Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement

Karen Klein[3] **Guillermo Pascual-Perez**[3] Michael Walter[3]
Chethan Kamath Margarita Capretto[2] Miguel Cueto[3]
Ilia Markov[3] Michelle Yeo[3] Joël Alwen[1] Krzysztof Pietrzak[3]

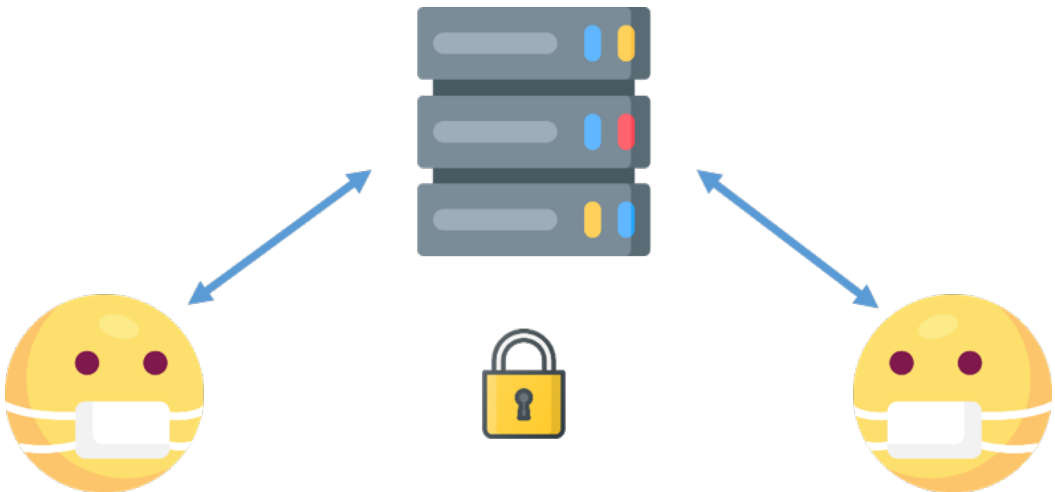
1 - Wickr

2 - Universidad Nacional de Rosario

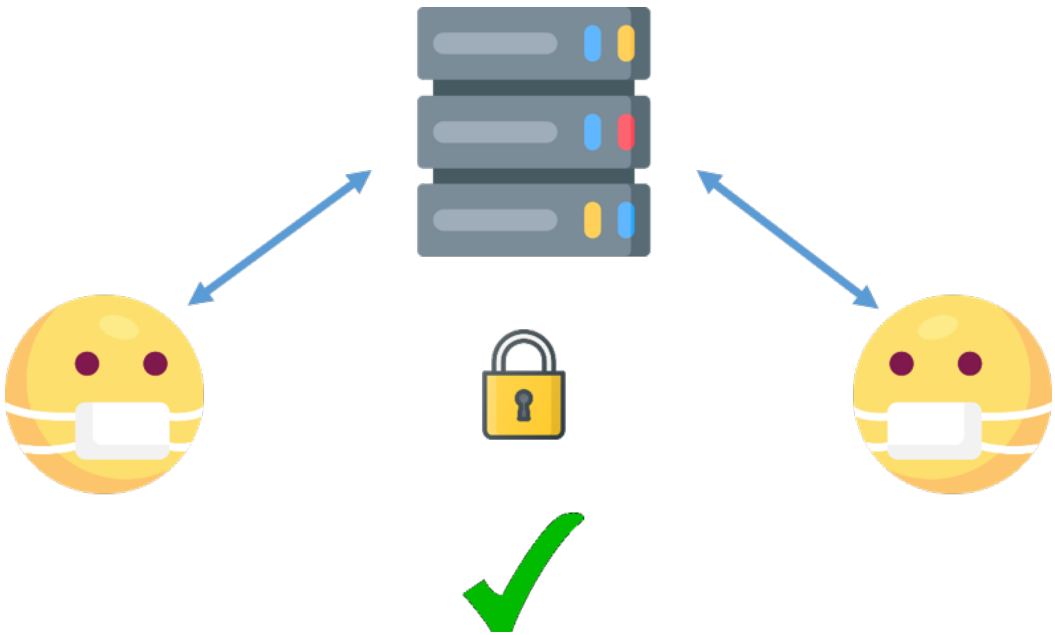
3- IST Austria

{kklein, gpascual, mwalter, pietrzak}@ist.ac.at, jalwen@wickr.com

Secure messaging

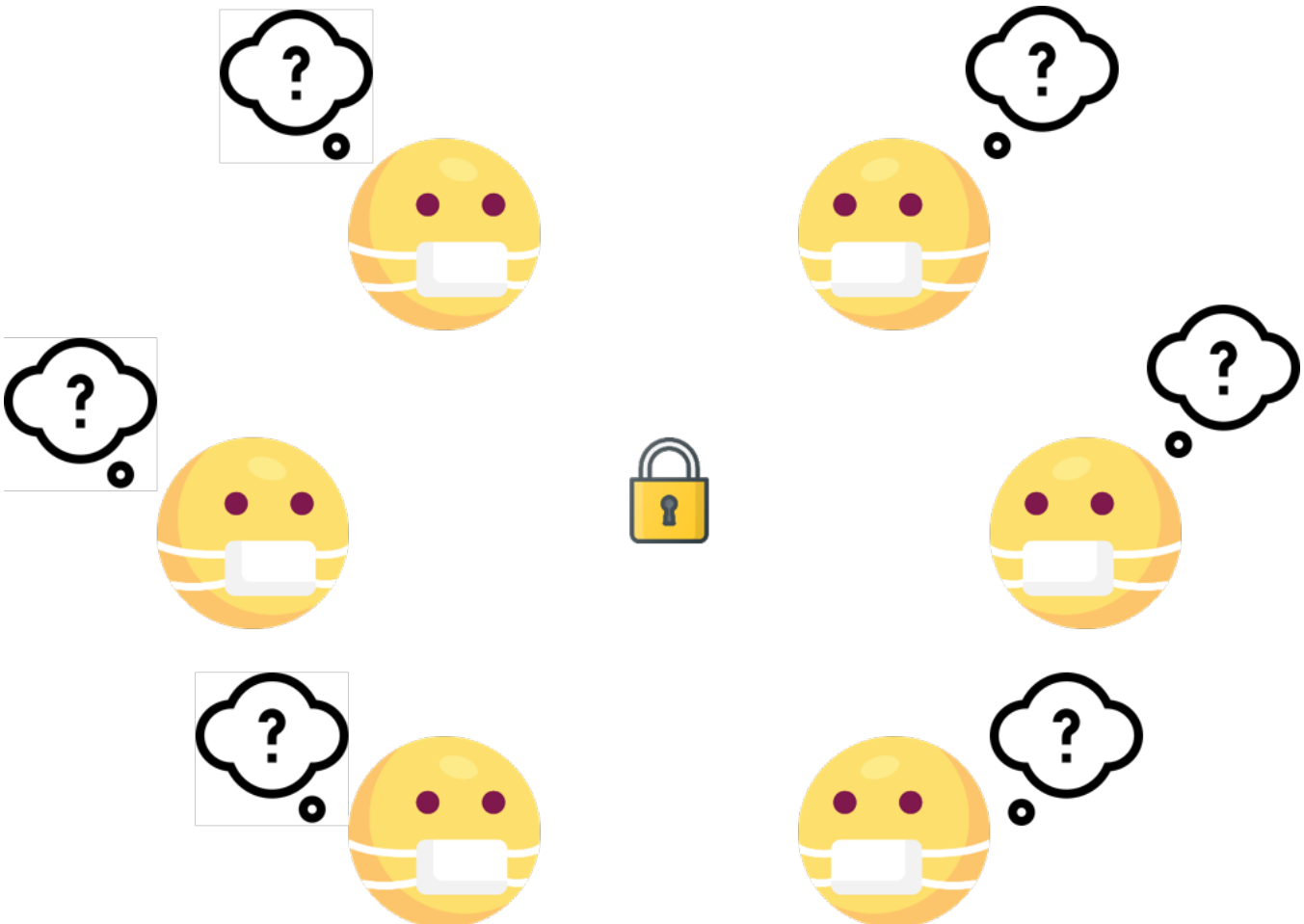


Secure messaging



M. Marlinspike and T. Perrin - *The double ratchet algorithm*, Signal.
[BSJNS17, CCDGS17, DV18, JS18, PR18, ACD19,...]

Secure Messaging



In this talk I will...

- Overview **Continuous Group Key Agreement** (CGKA).
- Present **Tainted TreeKEM**, an efficient CGKA protocol.
- Discuss efficiency and security

Continuous Group Key Agreement (CGKA)

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports adding/removing members.

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports adding/removing members.

- **Asynchronous**
  - Untrusted server buffers messages.

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports adding/removing members.

- **Asynchronous**
  - Untrusted server buffers messages.

- **Secure**

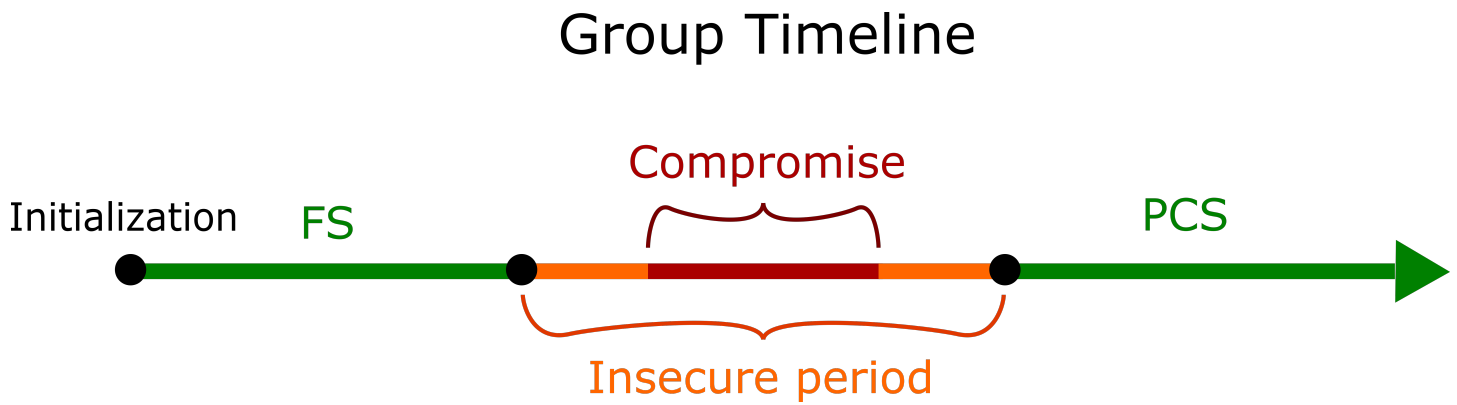
Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports adding/removing members.

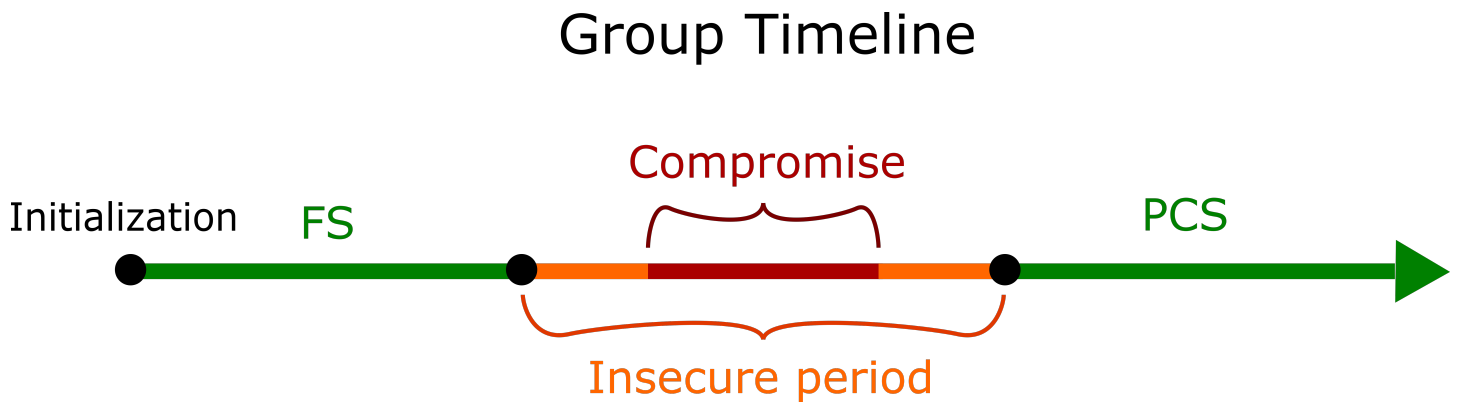
- **Asynchronous**
  - Untrusted server buffers messages.

- **Secure**
  - Forward Secrecy (FS)
  - Post-Compromise Security (PCS)

Forward Secrecy (FS) & Post-Compromise Security (PCS)

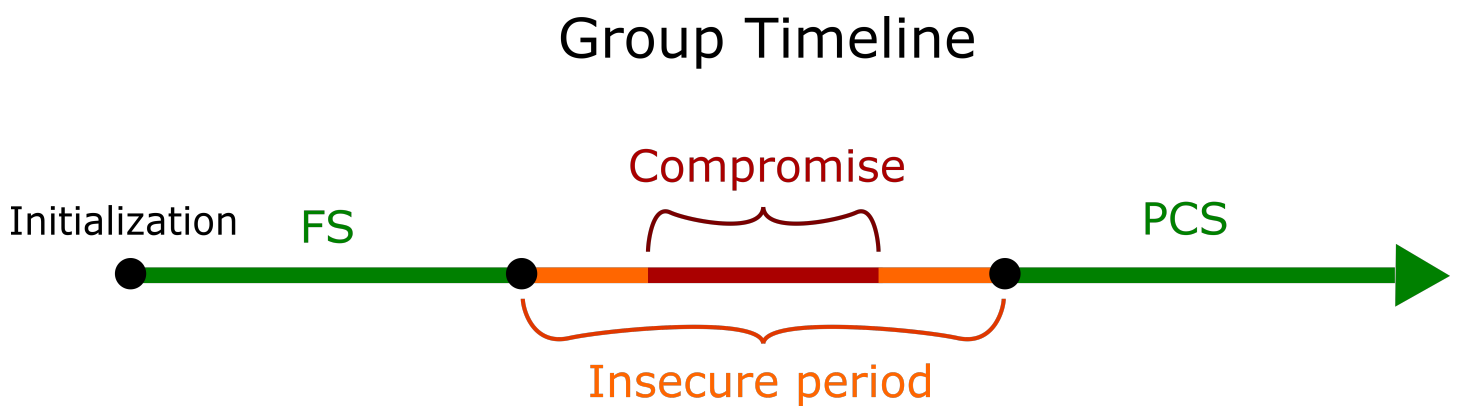


Forward Secrecy (FS) & Post-Compromise Security (PCS)



Need **key update** functionality

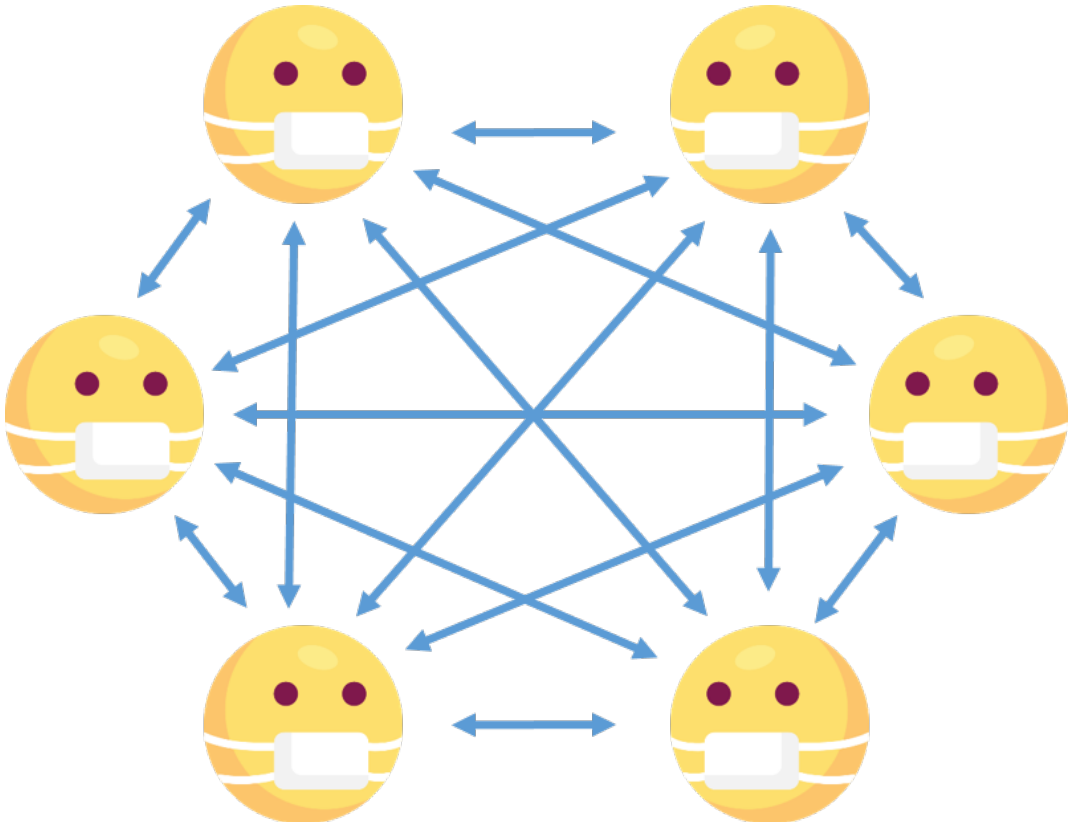
Forward Secrecy (FS) & Post-Compromise Security (PCS)



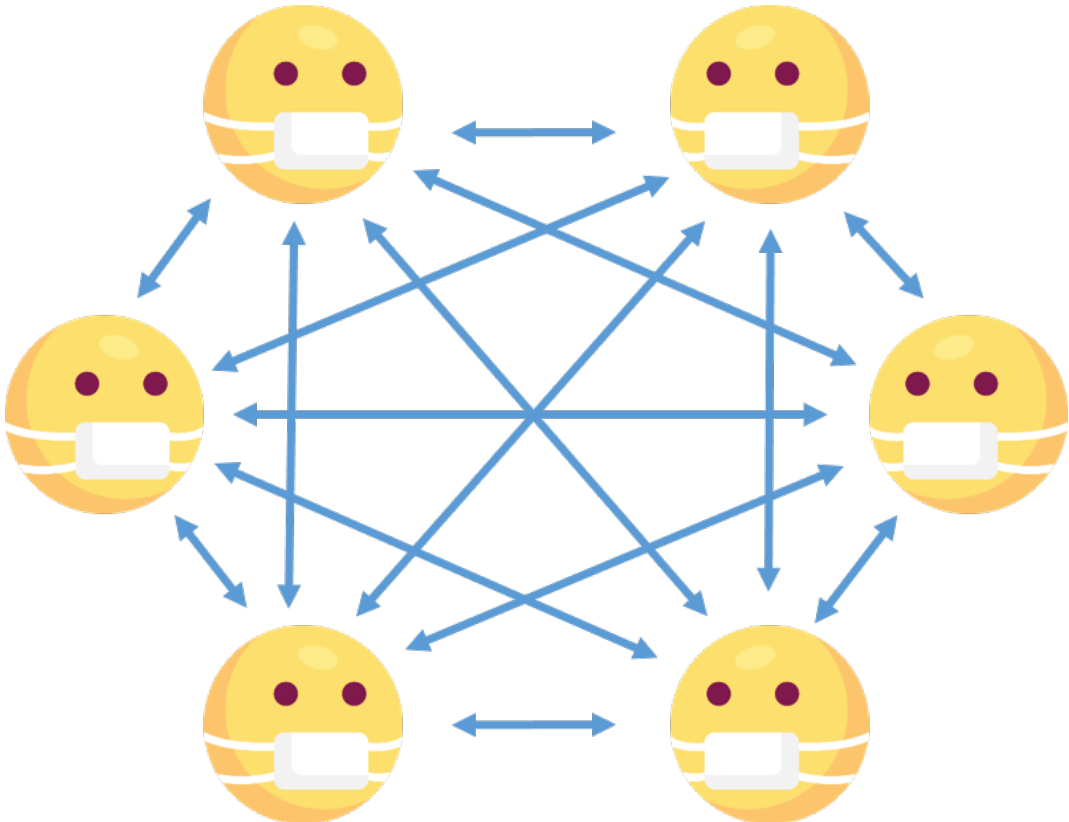
Need **key update** functionality

- **FS**: One-way deterministic enough.
- **PCS**: Needs **new randomness**.

n-party CGKA: Bidirectional channels?



n-party CGKA: Bidirectional channels?



Key updating incurs **linear communication cost**!

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports Add & Remove of members.

- **Asynchronous**
  - Untrusted server buffers messages.

- **Secure**
  - Forward Secrecy (FS)
  - Post-Compromise Security (PCS)

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports Add & Remove of members.

- **Asynchronous**
  - Untrusted server buffers messages.

- **Secure**
  - Forward Secrecy (FS)
  - Post-Compromise Security (PCS)
- Key updates with **efficient communication cost** (logarithmic).

Continuous Group Key Agreement (CGKA)

- **Dynamic Membership**
  - Supports Add & Remove of members.

- **Asynchronous**
  - Untrusted server buffers messages.

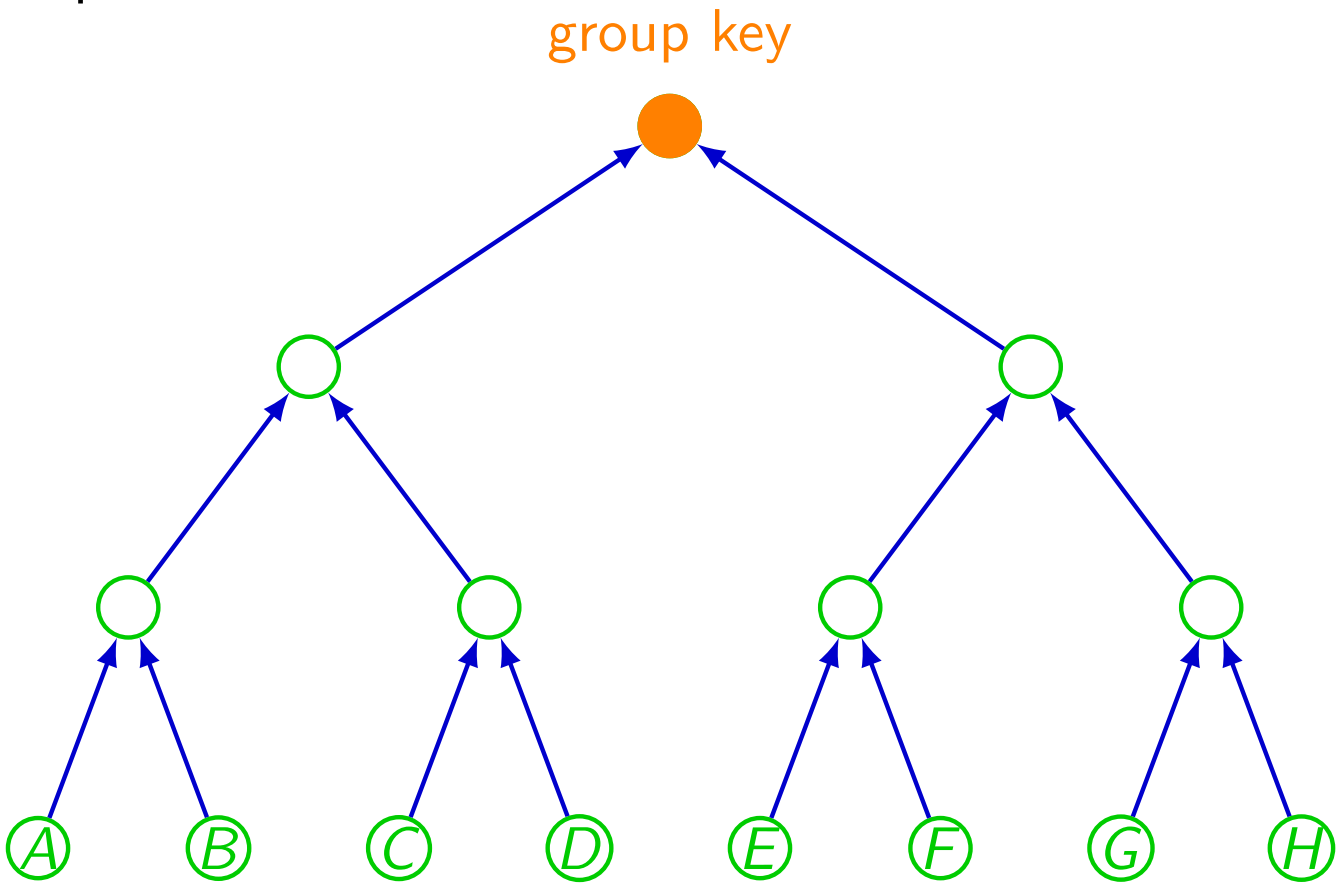
- **Secure**
  - Forward Secrecy (FS)
  - Post-Compromise Security (PCS)
- Key updates with **efficient communication cost** (logarithmic).
  - More frequent updates $\rightarrow$ better security.

Message Layer Security (MLS)

- IETF Working group
- Standard for Secure Group Messaging
- Support for groups $\leq$ 50k users.
- Current Proposal: **TreeKEM**.

TreeKEM Protocol (MLS)

PKE key-pair per node.



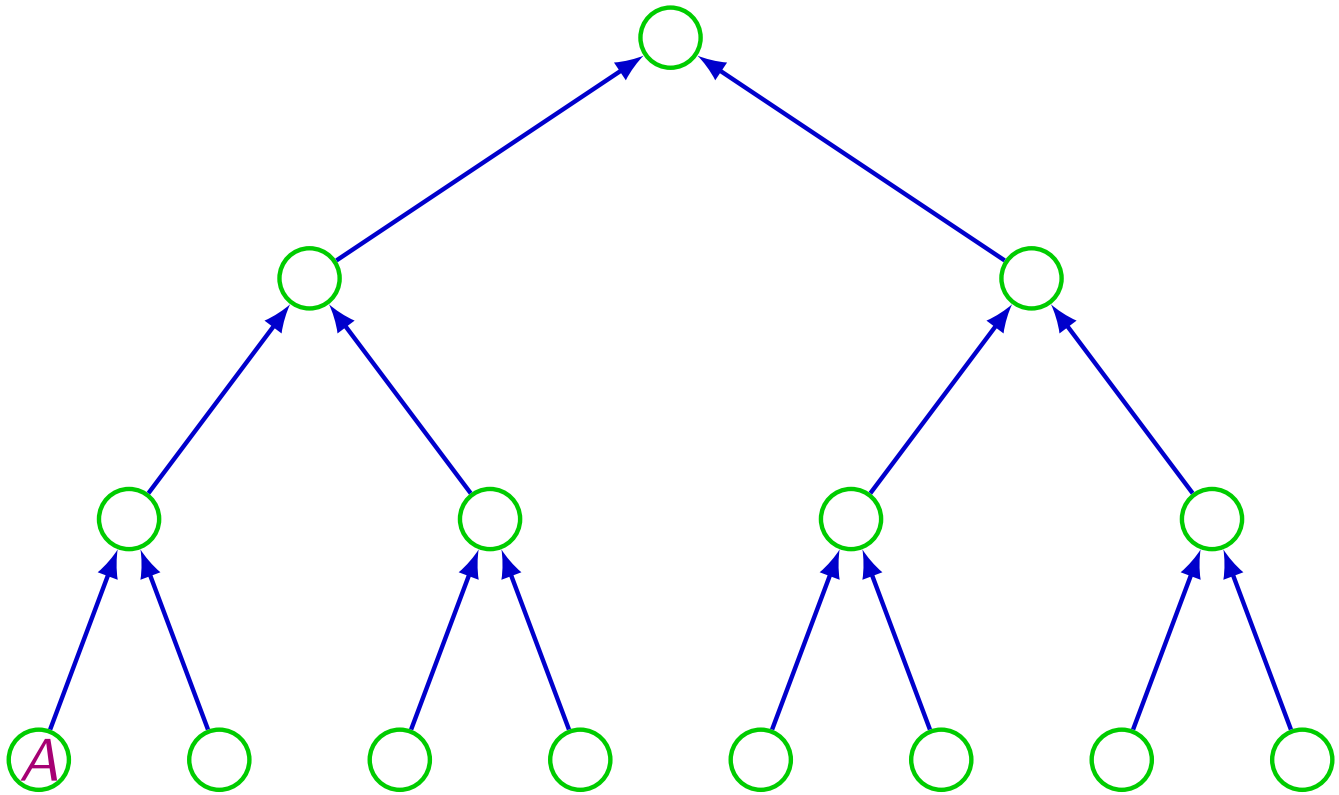
Edges meaning: Knowledge of source $\Rightarrow$ Knowledge of sink

$\Downarrow$

User knows secret keys on their **path to root**.

TreeKEM: Update (simplified)

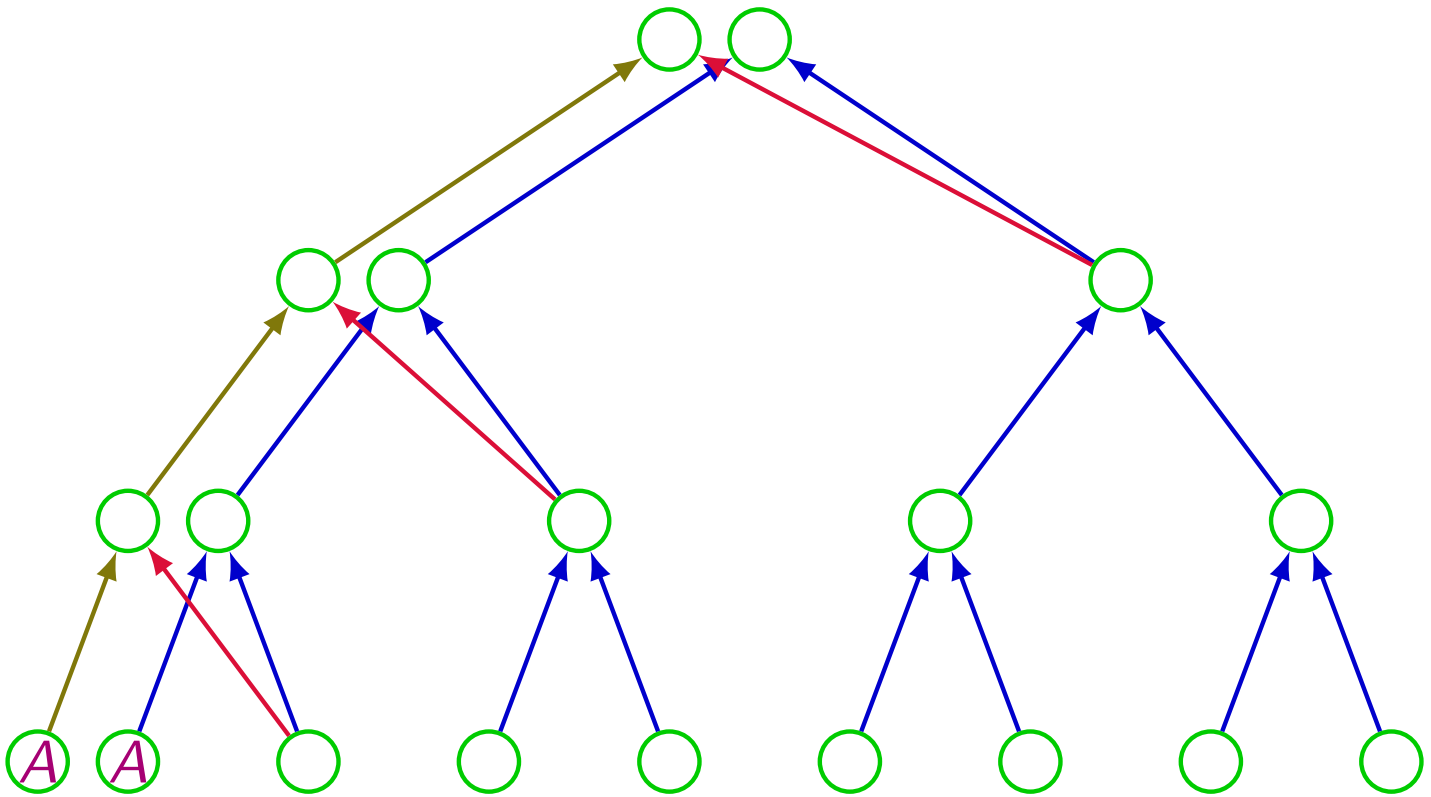
Alice updates



TreeKEM: Update (simplified)

Alice updates

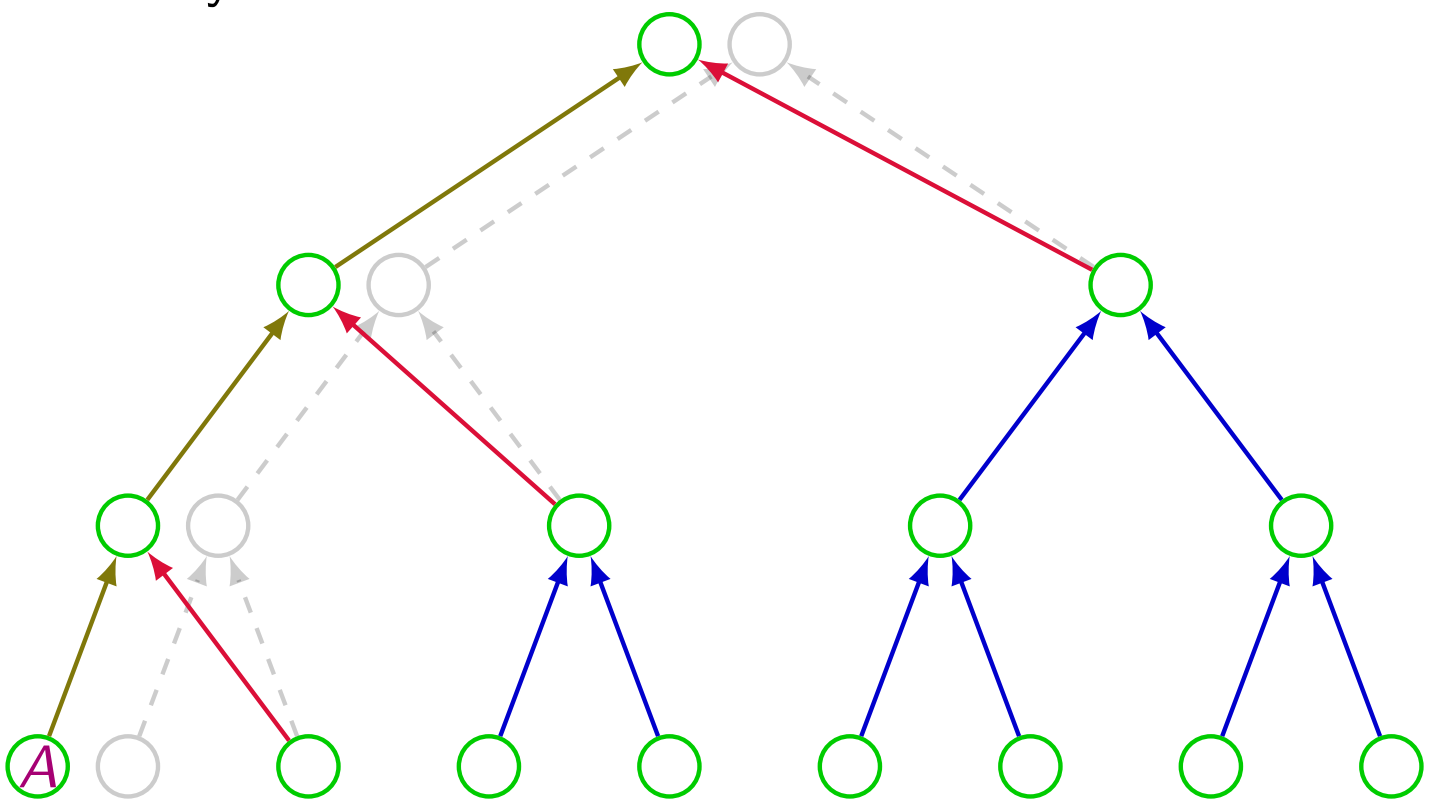
- chooses and encrypts fresh keys



TreeKEM: Update (simplified)

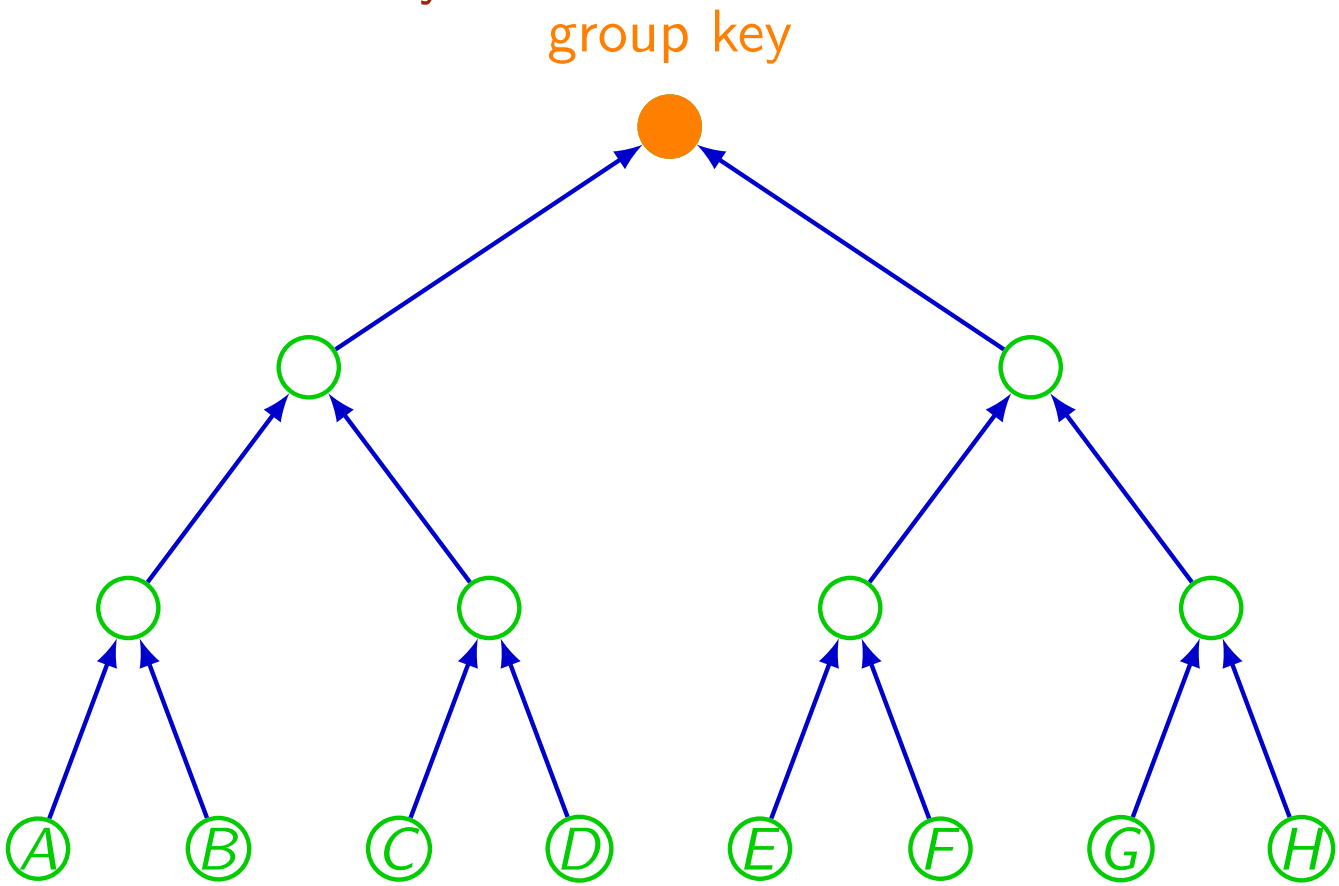
Alice updates

- chooses and encrypts fresh keys
- removes old keys



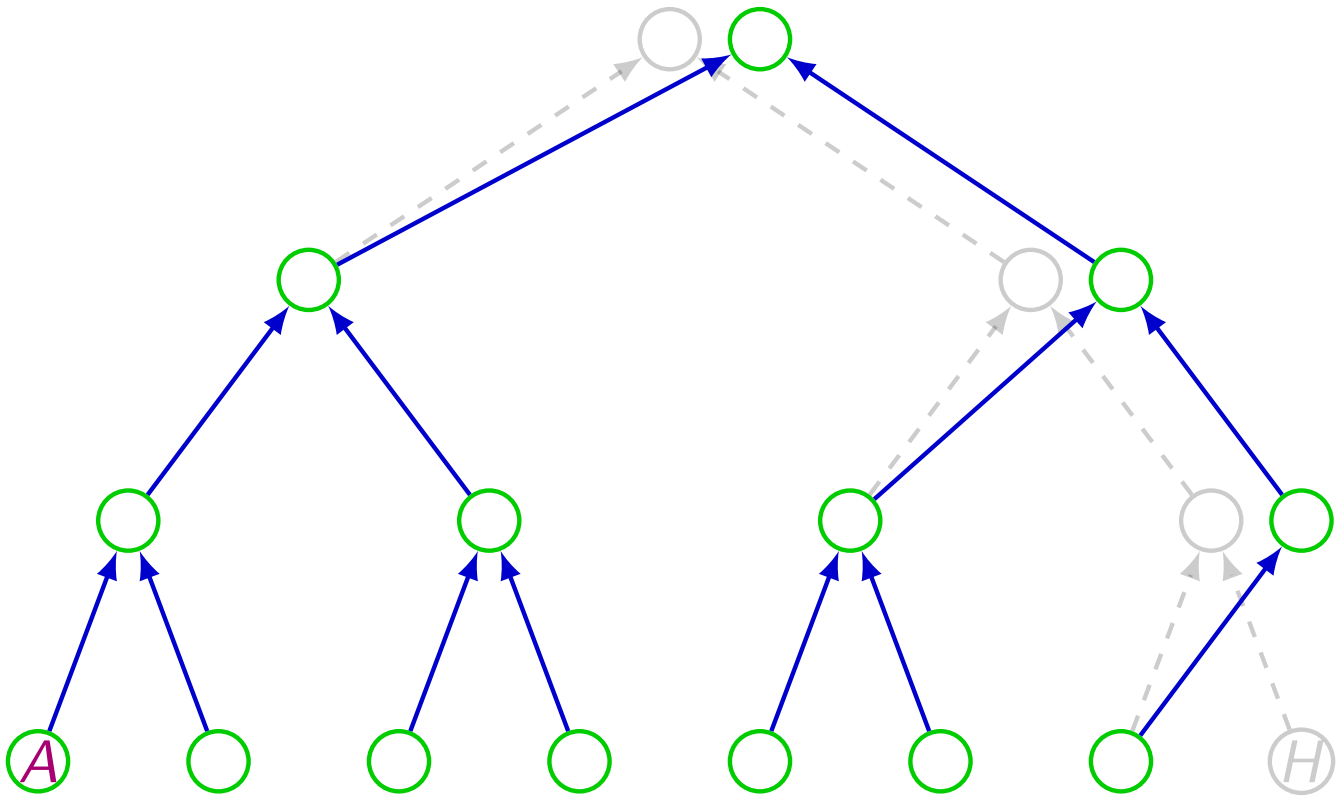
How to remove?

Alice wants to remove Henry



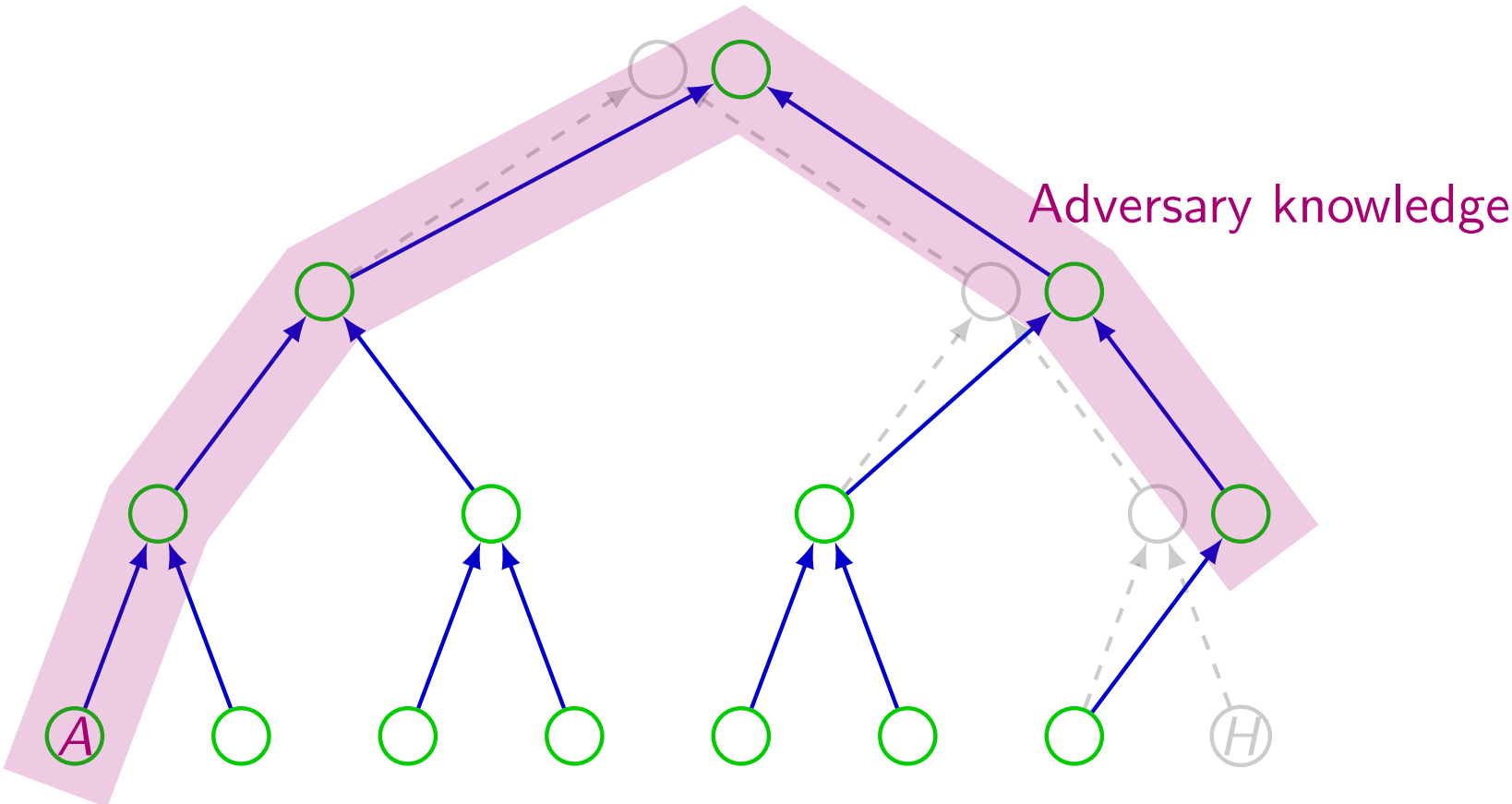
How to remove?

Alice needs to rotate keys in Henry 's path.



How to remove?

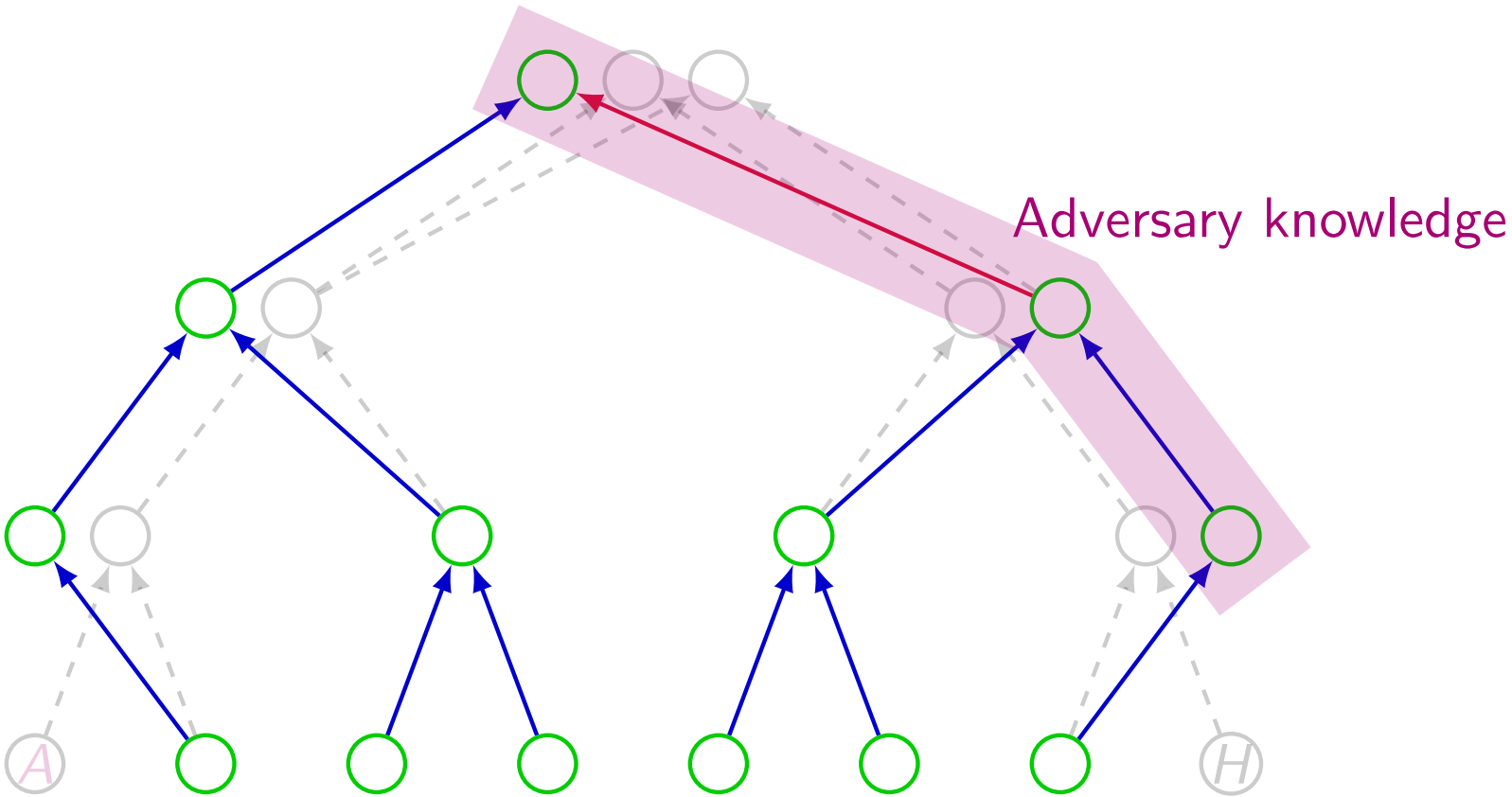
Alice needs to rotate keys in Henry 's path.



... if Alice corrupted, secret keys outside her path leak!

How to remove?

If Alice is now removed in the same way...

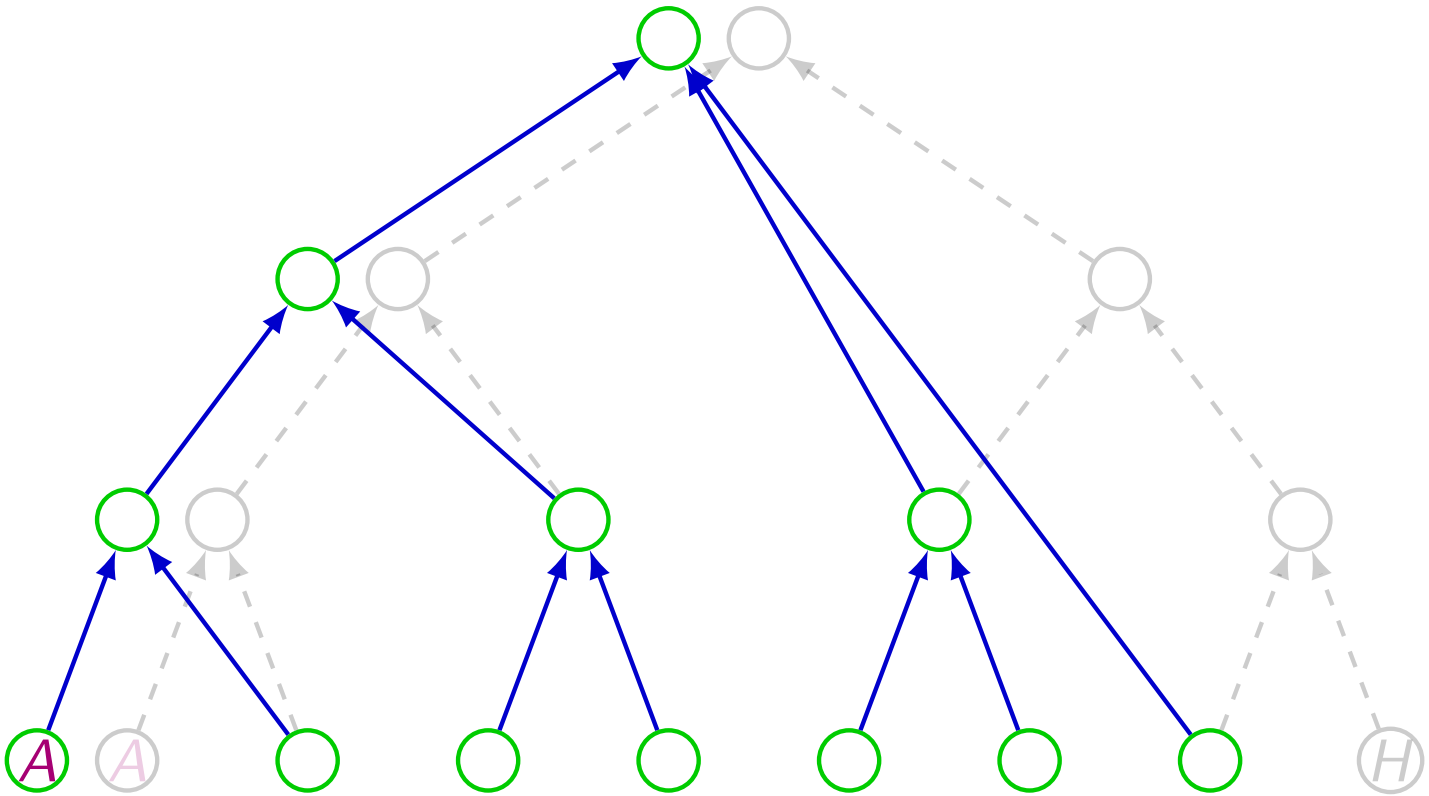


Adversary still has knowledge of the group key!

TreeKEM: Remove

Alice removes Henry by:

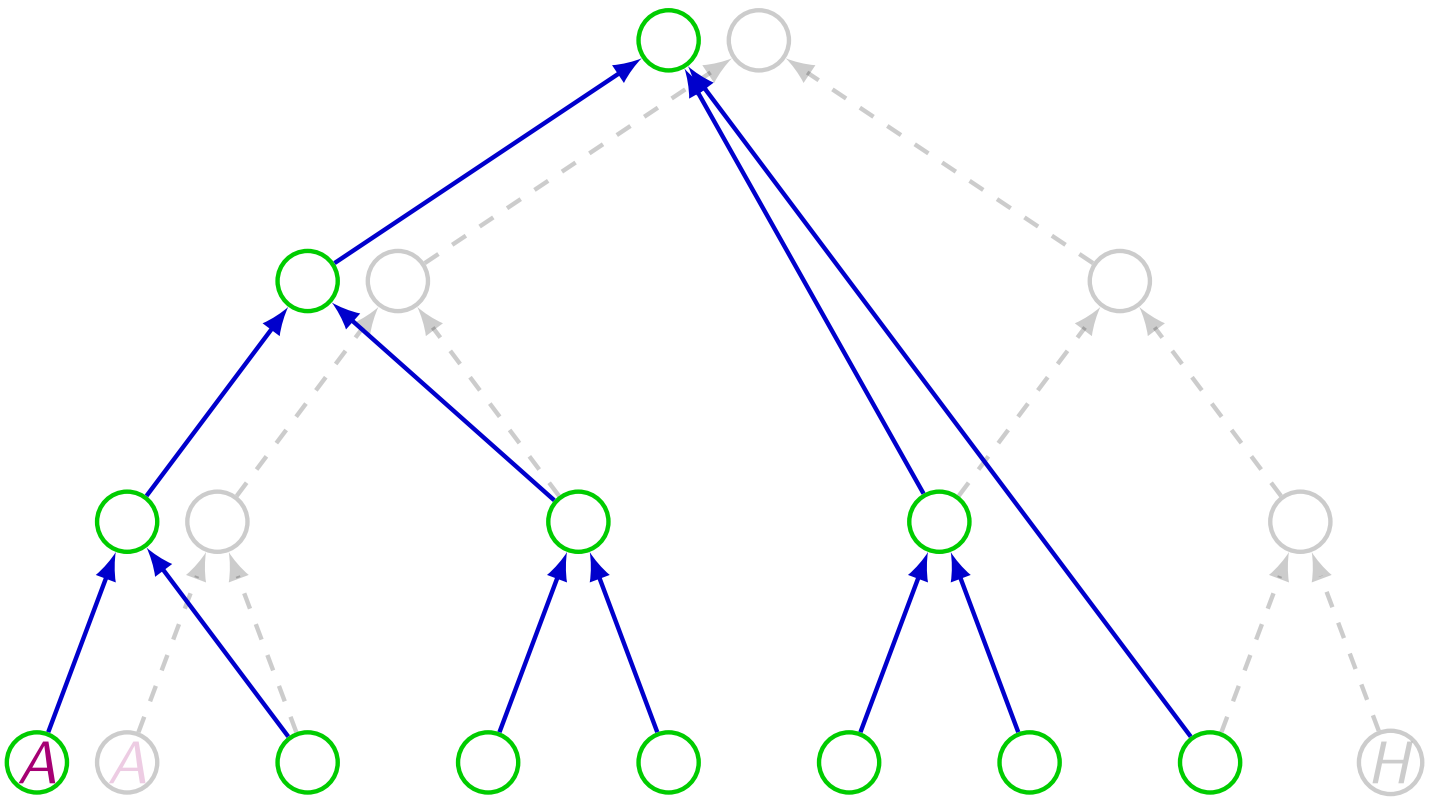
- "blanking"/deleting all nodes along Henry 's path.
- sampling a new group key.



TreeKEM: Remove

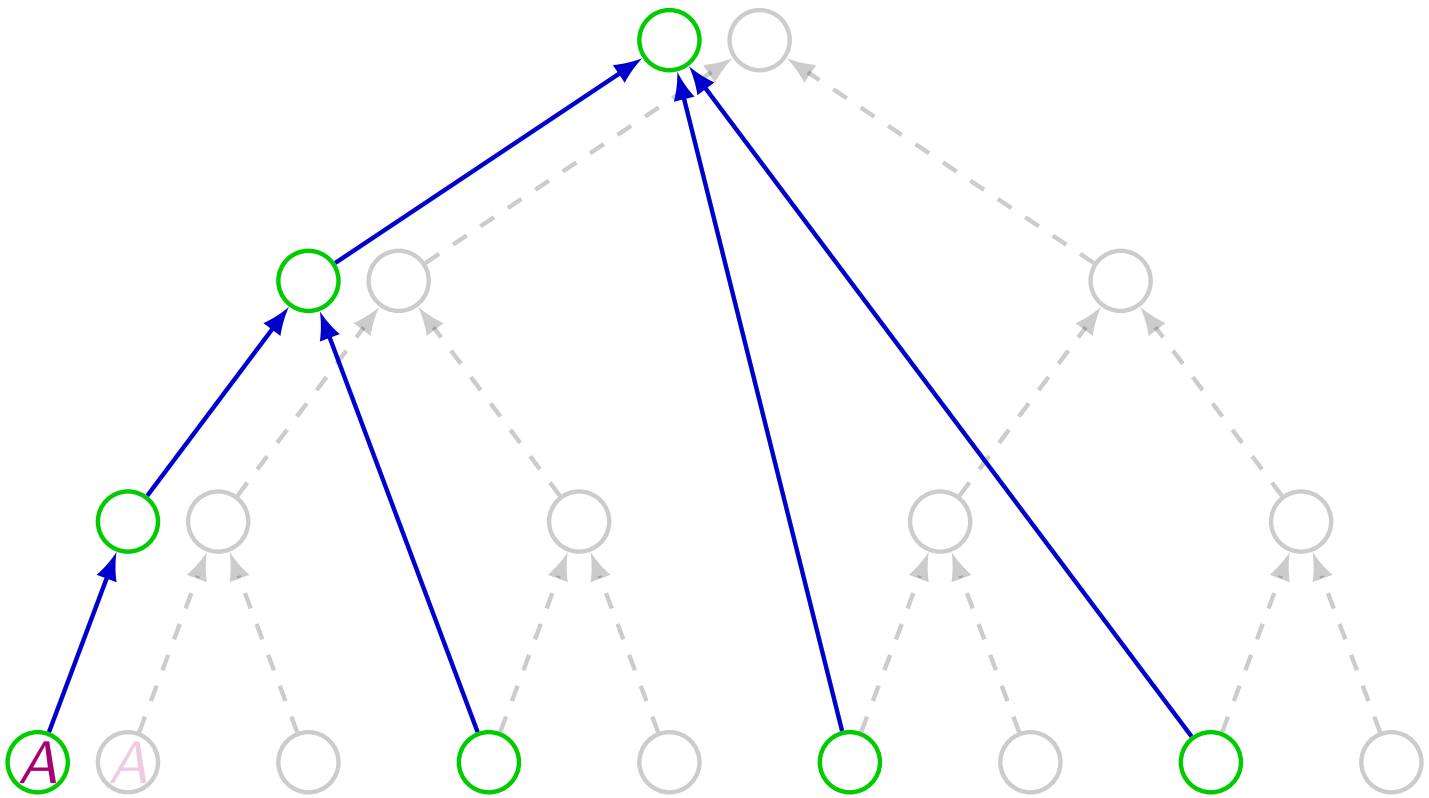
Alice removes Henry by:

- "blanking"/deleting all nodes along Henry 's path.
- sampling a new group key.



Blank nodes unblanked as parties update.

Problem: if many users removed...

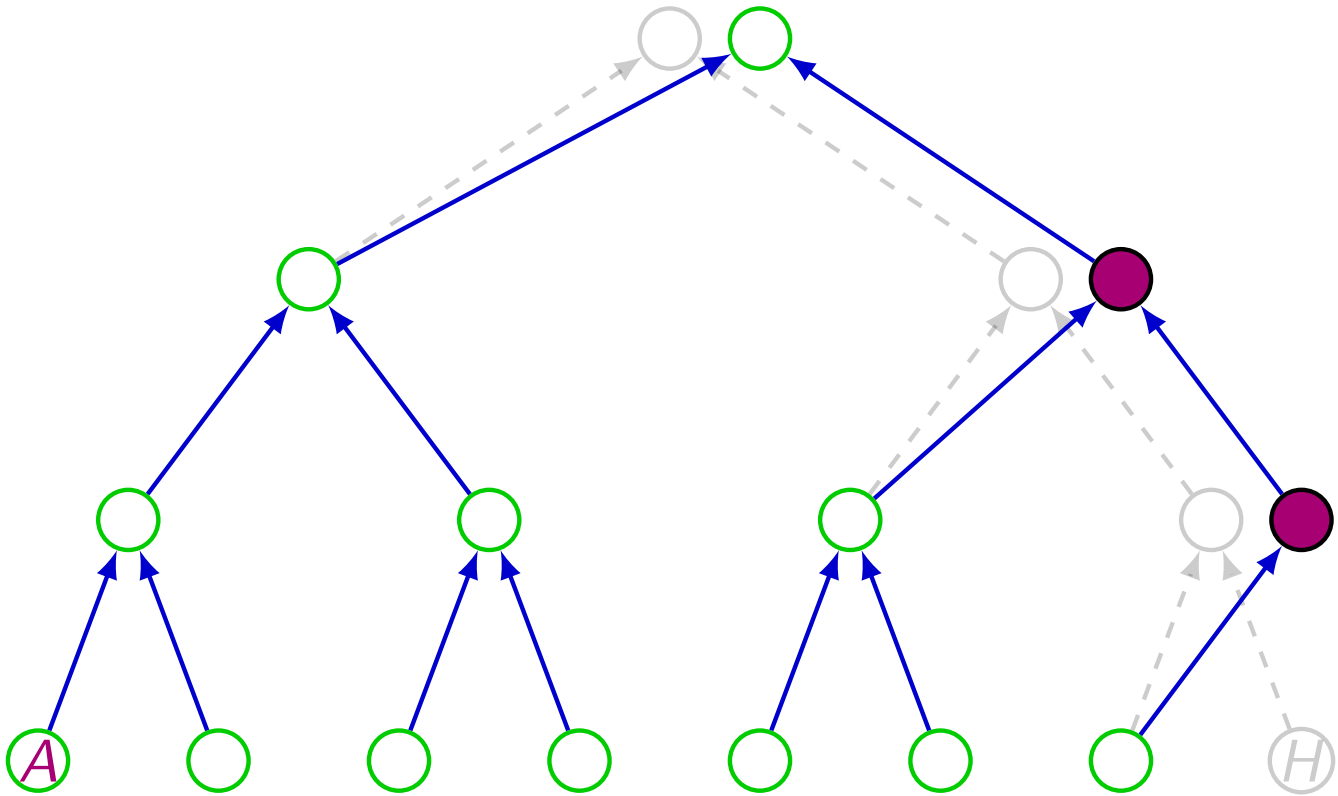


$\Rightarrow$ Linear packet size!!

Tainted TreeKEM (TTKEM) (this work)

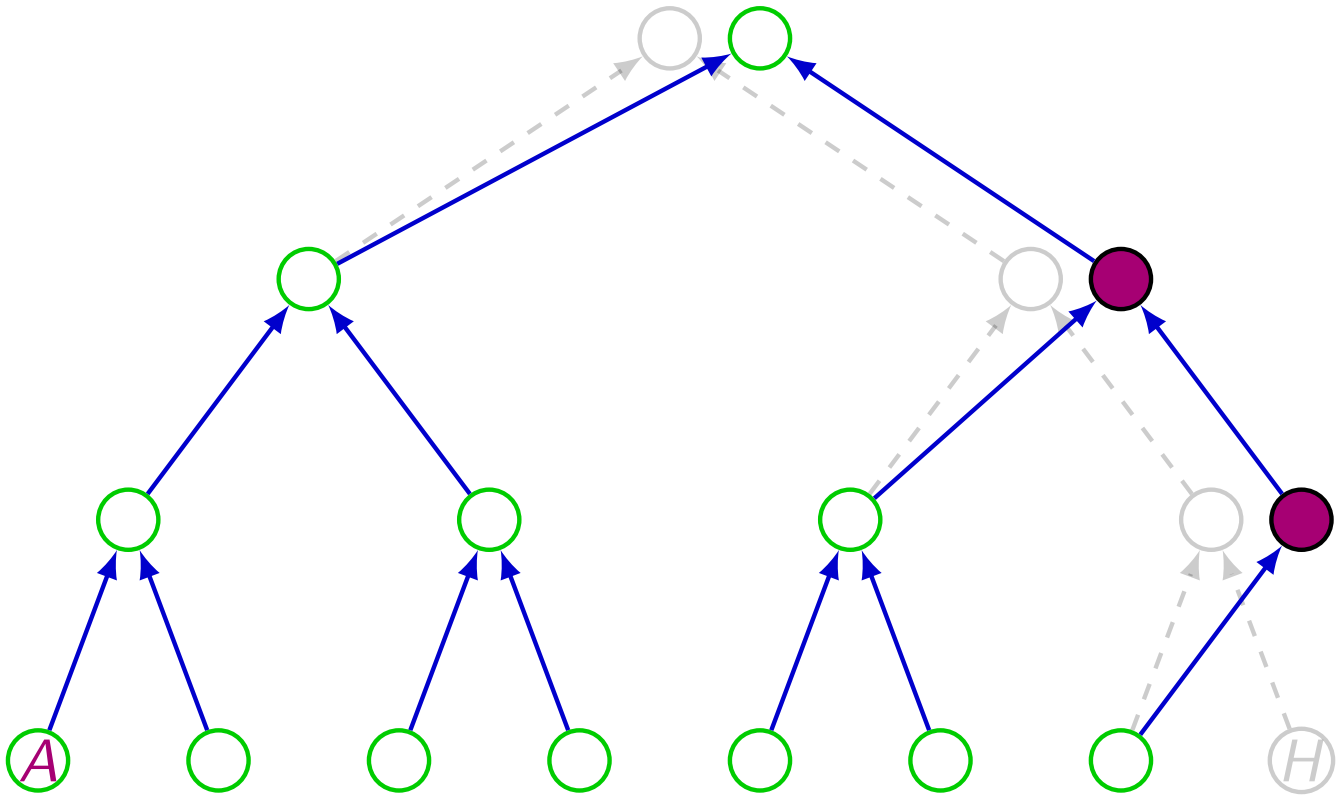
- CGKA variant of TreeKEM **without blanking**.
- More **efficient** under natural distributions of group operations.
- Secure against **adaptive** adversaries with **full network control**.
  - First adaptive proof for a CGKA/TreeKEM-related protocol with **polynomial loss**.

Tainted TreeKEM (TTKEM): Removal



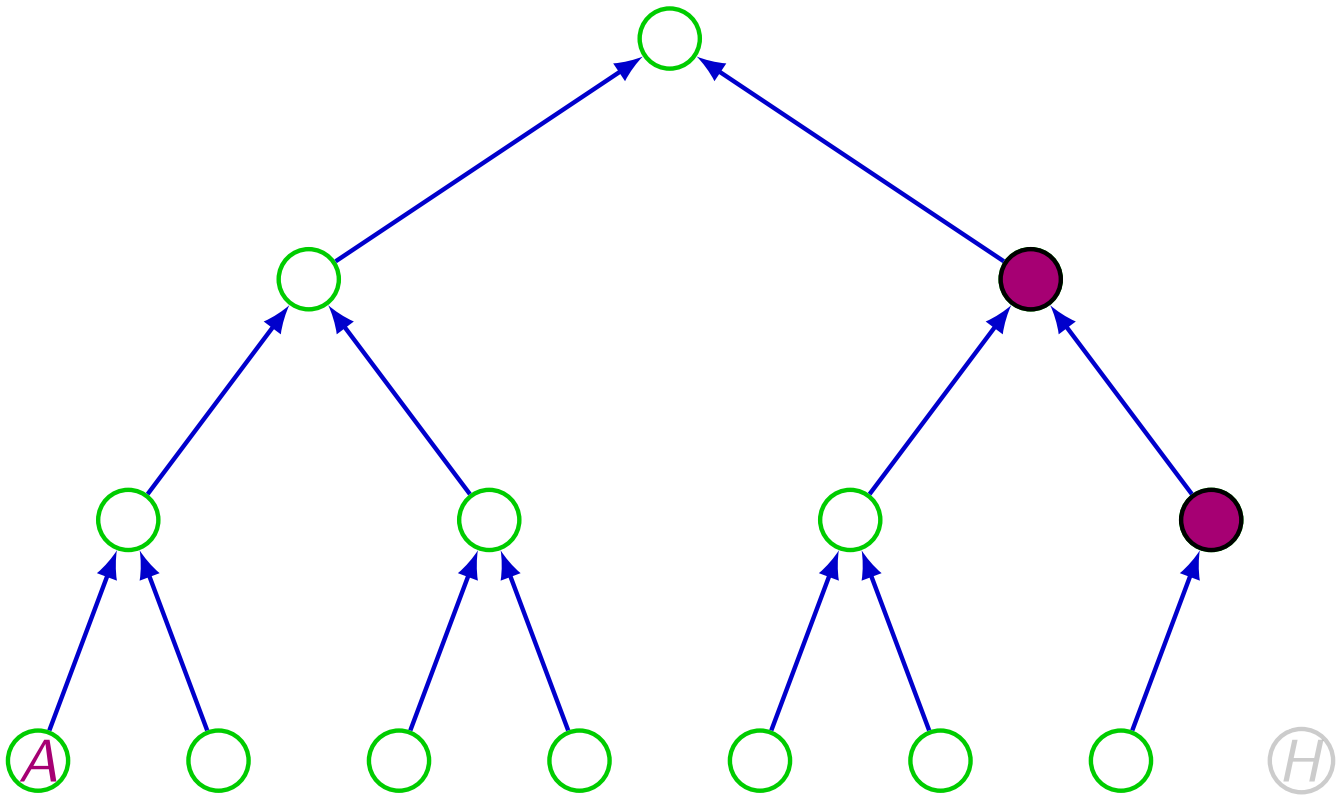
Tainted TreeKEM (TTKEM): Removal

- Allowed to sample keys outside own path $\rightarrow$ tainted nodes.
- Keep track of tainted nodes.



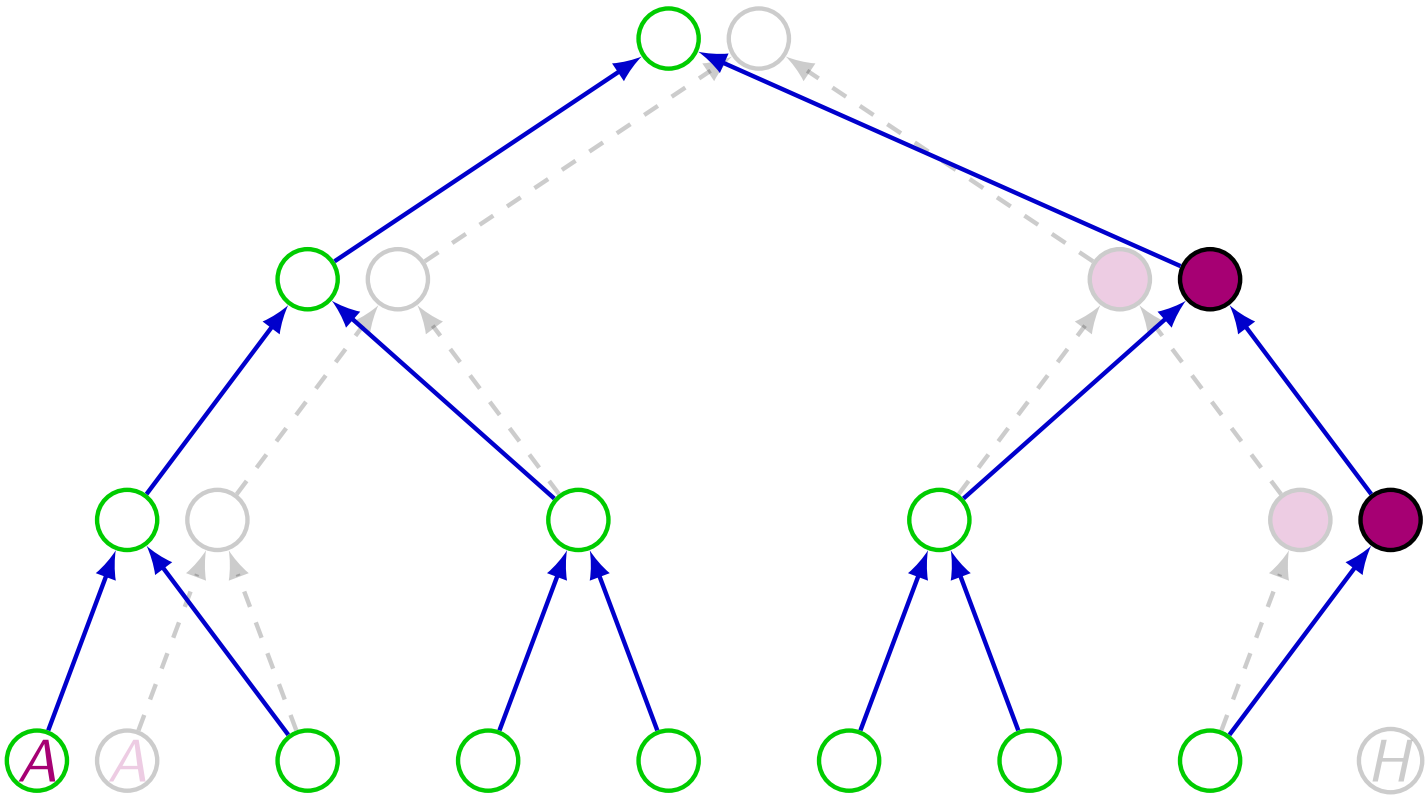
TTKEM: Update

Alice updates having tainted nodes



TTKEM: Update

Tainted nodes need to be re-sampled:



Efficiency Trade-offs

- Who is **affected** by it?
  - A **blank** affects *anyone* whose co-path contains it.
  - A **taint** affects *only the tainter*, but irrespective of position.

Efficiency Trade-offs

- Who is **affected** by it?
  - A **blank** affects *anyone* whose co-path contains it.
  - A **taint** affects *only the tainter*, but irrespective of position.
- When does a node **heal**?
  - A **blank** requires user in sub-tree to sample a new key for it.
  - A **taint** *also* requires all its children to be untainted.

Efficiency Comparison

TreeKEM recent version uses **Commit framework**:

- Group operations bundled into batches.
- Executed at once together with an update.

Efficiency Comparison

TreeKEM recent version uses **Commit framework**:

- Group operations bundled into batches.
- Executed at once together with an update.

Compared **TTKEM** against two variants of TreeKEM:

Efficiency Comparison

TreeKEM recent version uses **Commit framework**:

- Group operations bundled into batches.
- Executed at once together with an update.

Compared **TTKEM** against two variants of TreeKEM:

- **TKEM**: Ignores the update following each Commit.
  - **More efficient** than TreeKEM.

Efficiency Comparison

TreeKEM recent version uses **Commit framework**:

- Group operations bundled into batches.
- Executed at once together with an update.

Compared **TTKEM** against two variants of TreeKEM:

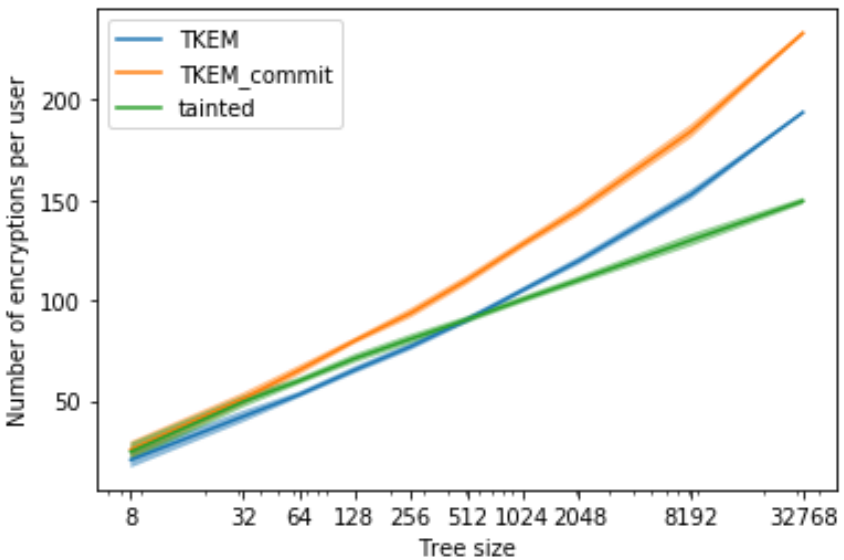
- **TKEM**: Ignores the update following each Commit.
  - **More efficient** than TreeKEM.
- **TKEM_commit**: Each Commit contains a single operation.
  - **Less efficient** than TreeKEM.

Efficiency Comparison, setting I: No administrators

- Adders and Removers sampled uniformly.
- Updaters follow either **Zipf** or **uniform** distribution.

Efficiency Comparison, setting I: No administrators

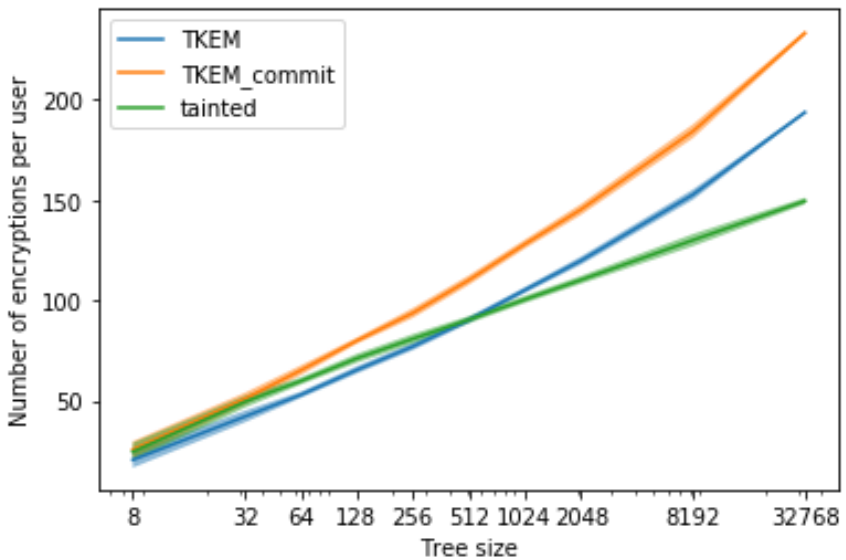
- Adders and Removers sampled uniformly.
- Updaters follow either **Zipf** or **uniform** distribution.



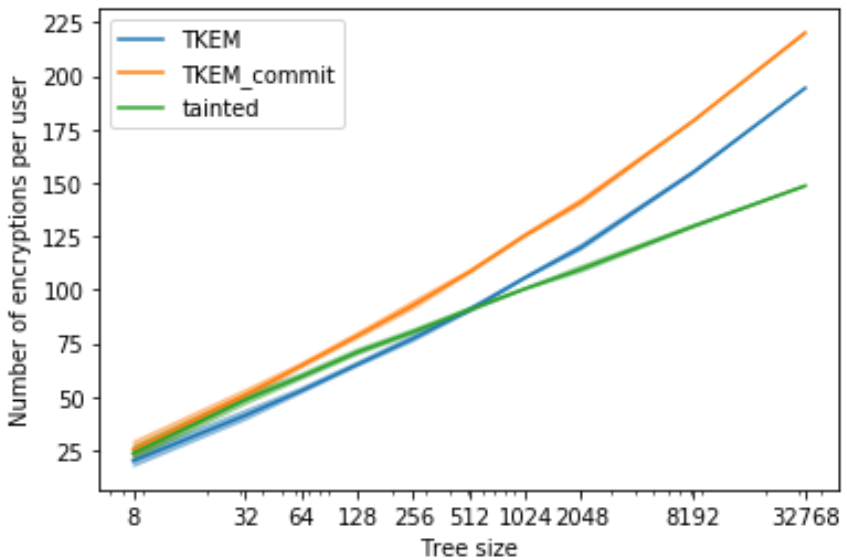
Updaters follow Zipf dist.

Efficiency Comparison, setting I: No administrators

- Adders and Removers sampled uniformly.
- Updaters follow either **Zipf** or **uniform** distribution.



Updaters follow Zipf dist.



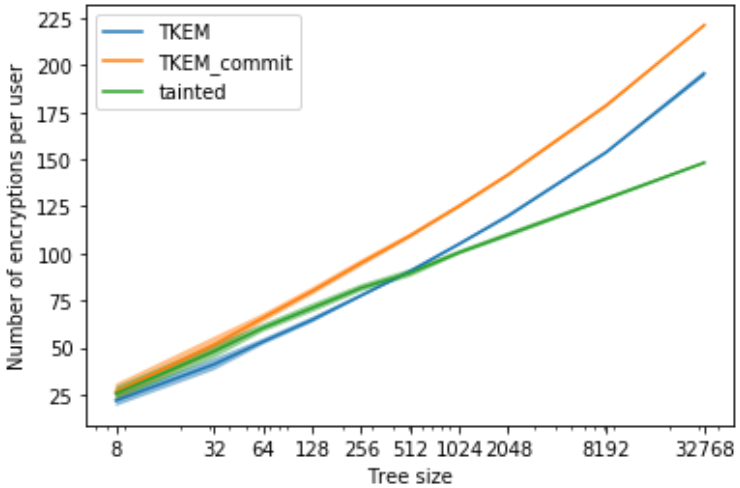
Updaters follow uniform dist.

Efficiency Comparison, setting II: Administrators

- Adds/Removes performed by a small set of **administrators**
- Updaters sampled uniformly.

Efficiency Comparison, setting II: Administrators

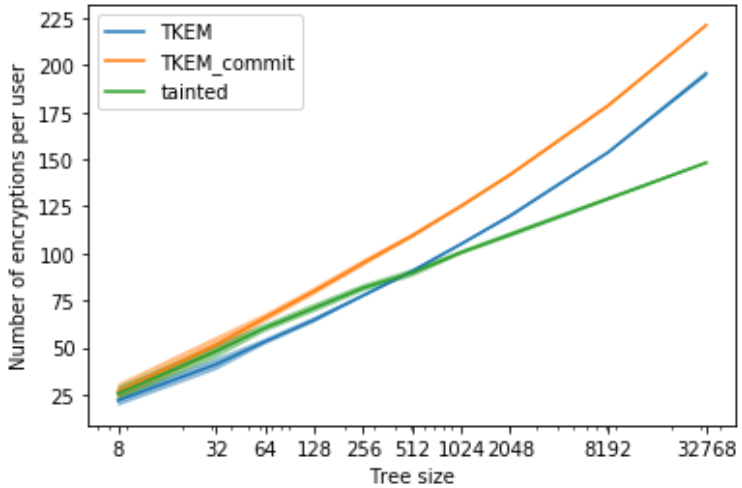
- Adds/Removes performed by a small set of **administrators**
- Updaters sampled uniformly.



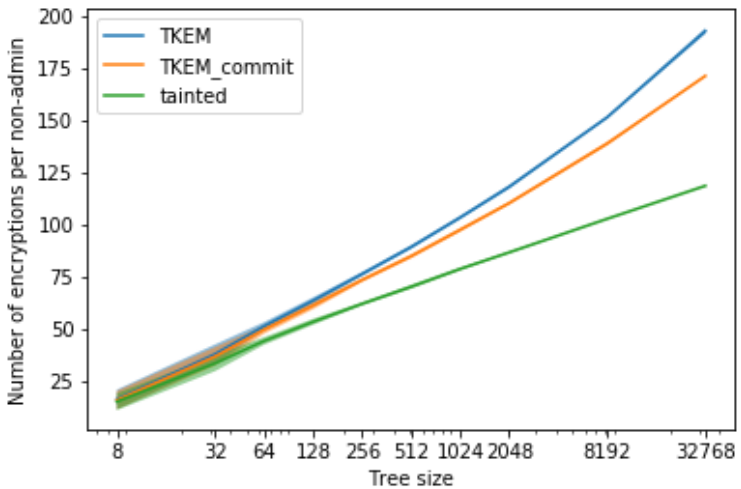
Average cost per user

Efficiency Comparison, setting II: Administrators

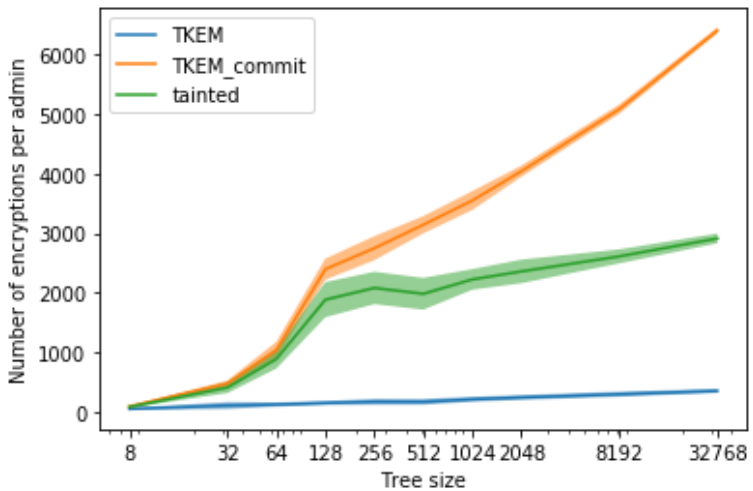
- Adds/Removes performed by a small set of **administrators**
- Updaters sampled uniformly.



Average cost per user



Cost for non-administrators



Cost for administrators

Security overview

Security overview

Adversarial Model:

Security overview

Adversarial Model:

- Controls protocol execution and can corrupt users **adaptively**.

Security overview

Adversarial Model:

- Controls protocol execution and can corrupt users **adaptively**.
- Corruption window:
  - leaks all user state.
  - randomness used while corrupted.

Security overview

Adversarial Model:

- Controls protocol execution and can corrupt users **adaptively**.
- Corruption window:
  - leaks all user state.
  - randomness used while corrupted.
- **"Partially" active**:
  - Full network control: can force parties into inconsistent states.
  - Not allowed to craft messages.

Security overview

Adversarial Model:

- Controls protocol execution and can corrupt users **adaptively**.
- Corruption window:
  - leaks all user state.
  - randomness used while corrupted.
- **"Partially" active**:
  - Full network control: can force parties into inconsistent states.
  - Not allowed to craft messages.
- **Challenge**: Distinguish group key from random.
  - Challenge must not be trivial: define **safe** predicate

Security overview

Q - # of operations; n - # of users

Theorem (Standard Model)

Enc ϵ-*IND-CPA secure*, H ϵ-*pseudorandom*

$\Rightarrow$ *TTKEM* $\epsilon \cdot Q^{\log(n)}$-*CGKA-secure*.

Theorem (Random Oracle Model)

Enc ϵ-*IND-CPA secure*, H *random oracle*

$\Rightarrow$ *TTKEM* $\epsilon \cdot (Qn)^2$-*CGKA-secure*.

Security overview

Q - # of operations; n - # of users

Theorem (Standard Model)

Enc ϵ-*IND-CPA secure*, H ϵ-*pseudorandom*

$\Rightarrow$ *TTKEM* $\epsilon \cdot Q^{\log(n)}$-*CGKA-secure*.

Theorem (Random Oracle Model)

Enc ϵ-*IND-CPA secure*, H *random oracle*

$\Rightarrow$ *TTKEM* $\epsilon \cdot (Qn)^2$-*CGKA-secure*.

- Results apply to TreeKEM.

Summary of our results:

- New variant of TreeKEM with **tainting** instead of **blanking**.
- More efficient under natural distributions.
- First adaptive security proof for any CGKA with only **polynomial loss**.

Summary of our results:

- New variant of TreeKEM with **tainting** instead of **blanking**.
- More efficient under natural distributions.
- First adaptive security proof for any CGKA with only **polynomial loss**.

Open Problems:

- Can we extend security to malicious insiders?
- More efficient protocols? New approaches?
- Get better comparison using real world access patterns.

Thanks!