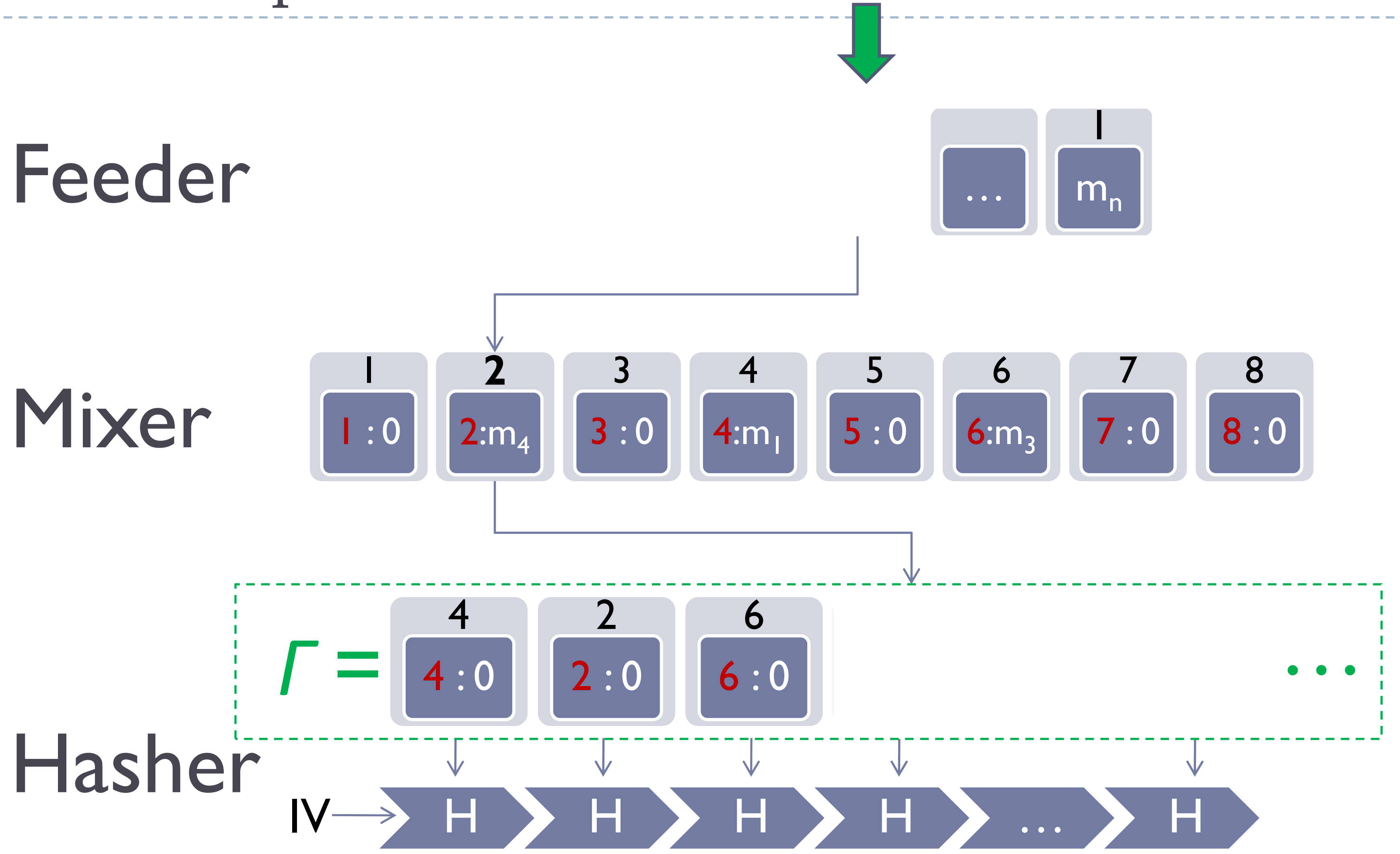
How it works
Step 2



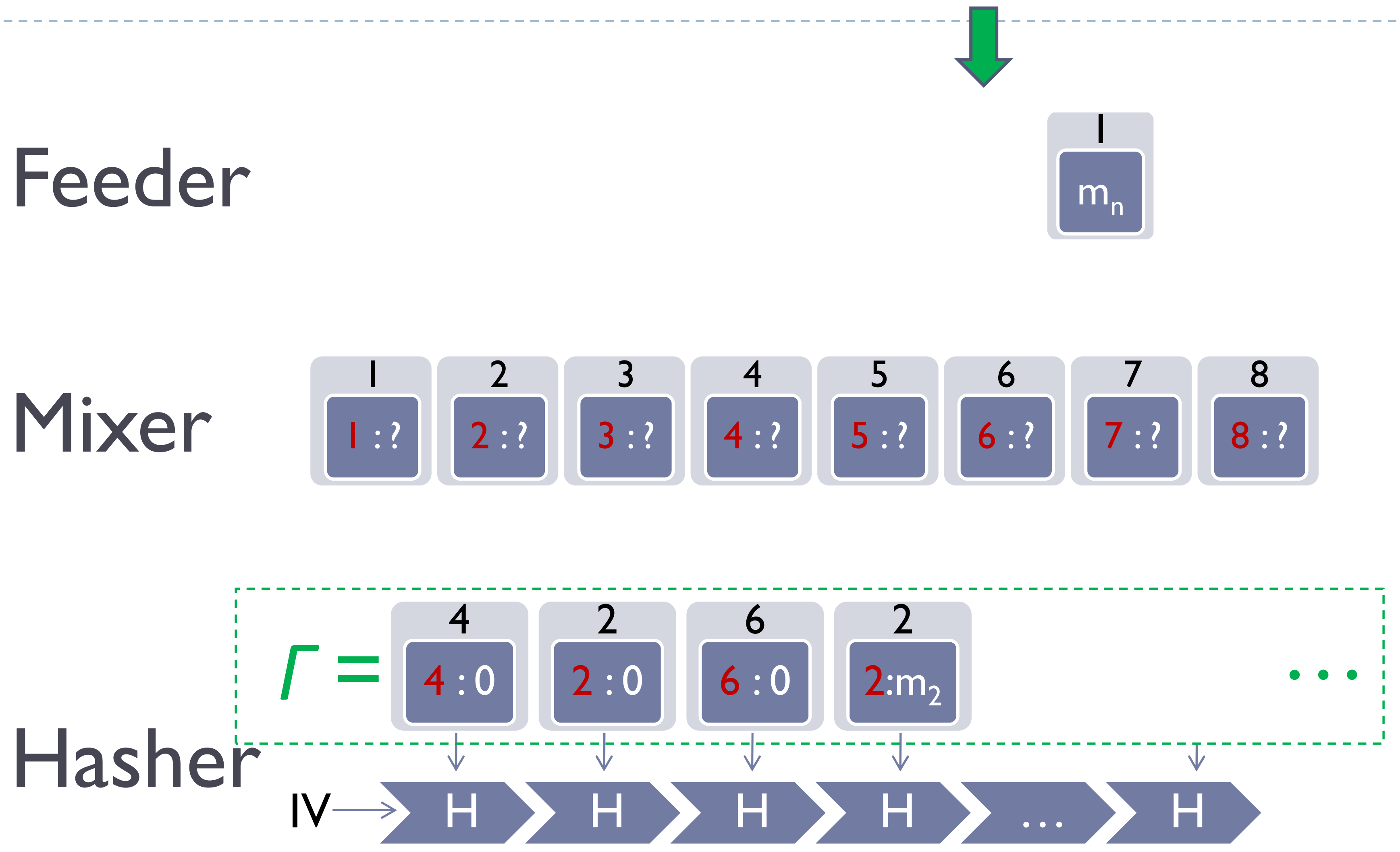
How it works
Step 3



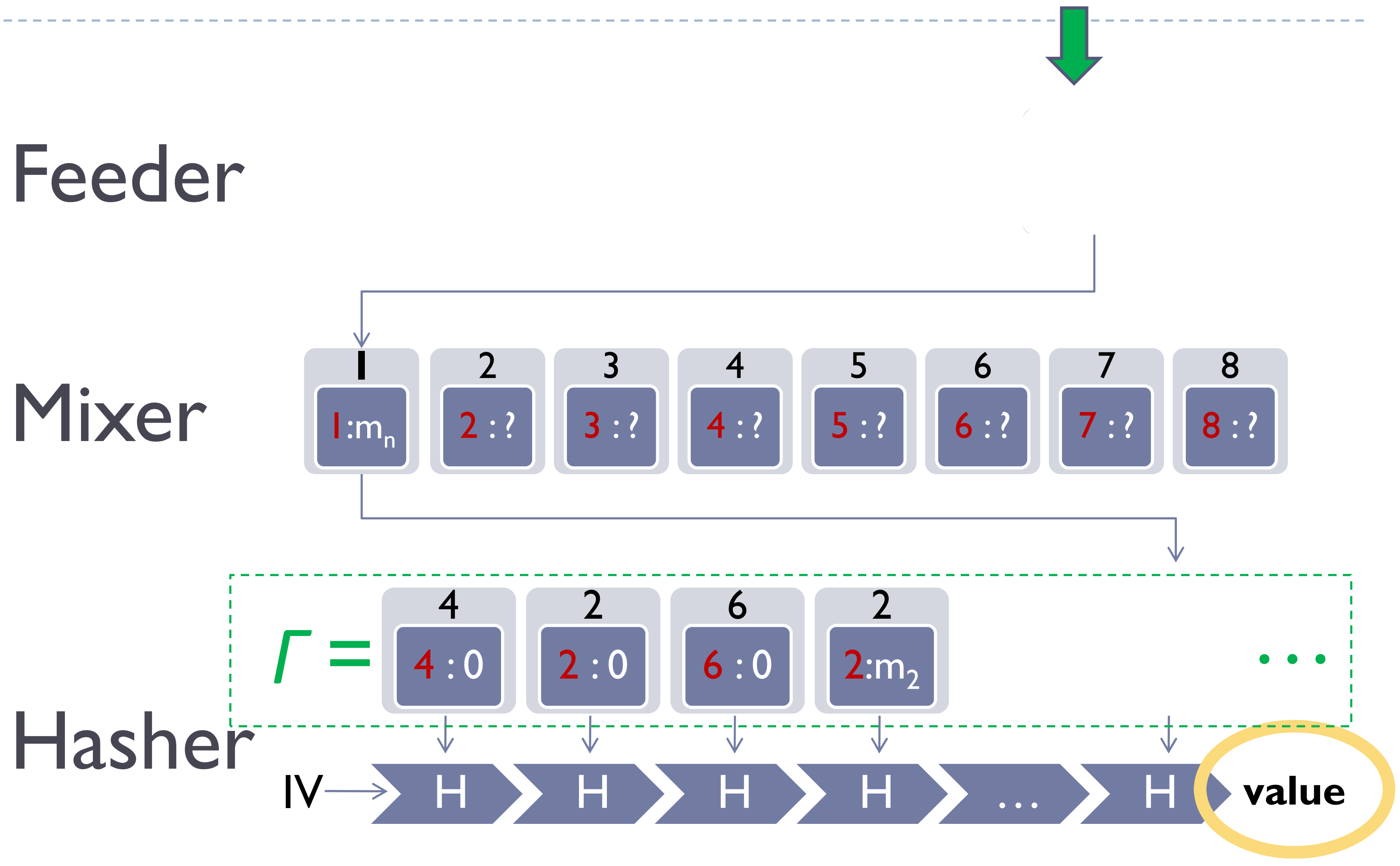
How it works
Step 4



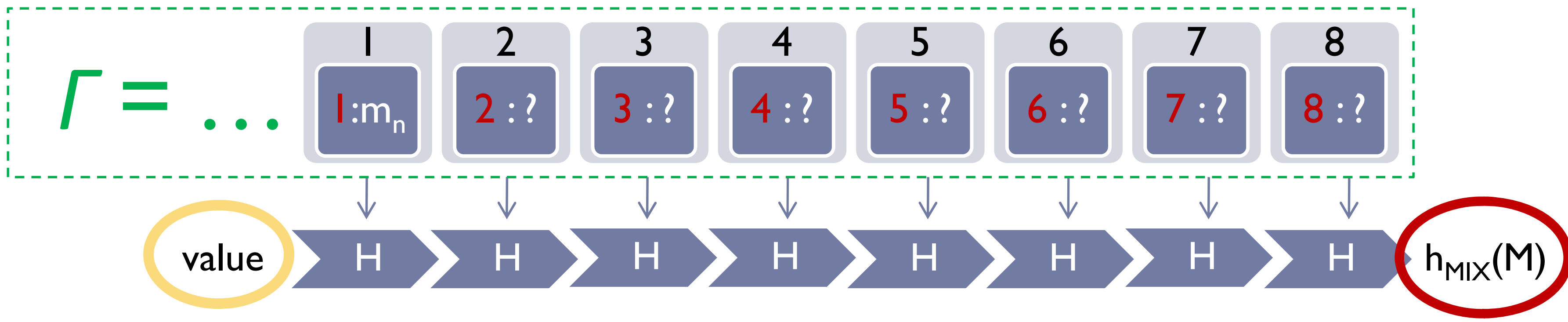
How it works
… continue



How it works
End state



How it works
Finalize



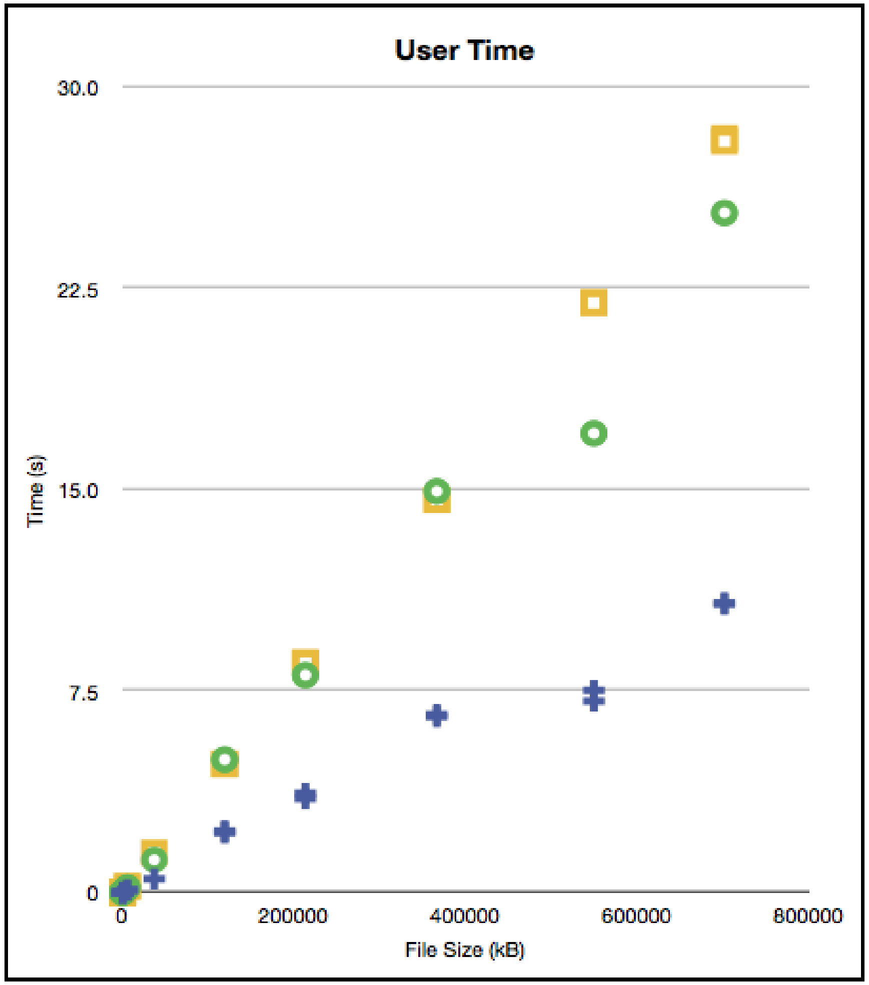
Implementation & Results

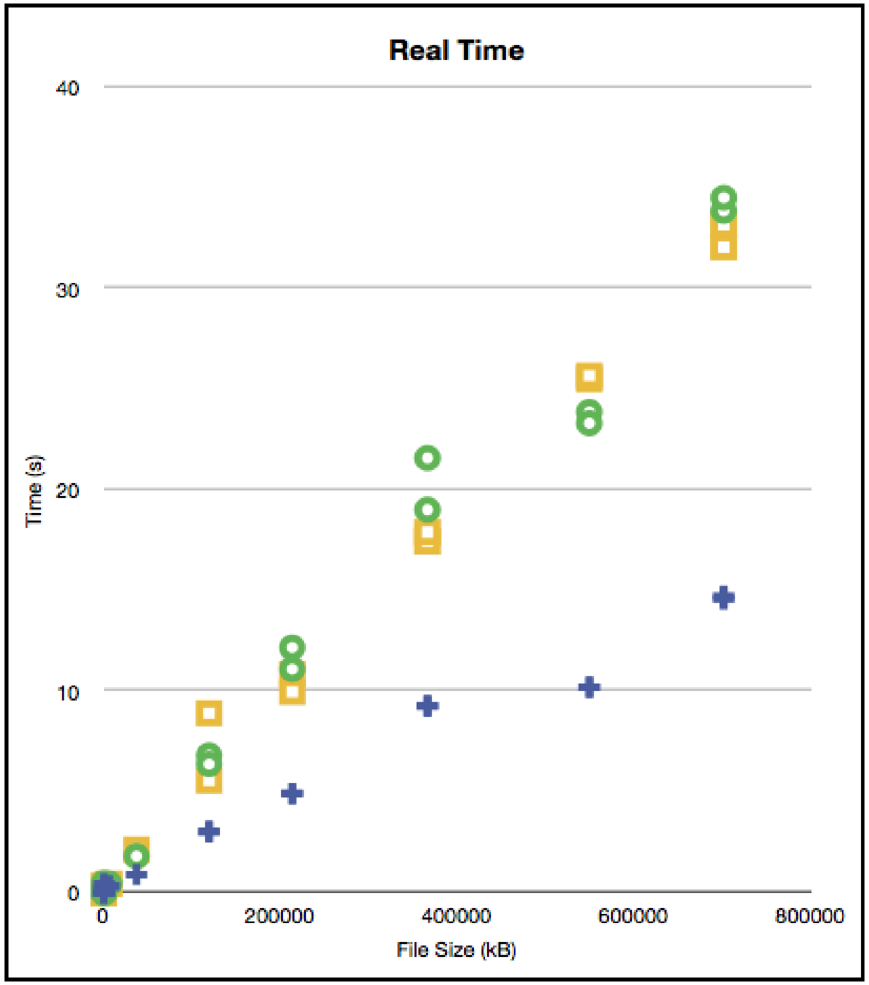
Implementation

- We implemented h_{MIX} in C
  - We used SHA-1 for both h_1 and h_2

- Expect runtime ~2.2 times SHA-1
  - All bits of the message are hashed twice
  - Extra time to move blocks
  - The e values add ~20% to the hashed material

Performance Results





Conclusion

- Theory – with one pass through *M*,
  - h_{MIX} is not provably secure against message extension attacks (see paper)
  - h_{MIX} is not immediately vulnerable to known multicollision attacks
- Practice
  - h_{MIX} is computationally equivalent to hashing *M* twice while reading the file once and using 0.5 KB of internal state