Generic Universal Forgery Attack on Iterative Hash-based MACs

Thomas Peyrin and Lei Wang

NANYANG TECHNOLOGICAL UNIVERSITY

EUROCRYPT 2014

Outline

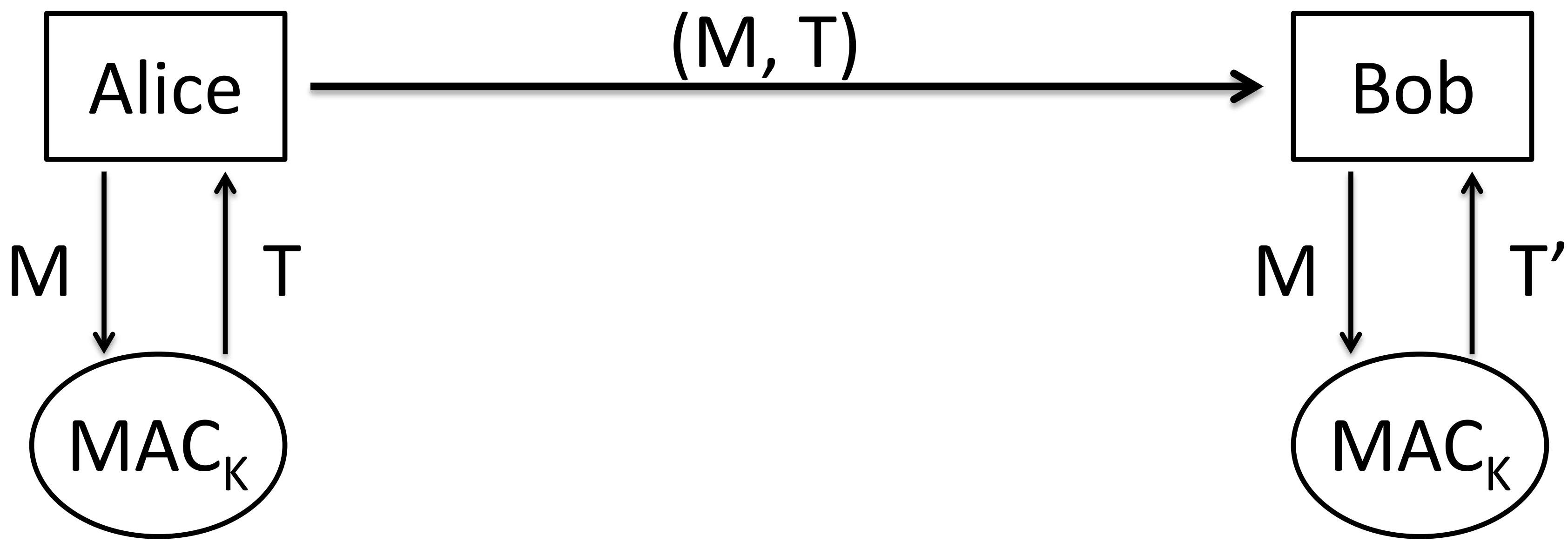
- Introduction
  - hash-based MACs
  - known results on hash-based MACs
  - our contributions
- Universal forgery attacks
  - attack overview
  - new technical ideas
- Conclusion

Outline

- **Introduction**
  - **hash-based MACs**
  - **known results on hash-based MACs**
  - **our contributions**
- Universal forgery attacks
  - attack overview
  - new technical ideas
- Conclusion

Message Authentication Code (MAC)

- **Symmetric-key** cryptographic protocol
  - ➢ Alice and Bob share a secret key **K**.
- Provide the **authenticity** and the **integrity**
  - ➢ Bob verifies if T=T' holds.



How to Build MACs

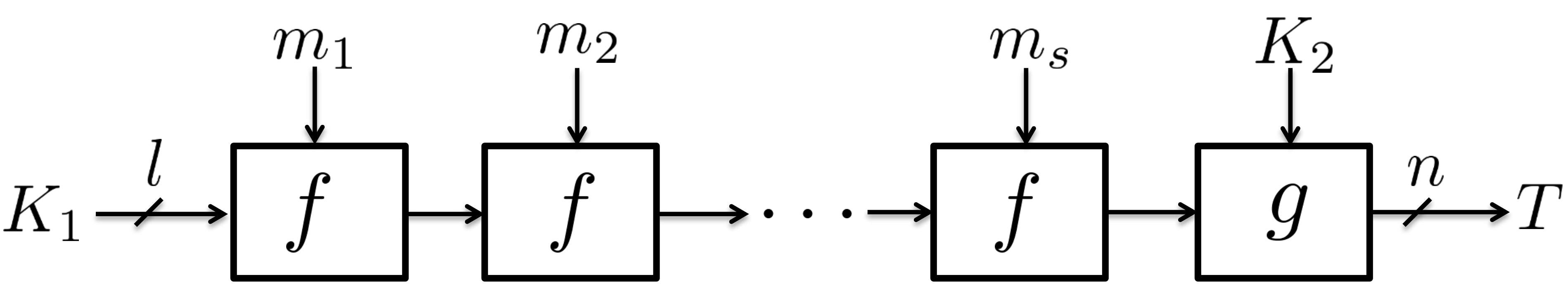
- From hash functions
  - HMAC, Sandwich-MAC, Envelop-MAC
- From block ciphers
  - CBC-MAC, CMAC, PMAC
- From universal hash functions
  - UMAC, VMAC, Poly1305
- Dedicated design
  - SQUASH, SipHash

How to Build MACs

- **From hash functions**
  - **HMAC, Sandwich-MAC, Envelop-MAC**
- From block ciphers
  - CBC-MAC, CMAC, PMAC
- From universal hash functions
  - UMAC, VMAC, Poly1305
- Dedicated design
  - SQUASH, SipHash

Iterative Hash-based MACs

- A simplified description
  - K_1, K_2 : initialization and finalization keys
  - f , g : public deterministic functions
  - l : internal state size
  - n : tag size



Well-known Example HMAC

- Designed by BCK96

- Standardized by ANSI, IETF, ISO, NIST

- Implemented in SSL, TLS, IPSec…

Known Results of Hash-based MACs

- Pseudo-Random-Function **proof**
  - **lower** security bound
  - up to the birthday bound $O(2^{l/2})$
  - implication to most security notions
  - HMAC, Sandwich-MAC, etc

Known Results of Hash-based MACs

- **Generic attacks** on each security notion
  - **upper** security bound
  - distinguishing-R: $O(2^{l/2})$
  - distinguishing-H: $O(2^{l/2})$
  - existential forgery: $O(2^{l/2})$
  - universal forgery: $O(2^{l})$
  - key recovery: $O(2^{k})$

Known Results of Hash-based MACs

- **Generic attacks** on each security notion
  - **upper** security bound
  - distinguishing-R: $O(2^{l/2})$ **tight**
  - distinguishing-H: $O(2^{l/2})$ **tight**
  - existential forgery: $O(2^{l/2})$ **tight**
  - universal forgery: $O(2^{l})$
  - key recovery: $O(2^{k})$

Our Contributions

- Generic attacks on each security notion
  - upper security bound
  - distinguishing-R: $O(2^{l/2})$ tight
  - distinguishing-H: $O(2^{l/2})$ tight
  - existential forgery: $O(2^{l/2})$ tight
  - **universal forgery**: $O(2^{l})$ $\mathbf{O(2^{5l/6})}$
  - key recovery: $O(2^{k})$

Our Technical Contributions

- Collision-detection-based attacks
  - ➢ dis-R and existential forgery by PvO96
  - ➢ dis-H in single-key setting by NSW+13
- **Functional-graph-based** attacks
  - ➢ indifferentiability of HMAC by DRS+12
  - ➢ dis-R/H and existential forgery of HMAC in related-key setting by PSW12
  - ➢ dis-H in single-key setting by LPW13
  - ➢ **universal forgery in this paper:**

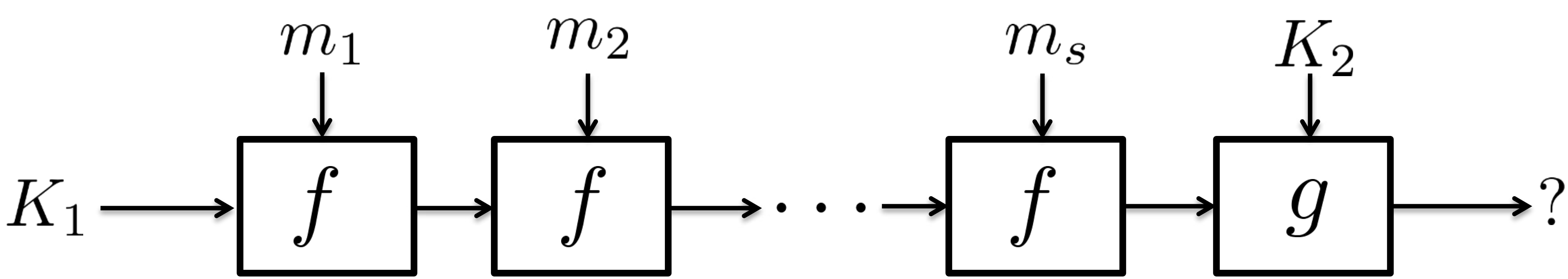
    extract more information than just cycle structure

Outline

- Introduction
  - hash-based MACs
  - known results on hash-based MACs
  - our contributions
- **Universal forgery attacks**
  - **attack overview**
  - new technical ideas
- Conclusion

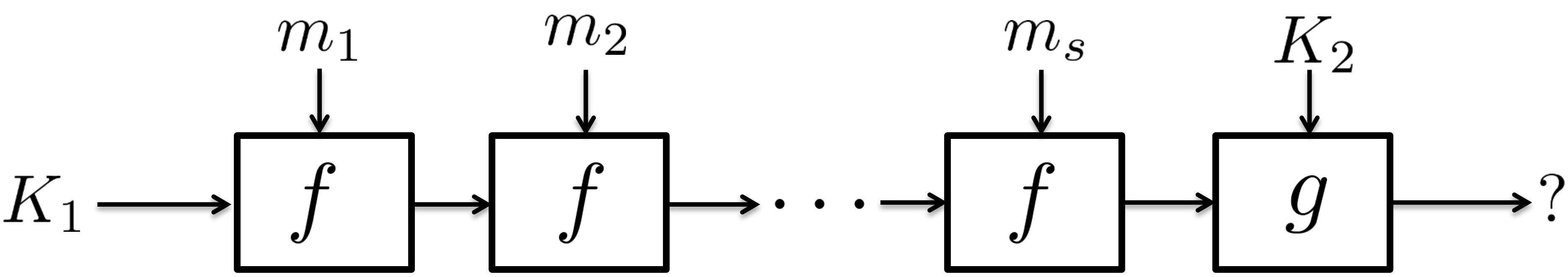
Universal Forgery Setting

- The adversary
  - **given** a message M $(=m_1||m_2||\bullet\bullet\bullet||m_s)$
  - can interact with MAC
  - can not query M to MAC
  - to produce a valid tag T for M



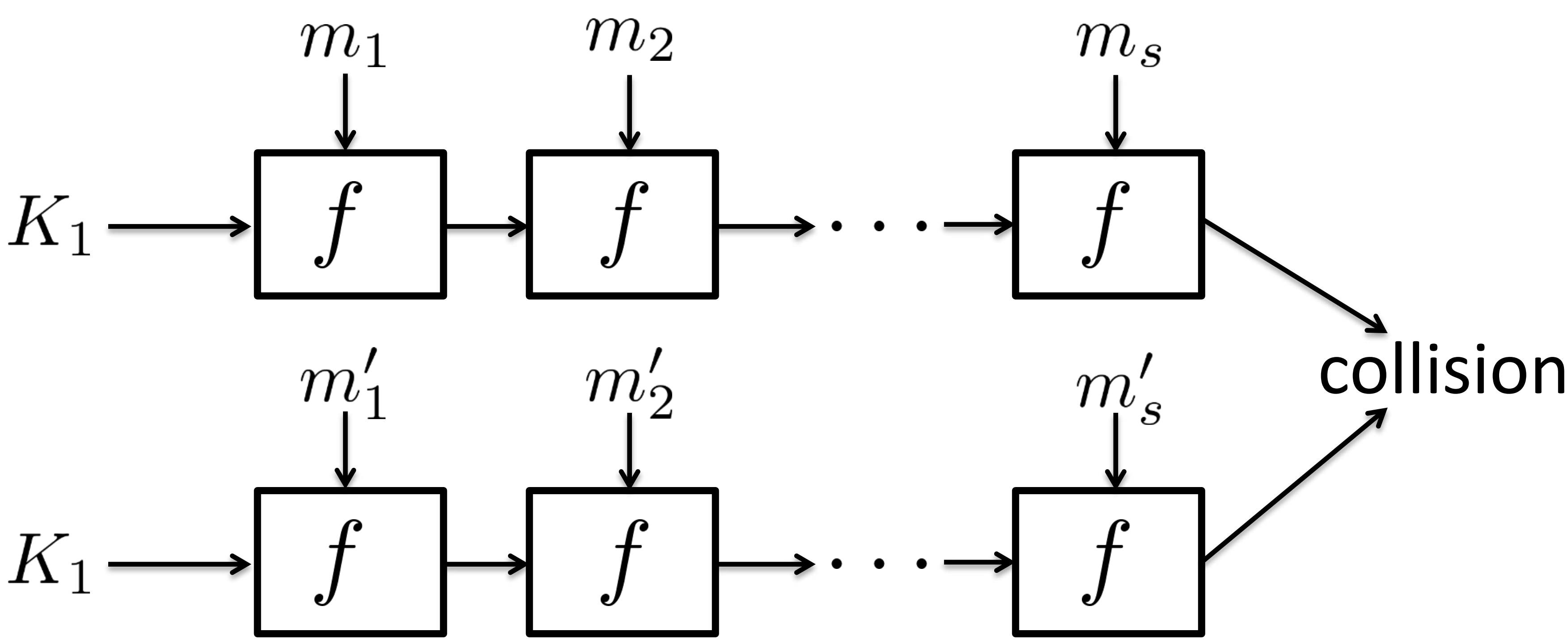
Universal Forgery Setting

- The adversary **must be able to forge any message**
  - **given** a message M (=$m_1||m_2||\bullet\bullet\bullet||m_s$)
  - can interact with MAC
  - can not query M to MAC
  - to produce a valid tag T for M



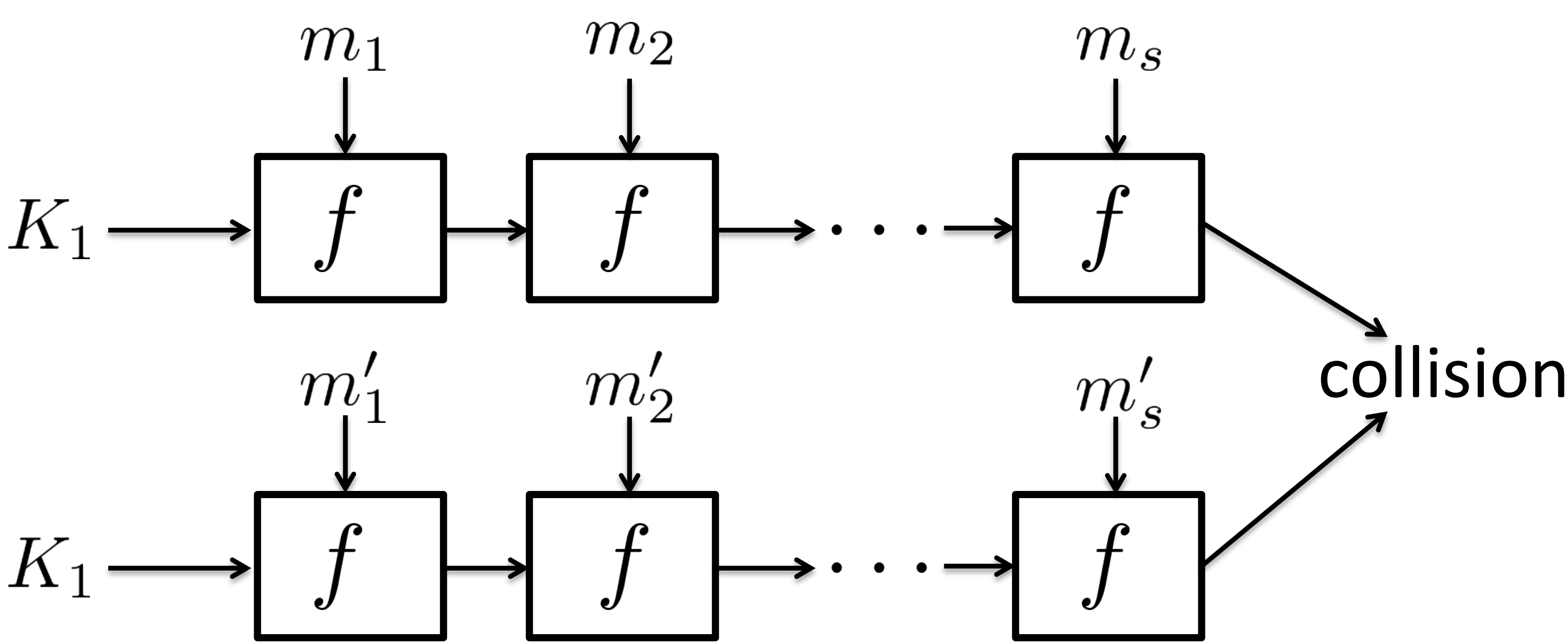
Main Idea

- Construct a second preimage M' for M
  - ➢ $\mathrm{MAC}_{K_1,K_2}(M)=\mathrm{MAC}_{K_1,K_2}(M')$
- Query M' to MAC to obtain a valid tag for M



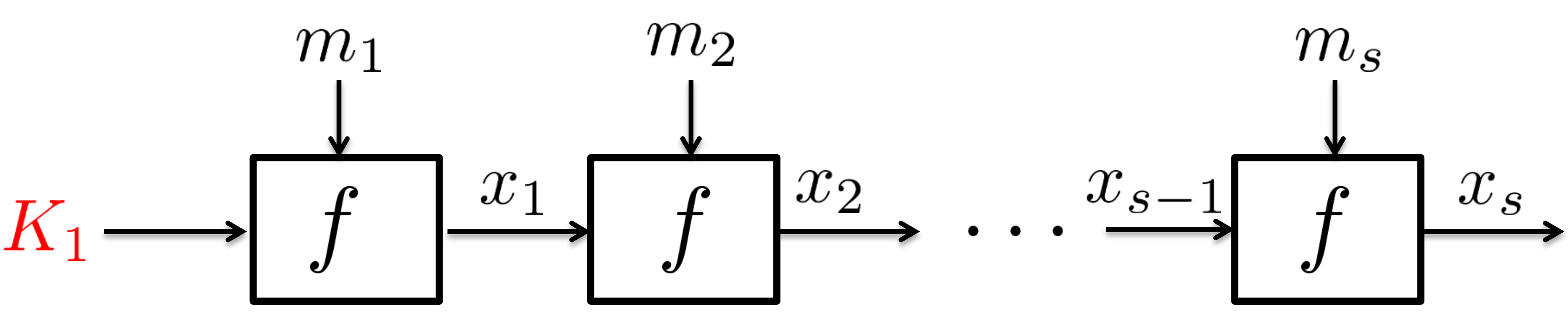
Main Idea

- Construct a second preimage M' for M
  - $\mathrm{MAC}_{K_1,K_2}(M) = \mathrm{MAC}_{K_1,K_2}(M')$
- Query M' to MAC to obtain a valid tag for M



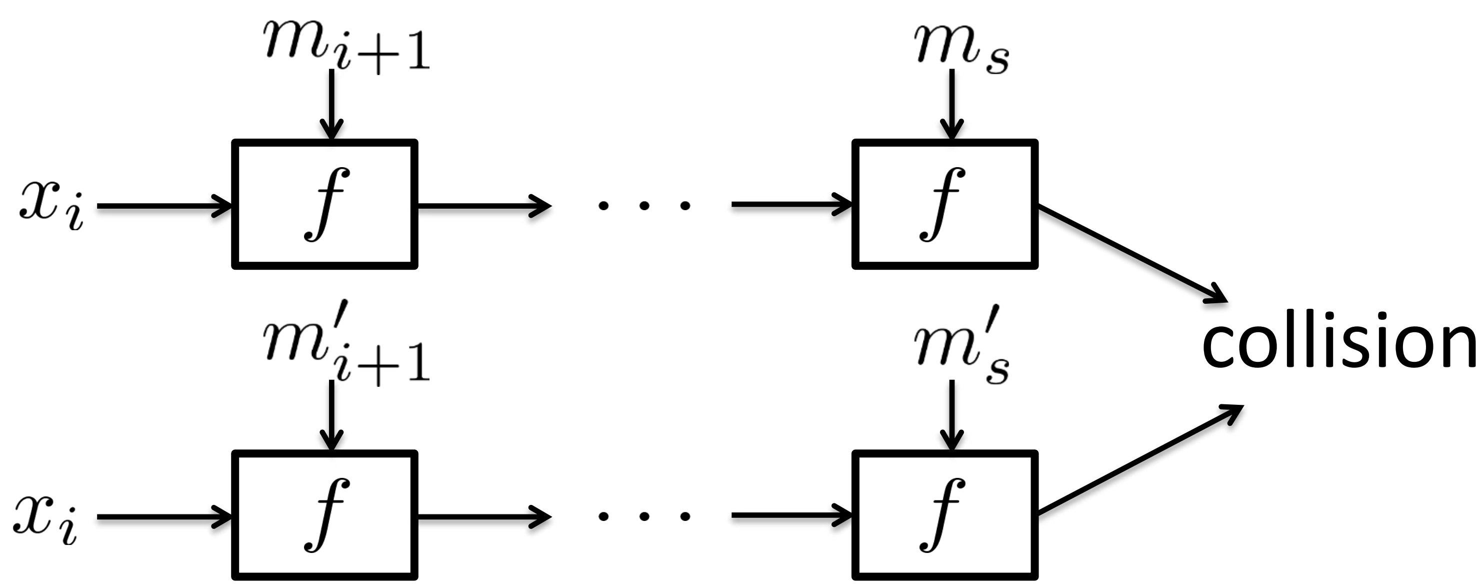
Difficulty of Constructing such a M'

- Essentially a second preimage attack on a **keyed** iterative hash function
  - ➢ internal states $x_1, \ldots, x_s$ are **unknown**
- Second preimage attack on **public** iterative hash function has been published by KS05
  - ➢ knowledge of internal states is **necessary**



How to Construct such a M'

- Recover some internal state x_i
  - ➢ states $x_{i+1}, \ldots, x_s$ are then known
- Apply previous second preimage attack on public iterative hash function to get

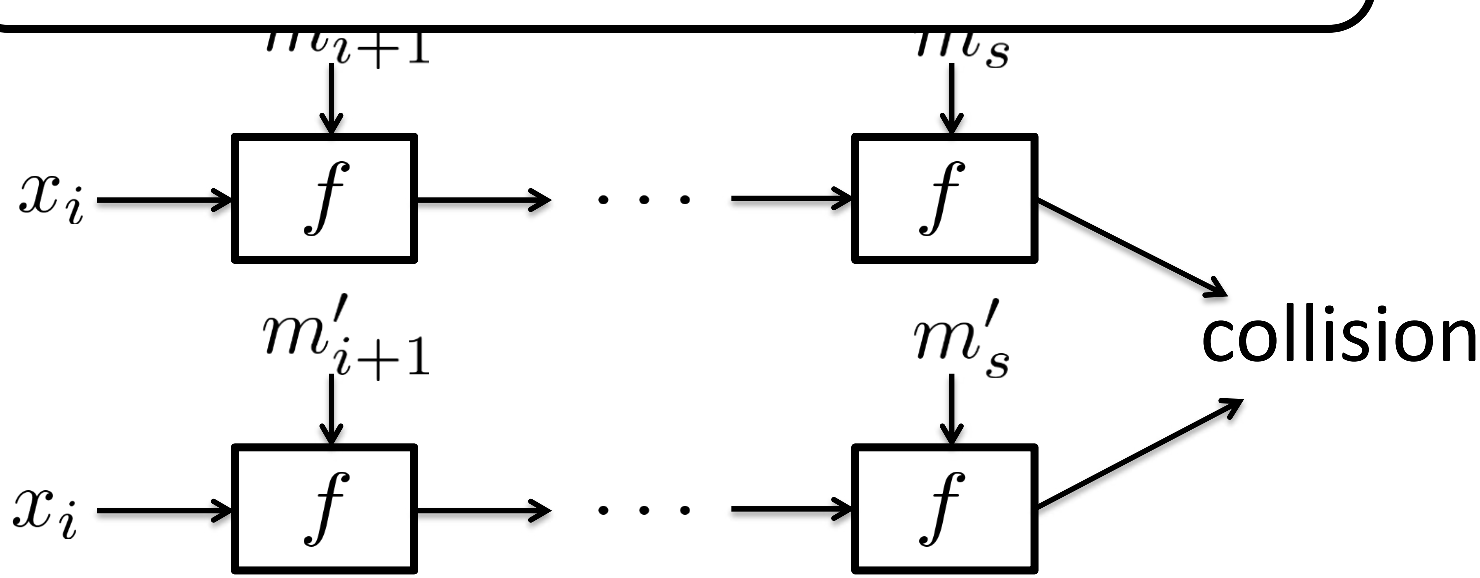


- Construct $M' = m_1 \| \cdots \| m_i \| m'_{i+1} \| \cdots \| m'_s$

How to Construct such a M'

- Recover some internal state x_i
  - state $x_{i+1}, \dots, x_s$ are then known
- App ublic iter

Our main technical contribution



- Construct $M' = m_1 \| \cdots \| m_i \| m'_{i+1} \| \cdots \| m'_s$

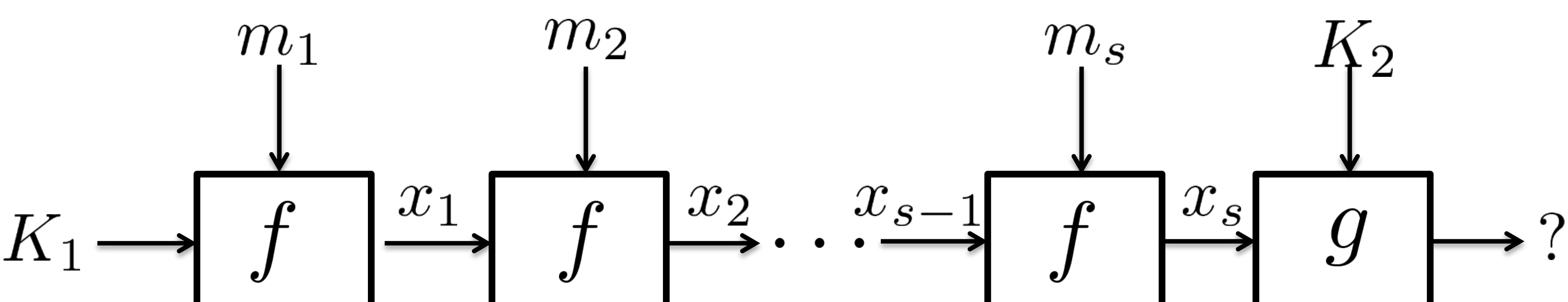
Overview of Our Attacks

- Firstly recover some internal state x_i

- Secondly find $m'_{i+1}\|\cdots\|m'_s$ so that

$$f(\cdots f(x_i, m_{i+1}), \cdots, m_s) = f(\cdots f(x_i, m'_{i+1}), \cdots, m'_s)$$

- Finally query $M' = m_1\|\cdots\|m_i\|m'_{i+1}\|\cdots\|m'_s$ to get a valid tag for the challenge message M

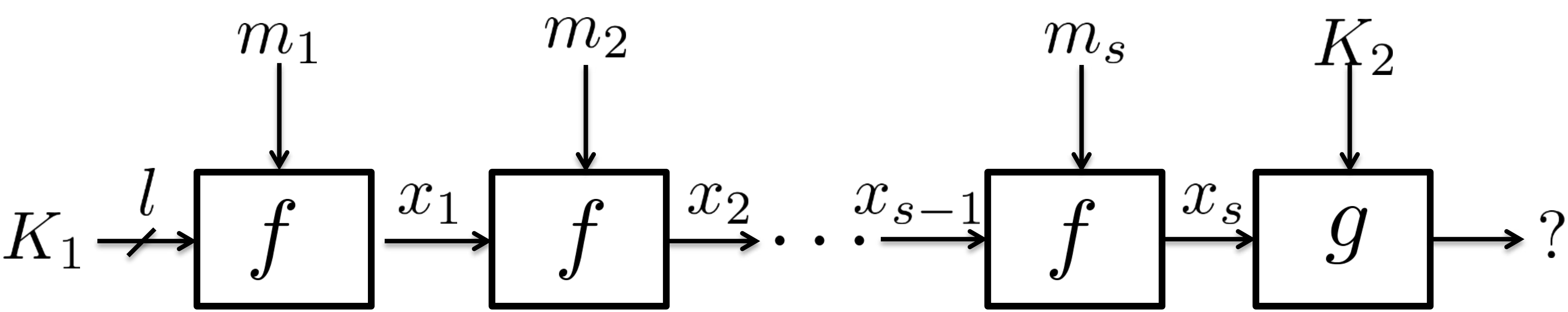


Outline

- Introduction
  - hash-based MACs
  - known results on hash-based MACs
  - our contributions
- Universal forgery attacks
  - attack overview
  - **new technical ideas**
- Conclusion

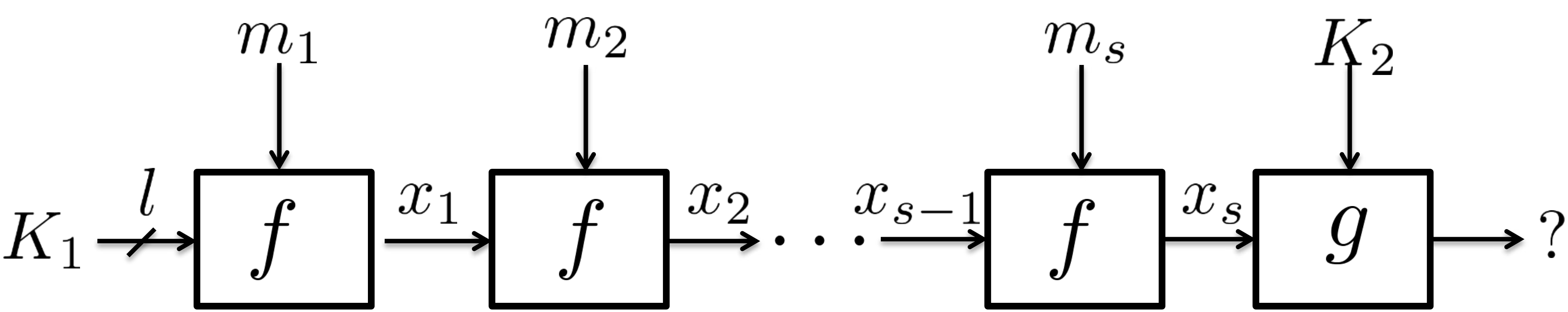
How to Recover an Internal State

- Offline select $2^l/s$ distinct values $y_1, \cdots, y_{2^l/s}$
  - one pair $x_i = y_j$ with a good probability



How to Recover an Internal State

- Offline select $2^l/s$ distinct values $y_1, \cdots, y_{2^l/s}$
  - one pair $x_i = y_j$ with a good probability
- Identity such a pair and get the value of x_i
  - in total 2^l pairs.
  - naive method to verify each pair costs 2^l



How to Recover an Internal State

- Offline select $2^l/s$ distinct values $y_1, \cdots, y_{2^l/s}$
  - ➢ one pair $x_i = y_j$ with a good probability
- Identity such a pair and get the value of x_i
  - ➢ in total 2^l pairs.
  - ➢ naive method to verify each pair costs 2^l
  - ➢ we use a new property to match $\{x_1, \ldots, x_s\}$ and $\{y_1, \cdots, y_{2^l/s}\}$ simultaneously

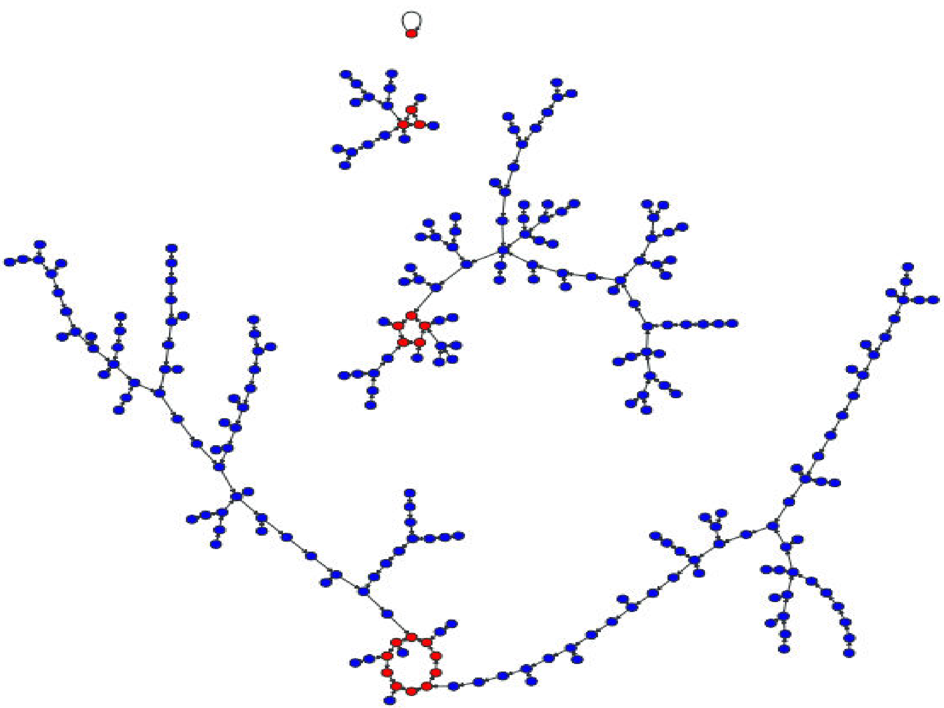
How to Recover an Internal State

- Offline select $2^l/s$ distinct values $y_1, \cdots, y_{2^l/s}$
  - one pair $x_i = y_j$ with a good probability
- Identity such a pair and get the value of x_i
  - in total 2^l pairs.
  - naive method to verify each pair costs 2^l
  - we use a new property to match $\{x_1, \ldots, x_s\}$ and $\{y_1, \cdots, y_{2^l/s}\}$ simultaneously

Height of nodes in functional graph

Functional Graph

- f : a l-bit to l-bit function
- iterate f: $x_i = f(x_{i-1})$



➢ #components: $O(l)$

➢ largest components:

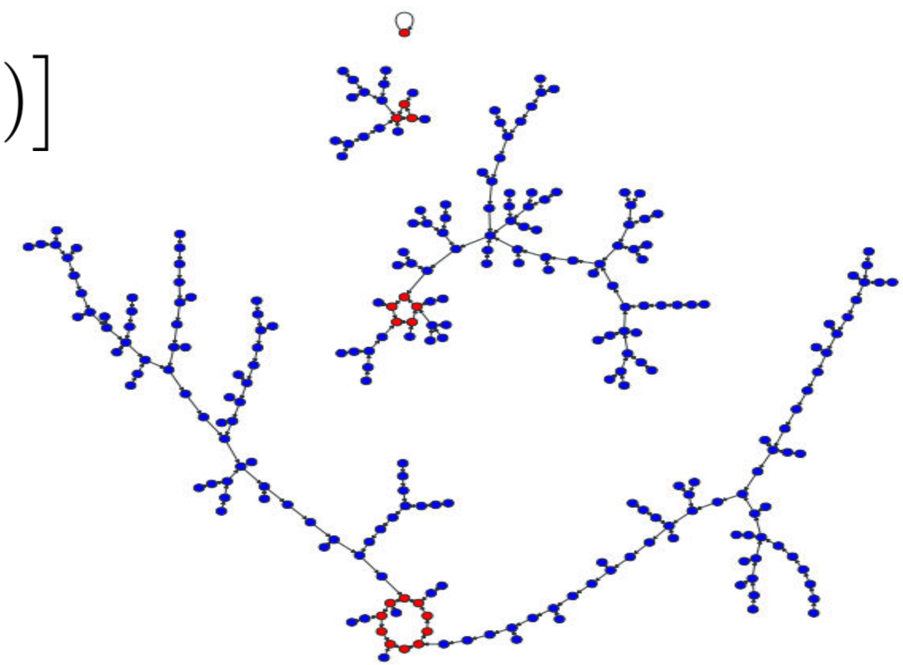
#nodes: $2/3 \cdot 2^l$

#cycle nodes: $2^{l/2}$

longest path: $O(2^{l/2})$

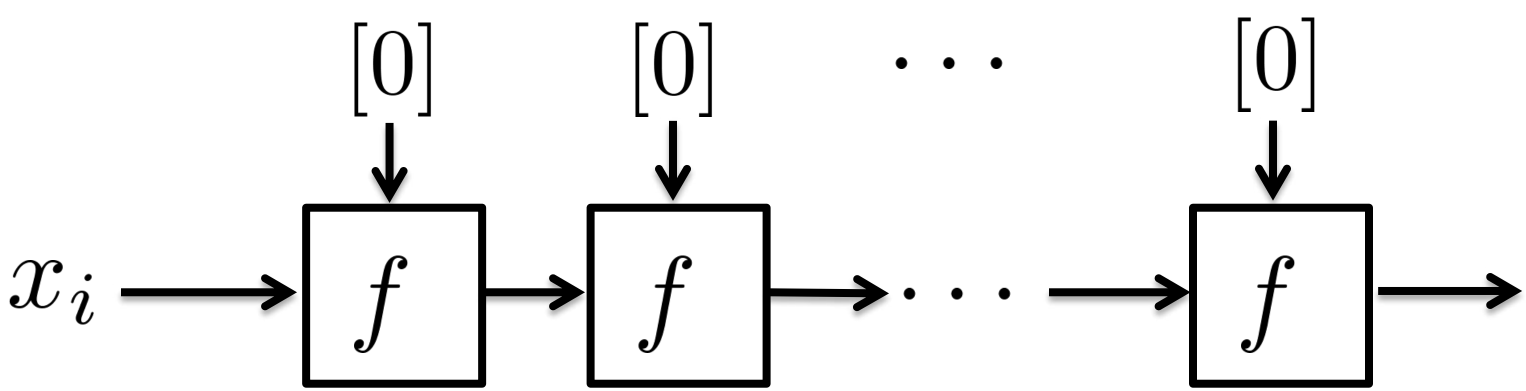
Height of Nodes in Functional Graph

- The height of a node x is the number of nodes from x to the cycle of its component.
  - ➢ each node has a single path to its cycle
  - ➢ height of cycle nodes is 0
- height range: $[0, O(2^{l/2})]$



How to Recover an Internal State

- Use functional graph of f with a constant message
  - e.g., $f(\cdot, 0)$: $l\text{-bit to } l\text{-bit}$ function
  - denoted as $f_{[0]}$



How to Recover an Internal State

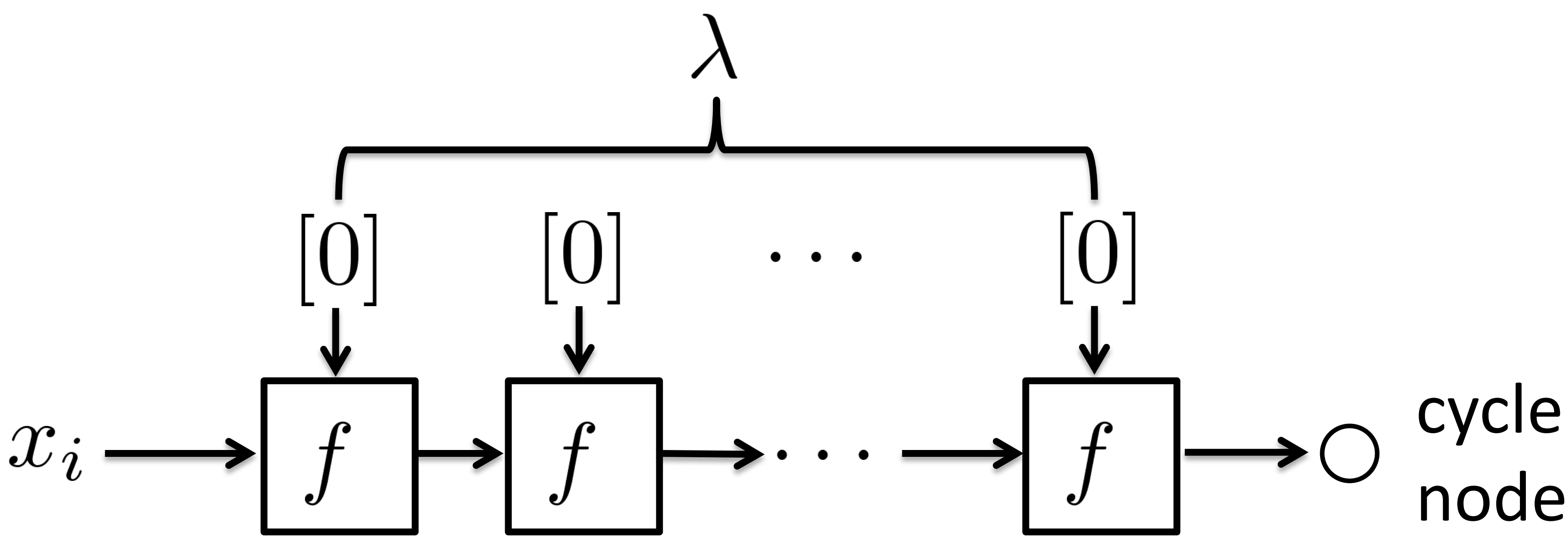
- Recover the height of $\{x_1, x_2, ..., x_s\}$
- Select $\{y_1, y_2, ..., y_{2^l/s}\}$ with their height
- **Match the height** between $\{x_1, x_2, ..., x_s\}$ and $\{y_1, y_2, ..., y_{2^l/s}\}$
  - ➢ #pairs left is upper bounded by $O(2^{5l/6})$
  - ➢ details are omitted, and referred to paper.
- Examine each remaining pair, and identify the pair $x_i = y_j$ to recover x_i

How to Recover an Internal State

- Recover the height of $\{x_1, x_2, \ldots, x_s\}$
- Select $\{y_1, y_2, \ldots, y_{2^l/s}\}$ with their height
- Match the height between $\{x_1, x_2, \ldots, x_s\}$ and $\{y_1, y_2, \ldots, y_{2^l/s}\}$
  - #pairs left is upper bounded by $O(2^{5l/6})$
  - details are omitted, and referred to paper.
- Examine each remaining pair, and identify the pair $x_i = y_j$ to recover x_i

How to Recover Height of x_i

- Find the **minimum** number of iterations λ so that the output value is a cycle node.

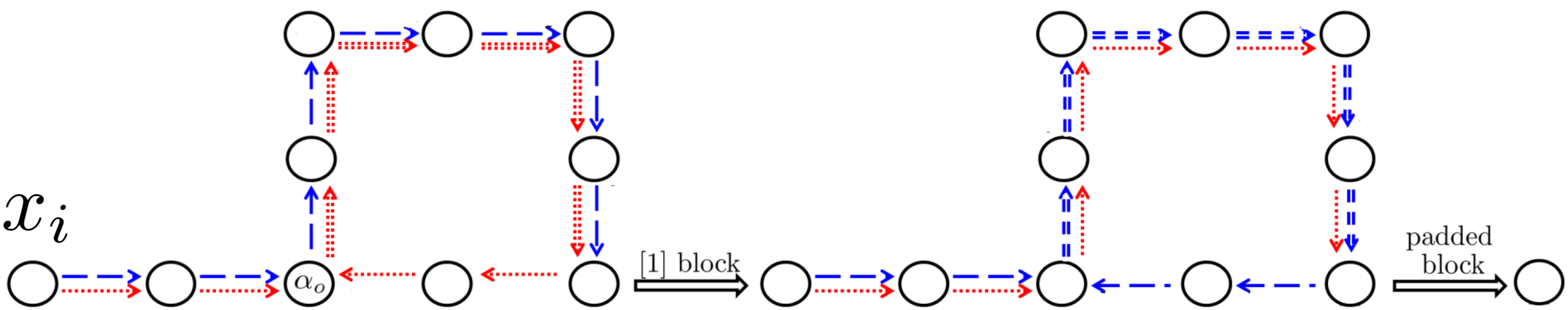


How to Recover Height of x_i

- Use two messages, constructed by appending $m_1 \| \cdots \| m_i$ with
  - L : the cycle length of the largest component

$$[0]^{2^{l/2}} \quad \| \quad [0]^{2L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{L}$$

$$[0]^{2^{l/2}} \quad \| \quad [0]^{L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{2L}$$

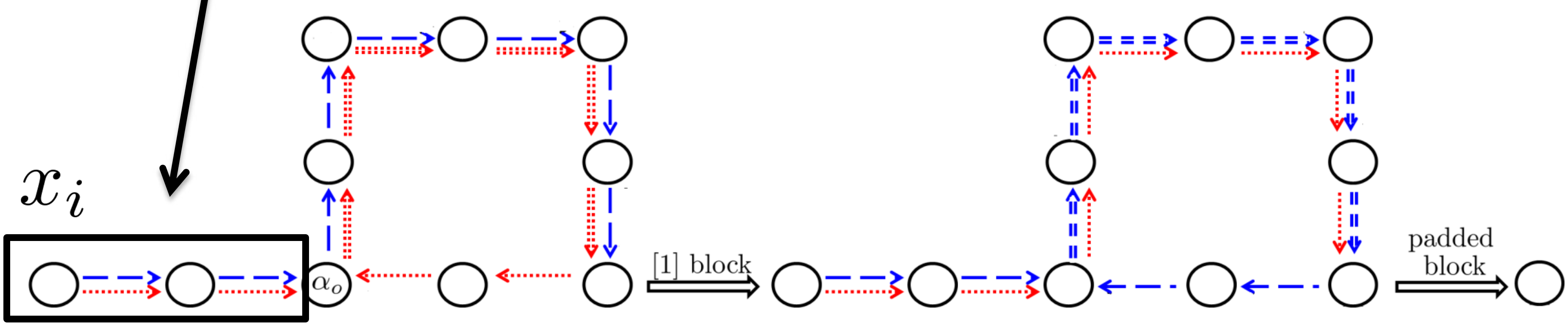


How to Recover Height of x_i

$$[0]^{2^{l/2}} \,\|\, [0]^{2L} \,\|\, [1] \,\|\, [0]^{2^{l/2}} \,\|\, [0]^{L}$$

$$[0]^{2^{l/2}} \,\|\, [0]^{L} \,\|\, [1] \,\|\, [0]^{2^{l/2}} \,\|\, [0]^{2L}$$

enter the cycle

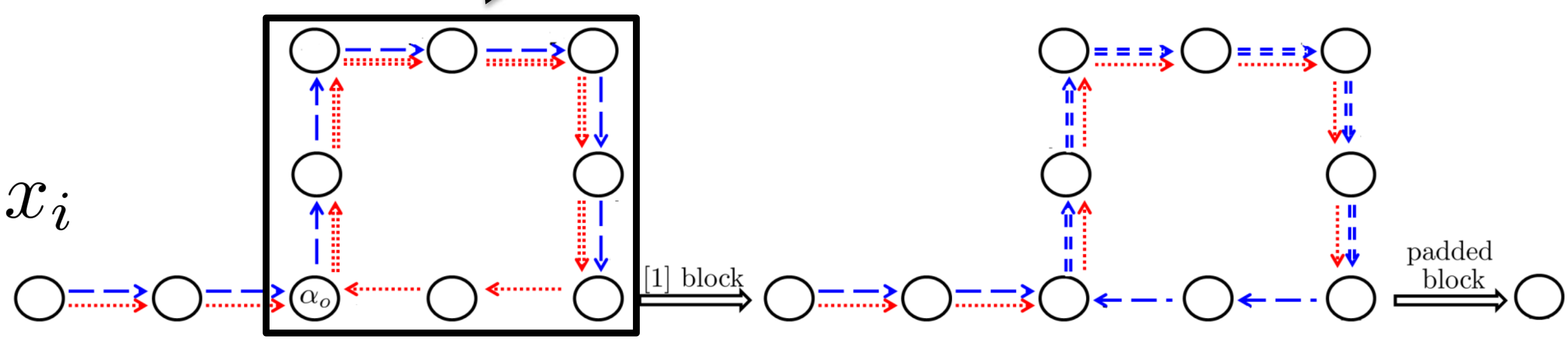


How to Recover Height of x_i

$$[0]^{2^{l/2}} \quad \| \quad [0]^{2L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{L}$$

$$[0]^{2^{l/2}} \quad \| \quad [0]^{L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{2L}$$

outputs collide

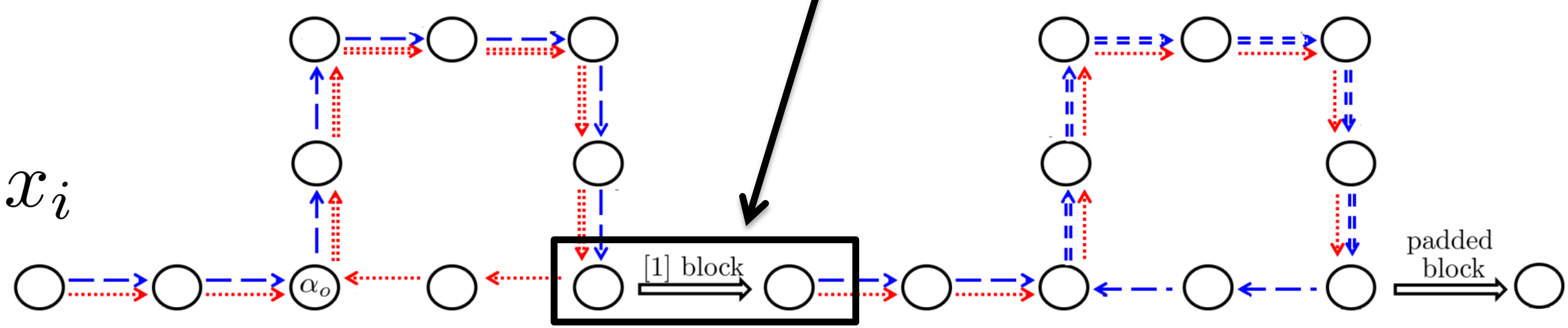


How to Recover Height of x_i

$$[0]^{2^{l/2}} \quad \| \quad [0]^{2L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{L}$$

$$[0]^{2^{l/2}} \quad \| \quad [0]^{L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{2L}$$

jump out the cycle

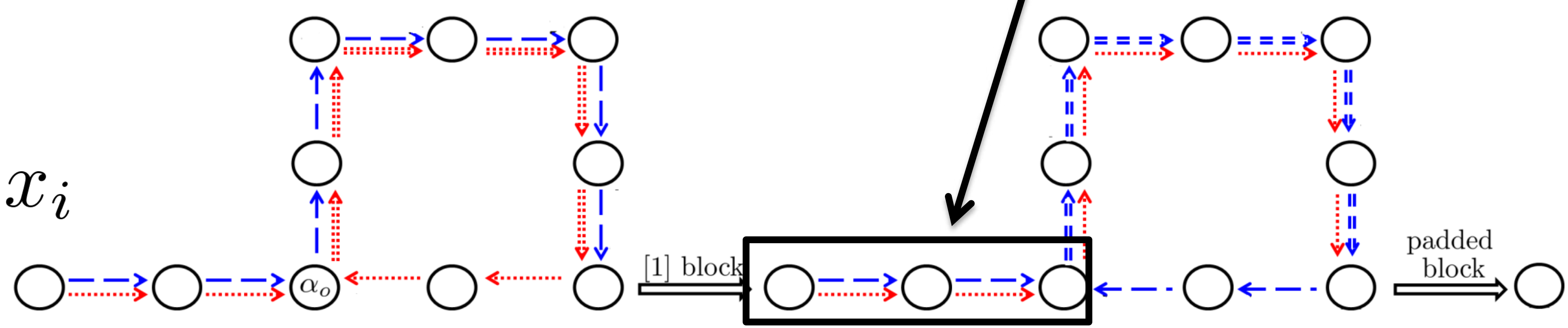


How to Recover Height of x_i

$$[0]^{2^{l/2}} \;\|\; [0]^{2L} \;\|\; [1] \;\|\; [0]^{2^{l/2}} \;\|\; [0]^{L}$$

$$[0]^{2^{l/2}} \;\|\; [0]^{L} \;\|\; [1] \;\|\; [0]^{2^{l/2}} \;\|\; [0]^{2L}$$

re-enter the cycle

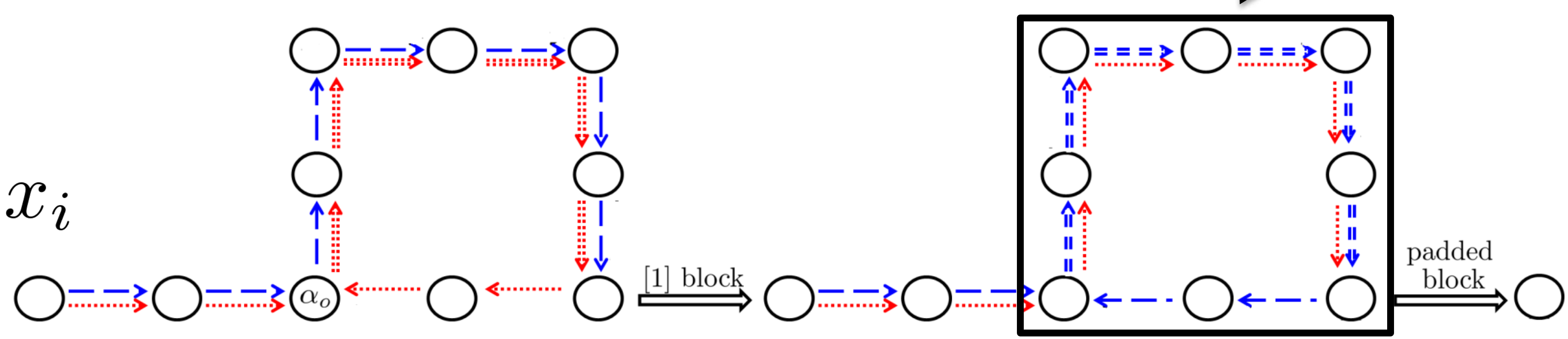


How to Recover Height of x_i

$$[0]^{2^{l/2}} \;\|\; [0]^{2L} \;\|\; [1] \;\|\; [0]^{2^{l/2}} \;\|\; [0]^{L}$$

$$[0]^{2^{l/2}} \;\|\; [0]^{L} \;\|\; [1] \;\|\; [0]^{2^{l/2}} \;\|\; [0]^{2L}$$

outputs collide

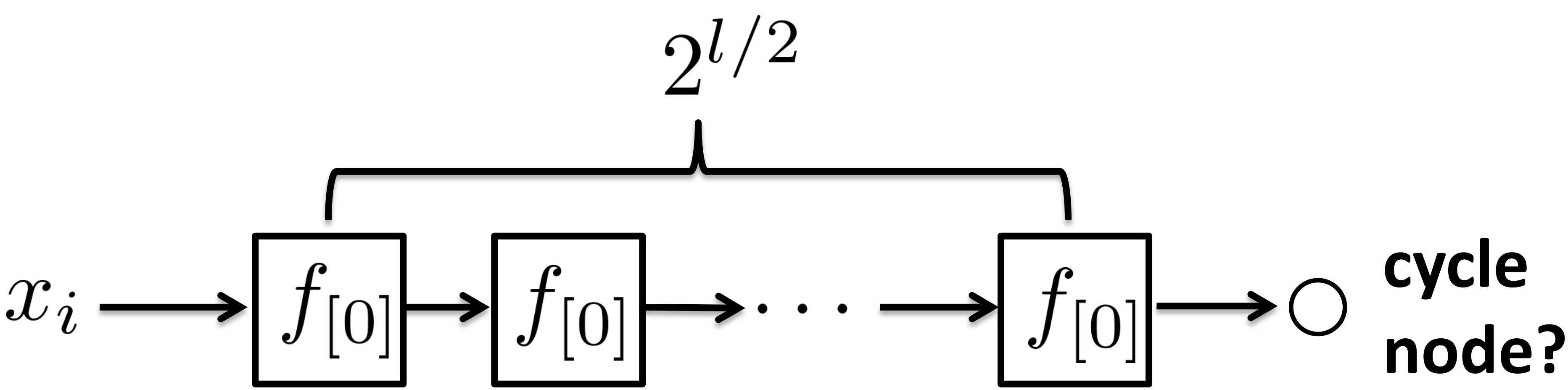


How to Recover Height of x_i

- Query the constructed message pair to MAC to check if they collide

$$[0]^{2^{l/2}} \quad \| \quad [0]^{2L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{L}$$

$$[0]^{2^{l/2}} \quad \| \quad [0]^{L} \quad \| \quad [1] \quad \| \quad [0]^{2^{l/2}} \quad \| \quad [0]^{2L}$$

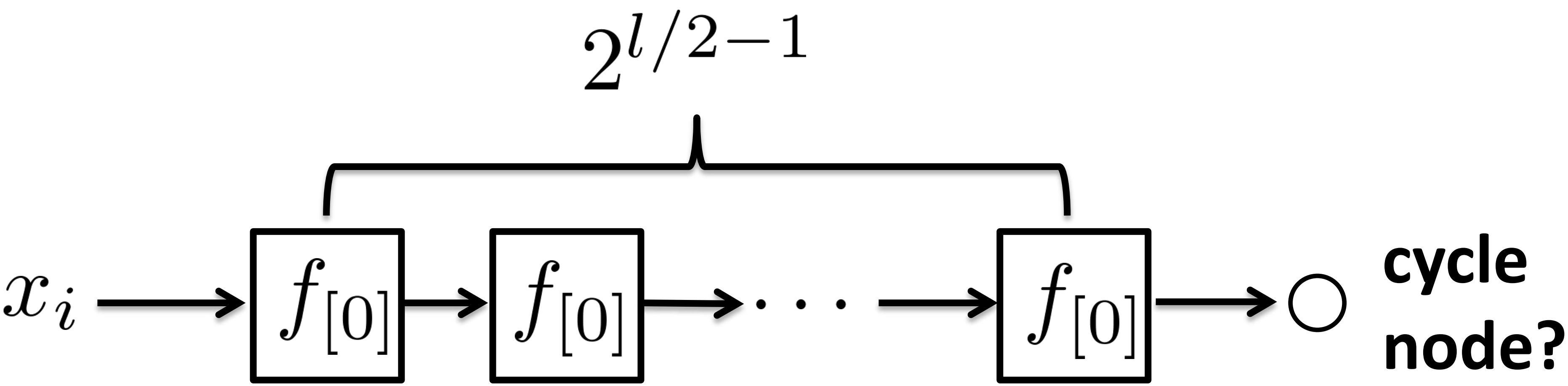


How to Recover Height of x_i

- A **binary search** to recover height
  - ➢ repeat the procedure by $l/2$ times

$$[0]^{2^{l/2-1}} \,\|\, [0]^{2L} \,\|\, [1] \,\|\, [0]^{2^{l/2}} \,\|\, [0]^{L}$$

$$[0]^{2^{l/2-1}} \,\|\, [0]^{L} \,\|\, [1] \,\|\, [0]^{2^{l/2}} \,\|\, [0]^{2L}$$



Outline

- Introduction
  - hash-based MACs
  - known results on hash-based MACs
  - our contributions
- Universal forgery attacks
  - attack overview
  - main technical ideas
- **Conclusion**

Conclusion and Open Problems

- Updated results of hash-based MACs

	proof	attack	tightness
➢ distinguishing-R:	$O(2^{l/2})$	$O(2^{l/2})$	yes
➢ distinguishing-H:	$O(2^{l/2})$	$O(2^{l/2})$	yes
➢ existential forgery:	$O(2^{l/2})$	$O(2^{l/2})$	yes
➢ **universal forgery:**	$O(2^{l/2})$	$\boldsymbol{O(2^{5l/6})}$	**no**
➢ key recovery:	$O(2^{l/2})$	$O(2^{k})$	**no**

Thank you for your attention!