Tight Upper and Lower Bounds for Leakage-Resilient Locally Decodable and Updatable Non-Malleable Codes

Dana Dachman-Soled

University of Maryland

Joint work with:

Mukul Kulkarni and Aria Shahverdi, University of Maryland

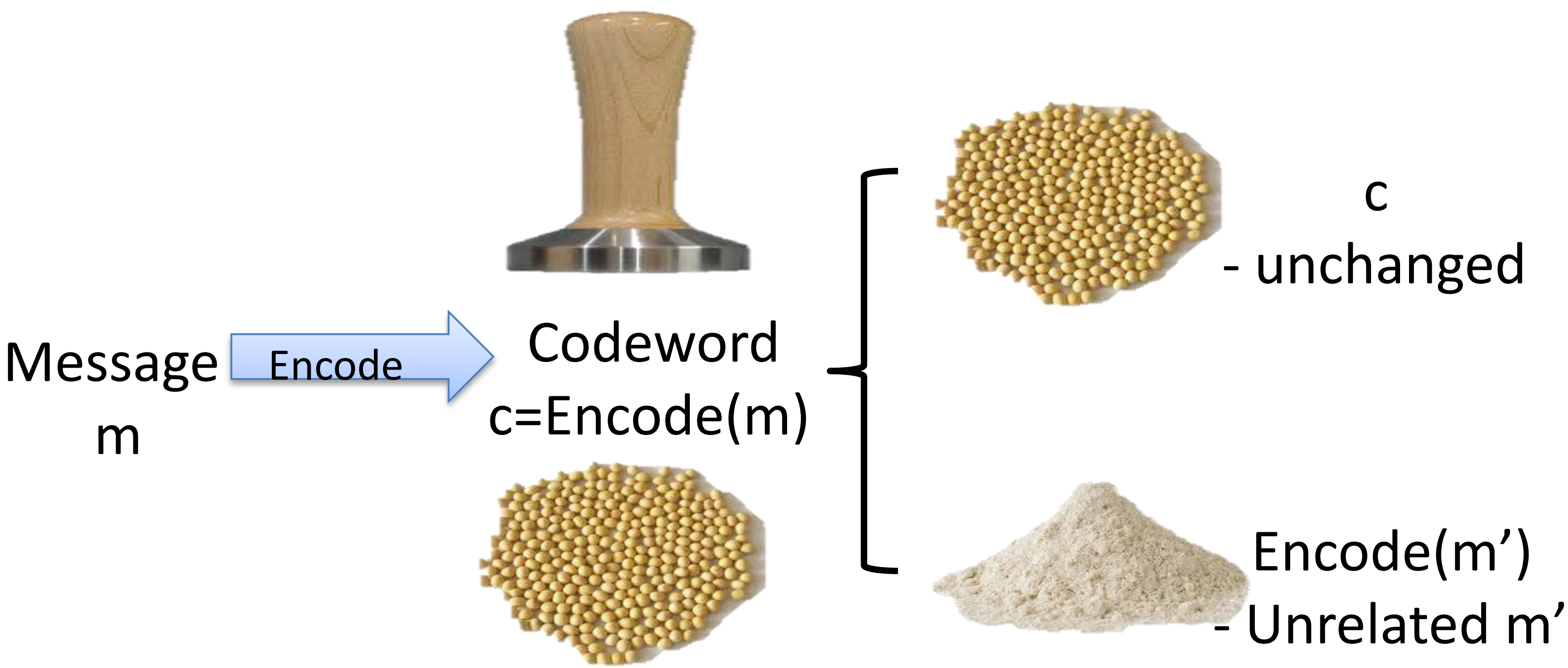
Talk is also based on joint work with: Feng-Hao Liu (FAU), Elaine Shi (Cornell) and Hong-Sheng Zhou (VCU)

Coding Schemes

- A coding scheme has two algorithms: (Encode, Decode)
  - Message m $\xrightarrow{\text{Encode}}$ Codeword C $\xrightarrow{\text{Decode}}$ Message m
- What properties do we expect from a coding scheme?
  - Error detection: If $< \ d$ bits of the codeword are modified, either the original message or $\perp$ is outputted
  - Error correction: If $< \ d/2$ bits of the codeword are modified, the original message is outputted
  - **Non-malleability**: Can potentially allow **all bits** of the codeword to be modified, but a valid message other than the original message may get outputted.

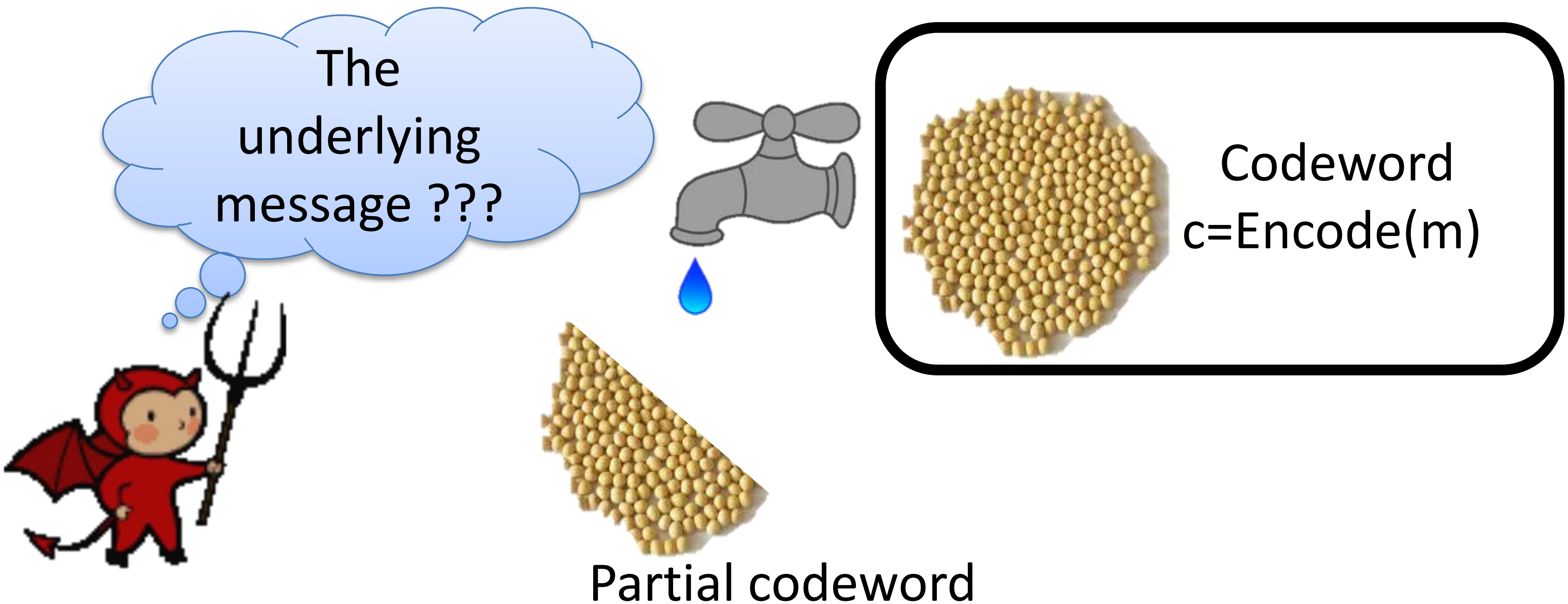
Non-Malleable Codes
[Dziembowski, Pietrzak, Wichs '10]

- Proposed as a generic way of protecting secret key stored in memory against tampering.
- Non-malleable codes: by tampering with the codeword, the underlying message is either the same or unrelated.
- Only **certain types** of tampering are allowed! (e.g. split-state)



Leakage Resilient Codes

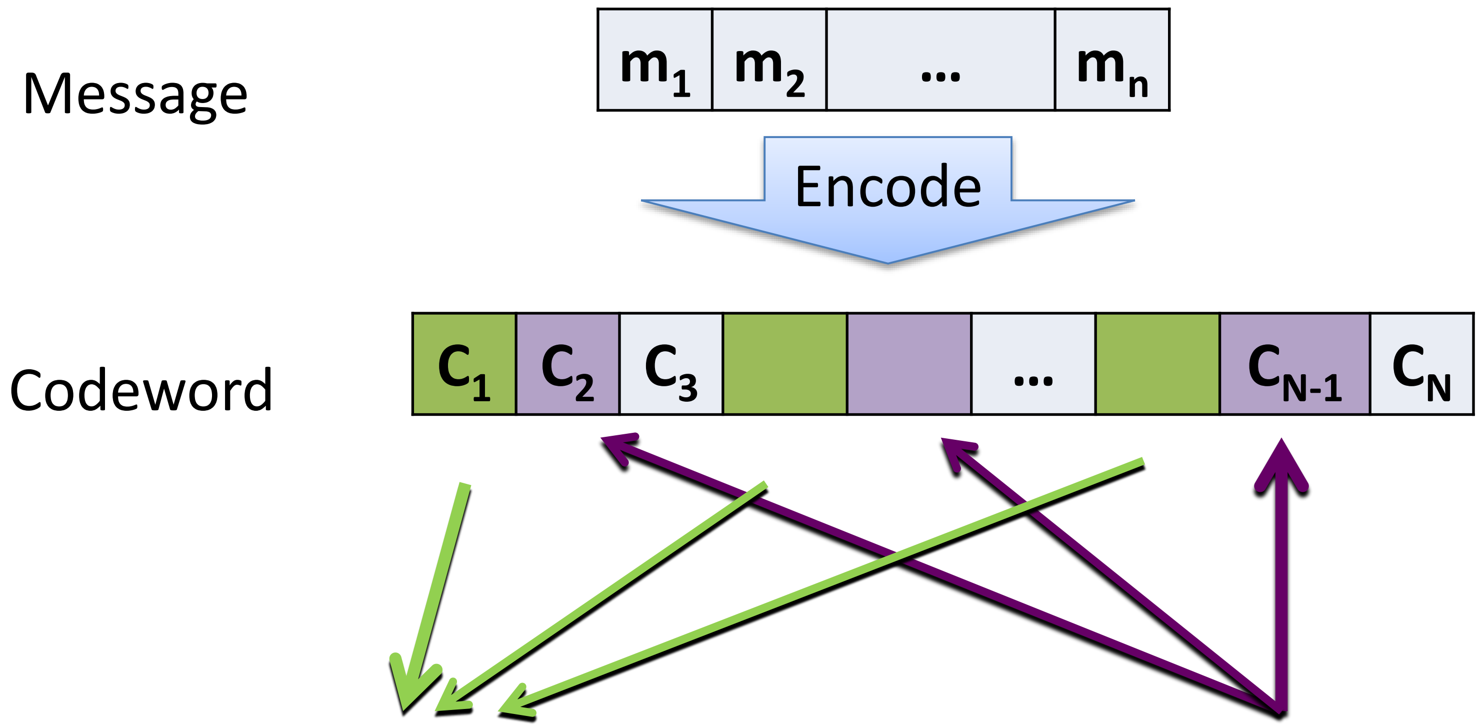
Getting partial information about the codeword does not reveal the underlying message



Problem

- Non-malleable codes are entirely unsuitable for random access computation!
- Message $\vec{m} = m_1, \dots, m_n$, encoded as $\vec{c} = \mathrm{c}_1, \dots, \mathrm{c}_\mathrm{N}$.
  - In order to decode and recover some m_i, the entire codeword needs to be accessed.
  - In order to update $m_i \to m'_i$, must re-encode the entire message $\vec{m}' = m_1, \dots, m'_i, \dots, m_n$.
- If non-malleable code is used to encode blocks of RAM individually, security guarantees do not hold.
  - Simple attacks against existing schemes.

Solution [D, Liu, Shi, Zhou ‘15]: Locally Decodable and Updatable Codes

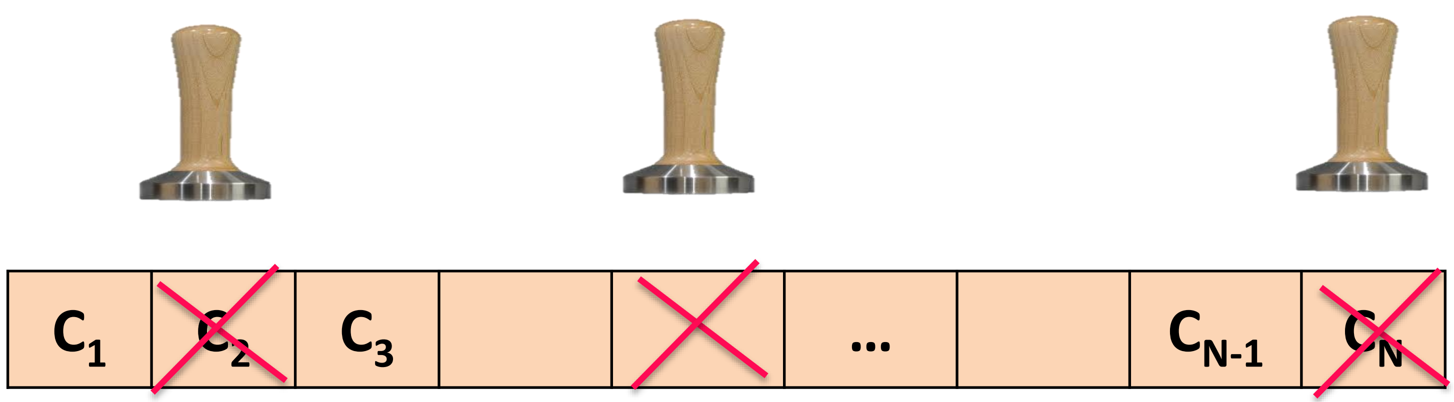


Decode(i):
Take input an index i, read a few blocks of the codeword and output m_i

Update(j, m’):
Take inputs an index j and a new message m’, update a few blocks of the codeword

Defining NM for Locally Decodable Codes

- Trickier to define NM
  - Decoding algorithm does not read all positions
  - Tampering function could destroy a few block(s) while keeping the other parts unchanged
  - The codeword is modified, but the underlying message could be very related to the original one, i.e. Decode(i)'s are the same for most i's.



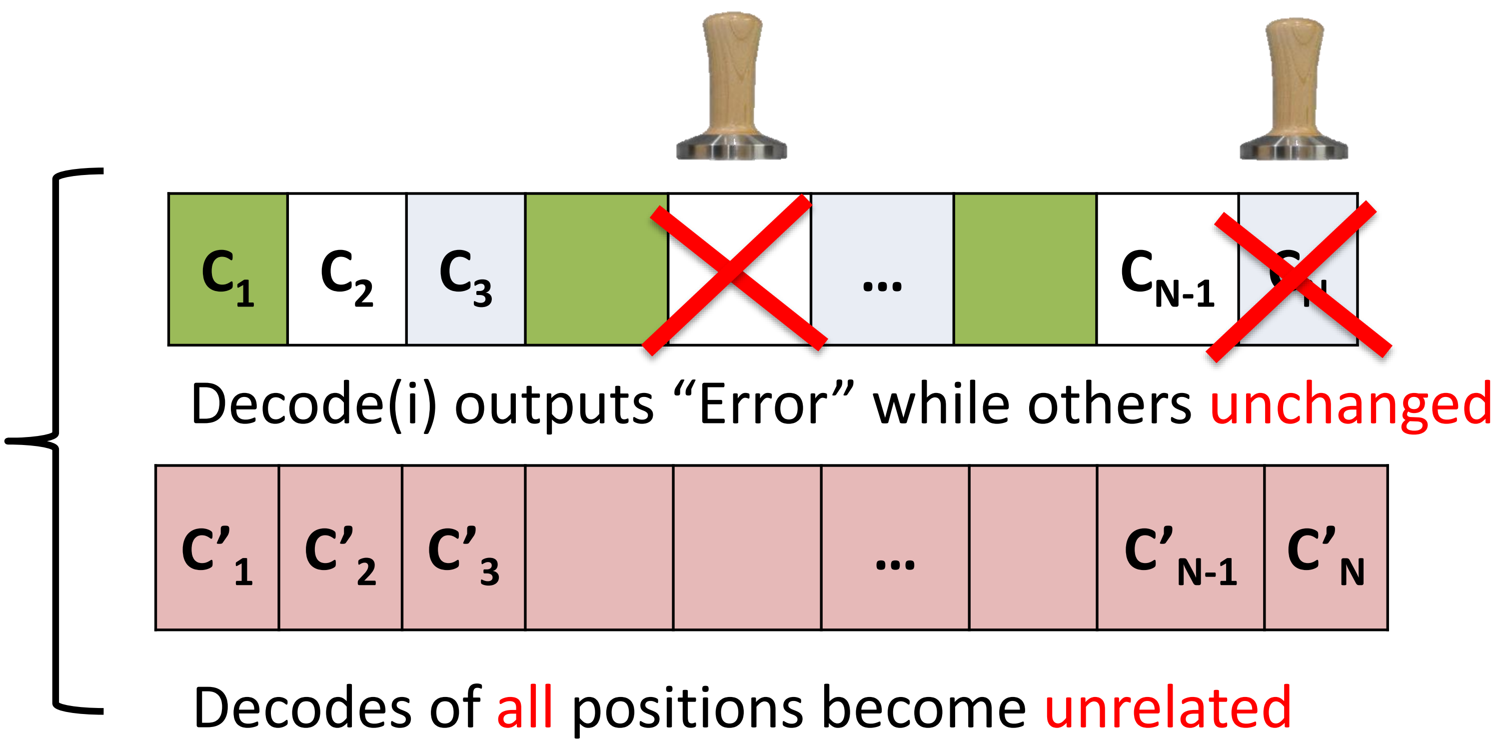
More Fine-grained Approach

- Tampering function can only do either:
  - Destroy a block (or blocks) of the underlying messages while keeping the other blocks unchanged
  - If it modifies a block of the underlying messages to some unrelated string, then it must have modified all blocks of the underlying messages to encodings of unrelated messages.

Putting It Together

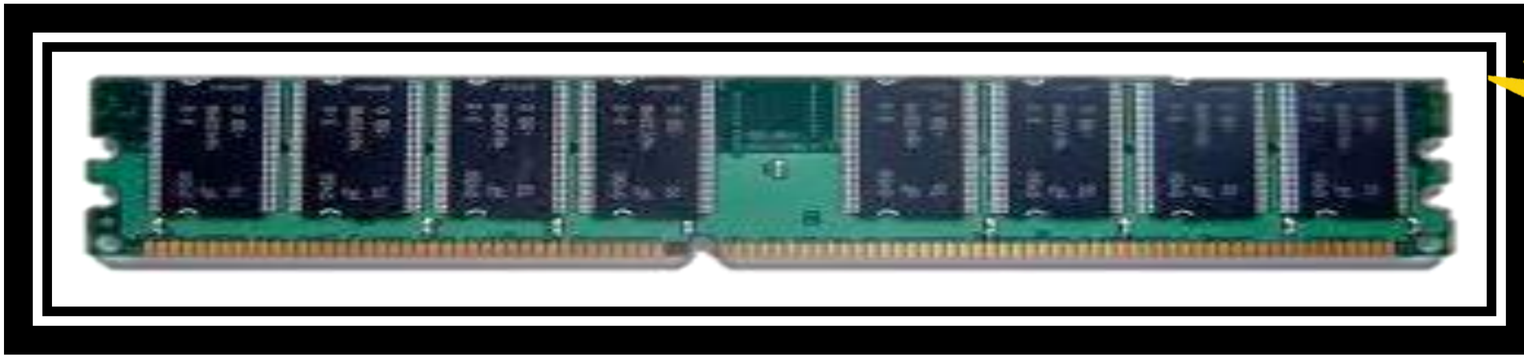
- Achieve all three properties!
  - Leakage resilience, non-malleability, locality
- Non-malleability in our setting: Tampering function either:

1. Destroy several blocks (keeps others unchanged), or
2. Change everything to unrelated messages



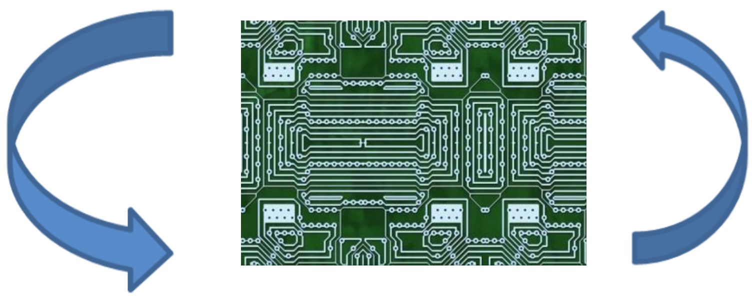
Tamper and Leakage Resilience For RAM Computation

Random Access Memory (RAM)



Store an encoding of Data in RAM-- Encode(ORAM(Data))

Write(j,m'):
Use Update(j,m')



Read(i):
Use Decode(i)

CPU

Our new code, together with an ORAM scheme, protects against physical attacks on random access memory.

Previous Work

- LR-LDUNMC with $\Omega(\log n)$ locality [D, Liu, Shi, Zhou '15]
  - Allows split-state tampering and split-state, bounded leakage.
  - Works in the continual setting.
- Information theoretically secure LDUNMC [Chandran, Kanukurthi, Raghuraman '16] in non-continual setting.

Formal Security Definition

Real Game: Round i

Adv $\xrightarrow{g^i \in \mathcal{G}}$ $\hat{C}^i$

$\xleftarrow{\ell^i := g^i(\hat{C}^i)}$

$\xrightarrow{f^i \in \mathcal{F}}$

Define: $\vec{m}^i :=$
$(Dec^{\hat{C}^{i+}}(1), \ldots, Dec^{\hat{C}^{i+}}(n))$

$\hat{C}^{i+} := f(\hat{C}^i)$

$\hat{C}^{i+1} := Update^{\hat{C}^+}(u^i, val^i)$

$Updater$: (u^i, val^i) $\xrightarrow{u^i}$

Output at the end of the game:
$(\ell^1, \cdots, \ell^r, \vec{m}^1, \ldots, \vec{m}^r, u^1, \ldots, u^r)$

Ideal Game*: Round i

M^i

Adv $\quad$ Sim

$$Adv \xrightarrow{g^i \in \mathcal{G}} Sim$$

$$Adv \xleftarrow{\ell^i} Sim$$

$$Adv \xrightarrow{f^i \in \mathcal{F}} Sim$$

Define $\vec{m}^i$ as follows:
If $I^i \neq [n]$, for $j \in I^i, \vec{m}^i[j] := \perp$
for $j \notin I^i, \vec{m}^i[j] := M^i[j]$
If $I^i = [n], \vec{m}^i = \vec{w}^i$

Outputs:
$(I^i, \vec{w}^i)$

$Updater$:
(u^i, val^i)

$$\xrightarrow{u^i}$$

$M^{i+1}[u^i]$
$:= val^i$

Output at the end of the game:
$(\ell^1, \cdots, \ell^r, \vec{m}^1, \ldots, \vec{m}^r, u^1, \ldots, u^r)$

Formal Definition—Intuition

- At round i, Sim outputs $\left(I^i, \vec{w}^i\right)$
  - If $I^i = [n]$, Sim thinks the whole codeword has been changed to an encoding of $\vec{w}$
  - Otherwise, Sim thinks only the positions in I^i have been modified to $\perp$, all other positions must remain $same$.
    - $same$ means most recently updated value in that position.

Rewind Attack

- Slowly leak part of the codeword corresponding to some message block j.
- Wait for an update to occur to message block j.
- Write back what was leaked.
- When decoding the j-th block, if original message is recovered (as opposed to most recently updated value) then non-malleability is broken.

How to Prevent Rewind Attacks

- Attacker can only leak a small amount in each round
- An update also occurs in each round.
- Goal: When the attacker writes back the leakage either
  - The information written back by the attacker is no longer consistent.
  - The information is consistent, but effectively overwrites the entire codeword.

Our Results—Lower Bound

Theorem: Let λ be security parameter and Π = (Encode, Decode, Upadate) be a locally decodable and updatable non-malleable code, in a security model which allows for a rewind attack.
Then for n = poly(λ), Π has locality $\delta(n) \in \omega(1)$.

**Holds for any polynomial block length*

***Requires the access patterns for decoding/updating to be non-adaptive*

****Result extends to randomized access patterns*

*****Lower bound holds even if only single bit is leaked in each round.*

Our Results—Upper Bound

Theorem: Let λ be security parameter. Then there exists a locally decodable and updatable non-malleable code Π = (Encode, Decode, Update), in a security model which allows for a rewind attack, such that Π has locality $\delta(n)$ for any $\delta(n) \in \omega(1)$.

**Requires block length* $\chi = \lambda^{1+\epsilon}$

***The access patterns for decoding/updating are non-adaptive*

****The access patterns are deterministic*

*****Allows for leakage of* $(1-\epsilon') \cdot \chi$ *bits per round.*

Upper and Lower Bound are “tight”.

Roadmap

- Tools for Lower Bound
- Lower Bound: Attack and Analysis
- Upper Bound
- Conclusions

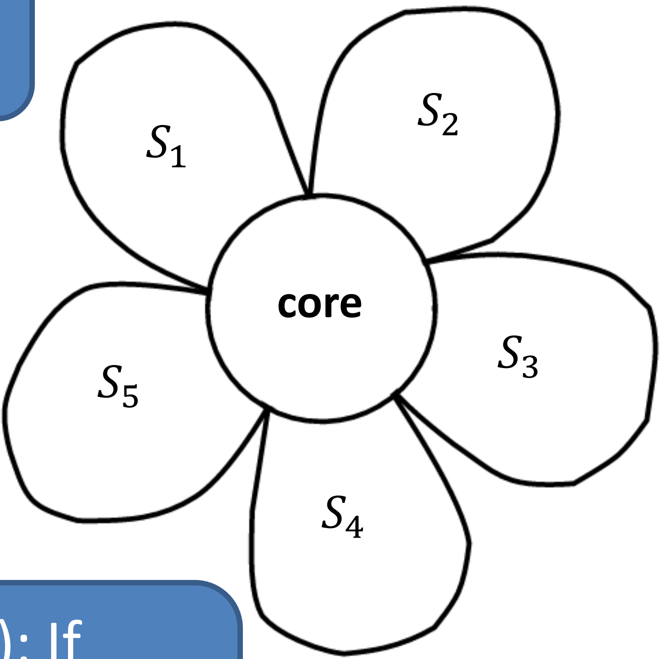
Roadmap

- Tools for Lower Bound
- Lower Bound: Attack and Analysis
- Upper Bound
- Conclusions

Sunflower Lemma

Definition: A **Sunflower** is a collection of sets such that the intersection of any pair is equal to the core.

- Consider $\Sigma := \{S_1, \ldots, S_n\}$
- S_i is the set of codeword blocks accessed during decode/update of the i-th message block.
- Size of each S_i is at most constant c.
- Size of each codeword block is $\chi := poly(\lambda)$



Sunflower Lemma (Erdös and Rado): If

$$n > c!\,(k)^c$$

then Σ contains a sunflower of size $k + 1$.

- Set $k \gg c \cdot \chi$
- n is polynomial in λ.

Compression Function

Given $\boldsymbol{SF} = \{S_{i_0}, S_{i_1}, \dots, S_{i_k}\}$, codeword $\hat{C}$

Define $F_{\hat{C}}(\cdot)\colon \{0,1,same\}^k \rightarrow \{0,1\}^{c\cdot\chi}$ as follows:

- On input $x_1, \dots, x_k \in \{0,1,same\}$
- For $j = 1$ to k
- If $x_j \neq same$, run $Update^{\hat{C}}(i_j, x_j)$
- Output the contents of the **core** of the Sunflower.

Why is this a compression function?
Recall that we chose n sufficiently large to guarantee that $k \gg c \cdot \chi$.

Distributional Stability

Theorem (Informal) [Drucker 12],(see also [Raz 98], [Shaltiel 10]):
Let $F_{\hat{C}}(X_1, \dots, X_k)\colon \{0,1,same\}^k \to \{0,1\}^{\leq t}$ be a randomized mapping, where $t \ll k$ and $X_1, \dots, X_k$ are independent random variables.

Then w.h.p. over choice of $i \sim [k]$, the two distributions

$$F_{\hat{C}}(X_1, \dots, X_k) \qquad\qquad F_{\hat{C}}(X_1, \dots, X_{i-1}, same, X_{i+1}, \dots, X_k)$$

are statistically close.

Roadmap

- Tools for Lower Bound
- Lower Bound: Attack and Analysis
- Upper Bound
- Conclusions

Attack on Code with Constant Locality c

Leak i_j-th petal

Attacker:

- Find the sunflower $\boldsymbol{SF} = \{S_{i_0}, S_{i_1}, \ldots, S_{i_k}\}$
- Choose $j \leftarrow [k]$
- In the first round, submit leakage function $g(\hat{C}) \coloneqq set_{i_j}(\hat{C}) \setminus \boldsymbol{core}$.
- Receive back leakage ℓ
- Wait until the $(k + 1)$-st round.
- In the $(k + 1)$-st round, choose tampering function f which replaces the current contents of $set_{i_j}(\hat{C}) \setminus \boldsymbol{core}$ with ℓ.

Replace i_j-th petal

Updater:

- Choose $x_1, \ldots, x_k \leftarrow \{0,1, same\}$
- In round $j = 1$ to k
- If $x_j \neq same$, request $Update^{\hat{C}}(i_j, x_j)$

*Small modification needed if adversary can leak only a single bit in each round.

Analysis

Lemma: For the attack and updater specified above:

Case 1: If the original message was $\vec{m} = \vec{0}$, then with probability at least 0.7, the decoding of position i_j in round $k+1$ is 0 in the real game.

Case 2: If the original message was $\vec{m} = \vec{1}$, then with probability at least 0.7, the decoding of position i_j in round $k+1$ is 1 in the real game.

Why is this sufficient to contradict non-malleability?

Proving the Lemma:
Case 1, $\vec{m} = \vec{0}$

Decoding of position i_j in the $(k+1)$-st round takes as input:

$$\left(\ell, core = F_{\hat{C}_0}(X_1, \dots, X_k)\right)$$

Hybrid Argument:

1. Consider

$$Dec\left(\ell, F_{\hat{C}_0}\left(X_1, \dots X_{j-1}, same, X_{j+1}, X_k\right)\right)$$

Output must be equal to 0. Why?

2. Consider

$$Dec\left(\ell, F_{\hat{C}_0}\left(X_1, \dots X_{j-1}, X_j, X_{j+1}, X_k\right)\right)$$

This must also be equal to 0 with high probability. Why?

Case 2, $\vec{m} = \vec{1}$ is analogous.

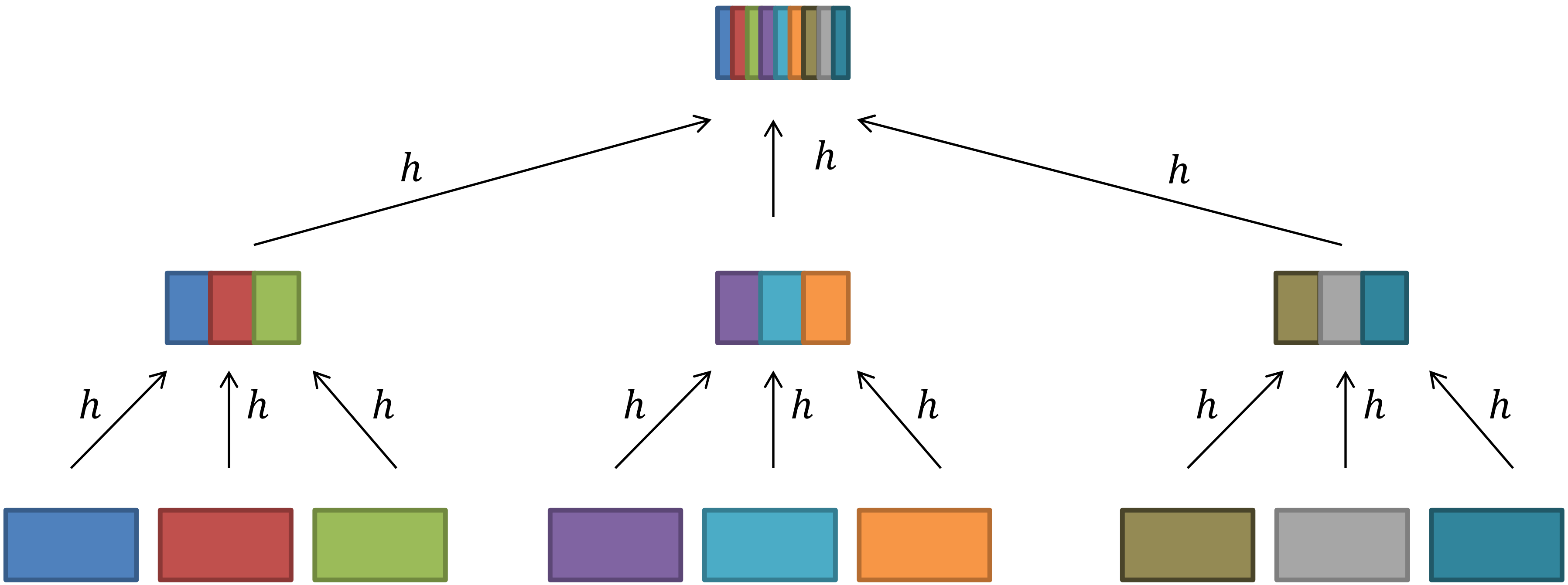
Roadmap

- Tools for Lower Bound
- Lower Bound: Attack and Analysis
- Upper Bound
- Conclusions

Upper Bound

- Recall the [DLSZ'15] construction:
  - Encrypt the data with an AE scheme
  - Compute the Merkle hash of the encrypted data
  - Encode the secret key, root of Merkle hash using regular (non-local) NMC.

t-Slice Merkle Tree



t-Slice Merkle Tree

- t-slice Merkle Tree is a t-ary tree where each node is hashed into a slice of its parent node.
  - We choose $t = \lambda^{\epsilon}$, for constant $0 < \epsilon < 1$.
- Update/Verify need to read only the path from root to leaf but not the siblings
  - Note that Update/Verify take time proportional to the height of the tree,
  - For $n = poly(\lambda), t = poly(\lambda)$ the height of the tree $< \delta(n)$, for any $\delta(n) \in \omega(1)$.

Roadmap

- Tools for Lower Bound
- Lower Bound: Attack and Analysis
- Upper Bound
- Conclusions

Conclusions

- We showed tight upper and lower bounds on locality for locally decodable and updatable codes in security models that allow for a rewind attack.
- Result holds for non-adaptive access patterns
  - In this talk: deterministic, non-adaptive access patterns
  - We have extended our result to randomized, non-adaptive access patterns.
- Future work:
  - Extend lower bound to adaptive setting.
  - Show an improved upper bound in adaptive setting.

Thank you!