Forward-Secure Encryption with Fast Forwarding

Yevgeniy Dodis **Daniel Jost** Harish Karthikeyan

New York University

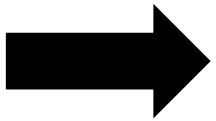
Motivation

Forward Security:

- Security of the past must not be affected by a compromise at the present time.

Coarse-grained FS:

- FS kicks in once protocol execution / session is over.
- Examples: Key-exchange, TLS



Fine-grained FS:

- Continuous / ongoing FS during protocol execution.
- Examples: Secure messaging

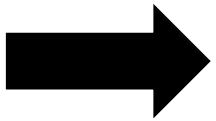
Motivation

Forward Security:

- Security of the past must not be affected by a compromise at the present time.

Coarse-grained FS:

- FS kicks in once protocol execution / session is over.
- Examples: Key-exchange, TLS



Fine-grained FS:

- Continuous / ongoing FS during protocol execution.
- Examples: Secure messaging

Typically involves notion of an epoch

Motivation

- What if a party is “stuck” in an old epoch?
- Many forward-secure primitives require sequential processing of missed messages!
- Why important:
  - Most recent messages often inherently more important
  - In group-chat protocols like MLS: sending requires the latest key!

Contributions

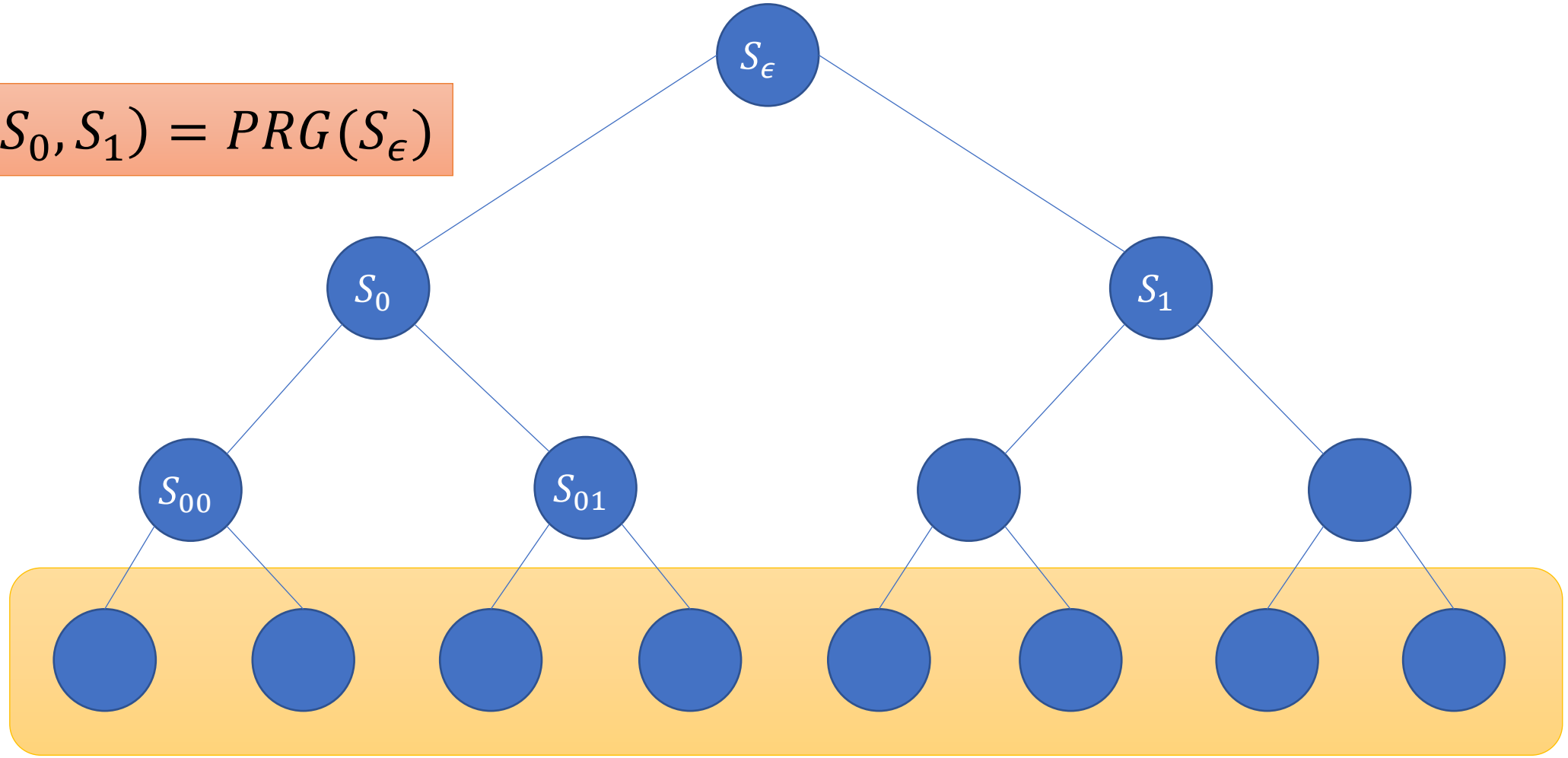
- We investigate a novel dimension of the price of forward-secure encryption: **fast-forwarding**.
- We ask if one can build forward-secure encryption with a **sublinear** fast-forward property.
  - Symmetric setting: PRG (and hence stream-cipher)
  - Asymmetric setting: Updatable PKE

Symmetric: Fast-Forwardable FS-PRG

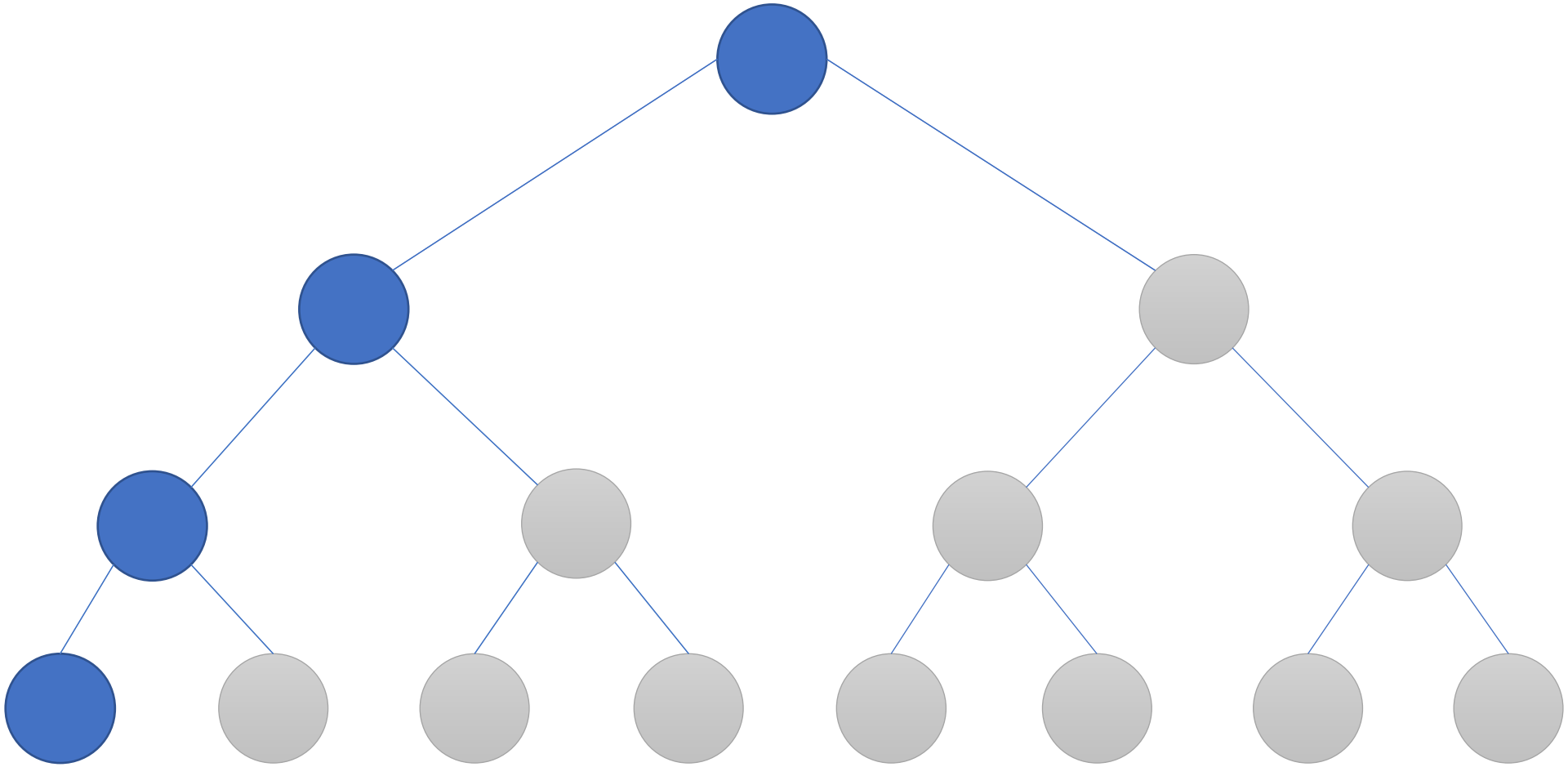
Observation: Folklore construction

- GGM construction
- Adapt the template by Bellare and Miner (C’99) and Malkin, Micciancio, and Miner (EC’02) for building forward-secure signature schemes

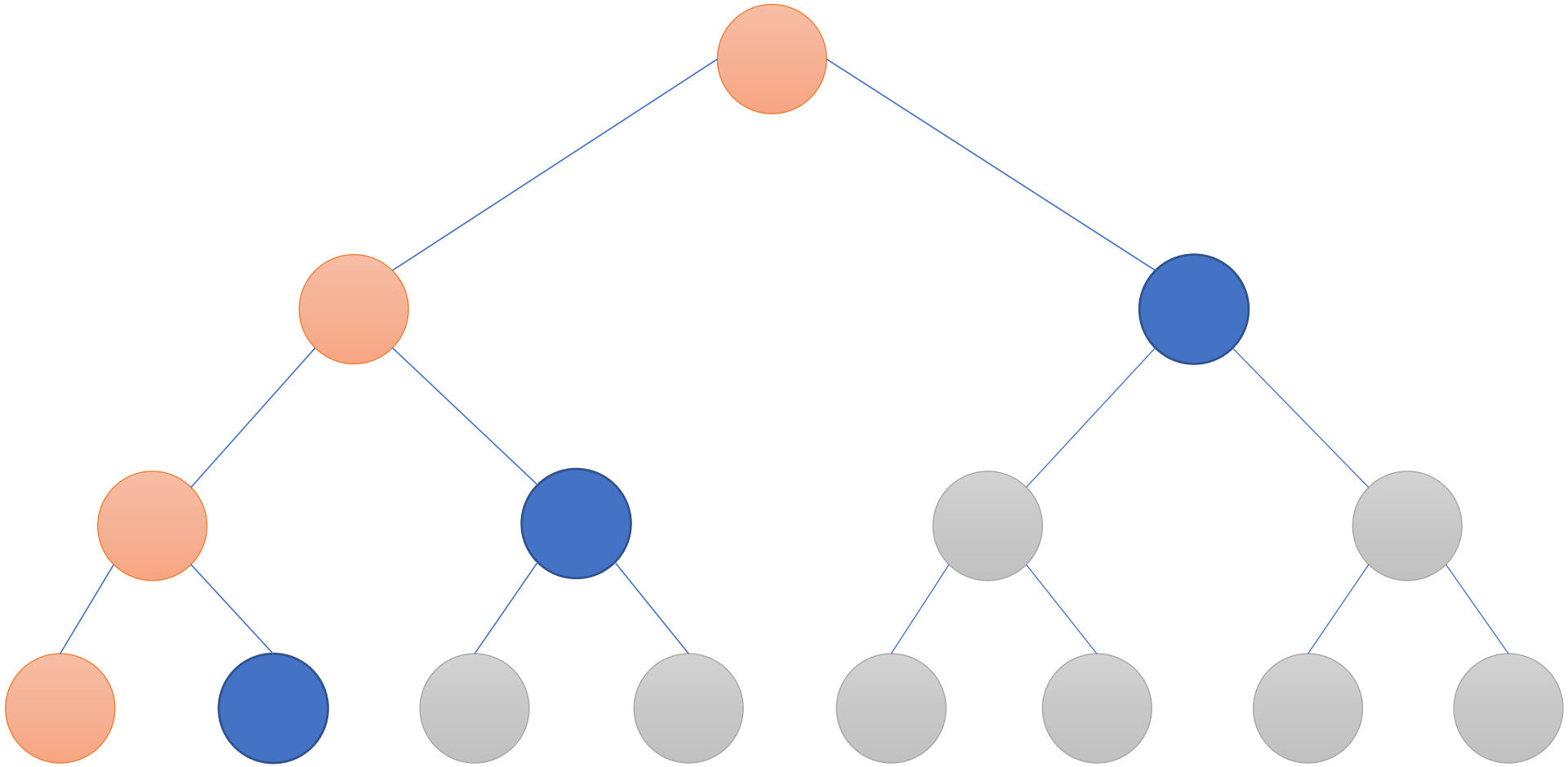
$$(S_0, S_1) = PRG(S_\epsilon)$$



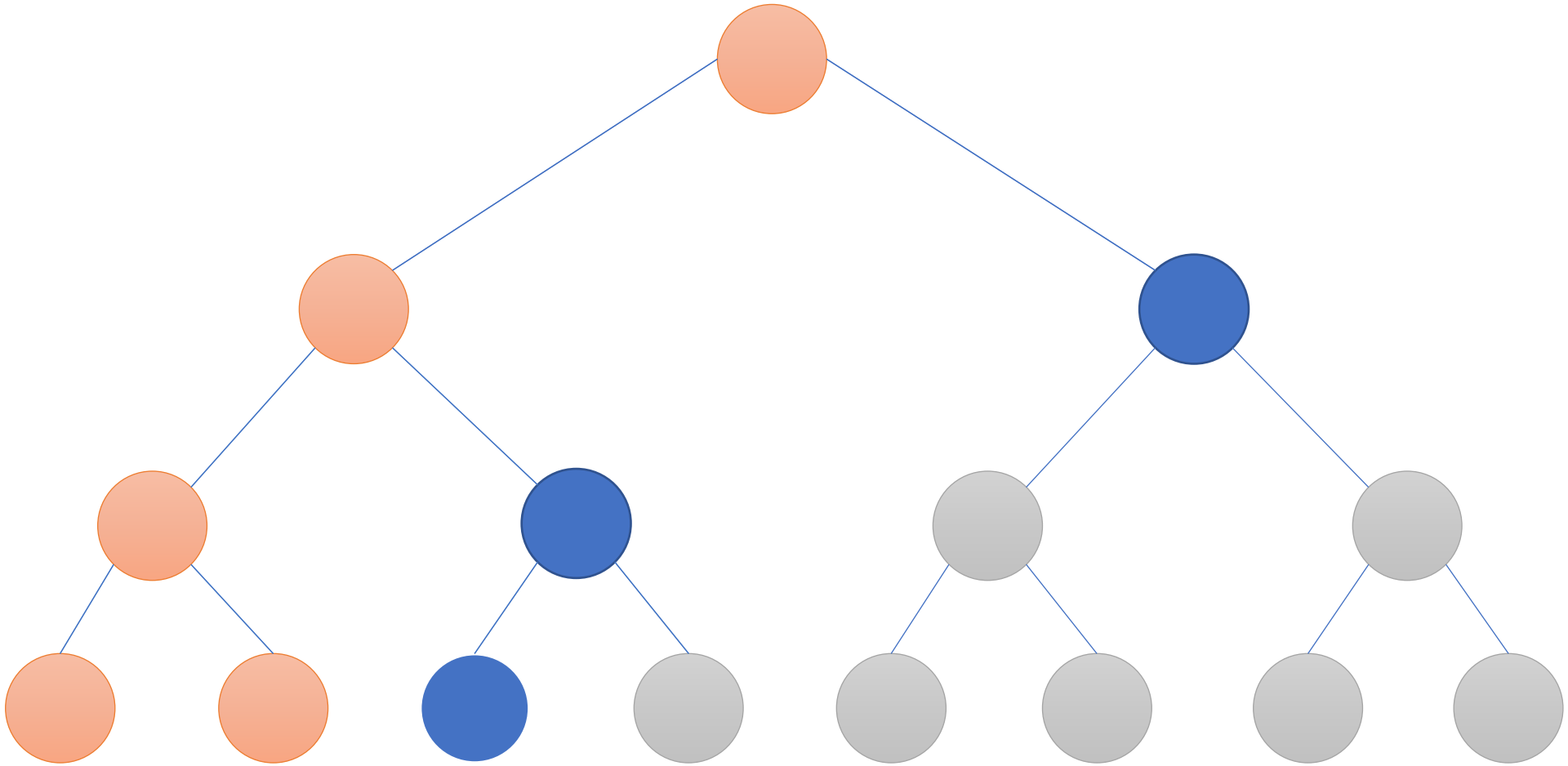
Epoch keys



e=1



e=2



e=3

Symmetric: Fast-Forwardable FS-PRG

This uses logarithmic local storage and logarithmic computation per (sequential) epoch change.

Question: Can we do better?

Symmetric: Fast-Forwardable FS-PRG

This uses logarithmic local storage and logarithmic computation per (sequential) epoch change.

Question: Can we do better?

Change of model:

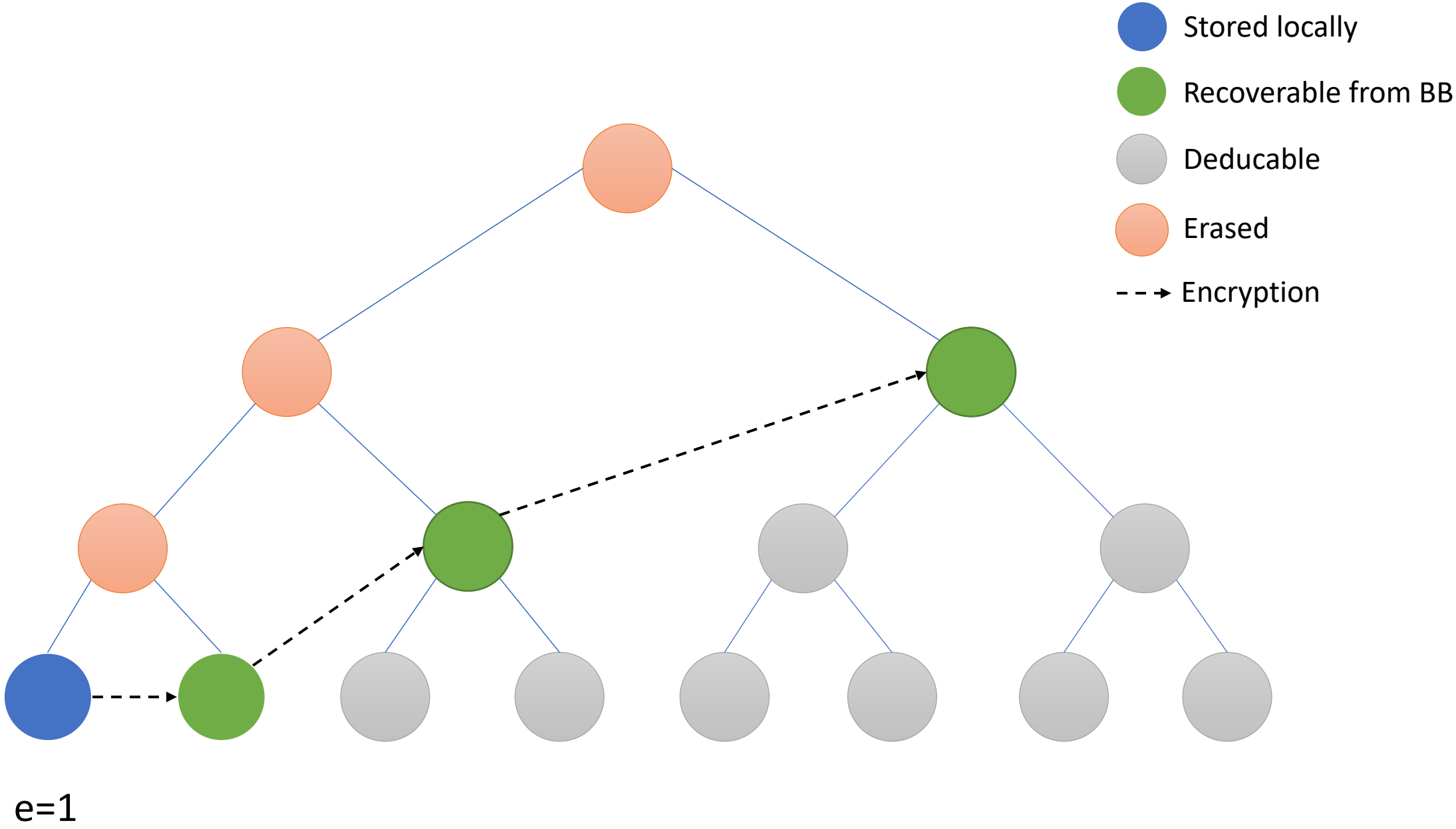
- Assume public bulletin board (honest but curious)
- Communication overhead!
  → can make sense if communication required anyway

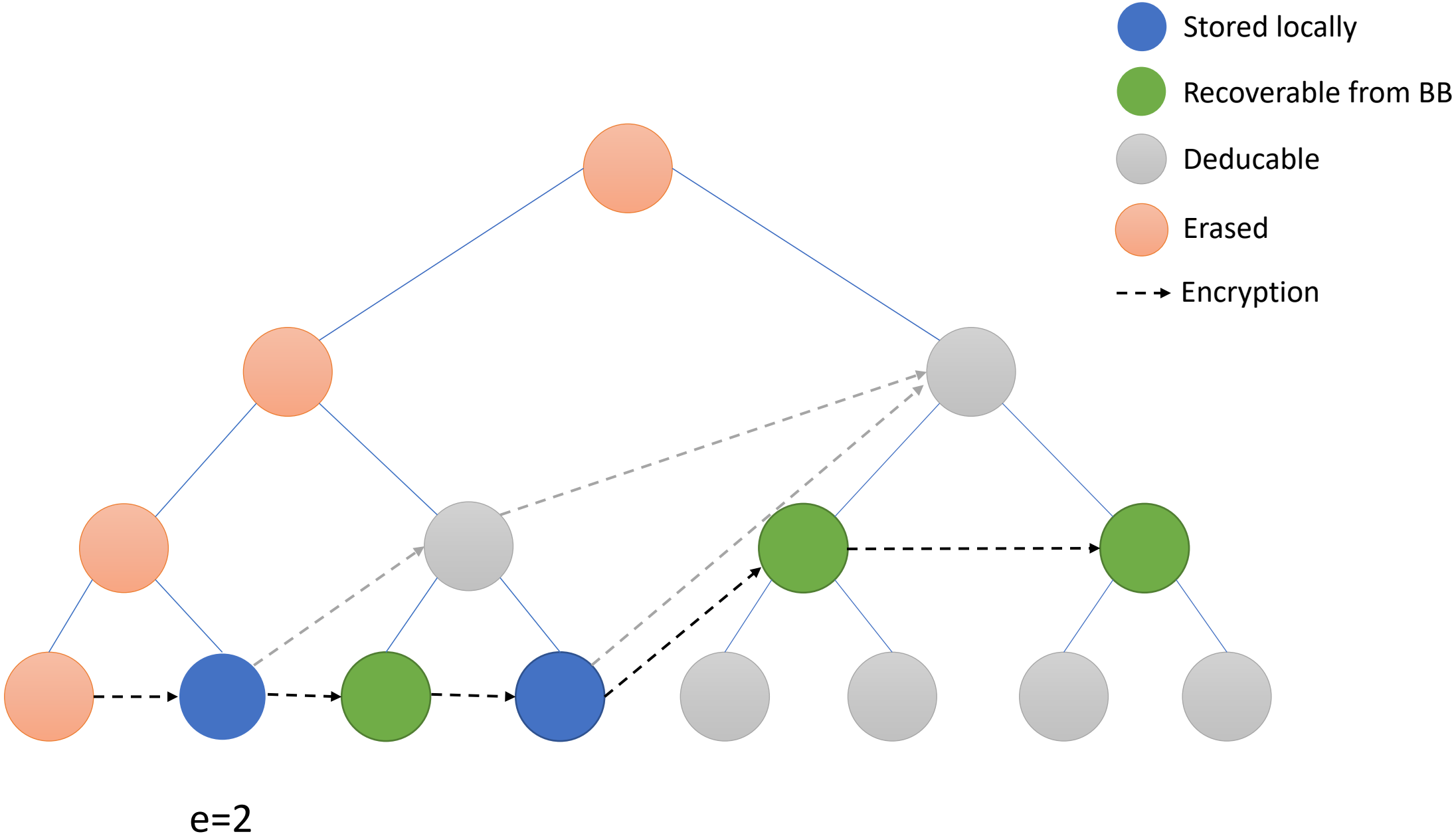
Symmetric: FF Forward-Secure PRG

Our result:

Fast-forwardable forward-secure PRG with

- **Local storage:** constant
- **Sequential Update:** constant computation and communication
- **Fast-forward:** logarithmic computation and communication


Stored locally
Recoverable from BB
Deducable
Erased
Encryption
e=1

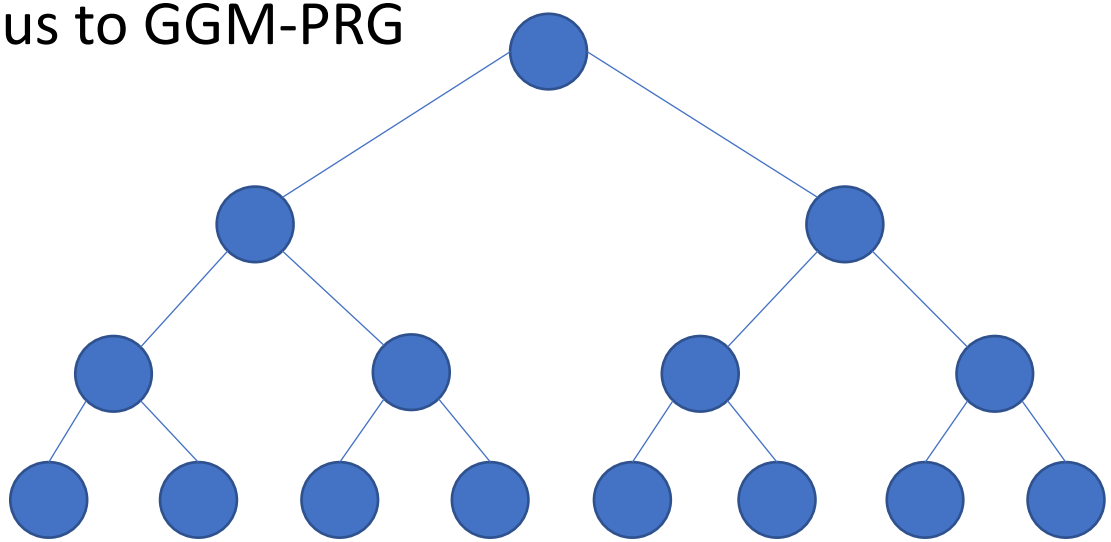

Stored locally
Recoverable from BB
Deducable
Erased
Encryption
e=2

Asymmetric Setting

- Observation:
  - Only known non-trivial FS-PKE: generic construction via hierarchical identity-based encryption (HIBE)
  - Allows for fast-forwarding analogous to GGM-PRG

Asymmetric Setting

- Observation:
  - Only known non-trivial FS-PKE: generic construction via hierarchical identity-based encryption (HIBE)
  - Allows for fast-forwarding analogous to GGM-PRG

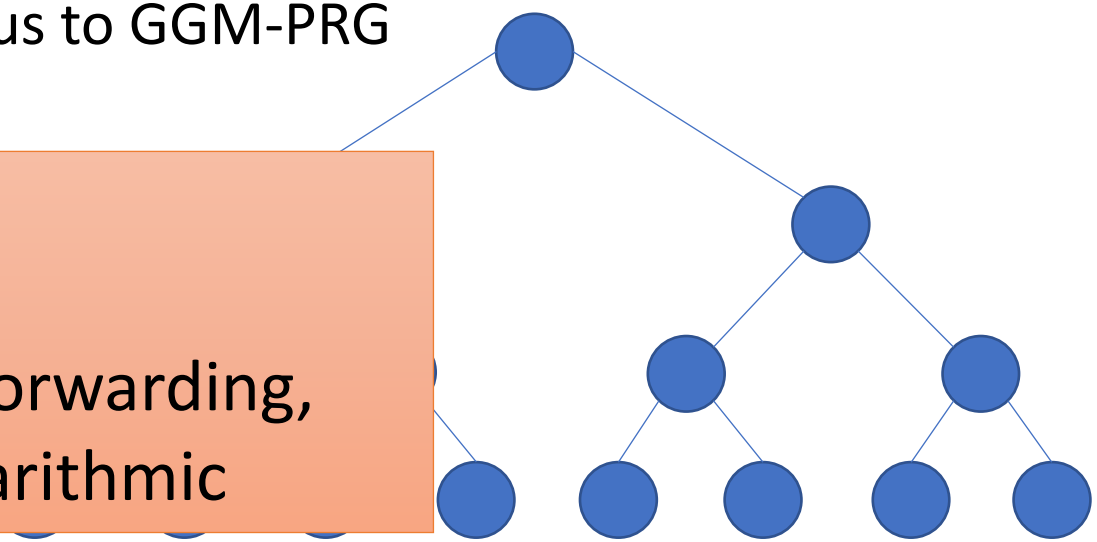


HIBE identity tree

Asymmetric Setting

- Observation:
  - Only known non-trivial FS-PKE: generic construction via hierarchical identity-based encryption (HIBE)
  - Allows for fast-forwarding analogous to GGM-PRG

Drawbacks:
- HIBE rather expensive
- With Bulletin-Board: after fast-forwarding, sequential updates become logarithmic



HIBE identity tree

Updatable PKE (JMM'19)

- KeyGen → pk_1, sk_1
- Encrypt(pk_i, M) → C
- Decrypt(C, sk_i) → M

PKE scheme

- UpdGen(pk_i) → $(\delta_{i\to i+1}, \Delta_{i\to i+1})$
- UpdatePK(pk_i, $\delta_{i\to i+1}$) → pk_{i+1}
- UpdateSK(sk_i,$\Delta_{i\to i+1}$) → sk_{i+1}

Updating mechanism for forward security

Update-Homomorphic UPKE

Schemes by:
Dodis, Karthikeyan, Wichs

- KeyGen → pk_1, sk_1
- Encrypt(pk_i, M) → C
- Decrypt(C, sk_i) → M

- UpdGen(pk_i) → $(\delta_{i\to i+1}, \Delta_{i\to i+1})$
- UpdatePK(pk_i, $\delta_{i\to i+1}$) → pk_{i+1}
- UpdateSK(sk_i,$\Delta_{i\to i+1}$) → sk_{i+1}

Upd-Comb($\Delta_{i\to j}$, $\Delta_{j\to k}$) → $\Delta_{i\to k}$
UpdateSK(sk_i,$\Delta_{i\to k}$) → sk_k

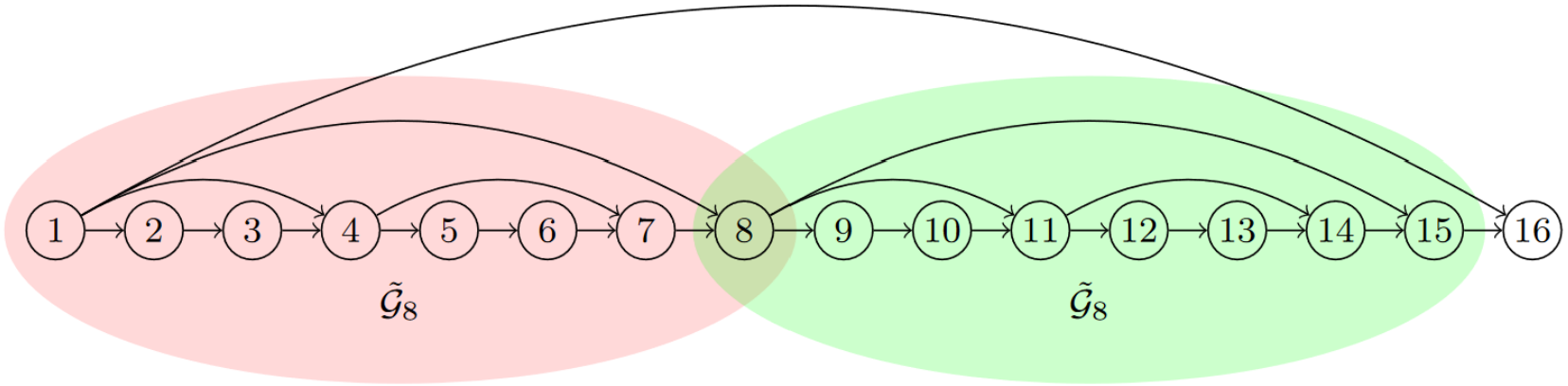
UpdateSK(sk_i,$\Delta_{i\to j}$) → sk_j
UpdateSK(sk_j,$\Delta_{j\to k}$) → sk_k

FF-UPKE from Update-Homomorphic UPKE

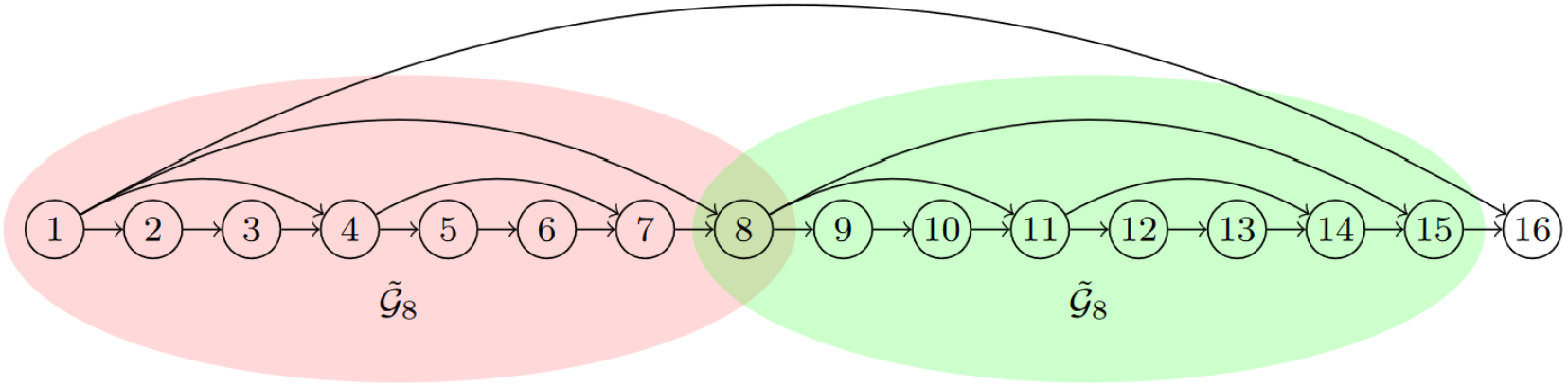
- Basic idea:
  - Senders produce “cumulative updates” stored in bulletin board that receiver can use for fast forwarding
  - Which cumulative updates to generate?

FF-UPKE from Update-Homomorphic UPKE

- Basic idea:
  - Senders produce “cumulative updates” stored in bulletin board that receiver can use for fast forwarding
  - Which cumulative updates to generate?



FF-UPKE from Update-Homomorphic UPKE



- Relevant quantities:
  - Diameter → efficiency of fast-forwarding
  - Cut (number of edges crossing two consecutive nodes)
    → Sender efficiency
  - In-degree → sender's communication complexity in single-sender setting

Open Problems

- Asymmetric setting:
  - Can we build more efficient update-homomorphic UPKE?
    →Need homomorphic encryption with message-space ≈ secret-key space
  - Can we build FF-UPKE more efficiently in general?

- What other applications are there where fast-forwarding might be relevant?