

Resolving Concurrency in Group Ratcheting Protocols

RUB



Secure Messaging Summit 2020

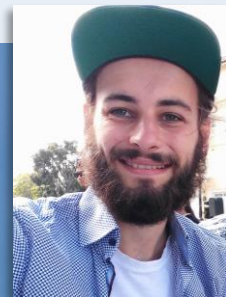
2020-09-03

Cryptography Group
New York University

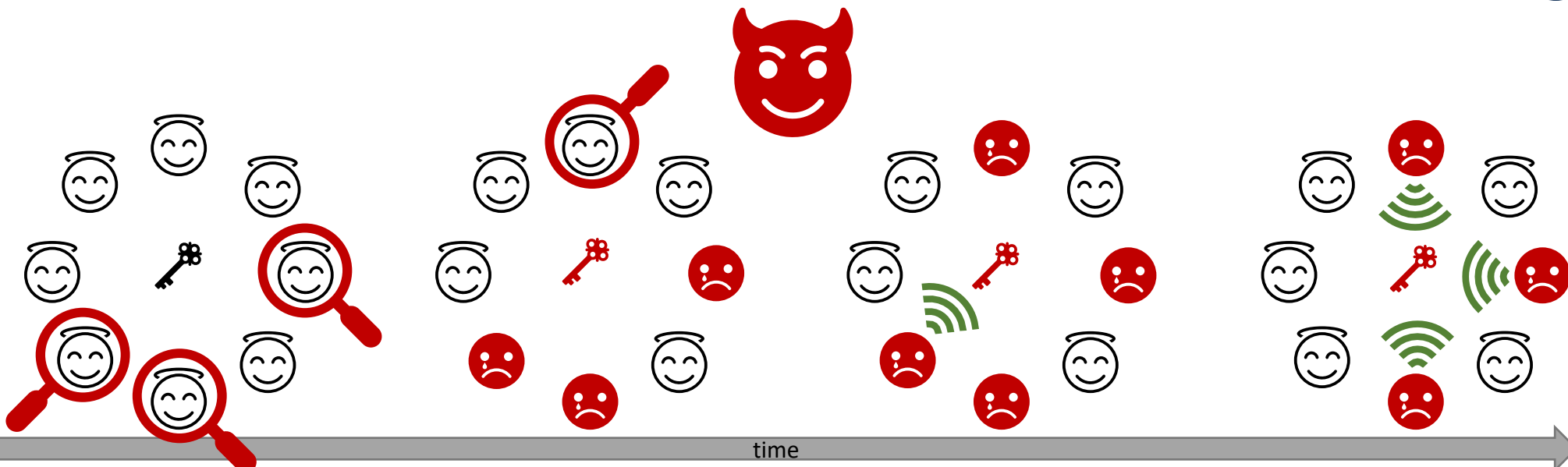
Alexander Bienstock, Yevgeniy Dodis,

Horst Görtz Institute for IT Security
Chair for Network and Data Security
Ruhr University Bochum

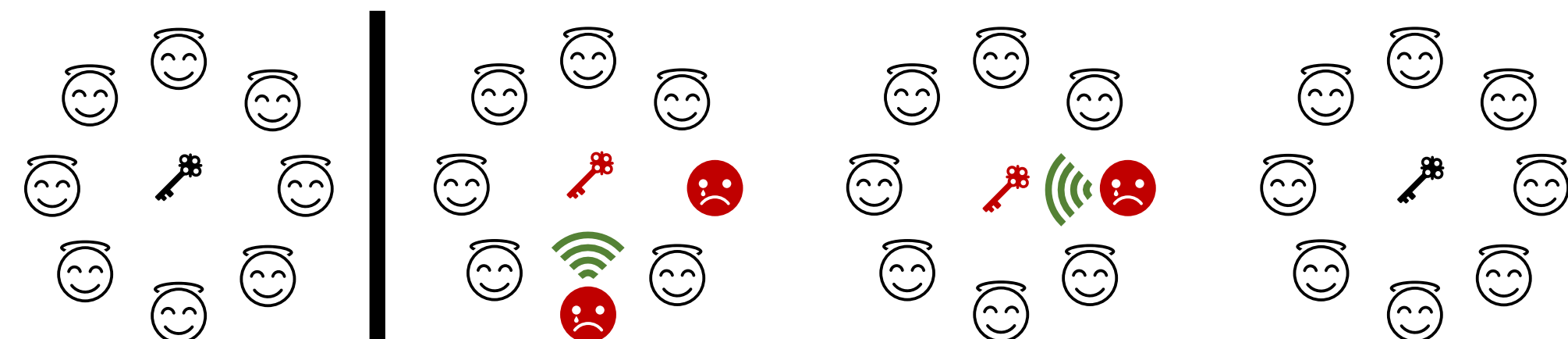
Paul Rösler



(Concurrent) Group Ratcheting



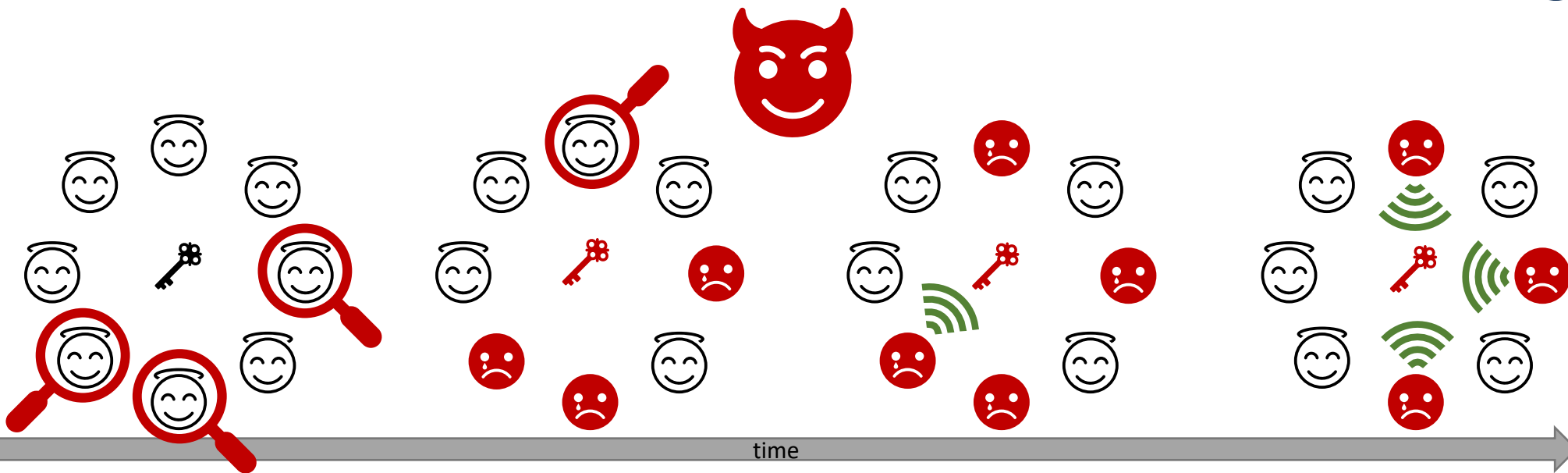
- Group computes joint keys
- Exposure of local state temporarily
 - Long-term sessions, mobile devices etc.
 - Leaks group key until all states recovered



- Recovery:
 - Generate new secrets
 - Share public values
- Concurrent recovery
 - Speedup
 - Merge recoveries

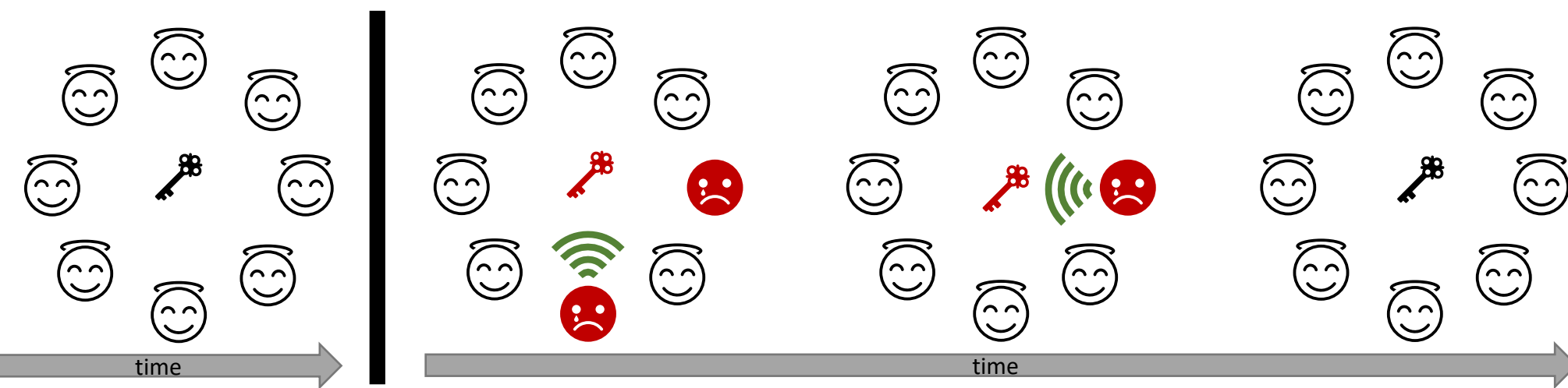
Here: one group, concurrent users.
[CHK'19]: Many groups, sequential users.

(Concurrent) Group Ratcheting



Target:

1. Post-Compromise Security
2. Small shares
3. Concurrency



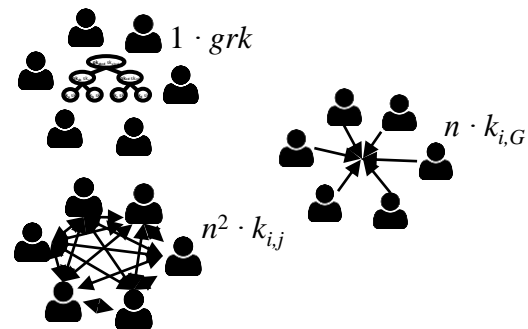
concurrent

sequential

Otherwise:

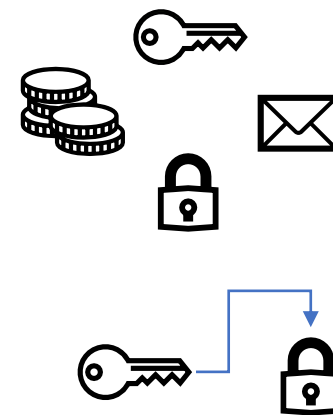
- Slow recovery from exposures
 - Consensus required
- Inapplicable to decentralized networks

Agenda



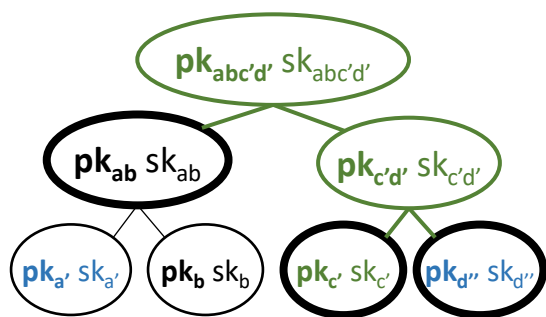
Previous Work:
What's the Problem?

Lower Bound:
What's the minimal overhead?



Upper Bound:
Almost optimal construction

Open Questions ...

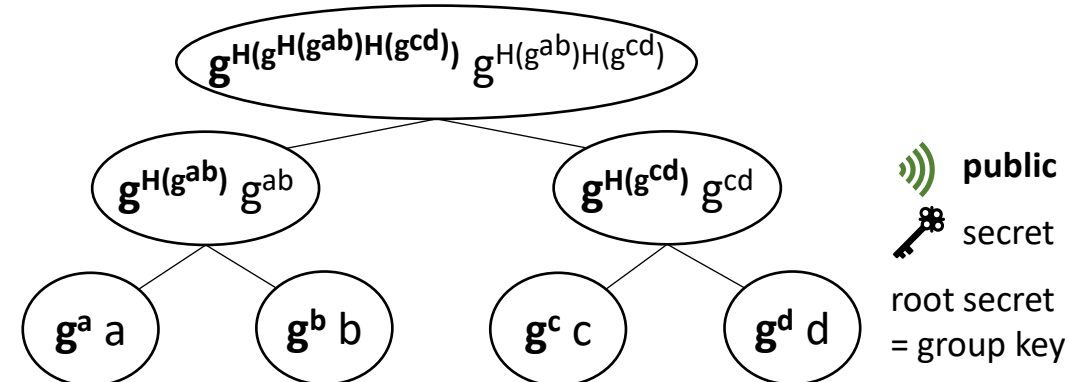
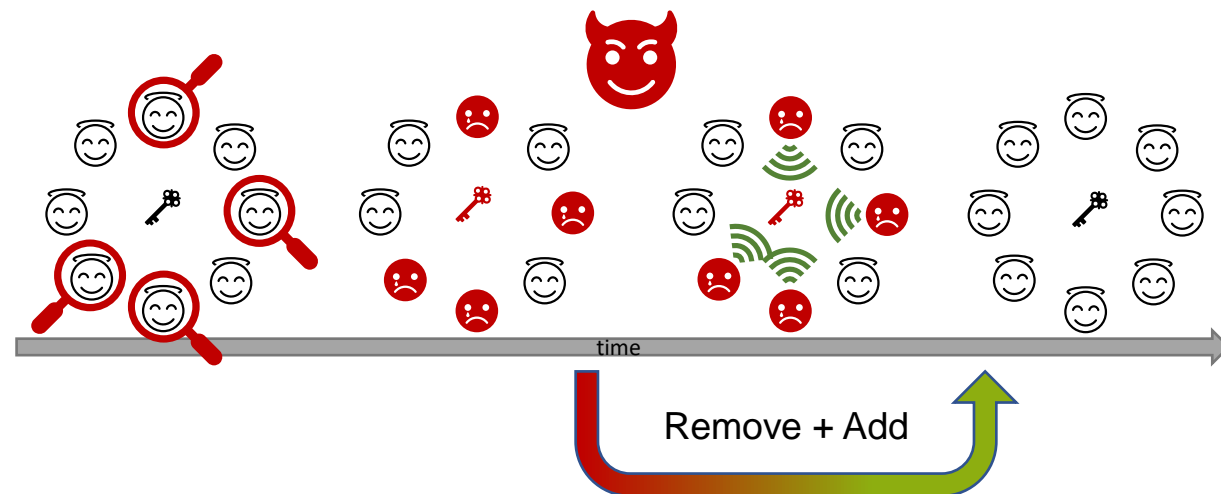


$\Omega(t)$ vs. $O(t \cdot (1 + \log(n/t)))$?
NIKE?
PCS-Delay?
Forward-secretity?
Application to MLS

Previous Work: What's the Problem?

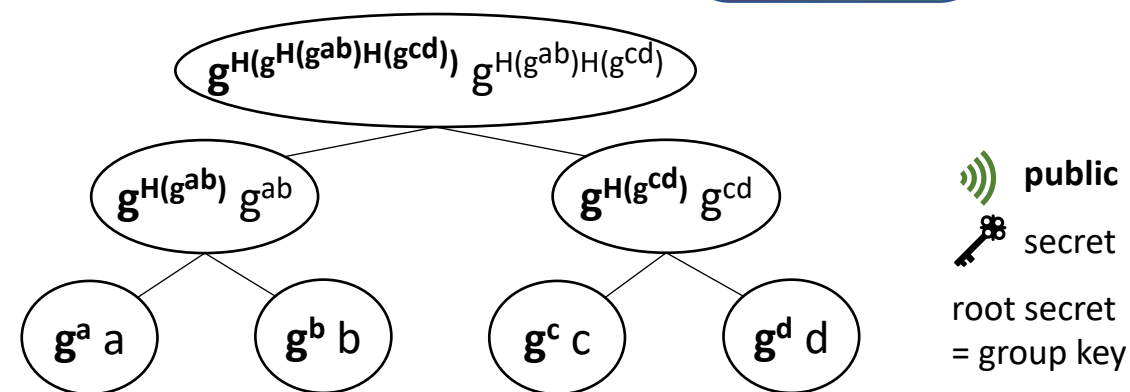
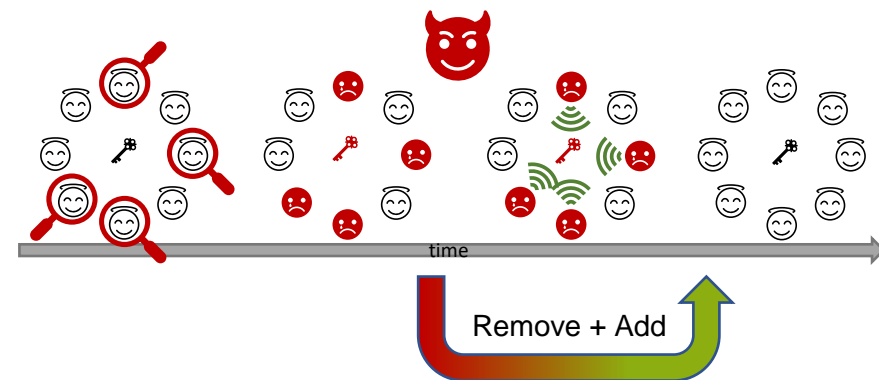
- Essentially: Dynamic group key exchange (DGKE)

- Expose = Unwanted member
 - Recovery = Remove + Add (R&A)
 - Many protocols from '80s – '00ers
 - Tree-based DGKE best suited for asynchronous settings:
 - Little communication for R&A: $O(\log n)$
 - (Almost) non-interactive for R&A
- First known DH-tree-based protocol [KPT'04]



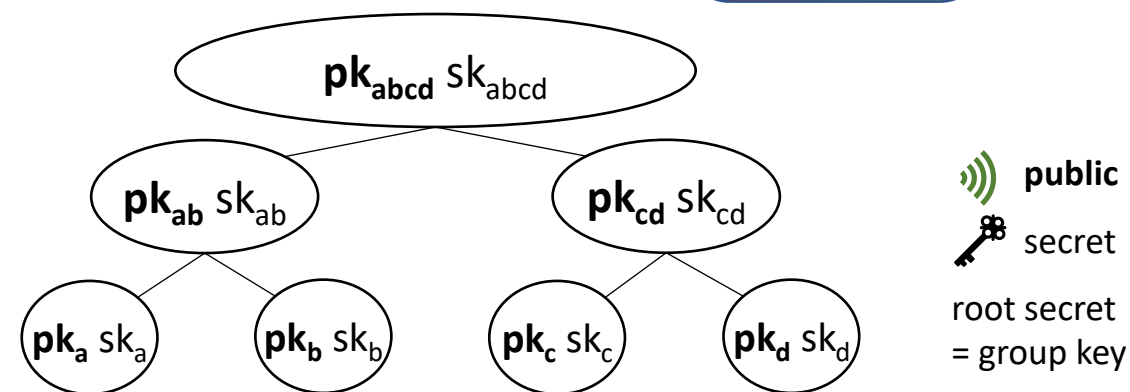
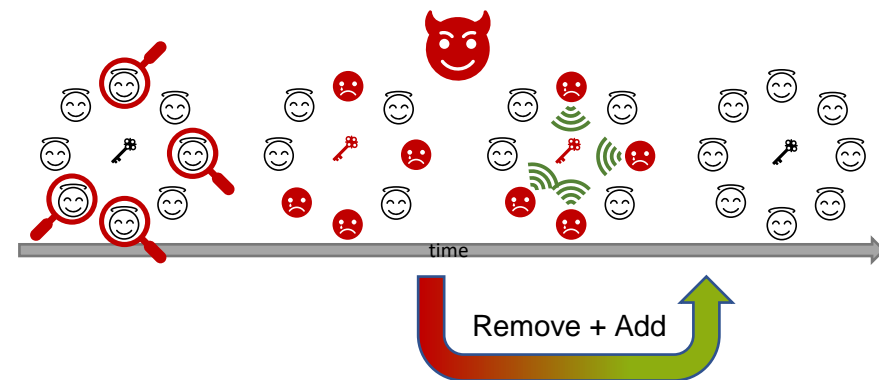
Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings
- Ratcheting in trees
 - Merge R&A [CCGMM'18]



Previous Work: What's the Problem?

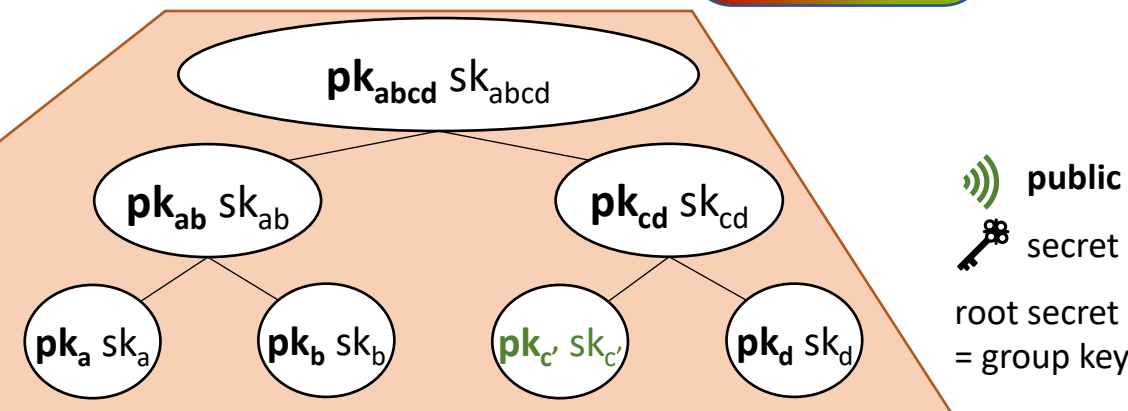
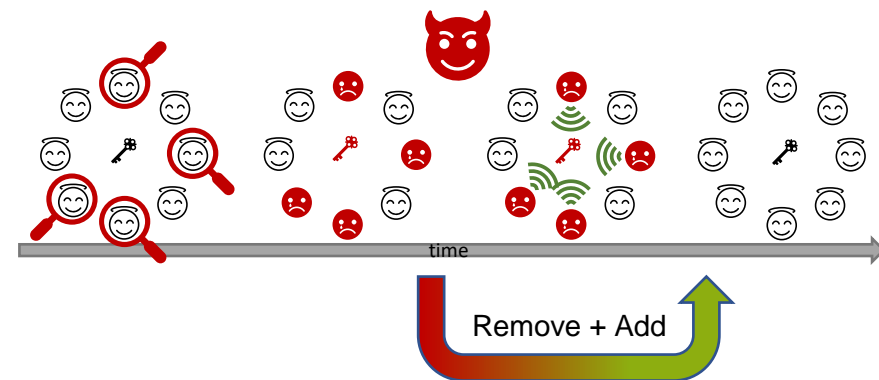
- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings
- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]



Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings
- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]

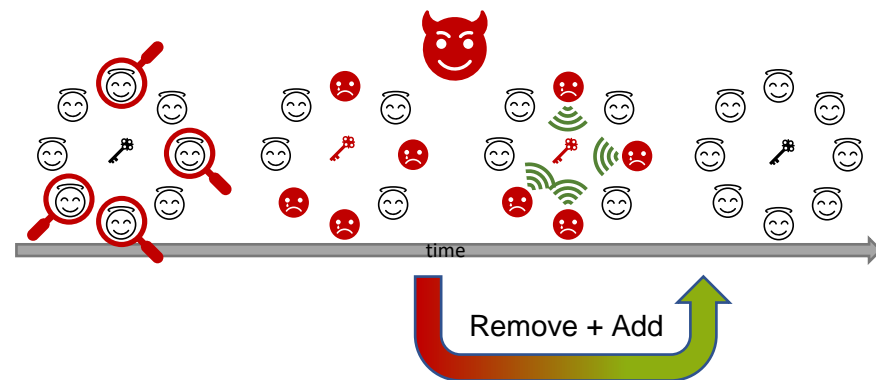
Recovery: sample $x_{c'}$, $sk_{c'} = x_{c'}$, $pk_{c'} = \text{gen}(sk_{c'})$,



Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)

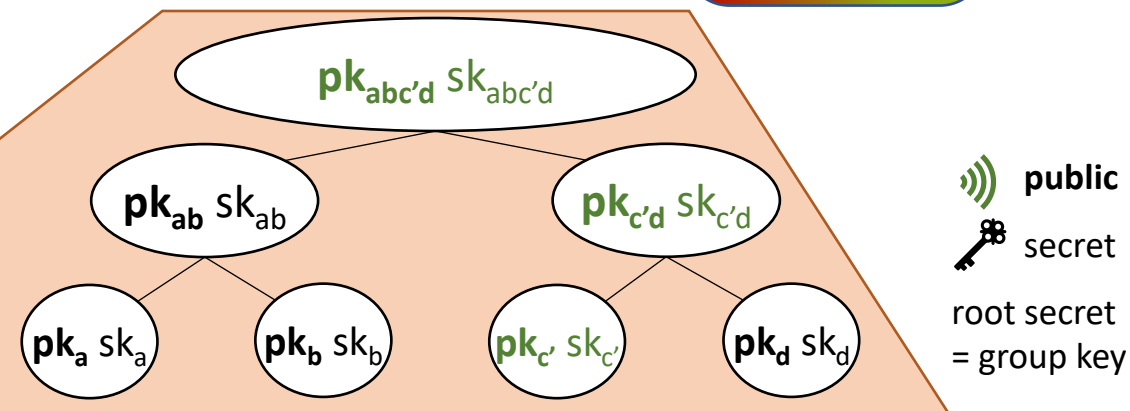
- Expose = Unwanted member
- Recover = Remove + Add (R&A)
- Tree-based DGKE best suited for asynchronous settings



- Ratcheting in trees

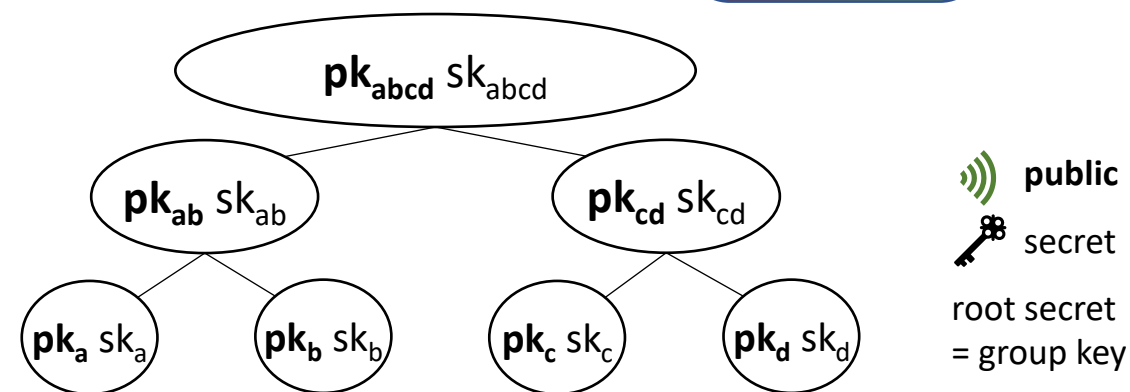
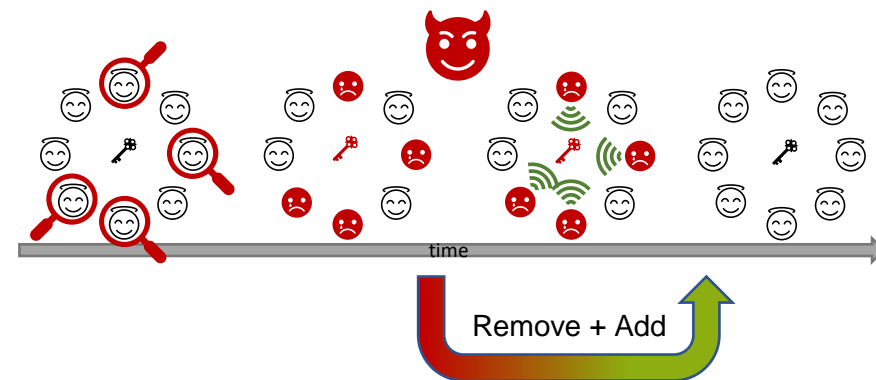
- Merge R&A [CCGMM'18]
- DH to KEM [BBR'18]

Recovery: sample $x_{c'}$, $sk_{c'}=x_{c'}$, $pk_{c'}=gen(sk_{c'})$, $x_{c'd}=H(x_{c'})$, $enc(pk_d, x_{c'd})$, $sk_{c'd}=x_{c'd}$, ...



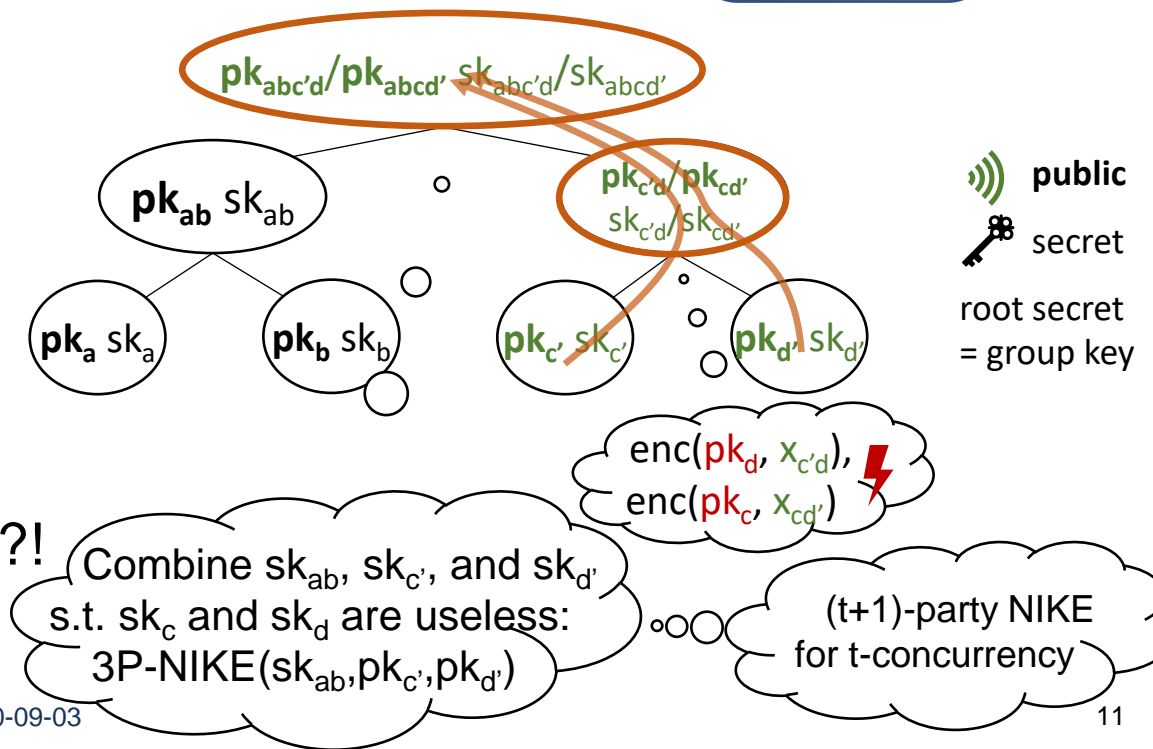
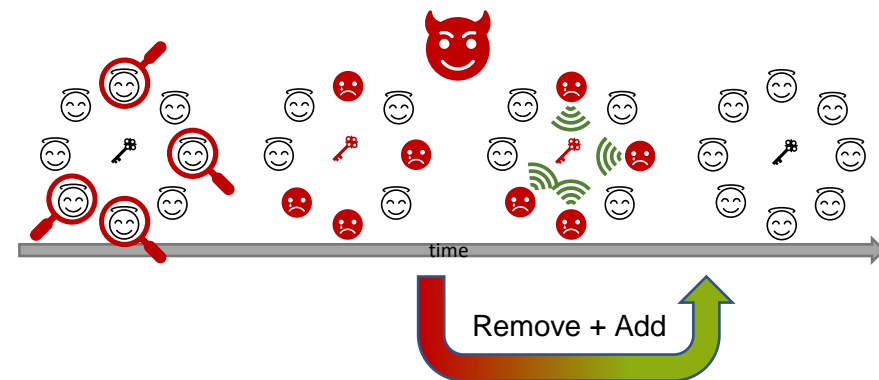
Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings
- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-secrecy [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]



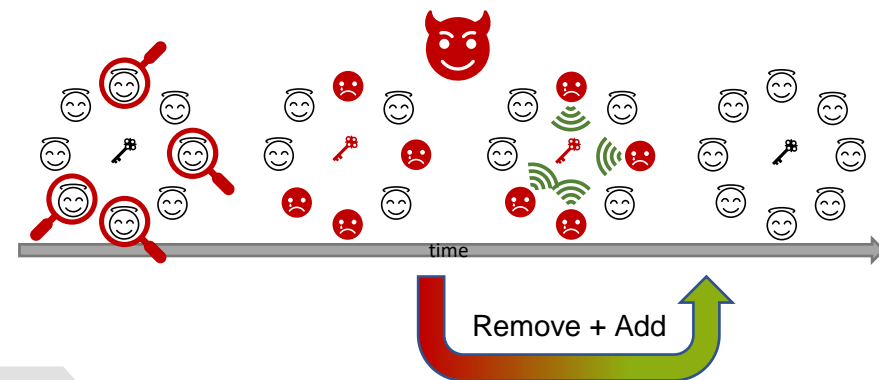
Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings
- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-secrecy [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]
- No concurrency
 - Intersection of concurrently updated paths
→ Merging under PCS without multiparty-NIKE?!

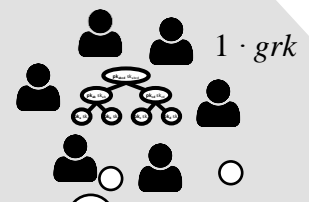


Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings



- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-secrecy [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]



PCS	Overhead	Concurrency
✓	$O(\log n)$	✗

MLSv9 worst-case:

(✓)	$O(n)$	(✓)
-----	--------	-----

- Rejects concurrent path updates
- Degrades to "n-tree"

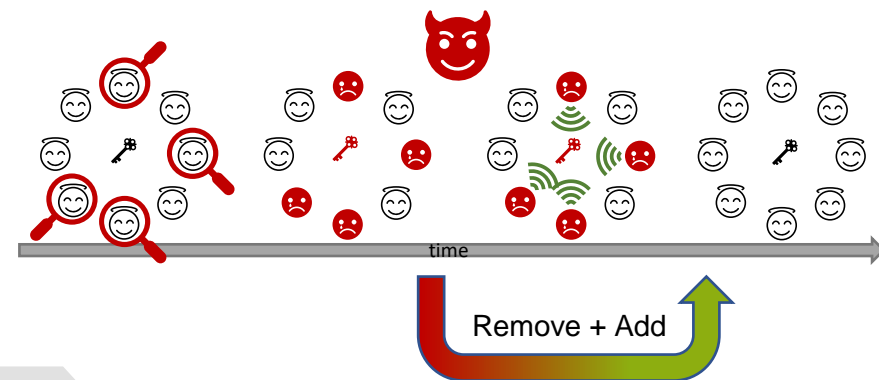
Merge DH Tree [Weidner'19]:

(✗)	$O(\log n)$	✓
-----	-------------	---

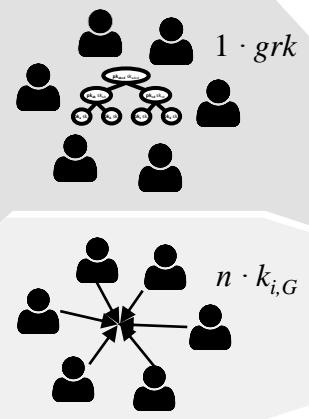
- New DH paths are merged
- Recovers only one user at a time

Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings



- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-secrecy [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]

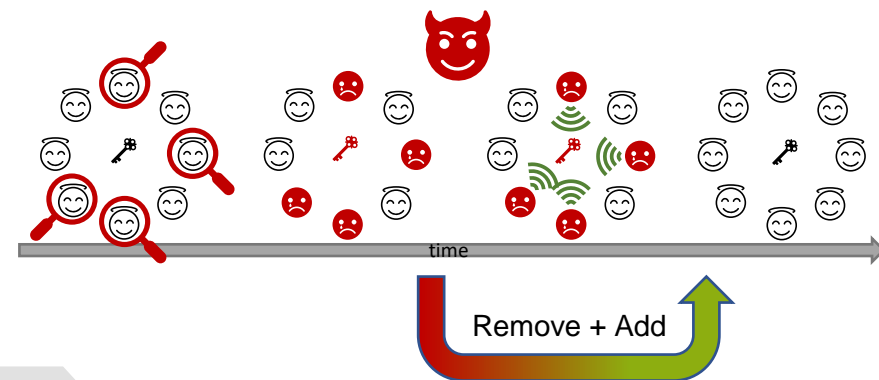


PCS	Overhead	Concurrency
✓	$O(\log n)$	✗
✗	$O(1)$	✓

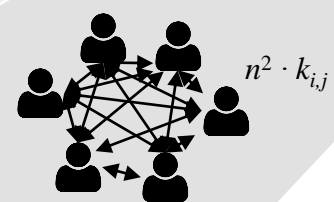
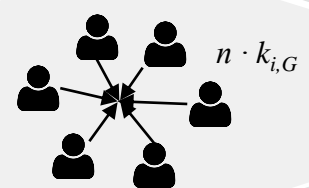
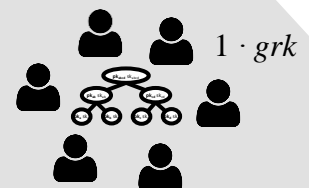
- Real-World
 - Forward-secure hash chain [WhatsApp]

Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings



- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-security [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]

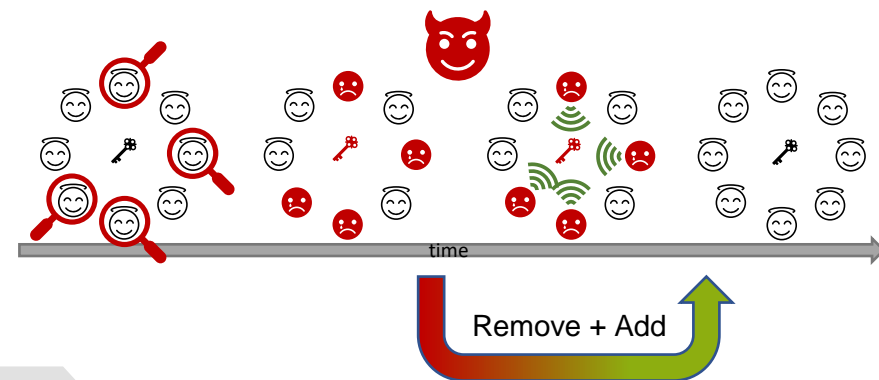


- Real-World
 - Forward-secure hash chain [WhatsApp]
 - Parallel pair-wise communication [Signal]

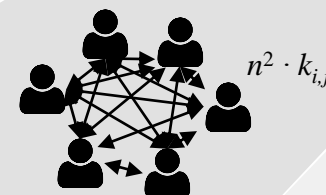
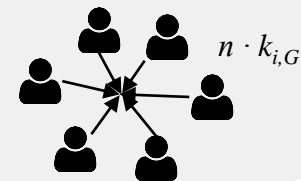
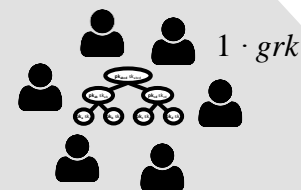
PCS	Overhead	Concurrency
✓	$O(\log n)$	✗
✗	$O(1)$	✓
✓	$O(n)$	✓

Previous Work: What's the Problem?

- Essentially: Dynamic group key exchange (DGKE)
 - Expose = Unwanted member
 - Recover = Remove + Add (R&A)
 - Tree-based DGKE best suited for asynchronous settings



- Ratcheting in trees
 - Merge R&A [CCGMM'18]
 - DH to KEM [BBR'18]
 - Better forward-security [ACDT'20]
 - Maintain balanced tree [ACCKKPPW'19]

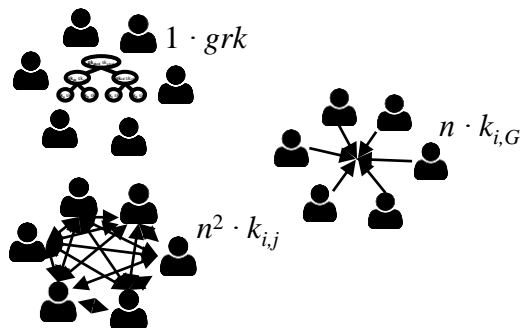


- Real-World
 - Forward-secure hash chain [WhatsApp]
 - Parallel pair-wise communication [Signal]

PCS	Overhead	Concurrency
✓	$O(\log n)$	✗
✗	$O(1)$	✓
✓	$O(n)$	✓
✓	?	✓

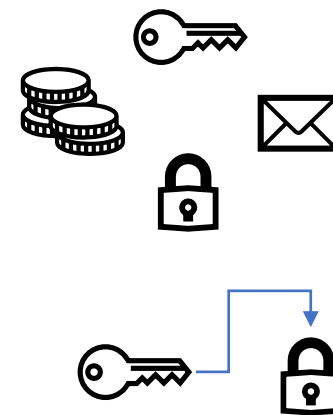
• We

Agenda

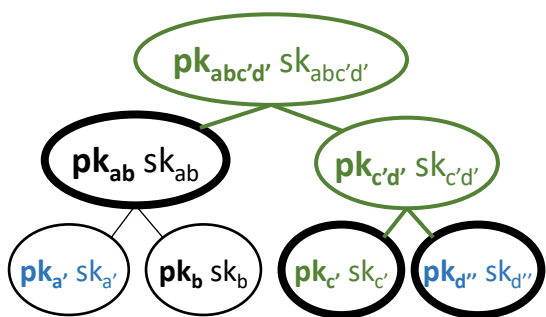


Previous Work:
What's the Problem?

Lower Bound:
What's the minimal overhead?



PCS & Concurrency & Small overhead
 PCS & Concurrency & Small overhead
 PCS & Concurrency & Small overhead



Upper Bound:
Almost optimal construction

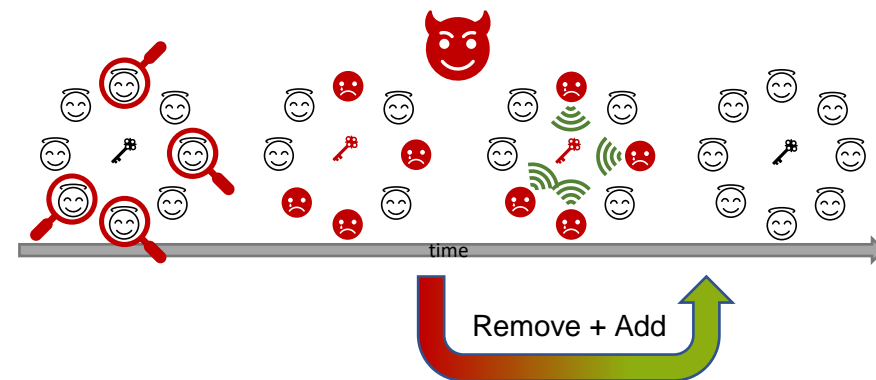
Open Questions ...

$\Omega(t)$ vs. $O(t \cdot (1 + \log(n/t)))$?
 NIKE?
 PCS-Delay?
 Forward-secretcy?
 Application to MLS

Lower Bound: What's the minimal overhead?

- Symbolic model

- Variables are symbols without bit representation or algebraic structure
- Algorithms follow “transition rules”
- Round based execution



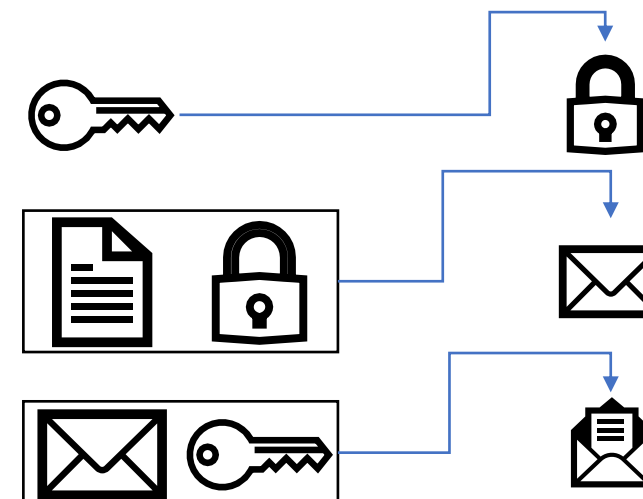
- Fixed set of allowed building blocks (for constructions with minimal overhead under PCS)

Our “transition rules” model:

- (Dual) pseudo-random functions
 - Key-updatable public key encryption (see [BRV20])
 - Broadcast encryption
- More than what previous constructions used

- Inspired by [MP04]:

Lower bound $O(\log n)$ for forward-secure DGKE



Lower Bound: What's the minimal overhead?

i-2 Exposure:

- No (shared) secrets

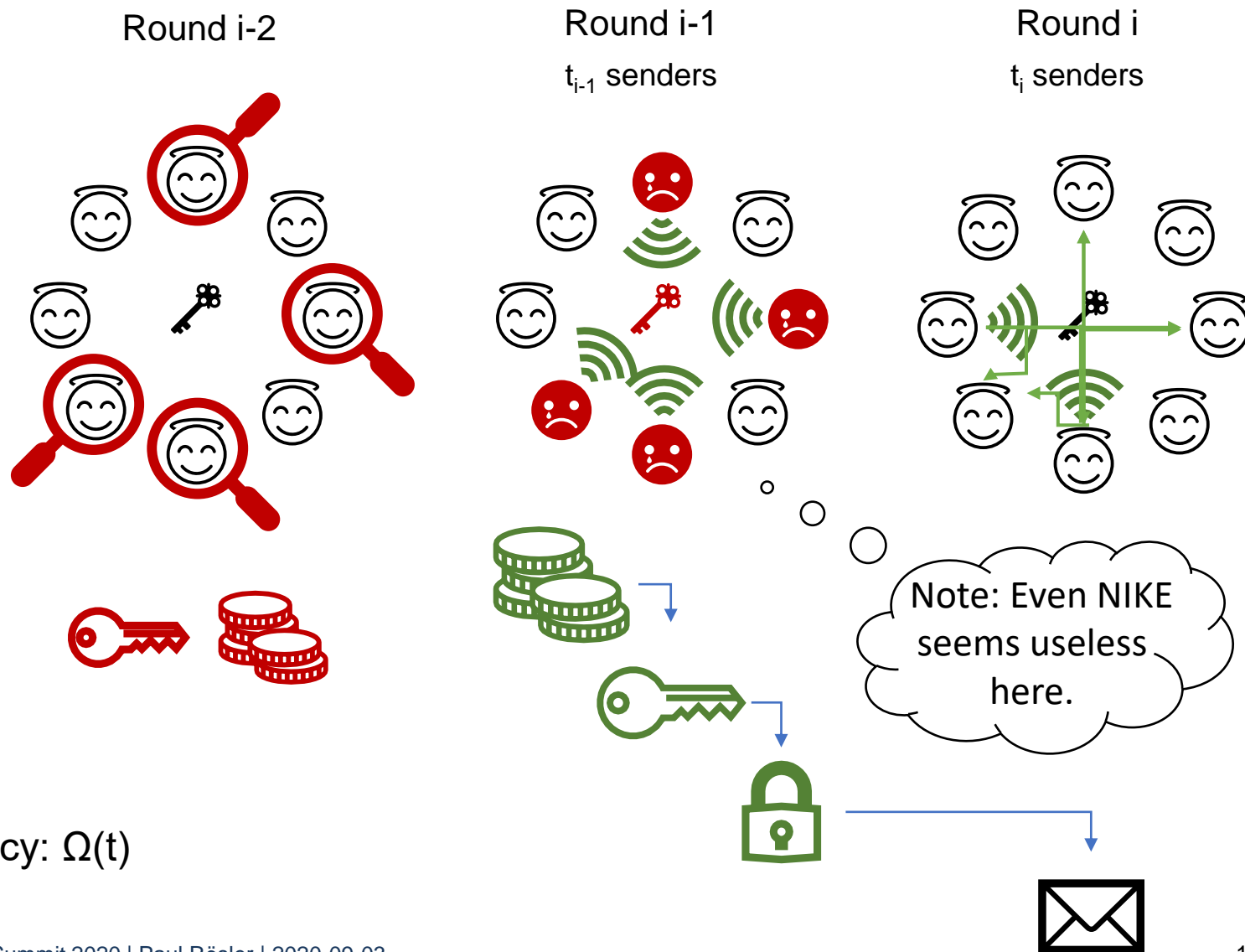
i-1 Recovery 1:

- Still no (shared) secrets
 - Sampling of new secrets
 - Sharing of derived values
- Still no (shared) secrets
 → Though, public values of shared secrets

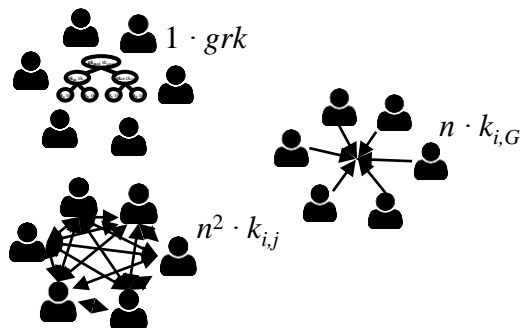
i Recovery 2:

- Respond to public values
 - All senders must respond as they cannot coordinate
- Each sender sends $\geq (t_{i-1}-1)$ responses
 → $\geq (t_{i-1}-1) \cdot t_i$ shares in round i

⇒ Overhead per recovery under t-concurrency: $\Omega(t)$

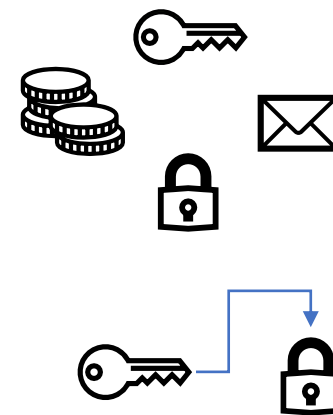


Agenda



Previous Work:
What's the Problem?

Lower Bound:
What's the minimal overhead?

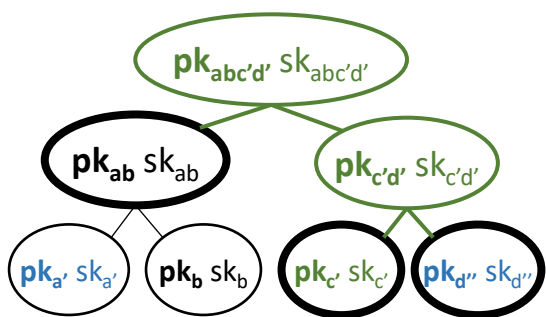


Realistic symbolic model:
No coordination + PCS
⇒ Ω(t)

PCS & Concurrency & Small overhead
PCS & Concurrency & Small overhead
PCS & Concurrency & Small overhead

Upper Bound:
Almost optimal construction

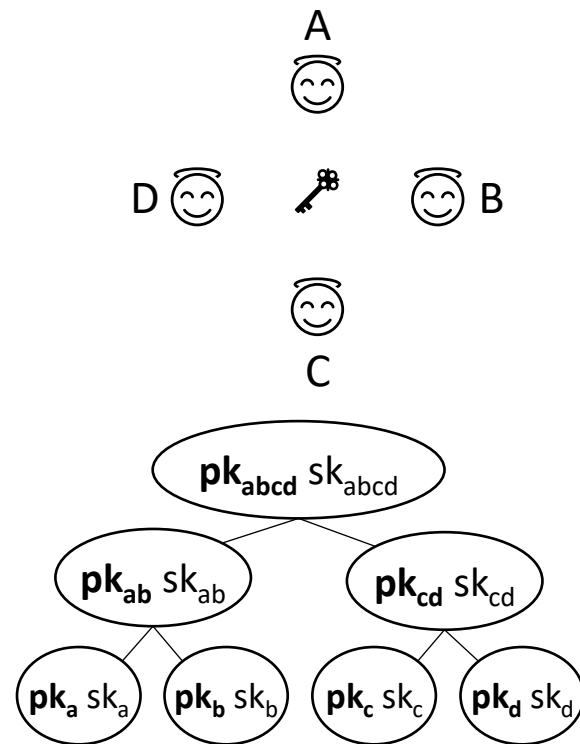
Open Questions ...



Ω(t) vs. O(t · (1 + log(n/t)))?
NIKE?
PCS-Delay?
Forward-secretly?
Application to MLS

Upper Bound: Almost optimal construction

Key tree (with updatable KEM)



 public
 secret
 root secret
 = group key

Upper Bound: Almost optimal construction

Key tree (with updatable KEM)

i-2 Exposure:

- Paths of c and d *public*: $sk_c, sk_d, sk_{cd}, sk_{abcd}$

i-1 Recovery 1:

- Generate and share new leaf key pairs: $(sk_{c'}, pk_{c'}), (sk_{d'}, pk_{d'})$

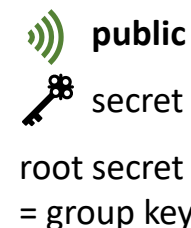
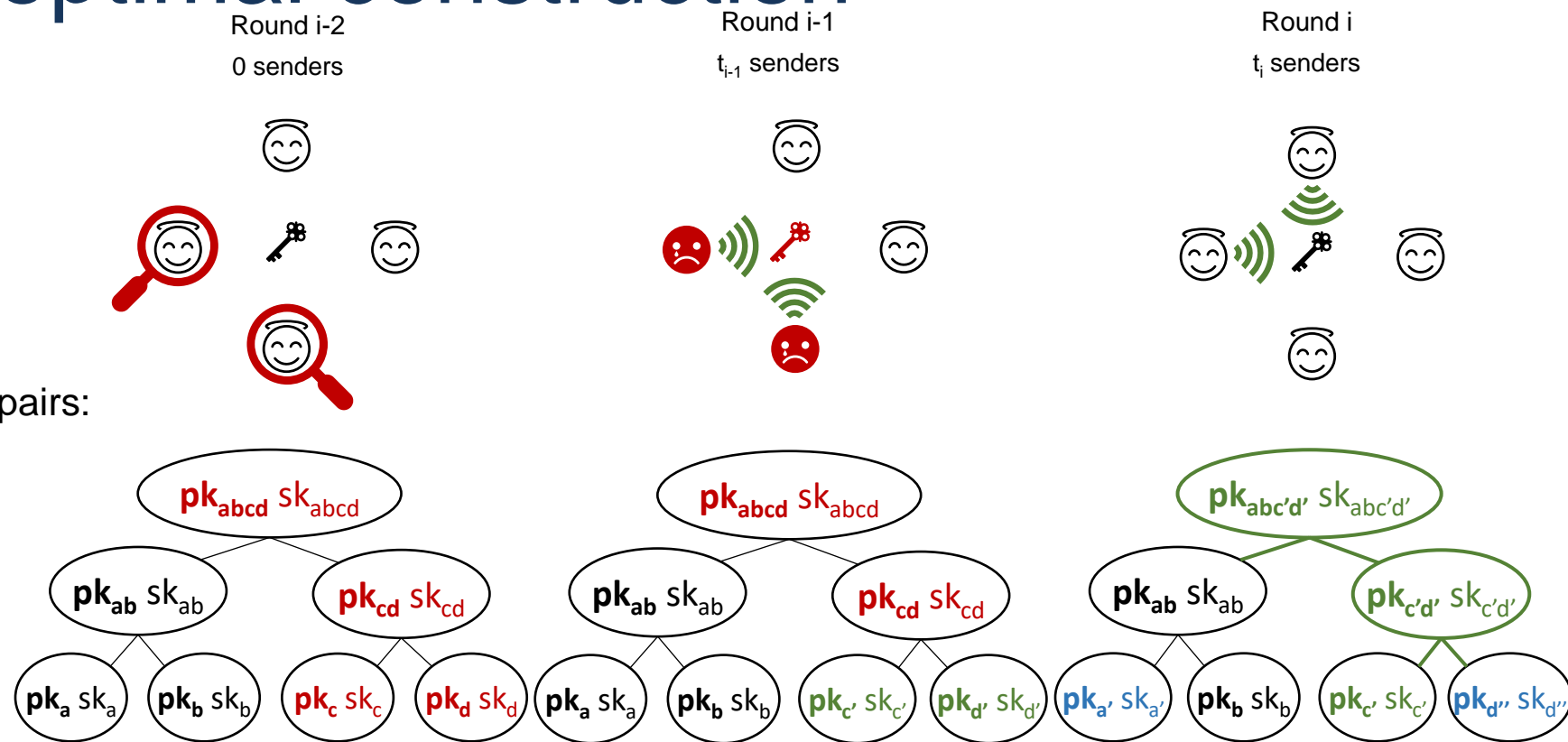
i Recovery 2:

a) See Recovery 1

- Each sender generates new paths for previous senders:

b) Sample $x_{c'd'}$

- c) Derive $sk_{c'd'} = x_{c'd'}$, $x_{abc'd'} = H(x_{c'd'})$, $sk_{abc'd'} = x_{abc'd'}$, $pk_{c'd'} = \text{gen}(sk_{c'd'})$, $pk_{abc'd'} = \text{gen}(sk_{abc'd'})$



Upper Bound: Almost optimal construction

Key tree (with updatable KEM)

i-2 Exposure:

- Paths of c and d *public*: $sk_c, sk_d, sk_{cd}, sk_{abcd}$

i-1 Recovery 1:

- Generate and share new leaf key pairs: $(sk_{c'}, pk_{c'}), (sk_{d'}, pk_{d'})$

i Recovery 2:

a) See Recovery 1

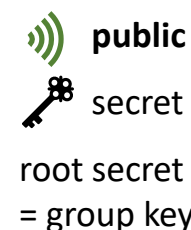
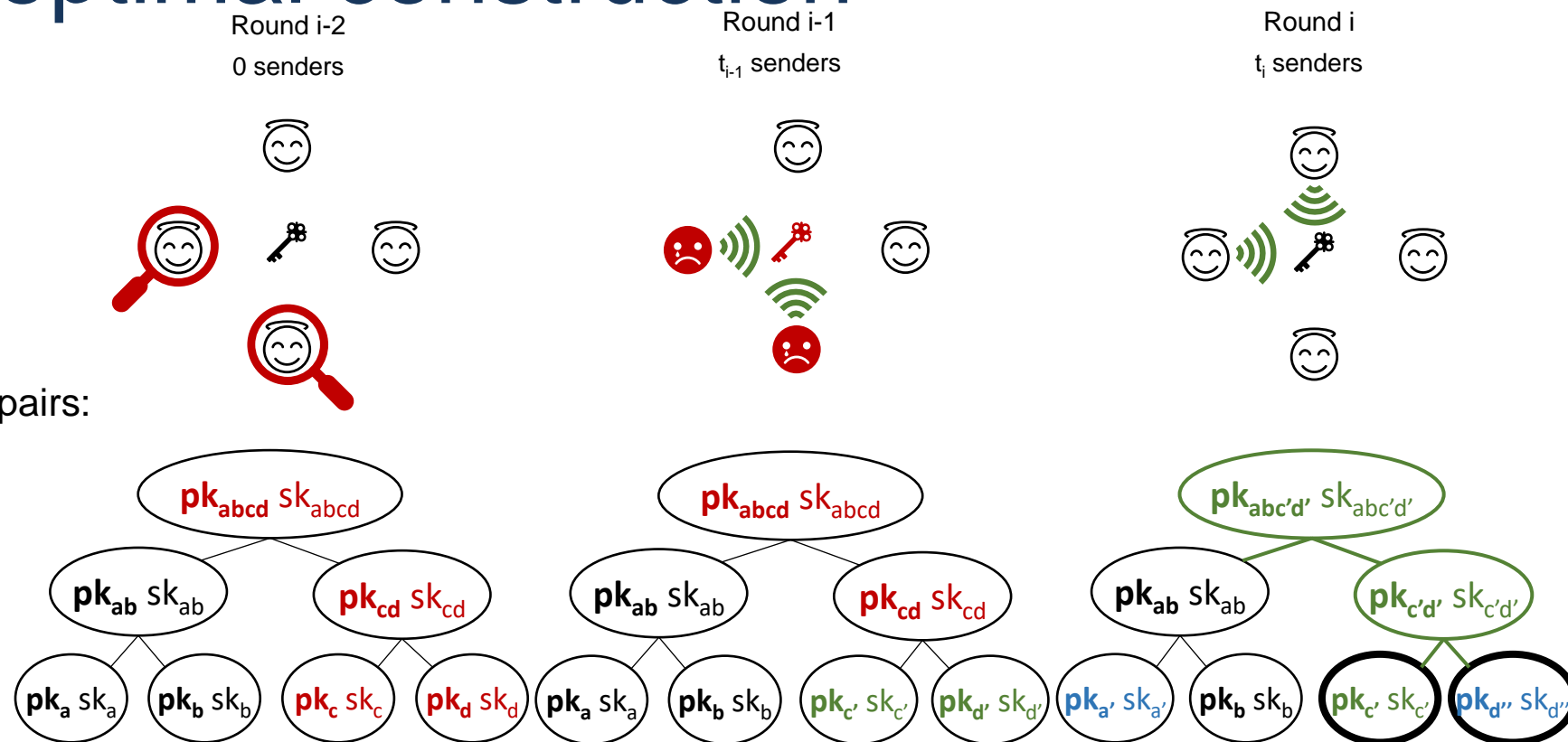
- Each sender generates new paths for previous senders:

b) Sample $x_{c'd'}$

c) Derive $sk_{c'd'} = x_{c'd'}$, $x_{abc'd'} = H(x_{c'd'})$, $sk_{abc'd'} = x_{abc'd'}$, $pk_{c'd'} = \text{gen}(sk_{c'd'})$, $pk_{abc'd'} = \text{gen}(sk_{abc'd'})$

d) Send $\text{enc}(pk_{c'}, x_{c'd'})$, $\text{enc}(pk_{d'}, x_{c'd'})$

→ **Number of leaves: t_{i-1}**



Upper Bound: Almost optimal construction

Key tree (with updatable KEM)

i-2 Exposure:

- Paths of c and d *public*: $sk_c, sk_d, sk_{cd}, sk_{abcd}$

i-1 Recovery 1:

- Generate and share new leaf key pairs: $(sk_{c'}, pk_{c'}), (sk_{d'}, pk_{d'})$

i Recovery 2:

a) See Recovery 1

- Each sender generates new paths for previous senders:

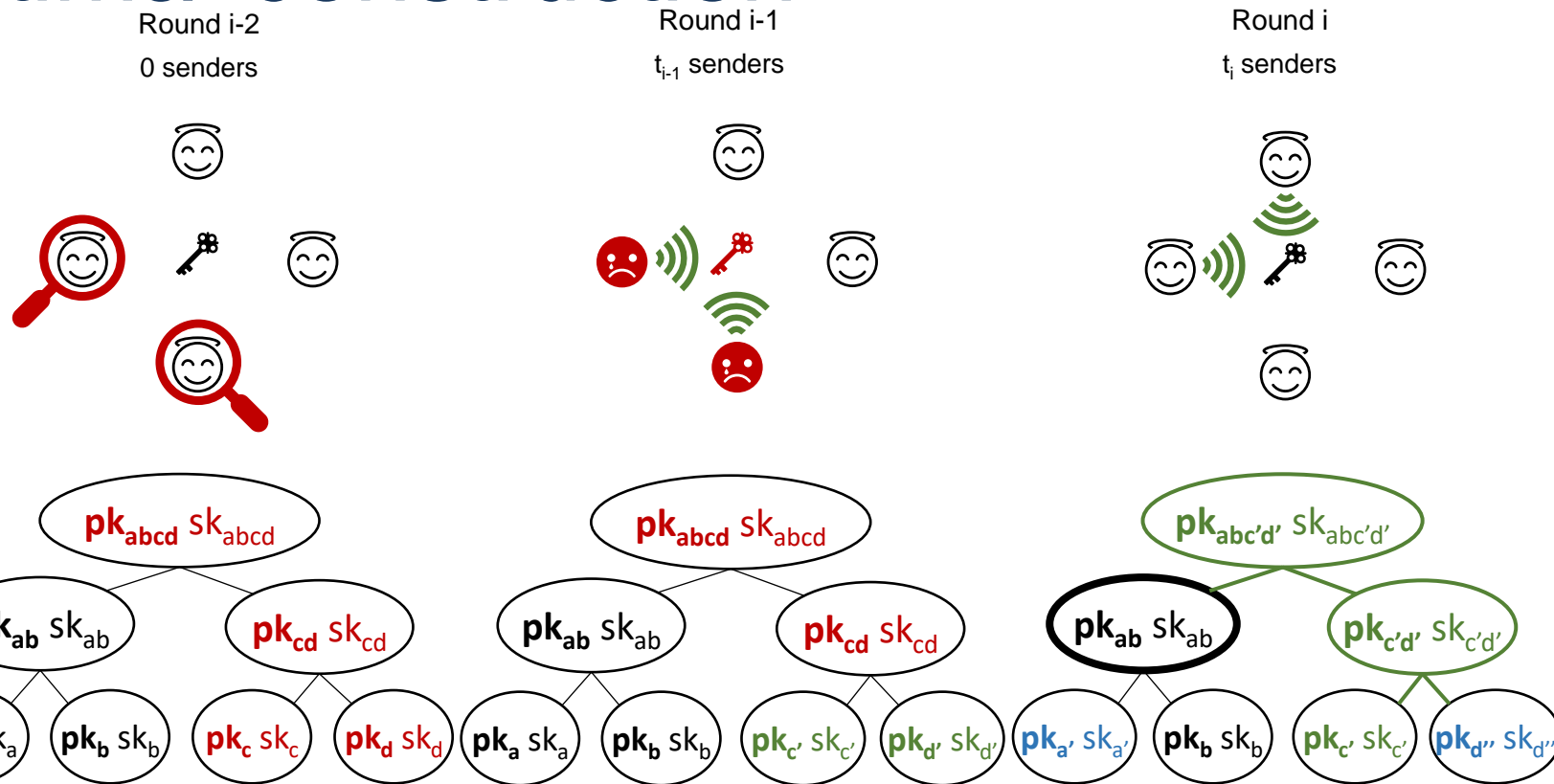
b) Sample $x_{c'd'}$

c) Derive $sk_{c'd'} = x_{c'd'}$, $x_{abc'd'} = H(x_{c'd'})$, $sk_{abc'd'} = x_{abc'd'}$, $pk_{c'd'} = \text{gen}(sk_{c'd'})$, $pk_{abc'd'} = \text{gen}(sk_{abc'd'})$

d) Send $\text{enc}(pk_{c'}, x_{c'd'})$, $\text{enc}(pk_{d'}, x_{c'd'})$, $pk_{c'd'}$, $\text{enc}(pk_{ab}, x_{abc'd'})$

→ Number of leafs: t_{i-1} , **number of update-tree-siblings: $O(t_{i-1} \cdot \log(n/t_{i-1}))$**

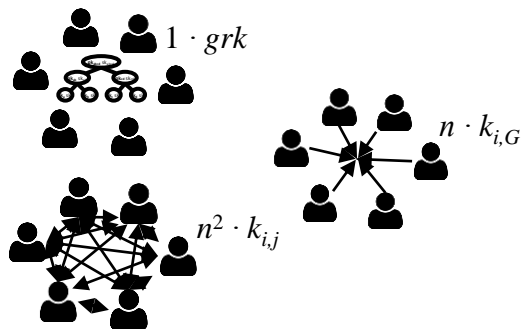
⇒ **Overhead per recovery under t-concurrency: $O(t + t \cdot \log(n/t))$**



1-"concurrency": $\rightarrow O(\log n)$
n-concurrency: $\rightarrow O(n)$

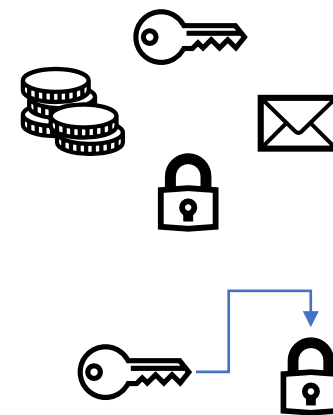
public
 secret
root secret = group key

Agenda



Previous Work:
What's the Problem?

Lower Bound:
What's the minimal overhead?



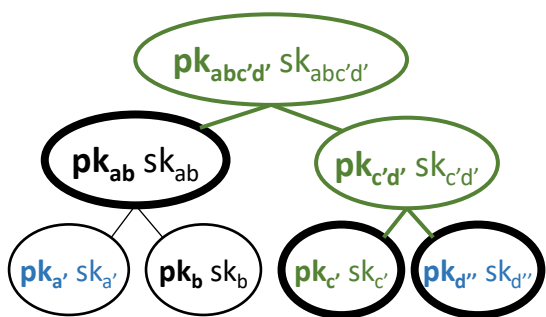
Realistic symbolic model:
No coordination + PCS
 $\Rightarrow \Omega(t)$

PCS & Concurrency & Small overhead
PCS & Concurrency & Small overhead
PCS & Concurrency & Small overhead

Upper Bound:
Almost optimal construction

Open Questions ...

$\Omega(t)$ vs. $O(t \cdot (1 + \log(n/t)))$?
NIKE?
PCS-Delay?
Forward-secrecy?
Application to MLS



Two-step recovery
 $\Rightarrow O(t \cdot (1 + \log(n/t)))$

@roeslpa Full details & formal proofs online *soonish*