Secure Data Deletion from Persistent Media

Joel Reardon, Hubert Ritzdorf, David Basin, Srdjan Capkun
ETH Zurich

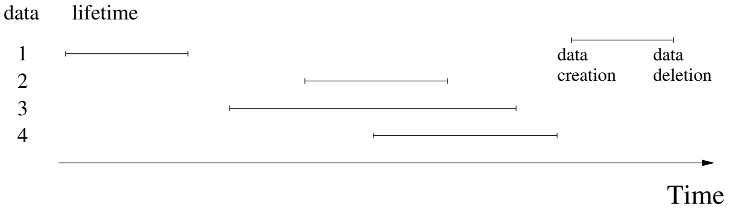
2013-11-05

Secure deletion: the task of deleting data from a physical medium so that the data is irrecoverable by an adversary.

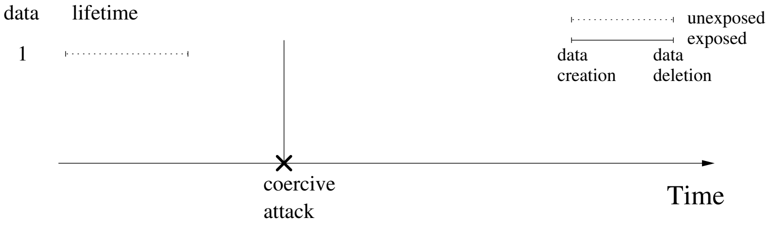
Security Model: Coercive Adversary

- A coercive adversary can:
  - Unexpectedly compromise all data stored on the user's storage media
  - Obtain any secret keys / passphrases to access this data
  - Strike multiple times
- Consequently:
  - No "extraordinary" actions can be taken to delete data
  - Any data the user can access when the adversary strikes is exposed to the adversary
- Models, for example, a subpoena

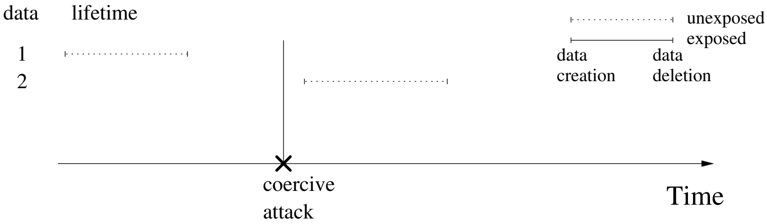
Data Life Timeline



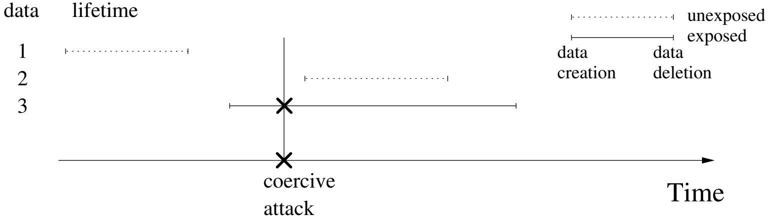
Data Life Timeline



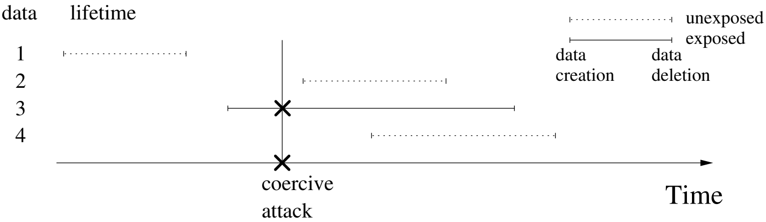
Data Life Timeline



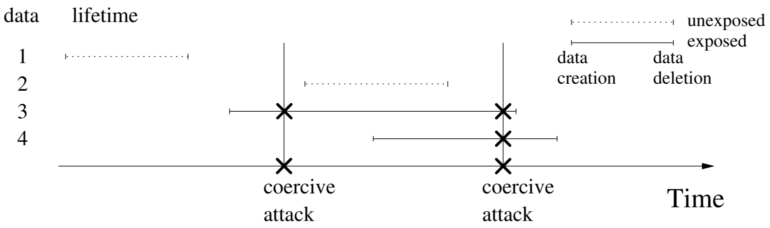
Data Life Timeline



Data Life Timeline



Data Life Timeline

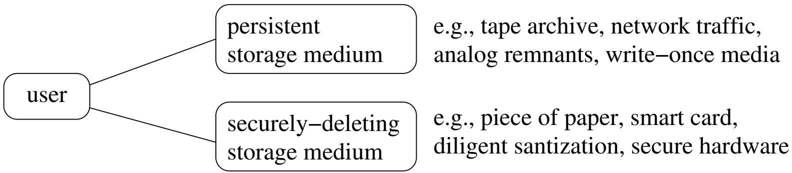


Persistent Storage Medium

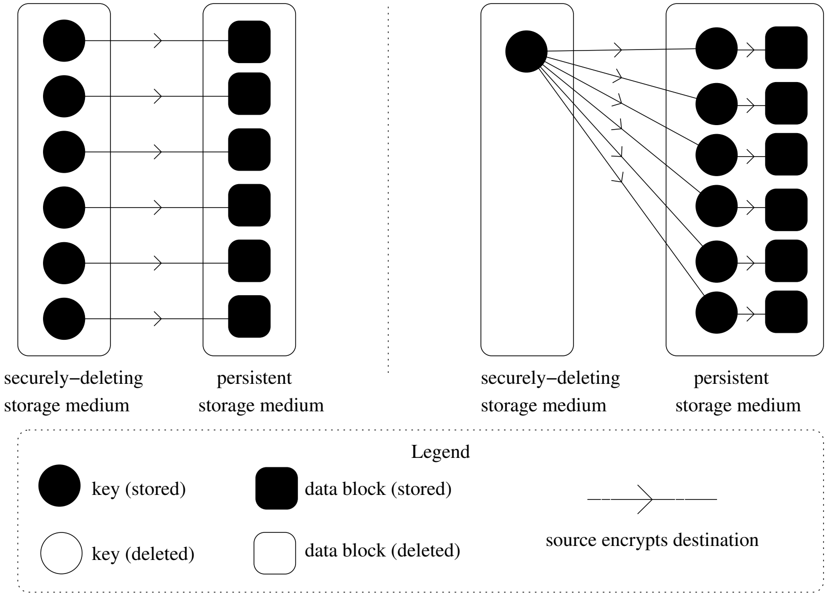
- **Persistent Storage Medium** is a storage medium that does not provide deletion
- Data written onto it is permanently available
- We assume that an adversary who compromises it sees the entire write history

So if we assume the adversary eventually compromises the persistent storage, how can we securely delete *anything*?

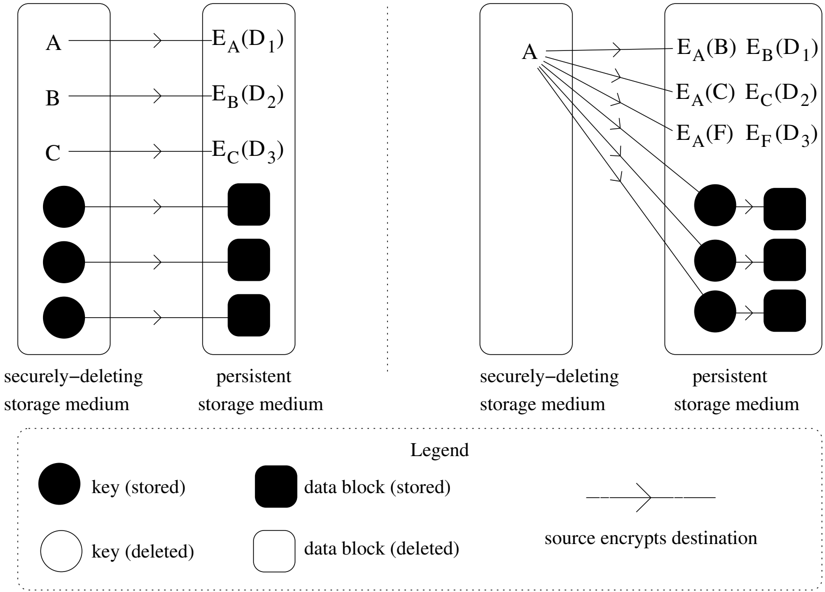
Our Model


user
persistent storage medium
securely–deleting storage medium
e.g., tape archive, network traffic, analog remnants, write–once media
e.g., piece of paper, smart card, diligent santization, secure hardware

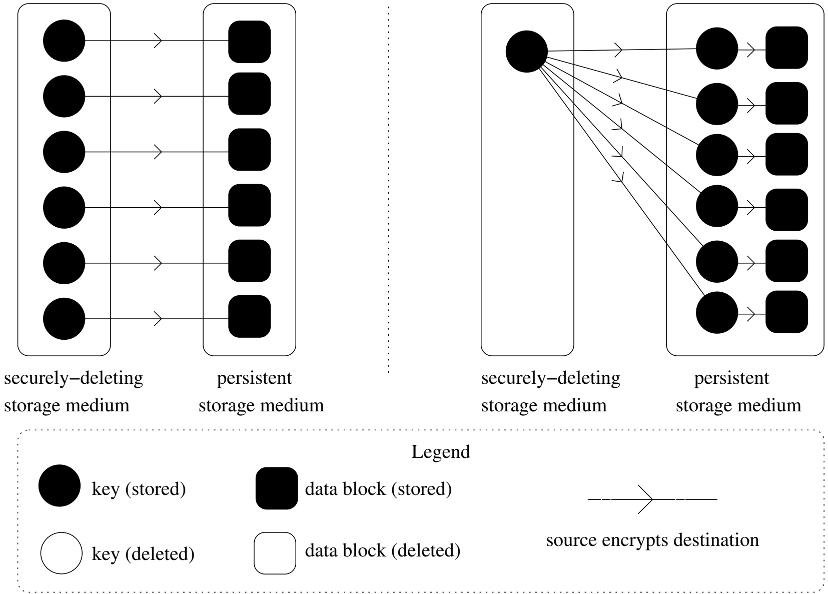
Background: Two Extreme Solutions



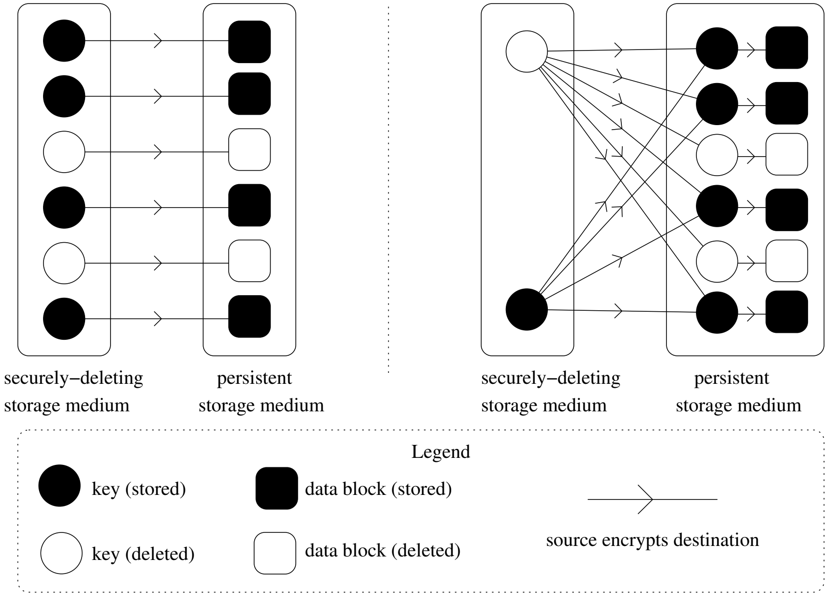
Background: Two Extreme Solutions



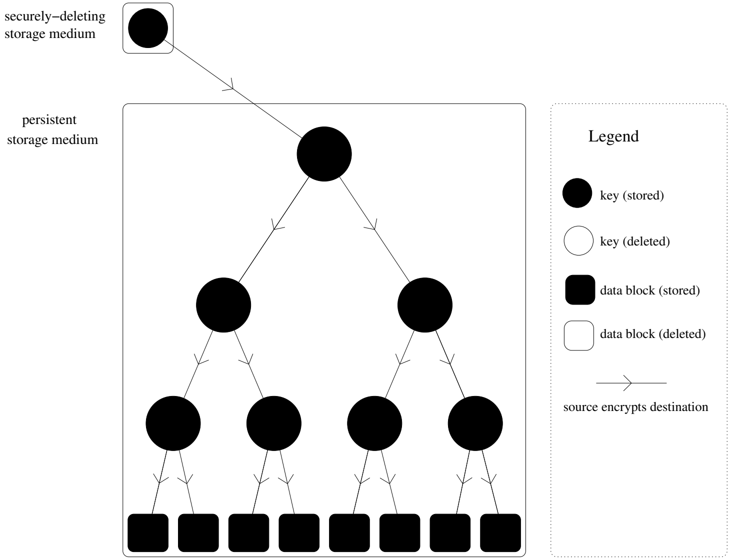
Background: Two Extreme Solutions



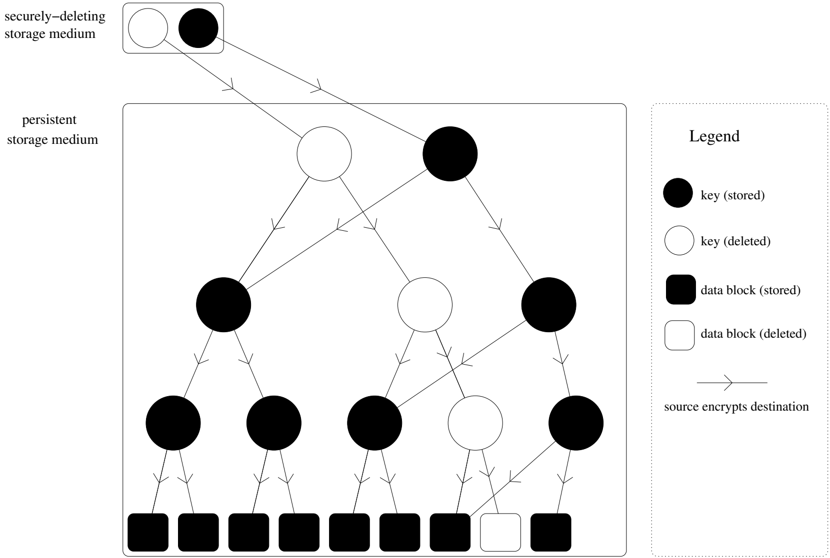
Background: Two Extreme Solutions



Background: Static Tree Solution



Background: Update Mechanism



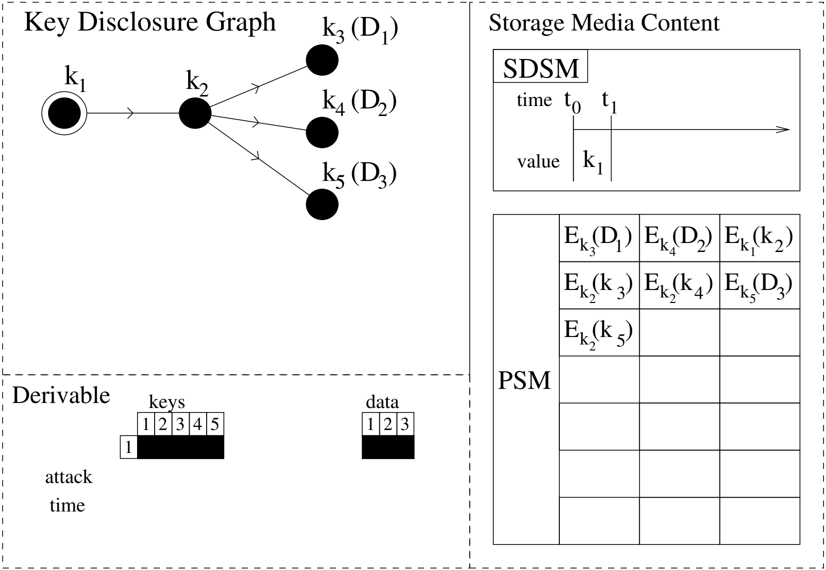
Static Tree Discussion

- Tree size/shape is fixed for eternity
  - Cannot accommodate new data
  - Tree-depth fixed even when sparse
- Proof of deletion is based on fixed shape
- Dynamic structures (e.g., B-Trees, balancing trees, etc) are more versatile
  - But the proofs become trickier with increased sophistication

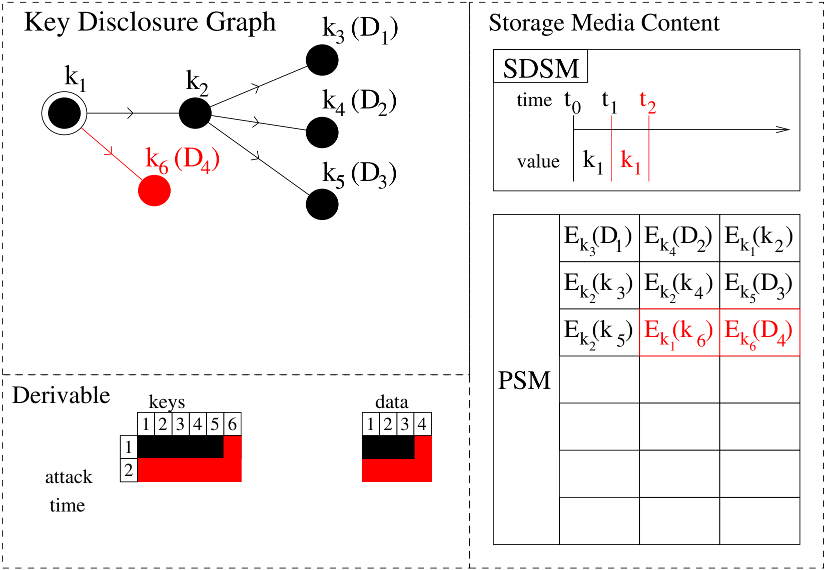
Our Contributions

- We present the **key disclosure graph**: a tool to model and reason about worst-case adversarial knowledge for persistent storage
- We present a generic **shadowing graph mutation**: a graph mutation that can express the update behaviour of arborescent data structures and facilitates secure deletion
- We characterize related work by their key disclosure graphs
- We instantiate this solution with a B-Tree

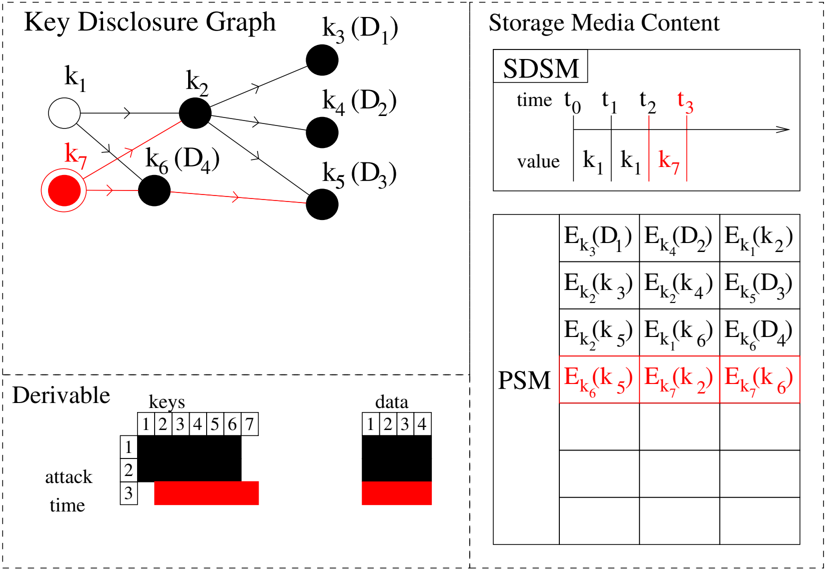
Key Disclosure Graph



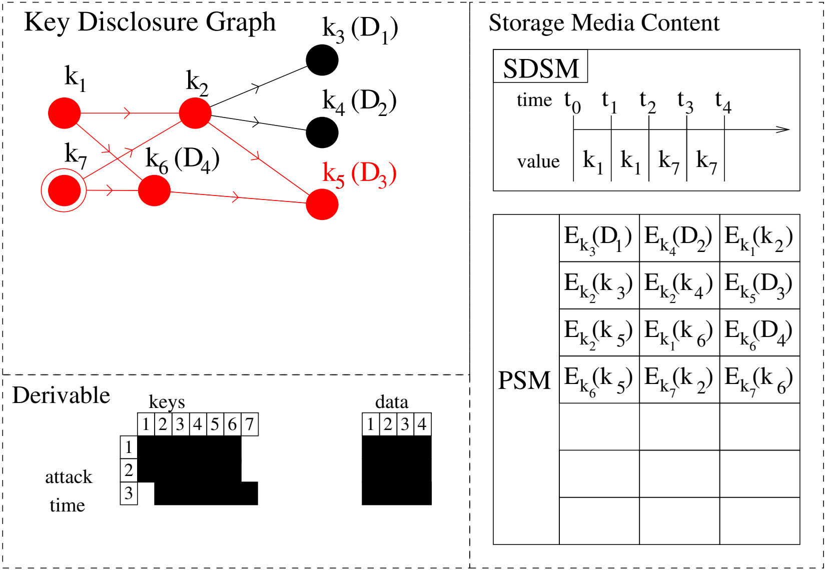
Key Disclosure Graph

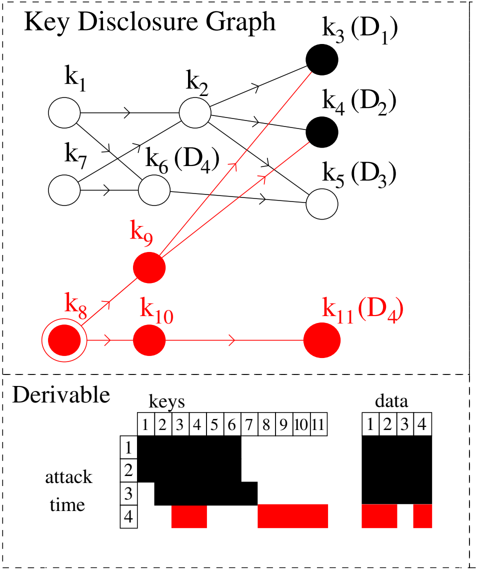


Key Disclosure Graph

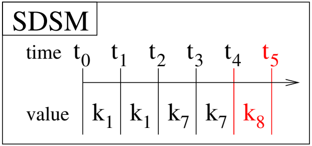


Key Disclosure Graph





Storage Media Content

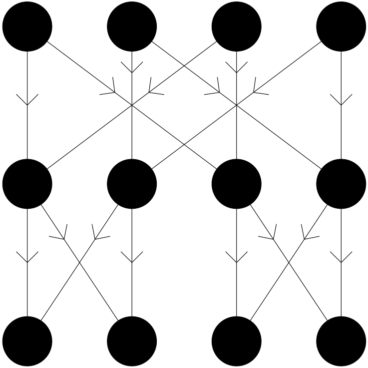


PSM			
	$E_{k_3}(D_1)$	$E_{k_4}(D_2)$	$E_{k_1}(k_2)$
	$E_{k_2}(k_3)$	$E_{k_2}(k_4)$	$E_{k_5}(D_3)$
	$E_{k_2}(k_5)$	$E_{k_1}(k_6)$	$E_{k_6}(D_4)$
	$E_{k_6}(k_5)$	$E_{k_7}(k_2)$	$E_{k_7}(k_6)$
	$E_{k_8}(k_9)$	$E_{k_8}(k_{10})$	$E_{k_9}(k_3)$
	$E_{k_9}(k_4)$	$E_{k_{10}}(k_{11})$	$E_{k_{11}}(D_4)$

Secure deletion

- Secure deletion of data requires:
  - when writing data, ensuring all previous values stored in the SDSM cannot derive its key
  - when deleting data, determining all of the derivable ancestors in the KDG and make them all underivable
  - ancestor relation based on the ever-growing KDG
- How do we avoid storing the entire KDG?
  - require that in the KDG there is at most one unique path that connects any pair of vertices
  - in graph theory, such a graph is called a **mangrove**

Mangroves

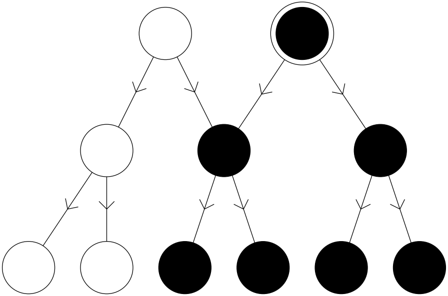


How do we ensure that the KDG is always a mangrove?
We use shadowed updates.

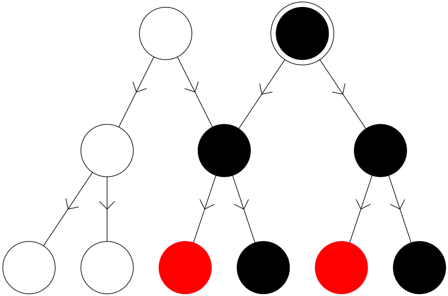
Shadowing

- Shadowed updates is a technique in file systems
  - New versions of data are written to new (empty) locations
  - Old versions remains but are no longer valid
  - Anything that references the old version is also shadow-updated by referring to the new location
- We use **keys** instead of versions
  - Any change results in a new key being generated
    to encrypt the new version
  - Key wrappers must then change to store the new key, etc.

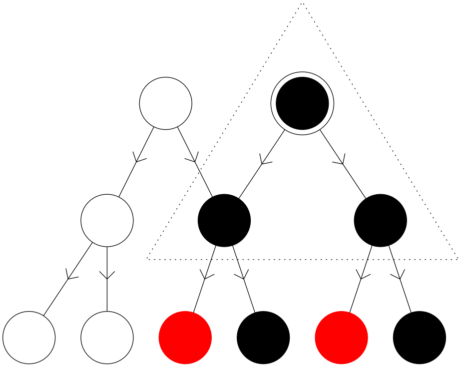
Shadowing Mutation



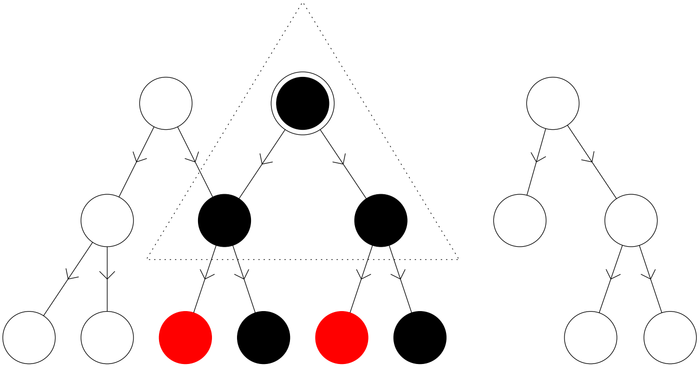
Shadowing Mutation



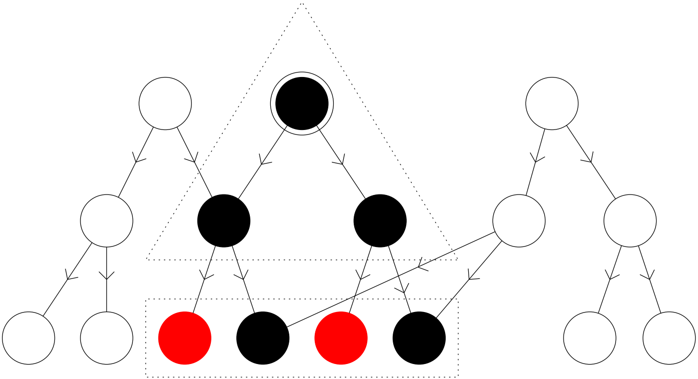
Shadowing Mutation



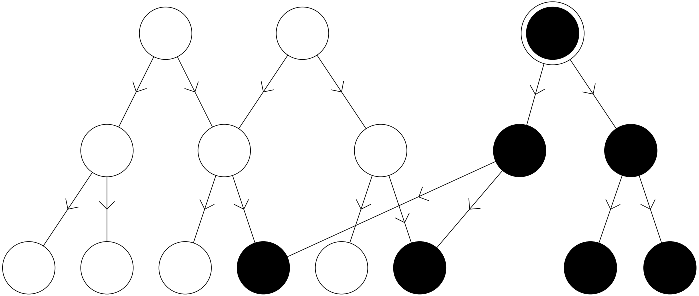
Shadowing Mutation



Shadowing Mutation



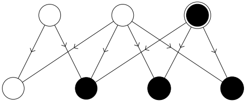
Shadowing Mutation



Shadowing Mutation

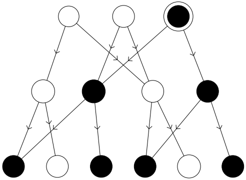
- Shadowing mutations can implement any arborescent data structure
- A shadowing mutation applied on a mangrove results in a mangrove
- Shadowing mutations do not permit old (pre-mutation) nodes to access new (post-mutation) nodes
- Computing the ancestors of a node requires only following its unique path

Related Work



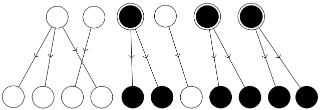
Boneh and Lipton

persistant storage: magnetic tape
securely–deleting: e.g., paper, floppy disk
update mechanism: re–encrypt keys with new master secret



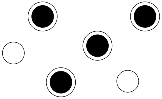
Di Crescenzo et al.

persistent and securely–deleting medium are explicitly considered
update mechanism: re–encrypt keys on static path to root



Perlman's Ephemerizer

persistant storage: communication channel
securely–deleting: trusted–third party
update mechanism: master keys correspond to expiration times

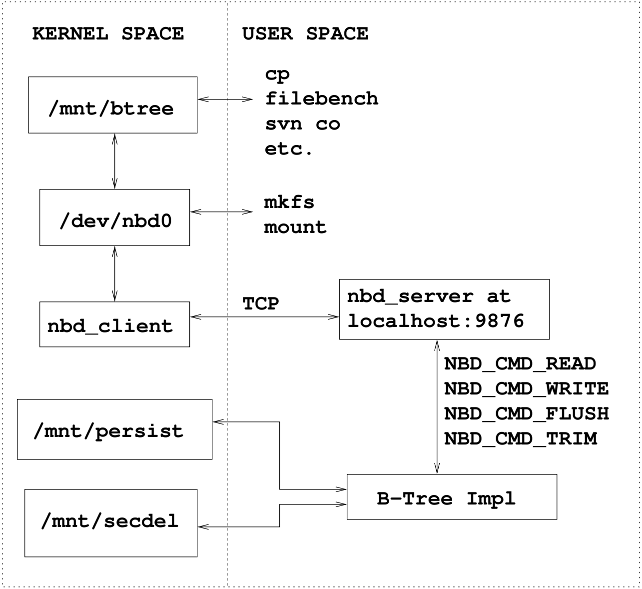


DNEFS

persistant storage: flash memory
securely–deleting: reserved area of flash
update mechanism: erase flash memory in reserved area

We implement a caching B-Tree version of this solution.

Implementation



Overhead Results

		B-Tree block size			
		4 KiB	16 KiB	64 KiB	256 KiB
general	total data blocks	6553600	1638400	409600	102400
	tree height	5	3	2	2
	cache size (nodes)	2048	512	128	32
	MiBs sharing path	0.16	2.65	42.6	682.5
sequent.	cache hits (%)	99.3	99.7	99.9	1
	storage overhead (%)	2.4	0.6	0.1	0.03
	comm overhead (%)	2.4	0.6	0.1	0.03
	block size ovrhd (%)	0	5.3	26.3	58.1
rand 1k	cache hits (%)	64.7	59	43.2	73.8
	storage overhead (%)	2.4	0.6	0.1	0.03
	comm overhead (%)	1308.5	3129	8623.5	20671.4
	block size ovrhd (%)	497.9	2293.2	9473	38191.8
rand 1m	cache hits (%)	99.2	98.9	96.5	95.5
	storage overhead (%)	2.47	0.59	0.14	0.03
	comm overhead (%)	4.9	3.7	7.8	17.7
	block size ovrhd (%)	1	7.7	34.6	82.1
svn	cache hits (%)	99.2	98.9	96.5	95.5
	storage overhead (%)	1.74	0.42	0.1	0.02
	comm overhead (%)	4.4	4.9	5.4	2.6
	block size ovrhd (%)	0	63.4	247.9	750.2

Summary and Future Work

- Contributions
  - We introduce the key disclosure graph to characterize adversarial knowledge growth for settings with a small securely-deleting medium and a large persistent medium
  - We prove that a generic shadowing graph mutation preserves the mangrove property on the KDG, facilitating secure deletion
  - We design and implement a B-Tree-based secure-deletion solution that shows promising performance
  - We characterize related work as instances of our general model
- Future Work
  - Extensively test our implementation with real-world workloads
  - Compare overheads against related work
  - Identify the best approaches for specific scenarios and workloads