On the Worst-Case Inefficiency of CGKA

Alexander Bienstock, Yevgeniy Dodis, Sanjam Garg, Garrison Grogan, Mohammad Hajiabadi, Paul Rösler

Motivation: Secure Group Messaging (SGM)

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications
- IETF “Messaging Layer Security” standardization in progress

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications
- IETF “Messaging Layer Security” standardization in progress
- Basic Func: Users send messages and add/remove others to a group

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications
- IETF “Messaging Layer Security” standardization in progress
- Basic Func: Users send messages and add/remove others to a group
- Users can go offline sometimes – delivery server facilitates interaction

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications
- IETF “Messaging Layer Security” standardization in progress
- Basic Func: Users send messages and add/remove others to a group
- Users can go offline sometimes – delivery server facilitates interaction
- Standard end-to-end security w.r.t. non-members (and more)
  - Including those just removed before or added immediately after sent msg

Motivation: Secure Group Messaging (SGM)

- Used by billions of users worldwide across many applications
- IETF “Messaging Layer Security” standardization in progress
- Basic Func: Users send messages and add/remove others to a group
- Users can go offline sometimes – delivery server facilitates interaction
- Standard end-to-end security w.r.t. non-members (and more)
  - Including those just removed before or added immediately after sent msg
- Continuous Group Key Agreement (CGKA) at the core of SGM
  - Creates shared secret for each group operation (used to encrypt msgs, etc)

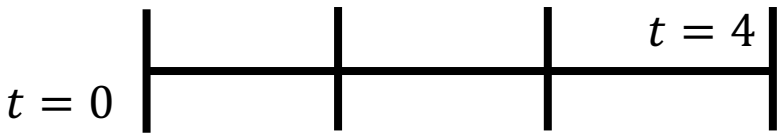
Security with State Leakages

Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time

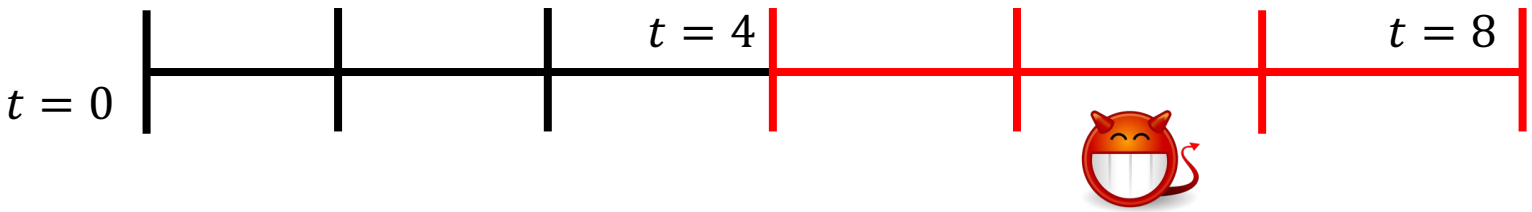
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time



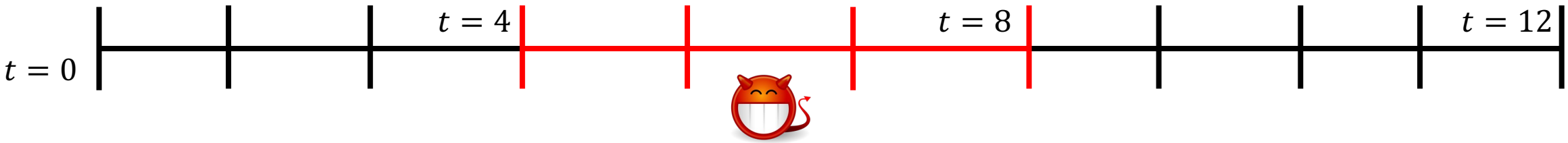
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time



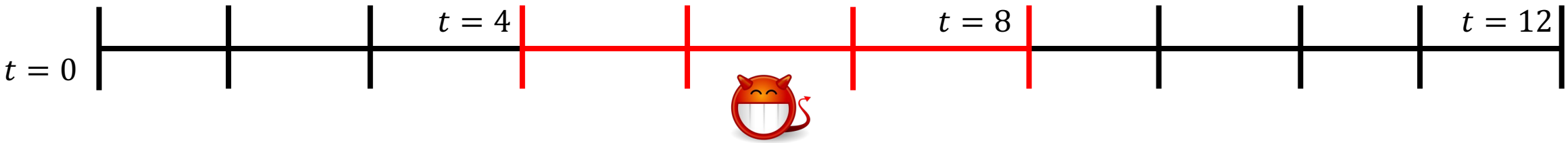
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time



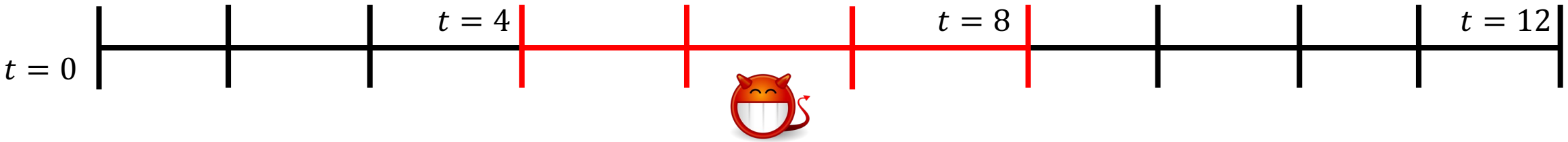
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time
- Users periodically issue state updates to “heal” from such leakages
  - Each such operation also creates a shared secret



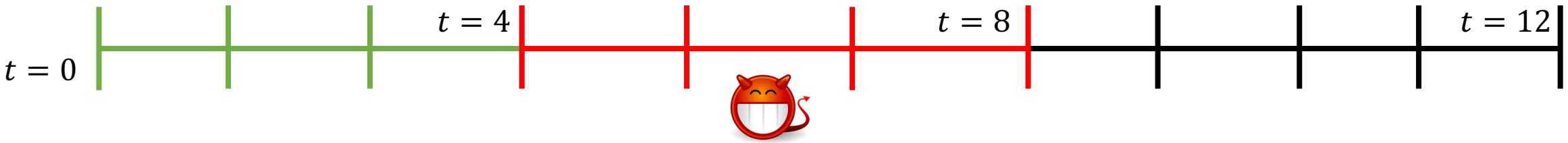
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time
- Users periodically issue state updates to “heal” from such leakages
  - Each such operation also creates a shared secret
- Forward Secrecy: state leakage of a user at some point in time does not reveal past group secrets



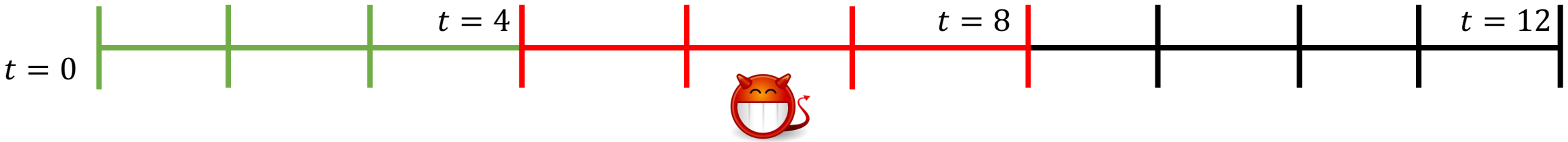
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time
- Users periodically issue state updates to “heal” from such leakages
  - Each such operation also creates a shared secret
- Forward Secrecy: state leakage of a user at some point in time does not reveal past group secrets



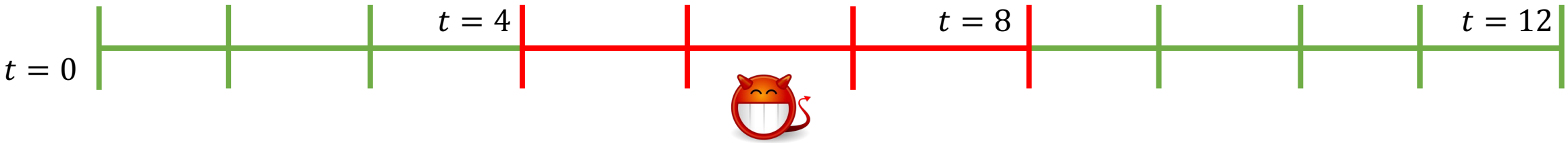
Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time
- Users periodically issue state updates to “heal” from such leakages
  - Each such operation also creates a shared secret
- Forward Secrecy: state leakage of a user at some point in time does not reveal past group secrets
- Post-Compromise Security: state update following leakage of a user at some point in time hides future group secrets

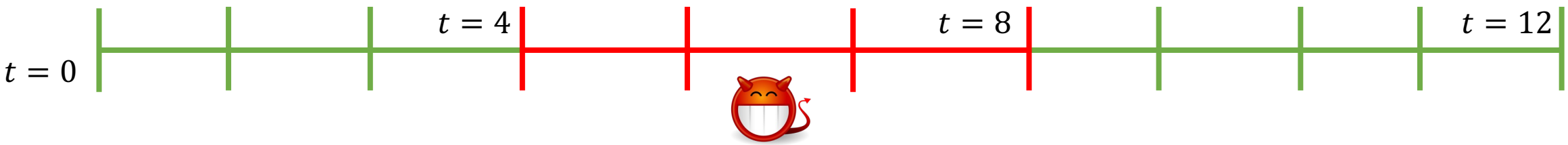


Security with State Leakages

- CGKA security models state leakages of users during time windows
  - Including sampled randomness during this time
- Users periodically issue state updates to “heal” from such leakages
  - Each such operation also creates a shared secret
- Forward Secrecy: state leakage of a user at some point in time does not reveal past group secrets
- Post-Compromise Security: state update following leakage of a user at some point in time hides future group secrets

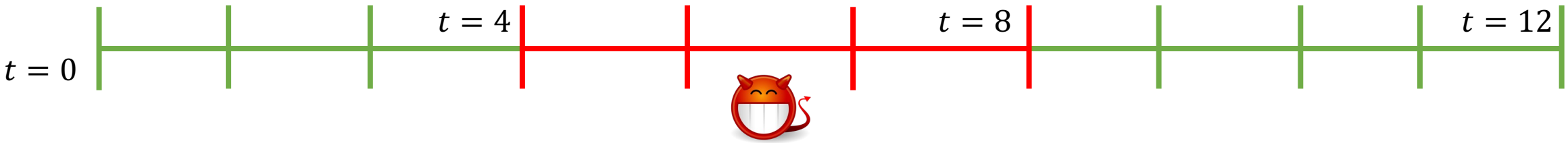


Important PCS Properties



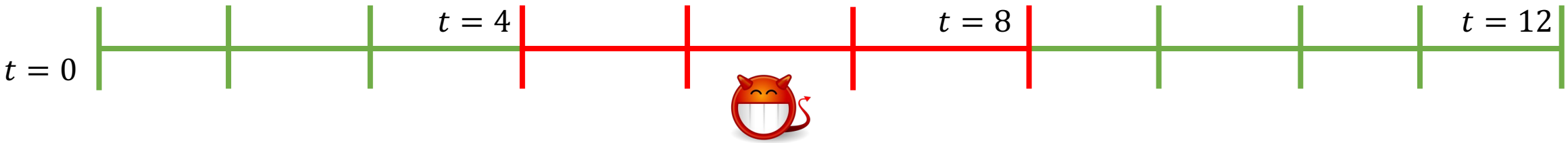
Important PCS Properties

- *Double-join prevention*: A malicious user may defy the protocol by not deleting old randomness and is removed at some point.



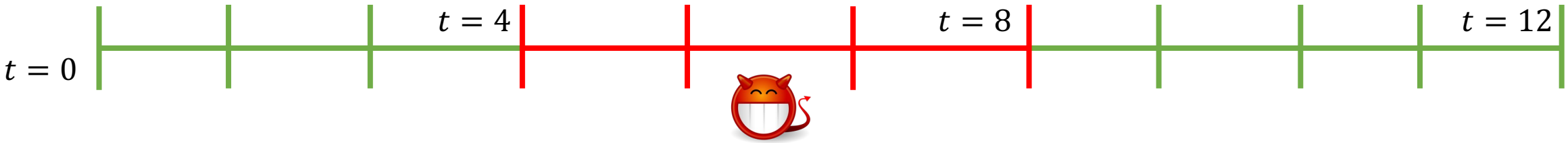
Important PCS Properties

- *Double-join prevention*: A malicious user may defy the protocol by not deleting old randomness and is removed at some point.
  - Saved randomness must not break security of future keys



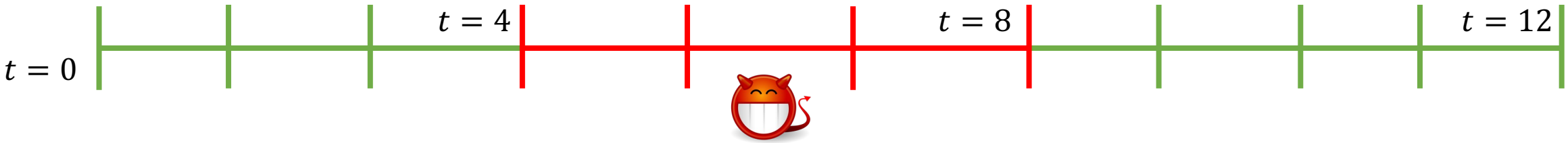
Important PCS Properties

- *Double-join prevention*: A malicious user may defy the protocol by not deleting old randomness and is removed at some point.
  - Saved randomness must not break security of future keys
- *Resilience to randomness leakage*: An honest user's device may leak sampled randomness (e.g., due to implementation flaws or a virus). If the leakage stops and the user updates:



Important PCS Properties

- *Double-join prevention*: A malicious user may defy the protocol by not deleting old randomness and is removed at some point.
  - Saved randomness must not break security of future keys
- *Resilience to randomness leakage*: An honest user's device may leak sampled randomness (e.g., due to implementation flaws or a virus). If the leakage stops and the user updates:
  - Leaked randomness must not break security of future keys



CGKA Efficiency

- $\mathrm{O}(\log n)$, or at least sub-linear, efficiency for operations
  - n = number of group members

CGKA Efficiency

- $\mathrm{O}(\log n)$, or at least sub-linear, efficiency for operations
  - n = number of group members
- Some protocols indeed claim "fair-weather" $O(\log n)$

CGKA Efficiency

- $\mathrm{O}(\log n)$, or at least sub-linear, efficiency for operations
  - n = number of group members
- Some protocols indeed claim "fair-weather" $O(\log n)$
- But all *known* schemes have $\Omega(n)$ worst-case efficiency
  - Even for communication!
  - Even amortized over long periods of time!
- We show: All CGKA schemes black-box from PKE must suffer from this worst-case inefficiency

High-Level Summary of Existing Schemes

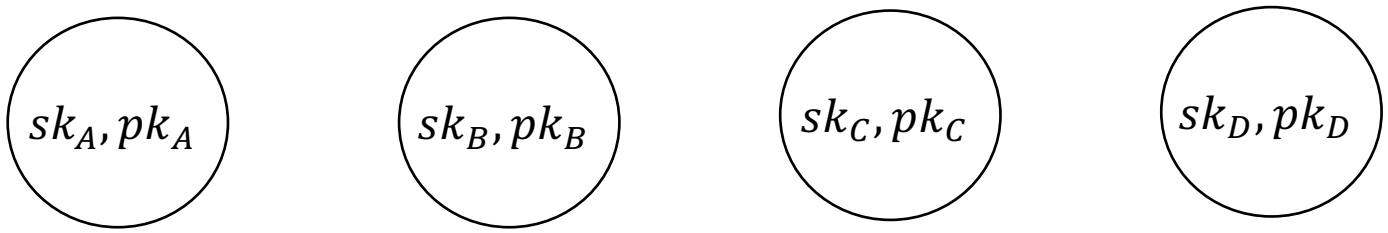
- TreeKEM Family

High-Level Summary of Existing Schemes

- TreeKEM Family
- User key pairs at leaves

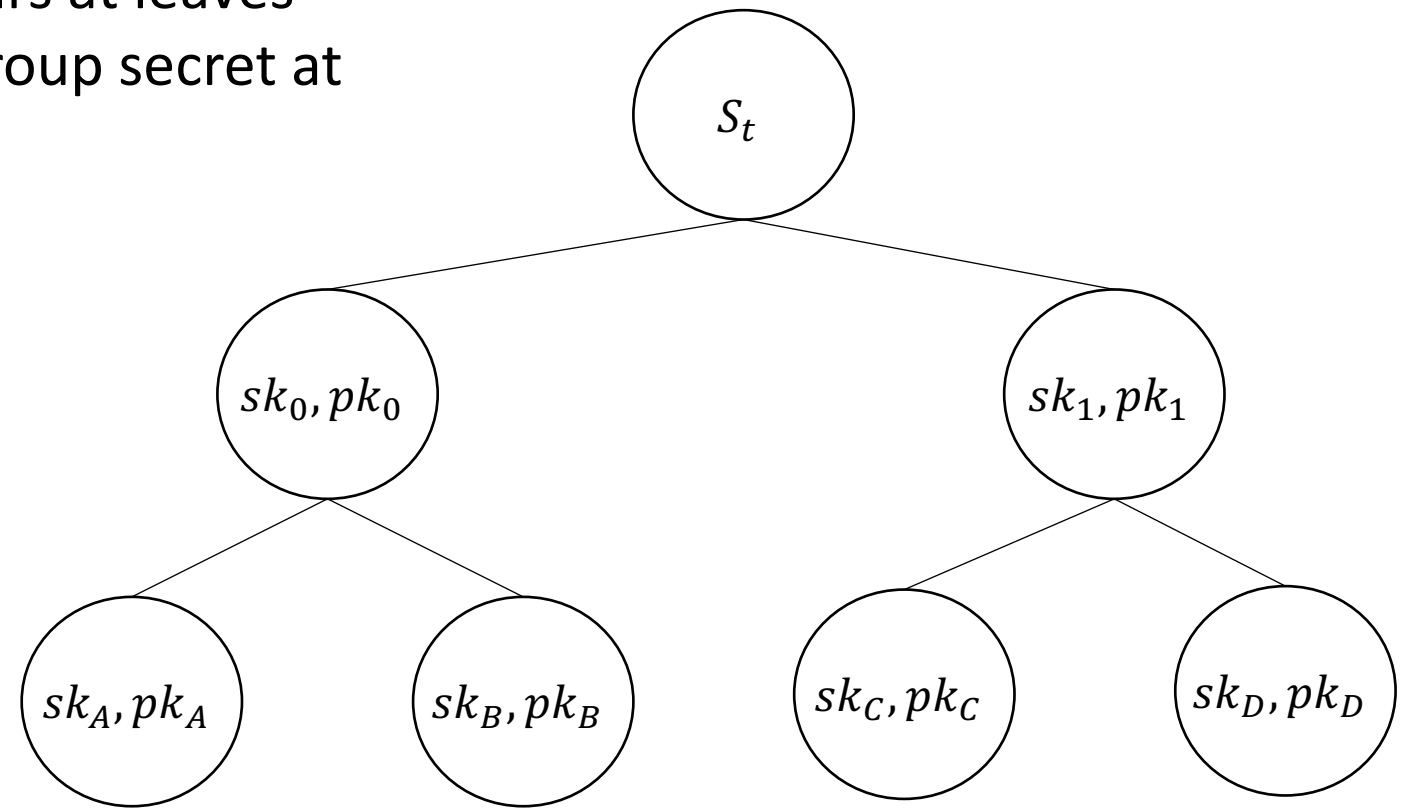
High-Level Summary of Existing Schemes

- TreeKEM Family
- User key pairs at leaves



High-Level Summary of Existing Schemes

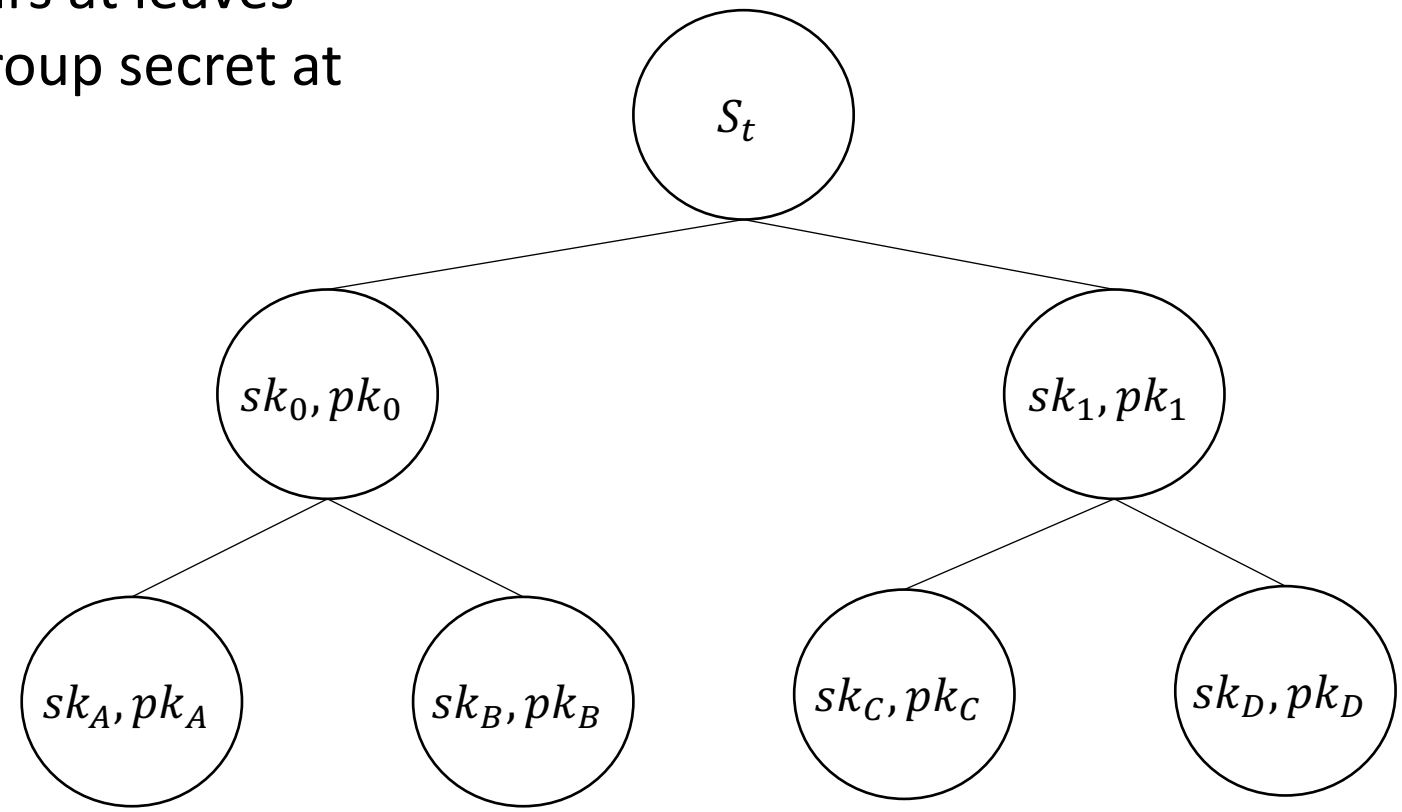
- TreeKEM Family
- User key pairs at leaves
- Root S_t is group secret at time t



High-Level Summary of Existing Schemes

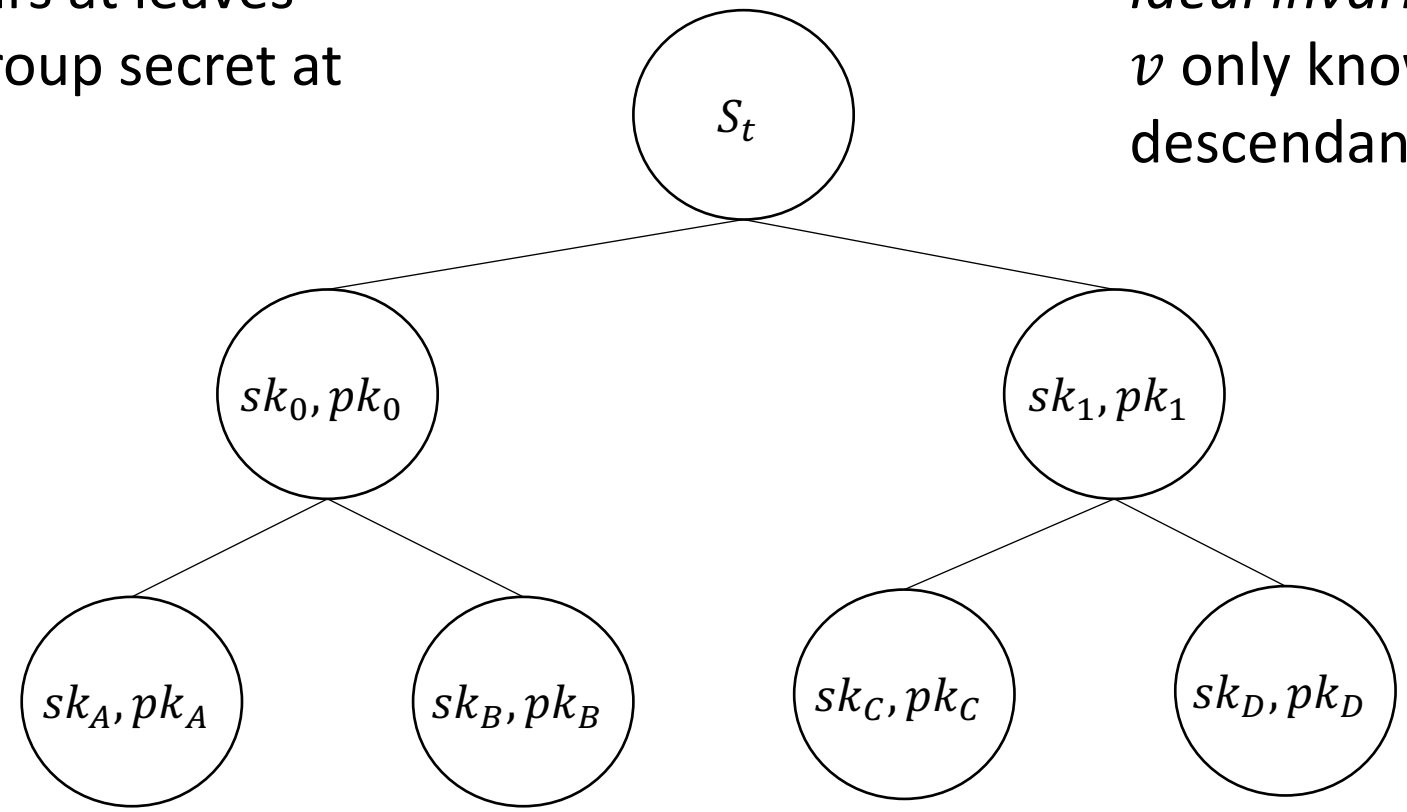
- TreeKEM Family
- User key pairs at leaves
- Root S_t is group secret at time t

- pk's known to all



High-Level Summary of Existing Schemes

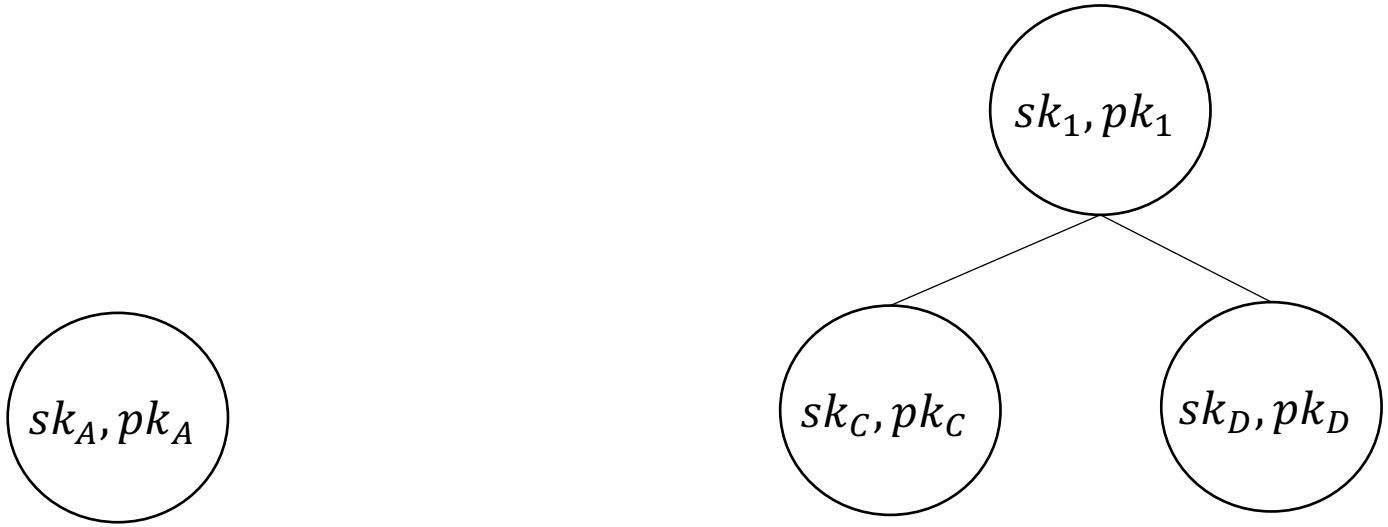
- TreeKEM Family
- User key pairs at leaves
- Root S_t is group secret at time t



- pk's known to all
- *Ideal invariant*: sk_v at node v only known to descendants

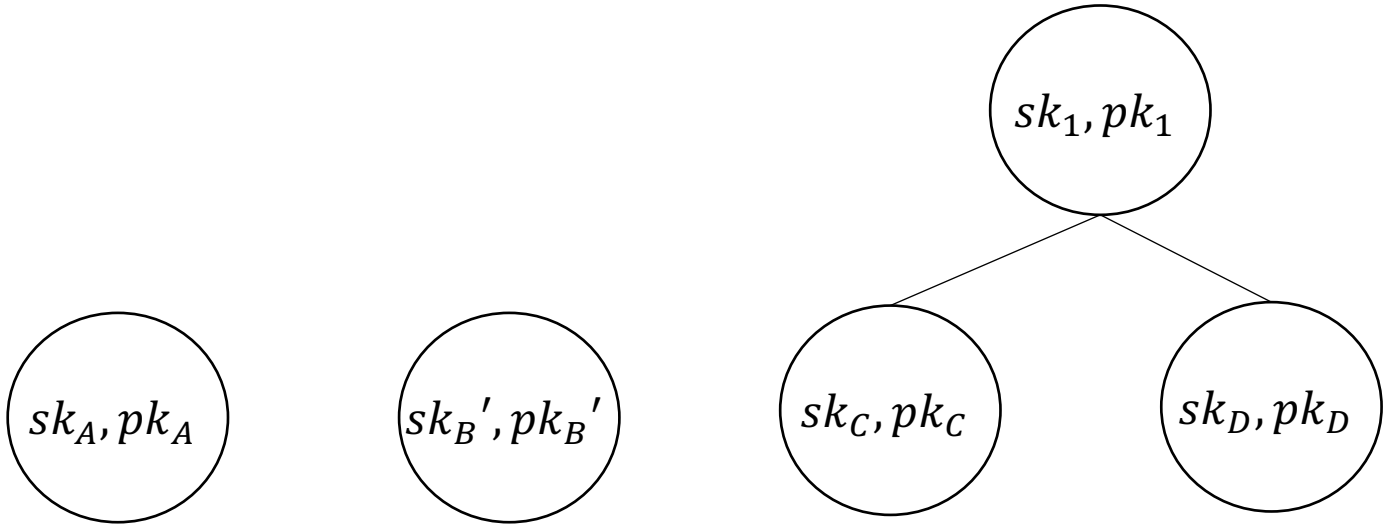
High-Level Summary of Existing Schemes

- Example op: B updates



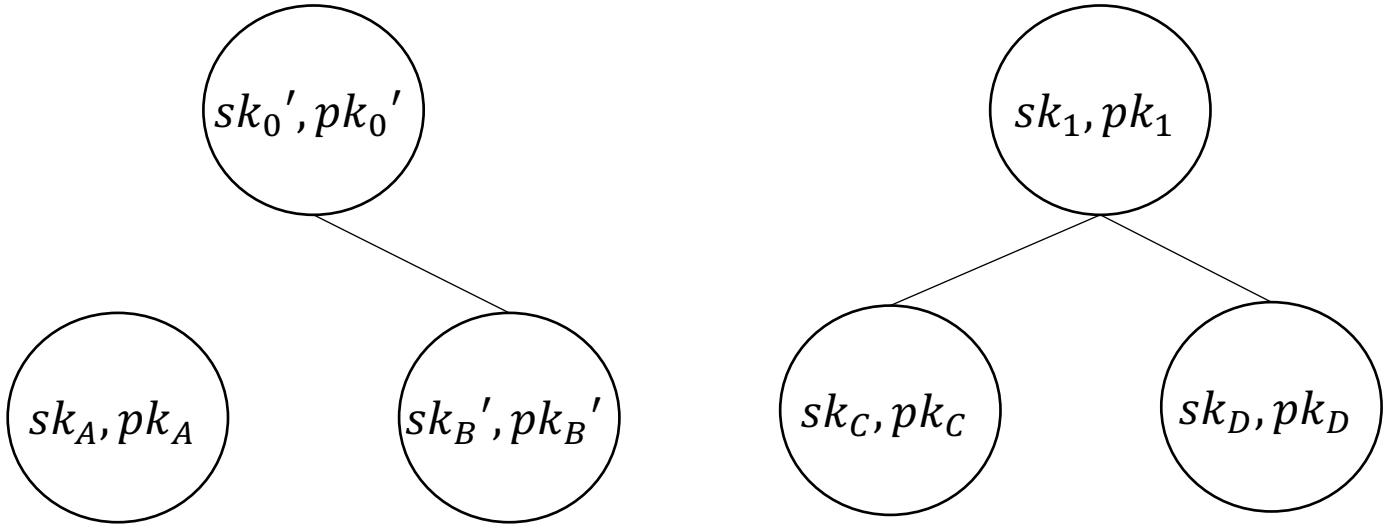
High-Level Summary of Existing Schemes

- Example op: B updates



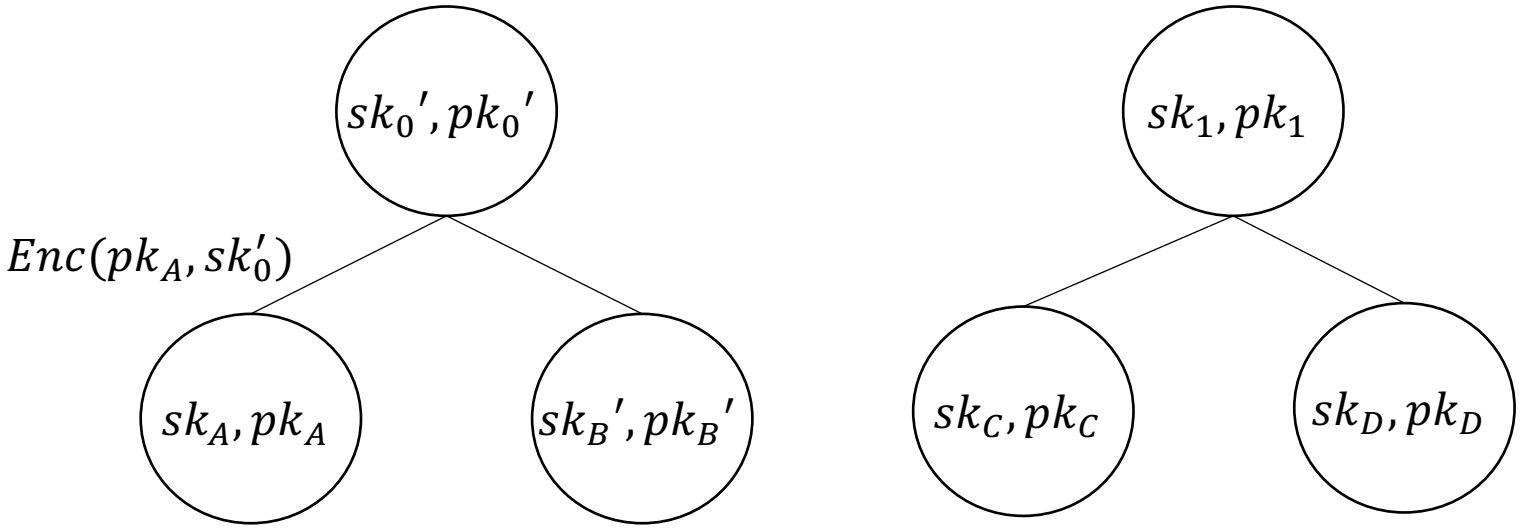
High-Level Summary of Existing Schemes

- Example op: B updates



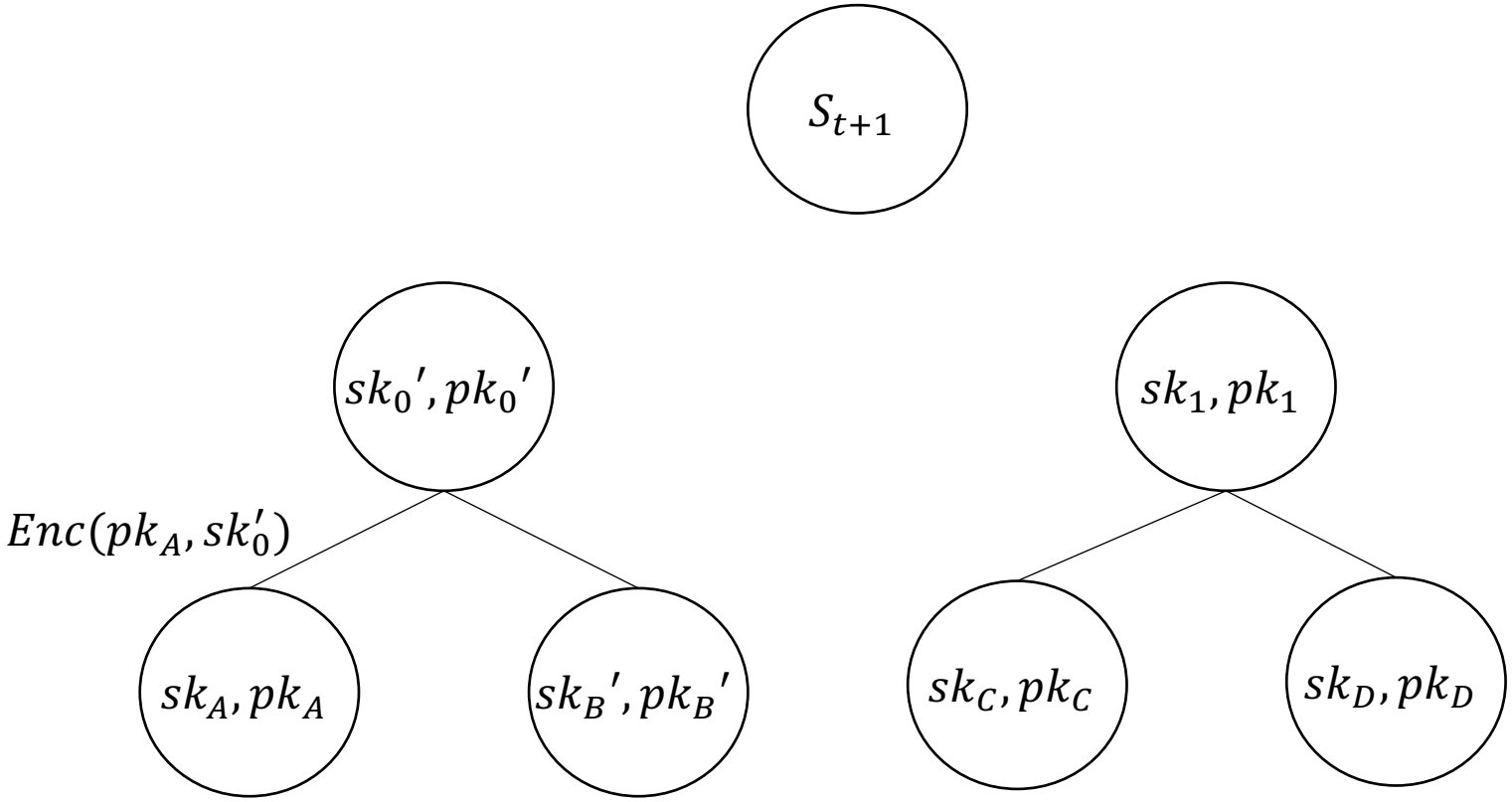
High-Level Summary of Existing Schemes

- Example op: B updates



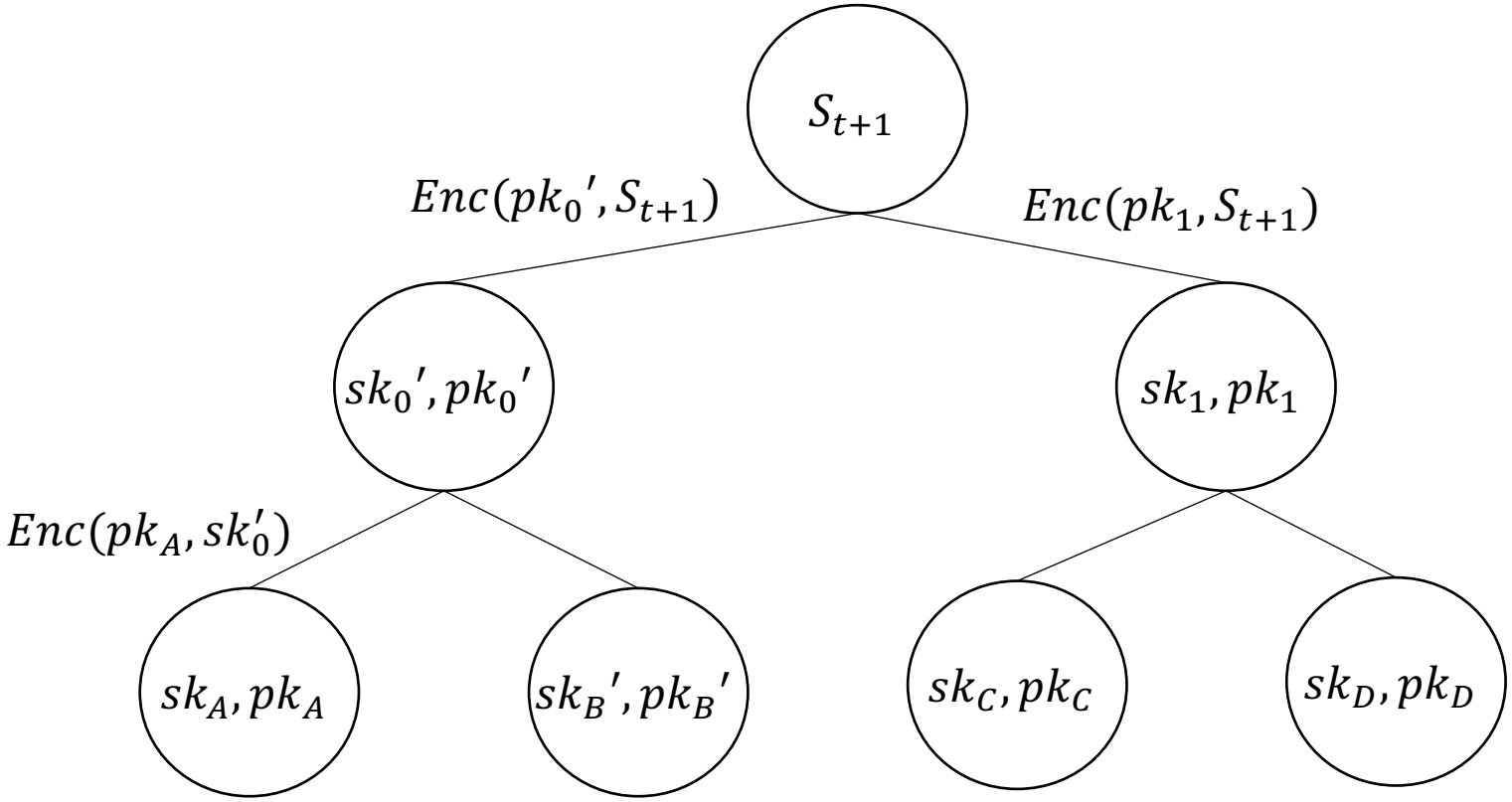
High-Level Summary of Existing Schemes

- Example op: B updates



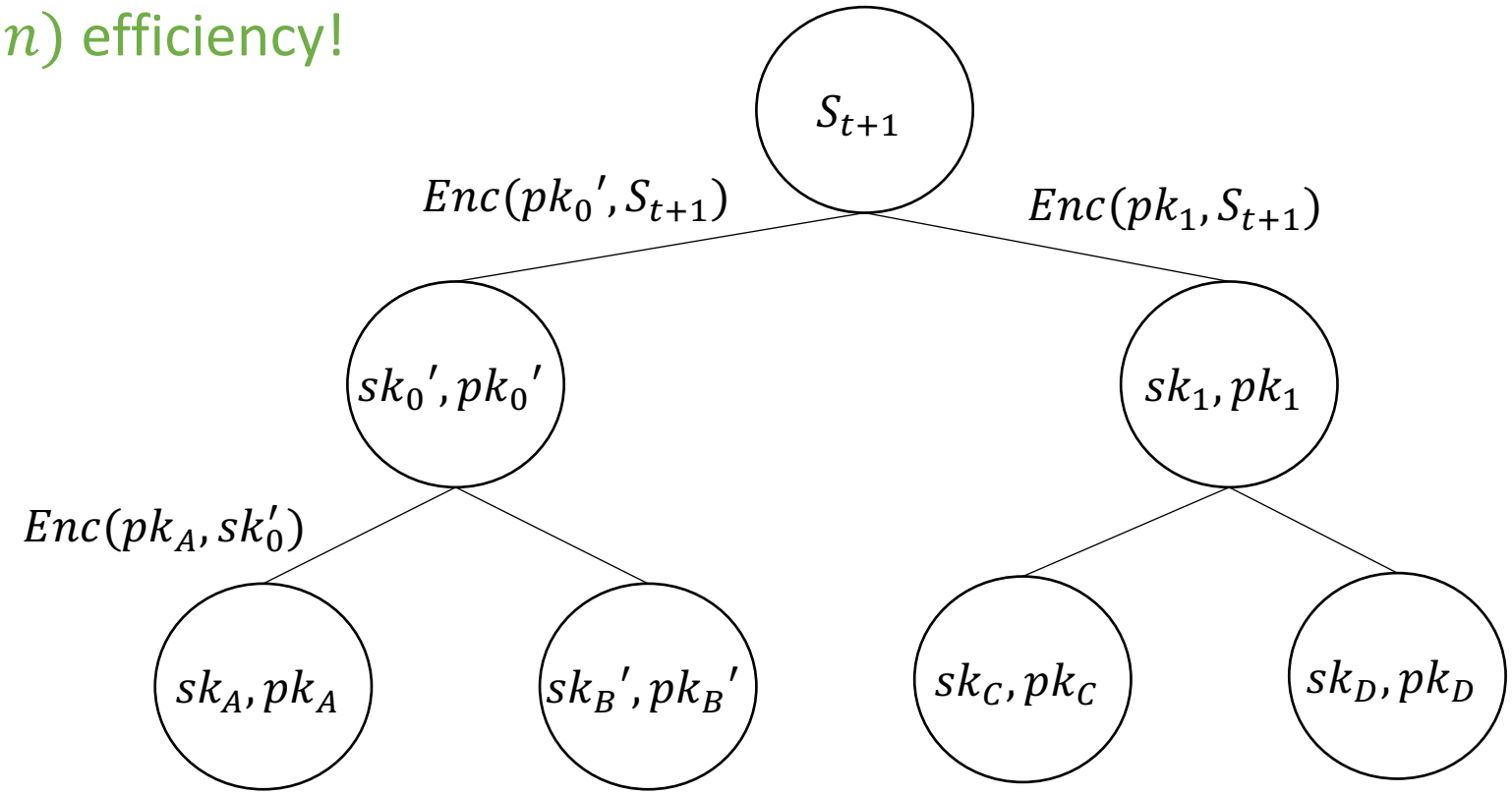
High-Level Summary of Existing Schemes

- Example op: B updates



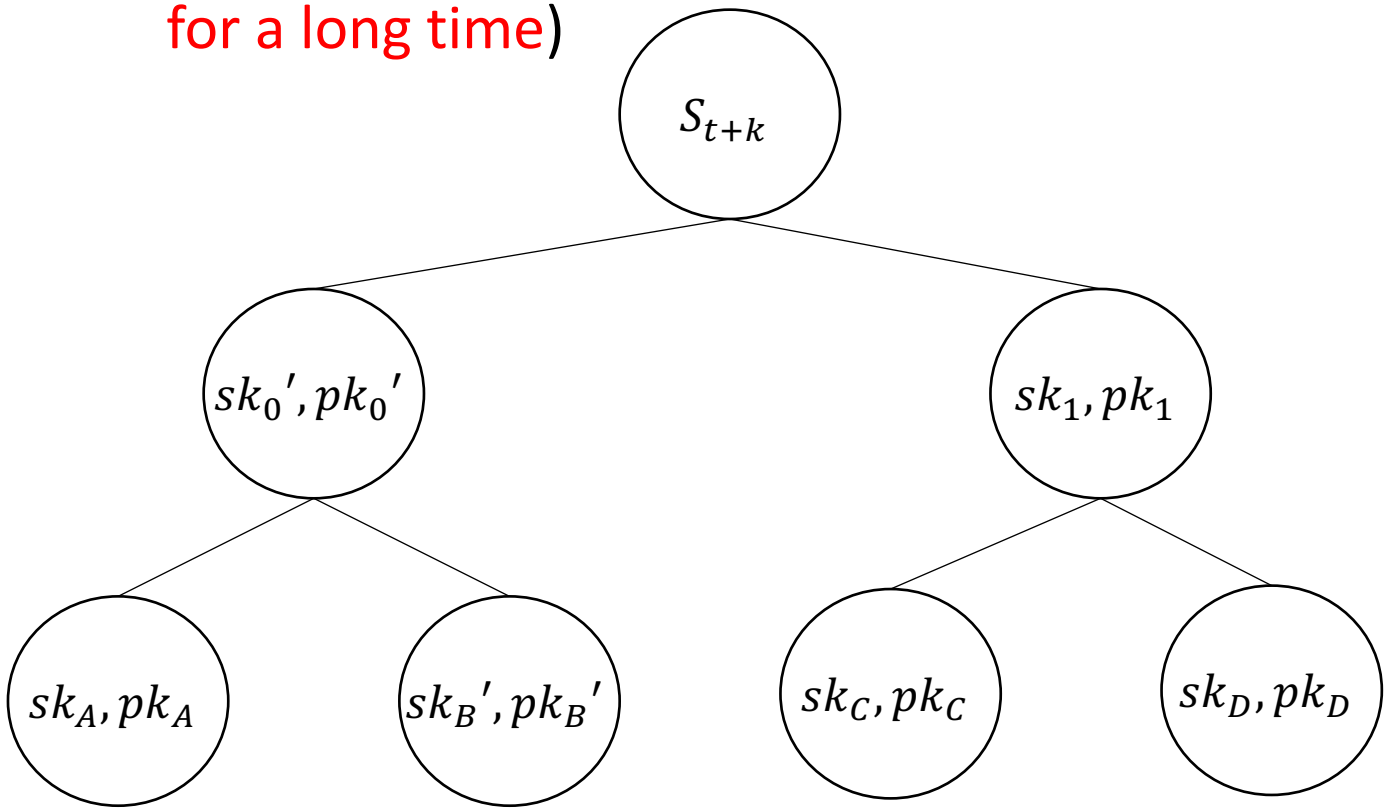
High-Level Summary of Existing Schemes

- Example op: B updates
- $O(\log n)$ efficiency!



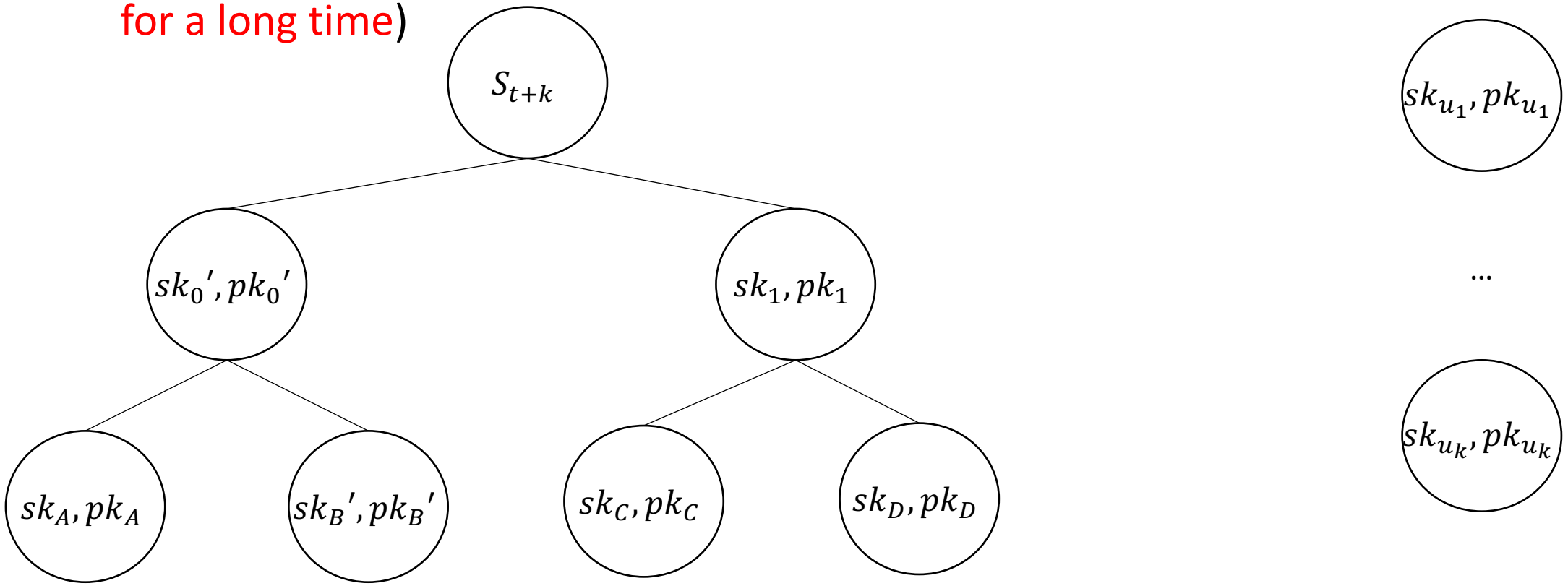
Worst-Case Inefficiency of Existing Schemes

- Consider: D adds k $(\Omega(n))$ users (who all remain offline for a long time)



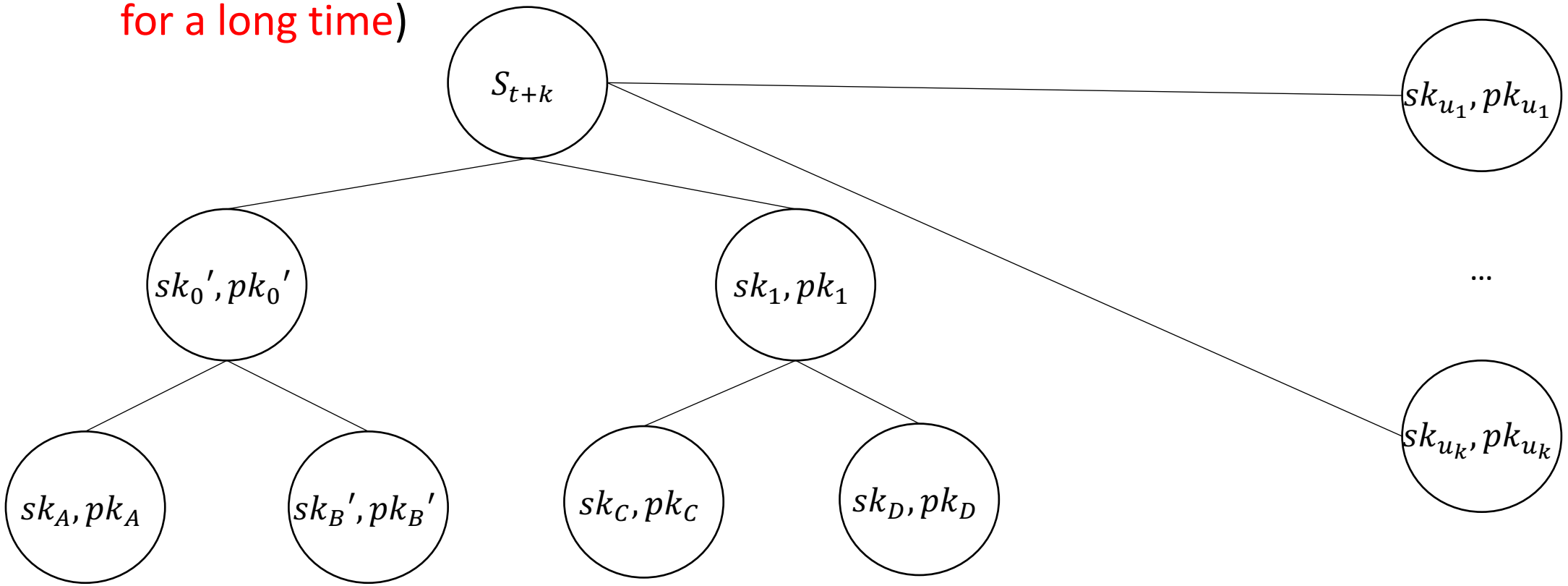
Worst-Case Inefficiency of Existing Schemes

- Consider: D adds k $(\Omega(n))$ users (who all remain offline for a long time)



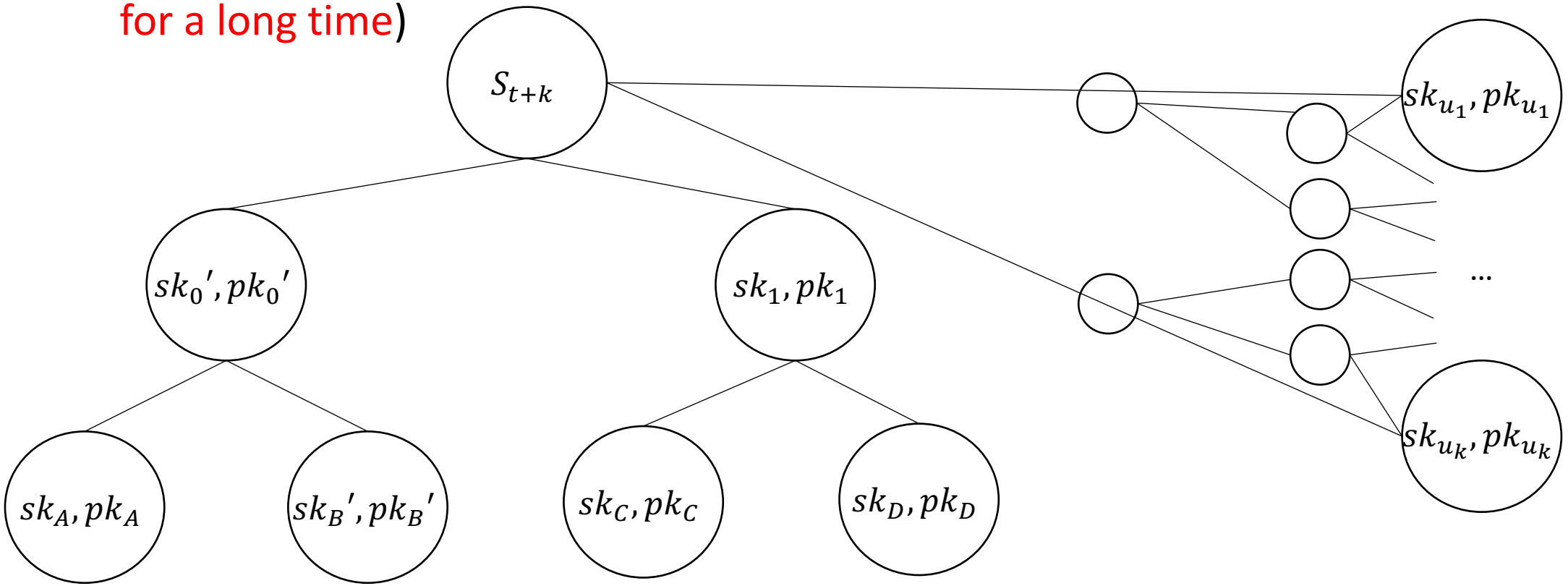
Worst-Case Inefficiency of Existing Schemes

- Consider: D adds k $(\Omega(n))$ users (who all remain offline for a long time)



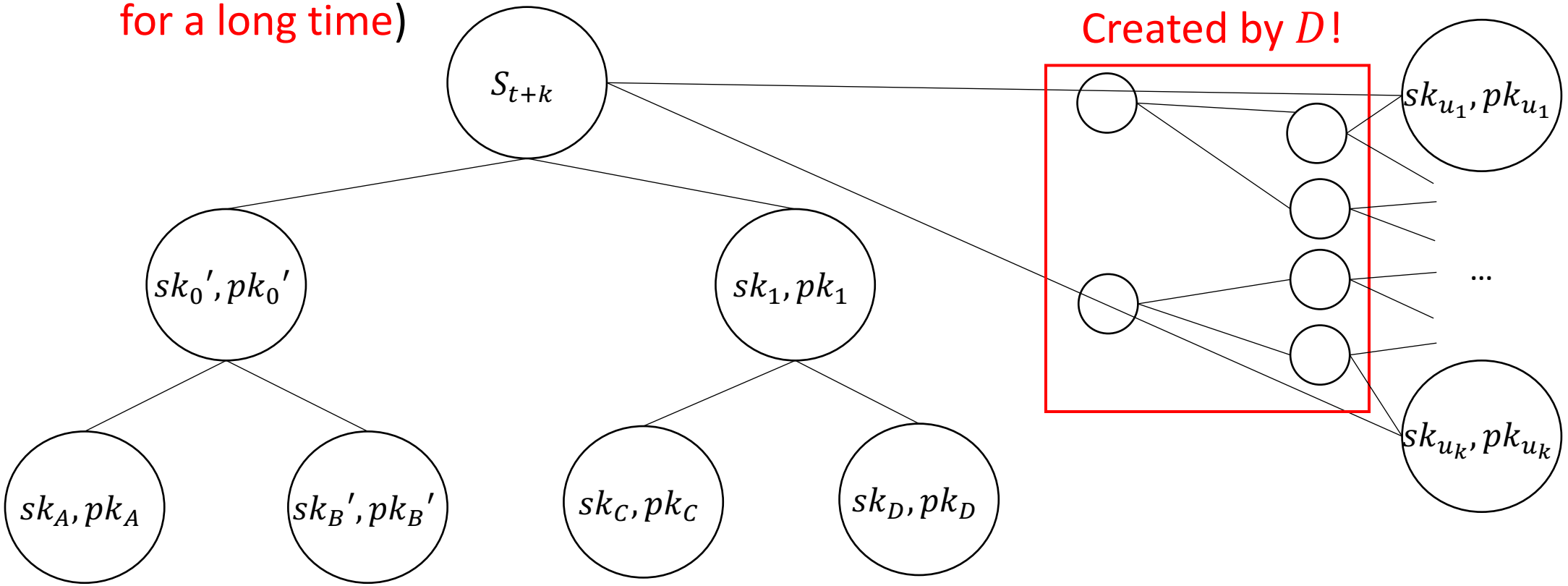
Worst-Case Inefficiency of Existing Schemes

- Consider: D adds k $(\Omega(n))$ users (who all remain offline for a long time)



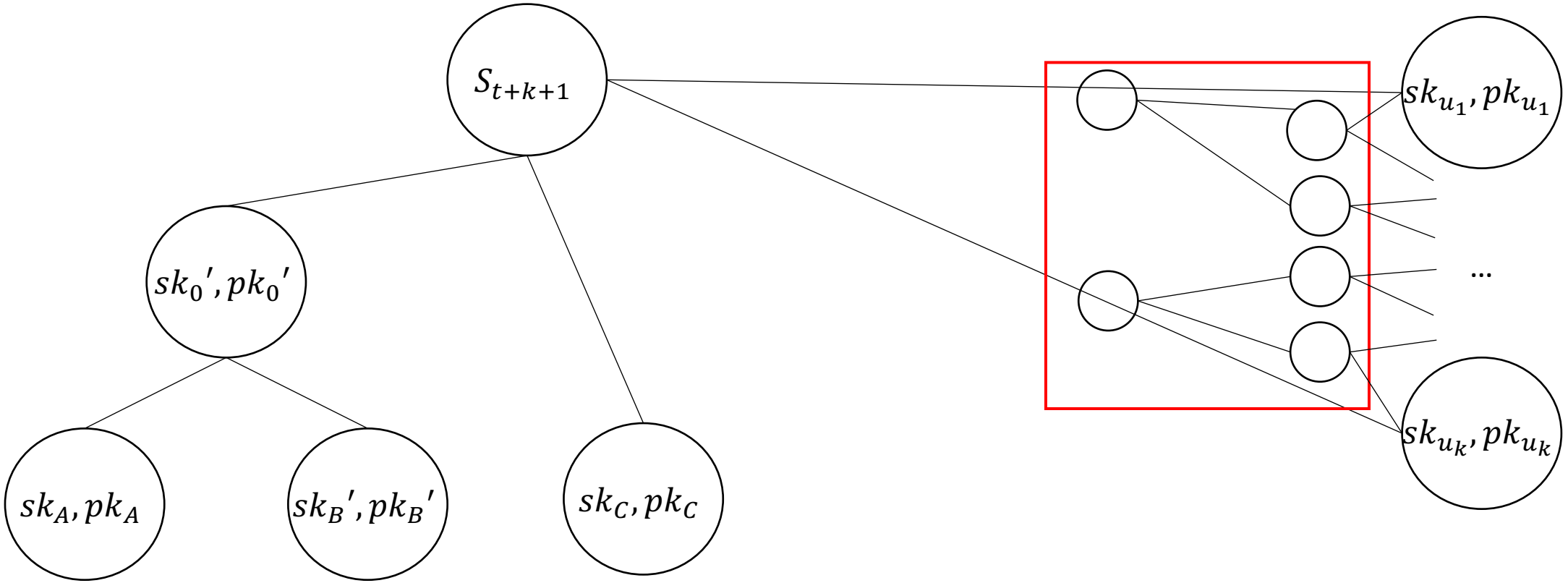
Worst-Case Inefficiency of Existing Schemes

- Consider: D adds k $(\Omega(n))$ users (who all remain offline for a long time)



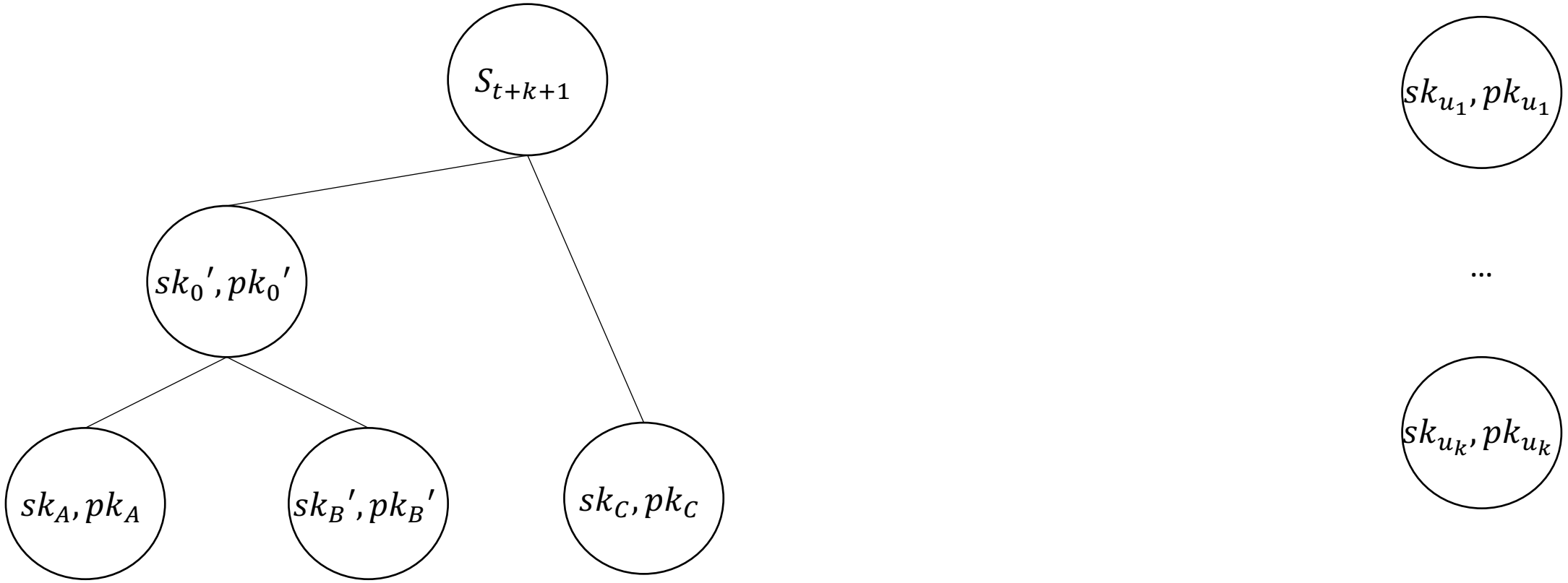
Worst-Case Inefficiency of Existing Schemes

- Consider: Now A removes D



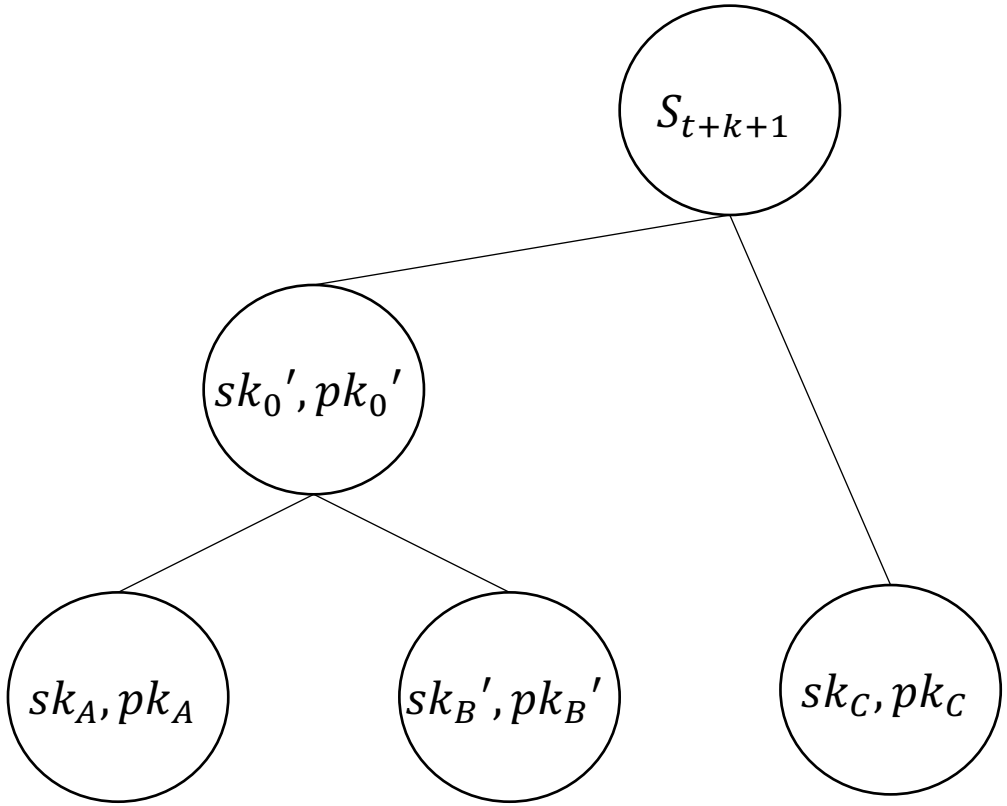
Worst-Case Inefficiency of Existing Schemes

- Consider: Now A removes D



Worst-Case Inefficiency of Existing Schemes

- Consider: Now A removes D

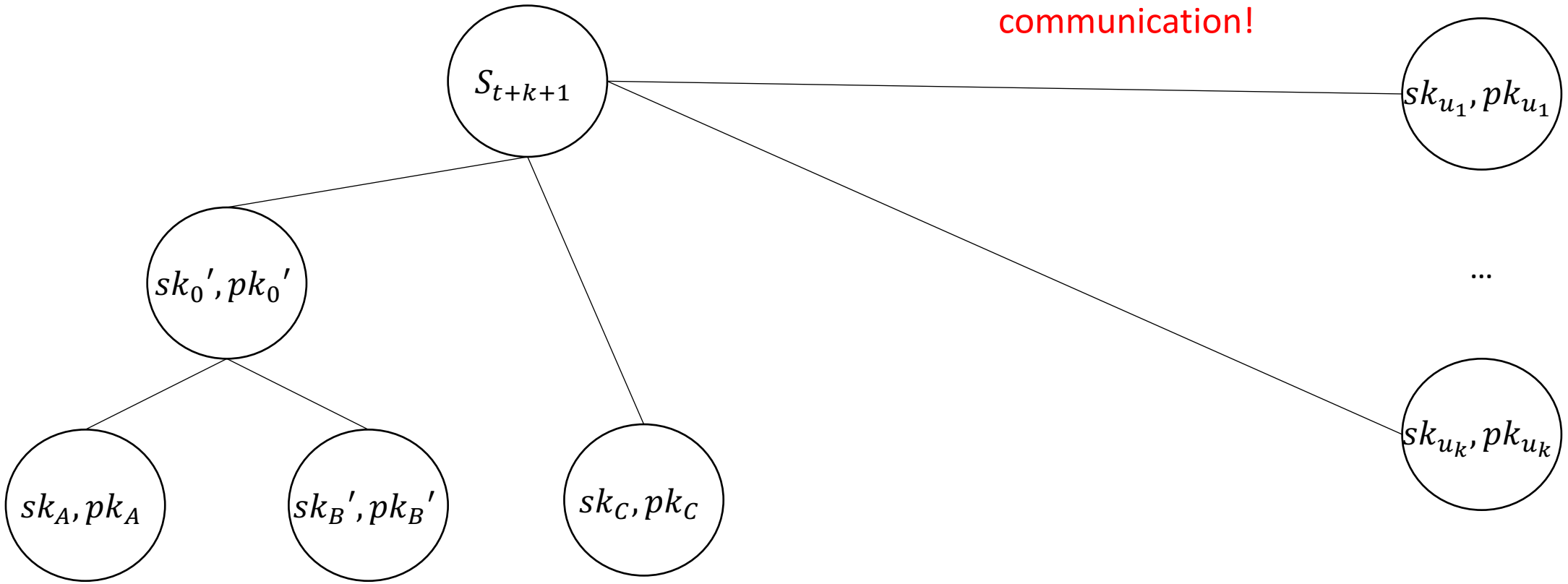


A needs to encrypt S_{t+k+1} to each added user: $\Omega(k)$ communication!

Worst-Case Inefficiency of Existing Schemes

- Consider: Now A removes D

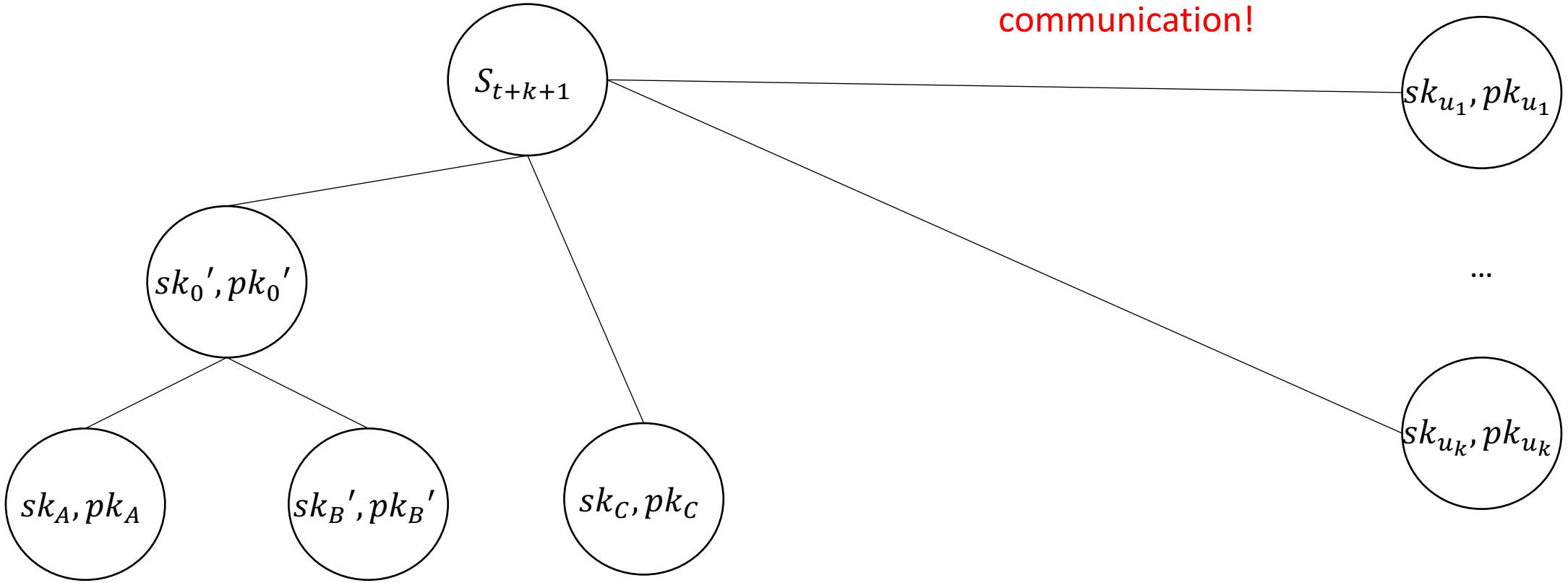
A needs to encrypt S_{t+k+1} to each added user: $\Omega(k)$ communication!



Worst-Case Inefficiency of Existing Schemes

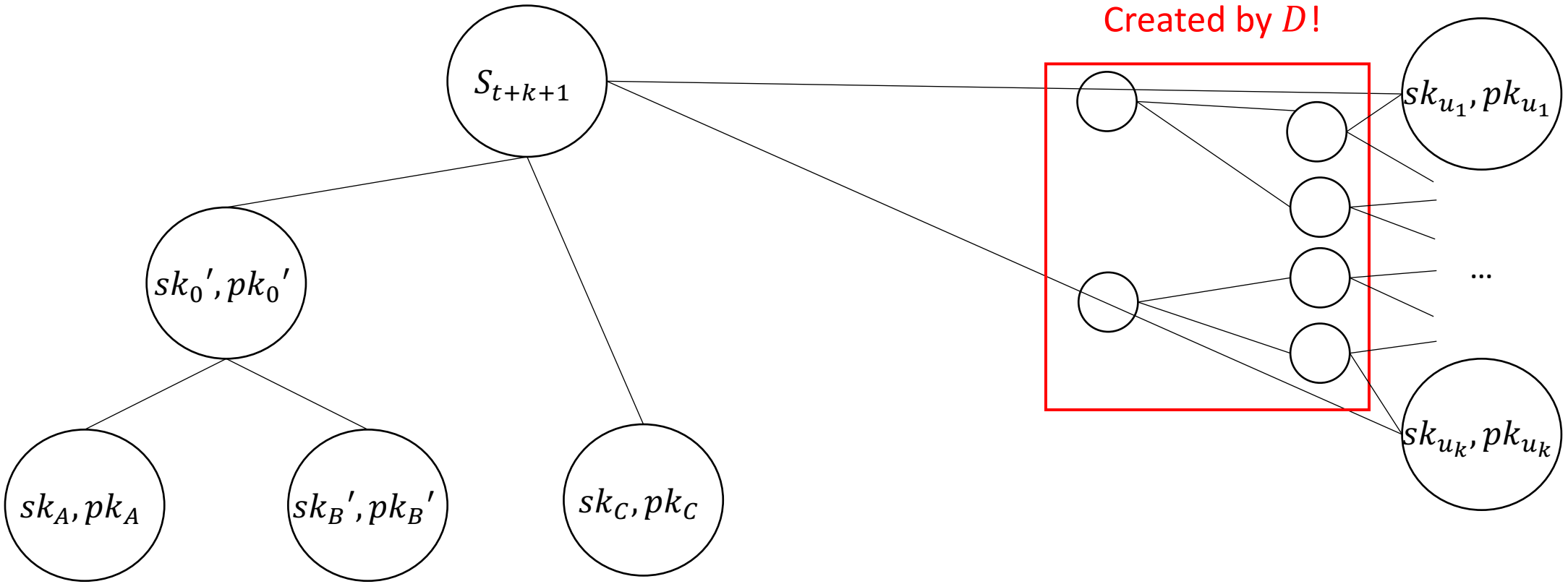
- Consider: Now A removes D
- Can continue *ad infinitum*

A needs to encrypt S_{t+k+1} to each added user: $\Omega(k)$ communication!



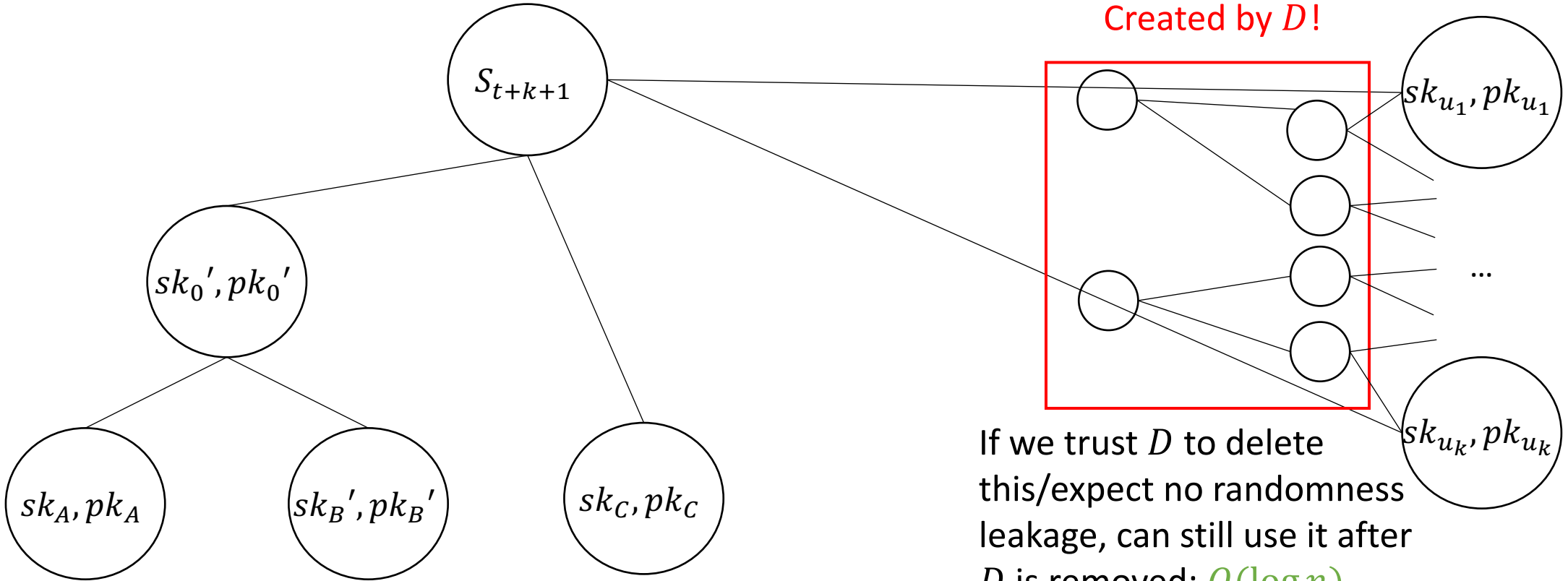
Trivial Efficiency without (standard) PCS

- Recall: PCS accounts for malicious users who are not trusted to delete "old" secrets & viruses/implementation issues that leak randomness



Trivial Efficiency without (standard) PCS

- Recall: PCS accounts for malicious users who are not trusted to delete "old" secrets & viruses/implementation issues that leak randomness



If we trust D to delete this/expect no randomness leakage, can still use it after D is removed: $O(\log n)$ efficiency

CGKA Worst-Case Lower Bound

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts
   - So k users' states not independent and can sample correlation successfully

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts
   - So k users' states not independent and can sample correlation successfully
3. Then show that CGKA => CKE, tightly in terms of communication

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts
   - So k users' states not independent and can sample correlation successfully
3. Then show that CGKA => CKE, tightly in terms of communication
   - Via example sequence from previous slides -- for *any* protocol the only state added users $u_1, \ldots, u_k$ share with other group members was created by D

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts
   - So k users' states not independent and can sample correlation successfully
3. Then show that CGKA => CKE, tightly in terms of communication
   - Via example sequence from previous slides -- for *any* protocol the only state added users $u_1, \ldots, u_k$ share with other group members was created by D
   - So after D removed, must communicate fresh key to k independent pk's

CGKA Worst-Case Lower Bound

1. We abstract out Compact Key Exchange (CKE) primitive to formally capture communicating fresh key S to k independent pk's
2. Show no CKE protocol black-box from PKE can achieve this with $o(k)$ communication
   - High-level: if ciphertext $o(k)$ length, can only fit $o(k)$ base PKE ciphertexts
   - So k users' states not independent and can sample correlation successfully
3. Then show that CGKA => CKE, tightly in terms of communication
   - Via example sequence from previous slides -- for *any* protocol the only state added users $u_1, \ldots, u_k$ share with other group members was created by D
   - So after D removed, must communicate fresh key to k independent pk's
   - Also a bit more general (not just D, but multiple users)

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication
  - Even amortized over long periods of time!

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication
  - Even amortized over long periods of time!
- Also in paper: no single CGKA protocol performs best on all sequences

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication
  - Even amortized over long periods of time!
- Also in paper: no single CGKA protocol performs best on all sequences
- Open Problems:

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication
  - Even amortized over long periods of time!
- Also in paper: no single CGKA protocol performs best on all sequences
- Open Problems:
  - Prove the lower bound w.r.t. stronger primitives/assumptions than PKE

Conclusion

- CGKA protocols black-box from PKE suffer worst-case $\Omega(n)$ communication
  - Even amortized over long periods of time!
- Also in paper: no single CGKA protocol performs best on all sequences
- Open Problems:
  - Prove the lower bound w.r.t. stronger primitives/assumptions than PKE
  - Prove better “average-case” communication upper bounds
  - What sequences are average? (huge space between trivially good and very bad)