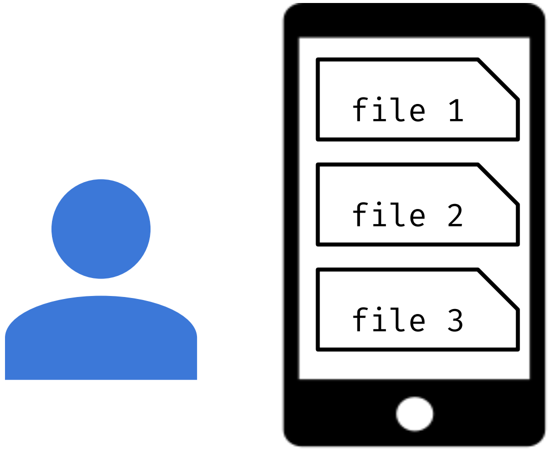
BurnBox

Self-Revocable Encryption in a World of Compelled Access

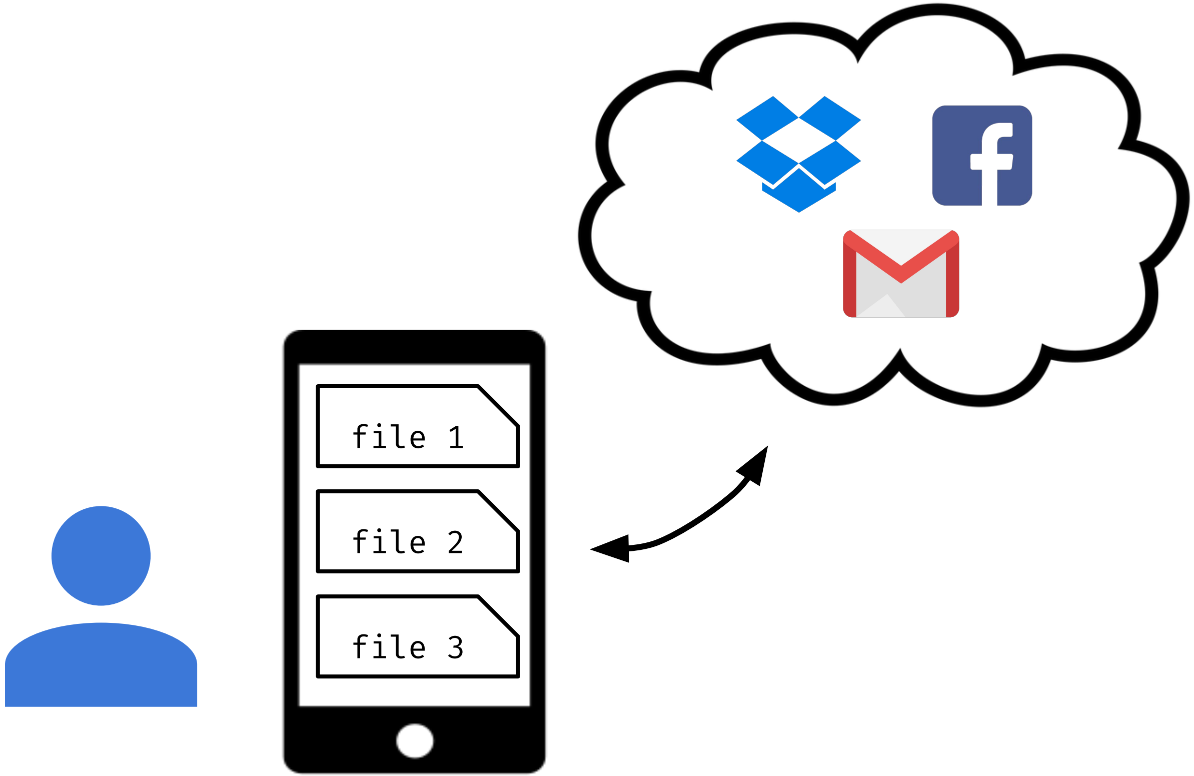
Nirvan Tyagi Muhammad Haris Mughees Thomas Ristenpart Ian Miers

Usenix Security 2018

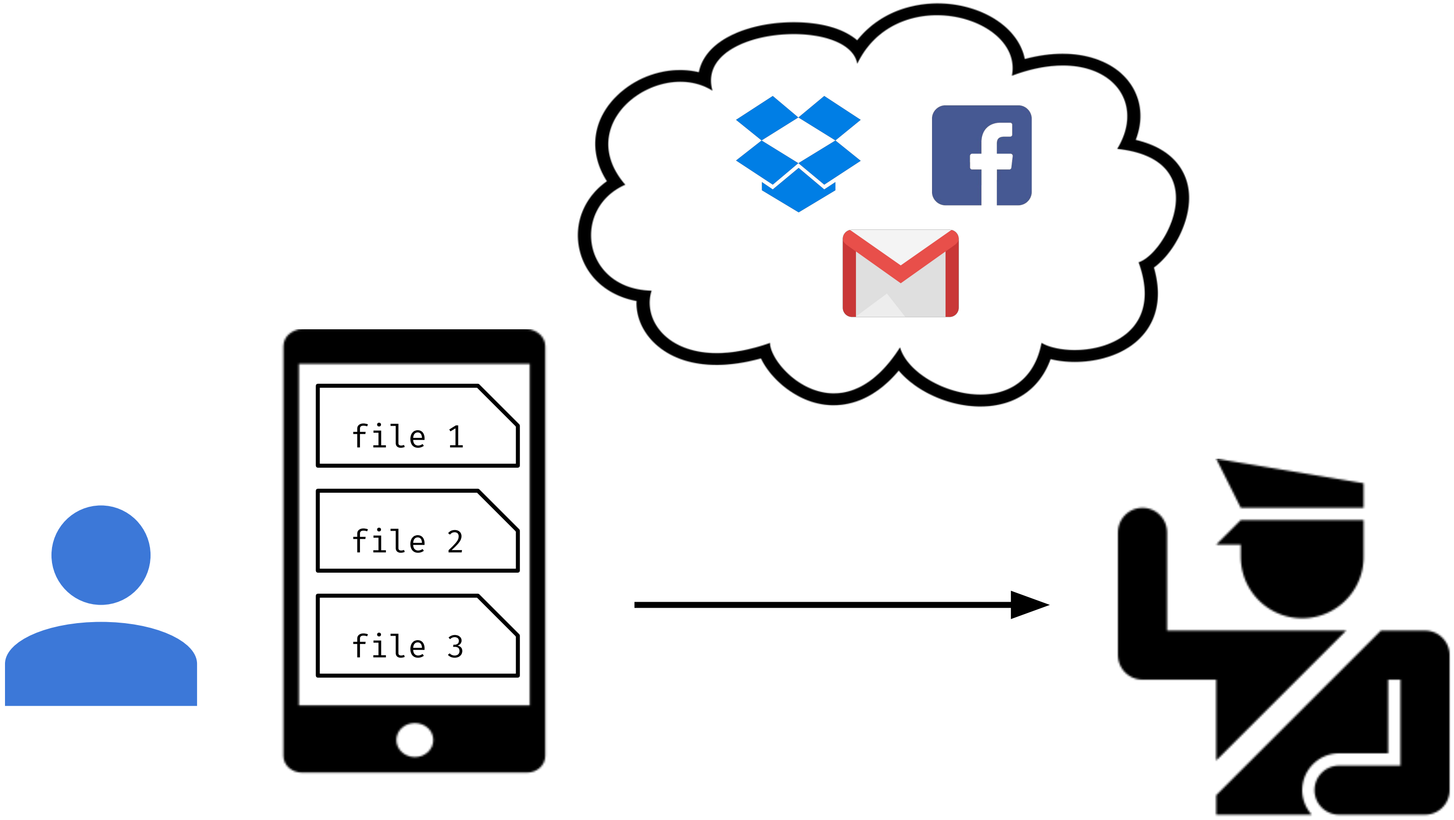
Compelled Access Setting



Compelled Access Setting

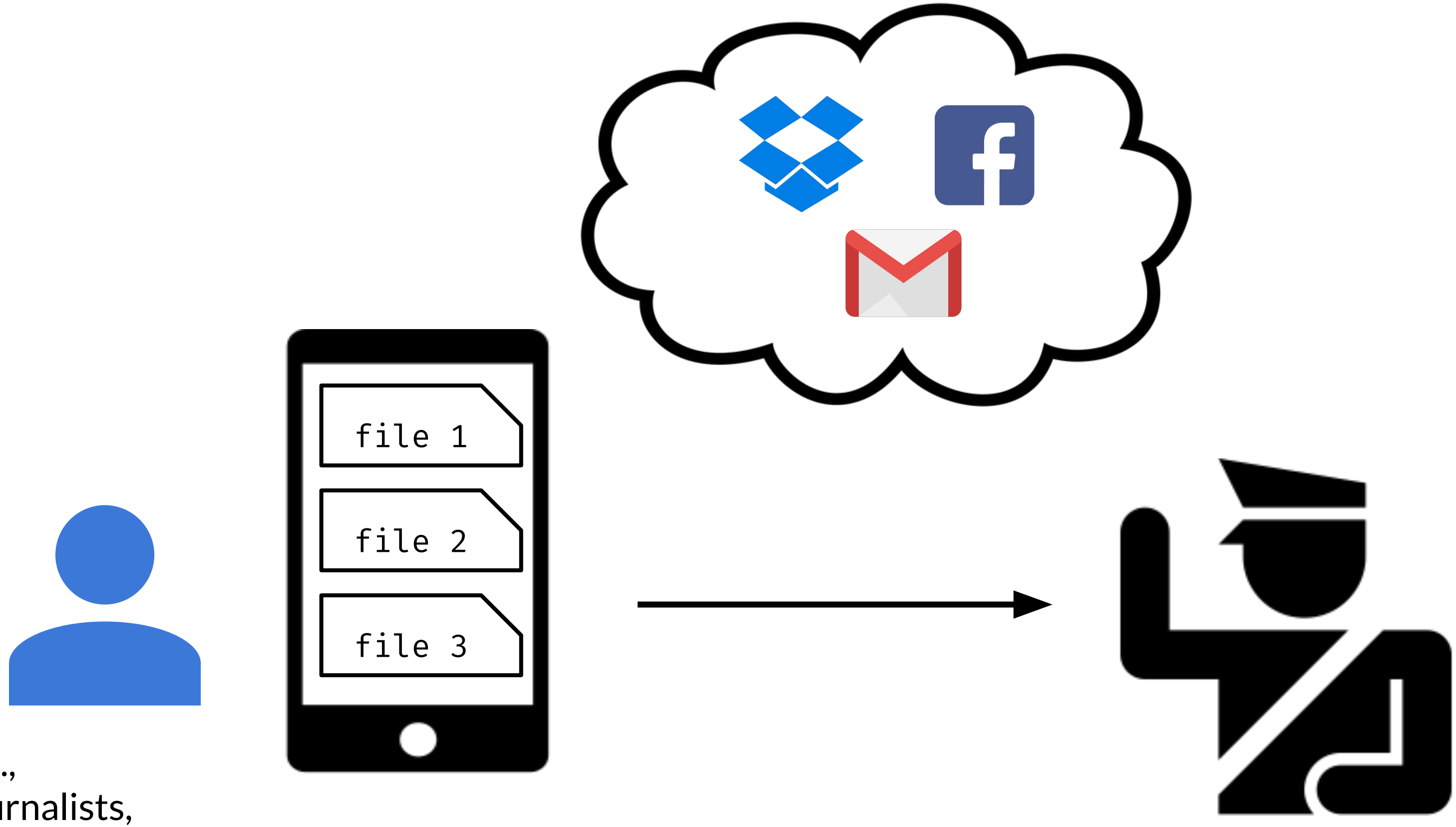


Compelled Access Setting



e.g.,
border crossing,
airport security,
police checkpoints

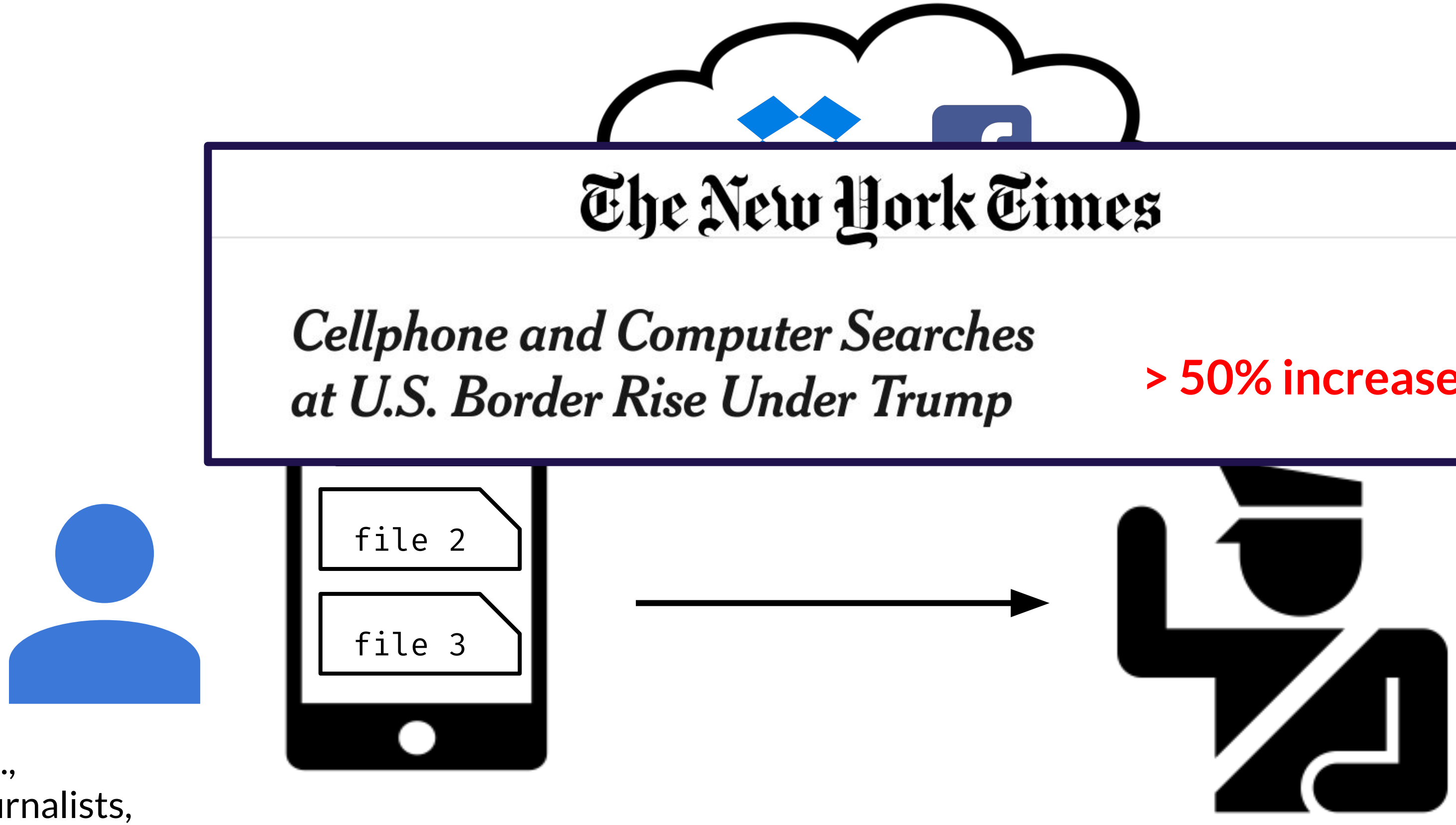
Compelled Access Setting



e.g.,
journalists,
dissidents, activists

e.g.,
border crossing,
airport security,
police checkpoints

Compelled Access Setting



The New York Times

Cellphone and Computer Searches at U.S. Border Rise Under Trump

> 50% increase

e.g.,
journalists,
dissidents, activists

e.g.,
border crossing,
airport security,
police checkpoints

Contributions

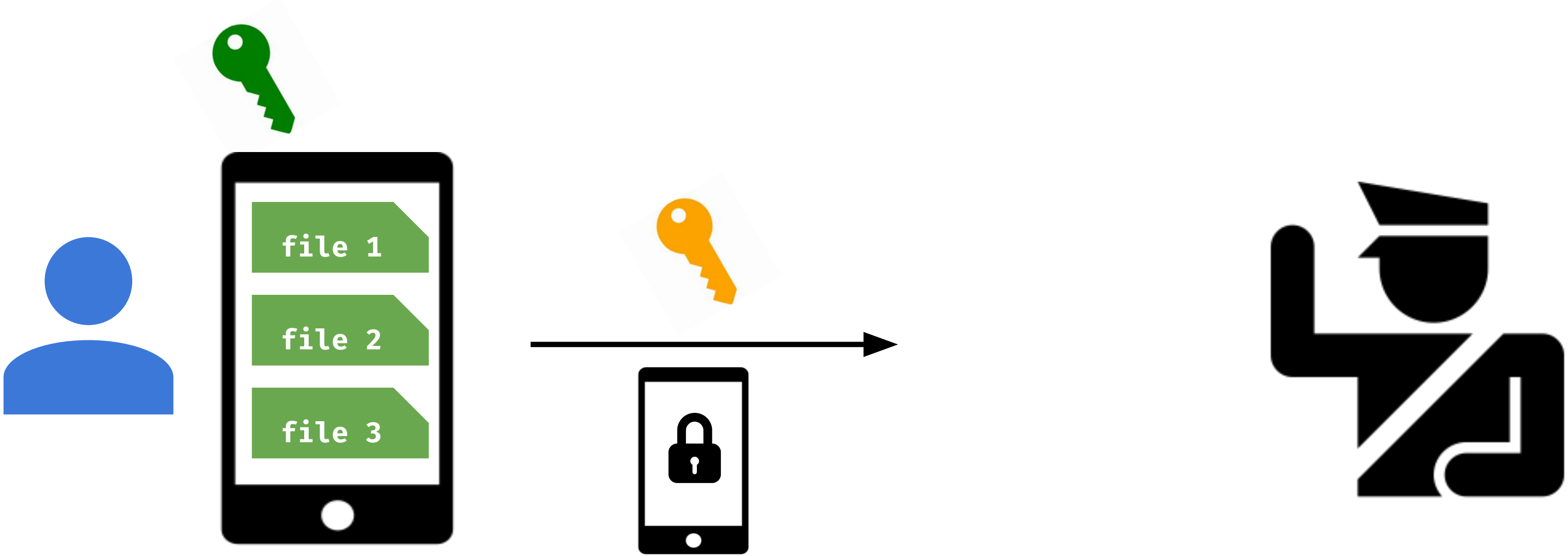
- BurnBox: Cloud storage secure in compelled access setting
  - Allow users to honestly comply with authorities while preserving confidentiality
  - Secure deletion: permanently delete files
  - Temporary revocation: self-revoke access to files temporarily
- Formal compelled access security notions and analysis
- Proof-of-concept prototype

Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]

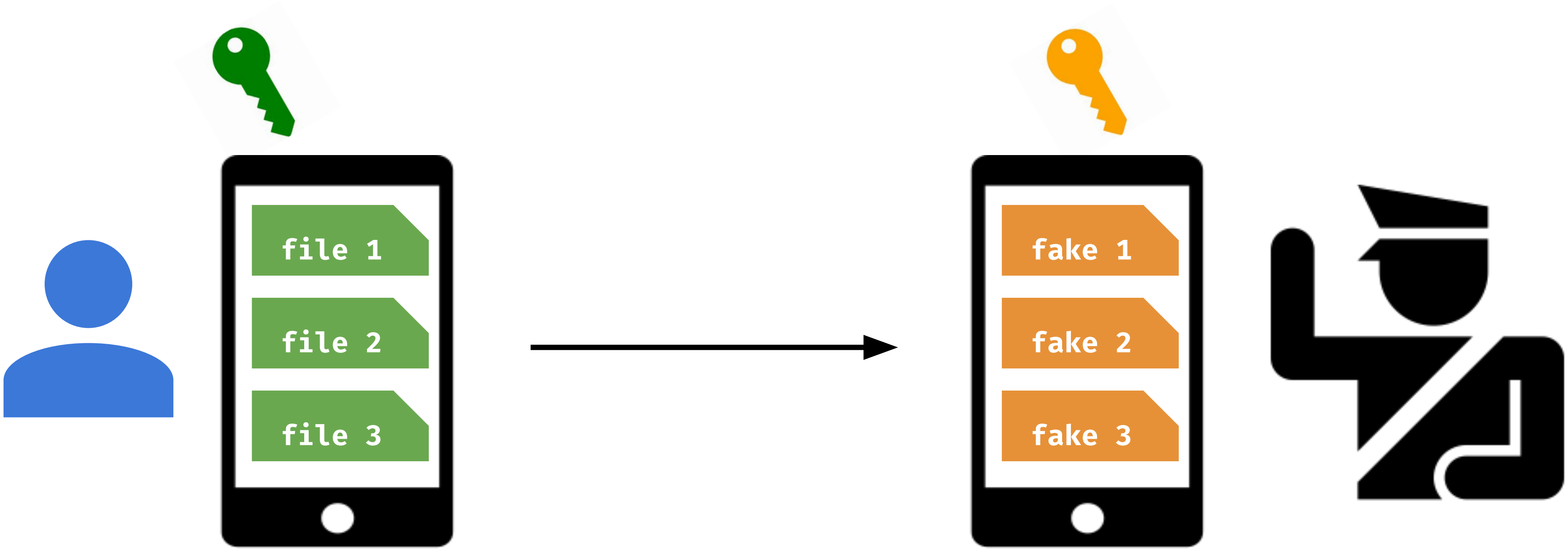
Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]



Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]

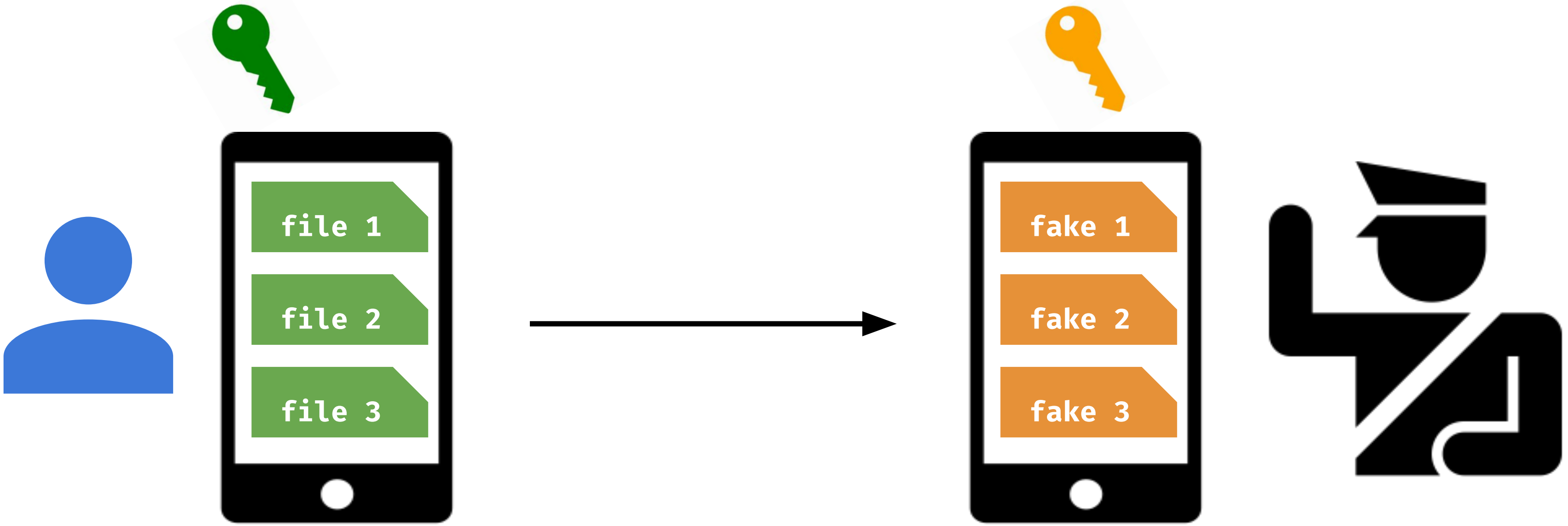


Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]

Limitation: High usability burden where deception is inherent to security

- Maintenance of “realistic-looking” fake content
- Ability to convincingly lie about duress key

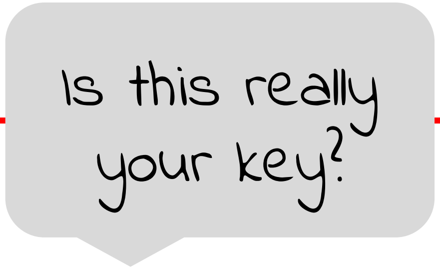


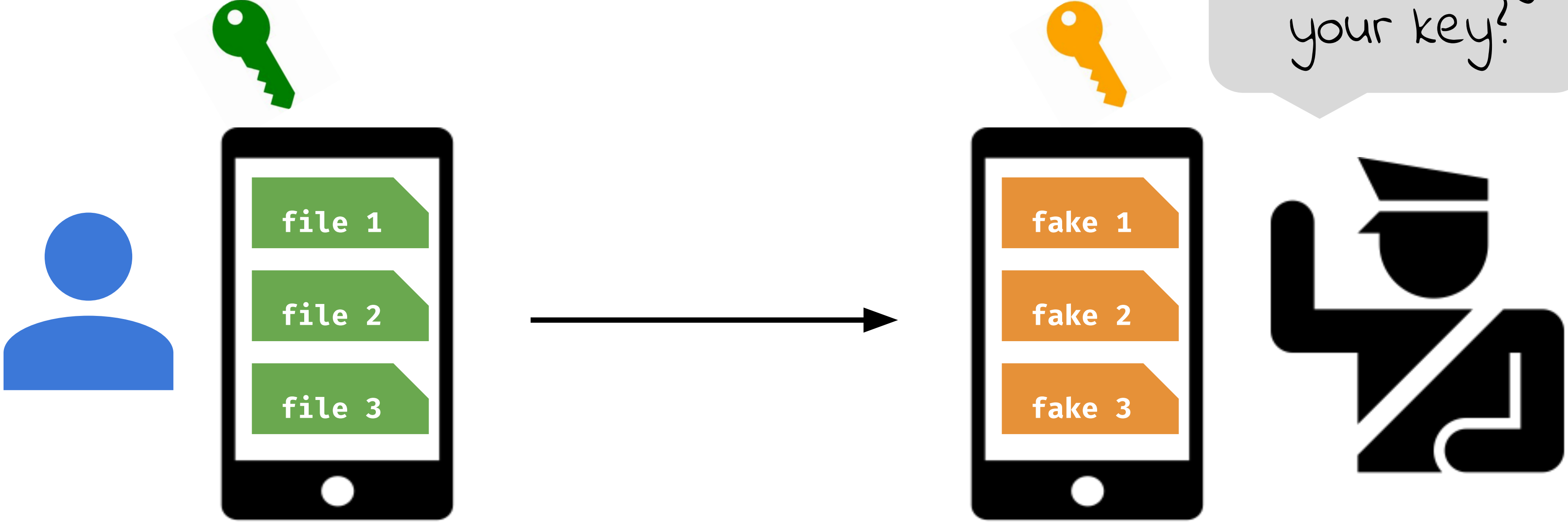
Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]

Limitation: High usability burden where deception is inherent to security

- Maintenance of “realistic-looking” fake content
- Ability to convincingly lie about duress key



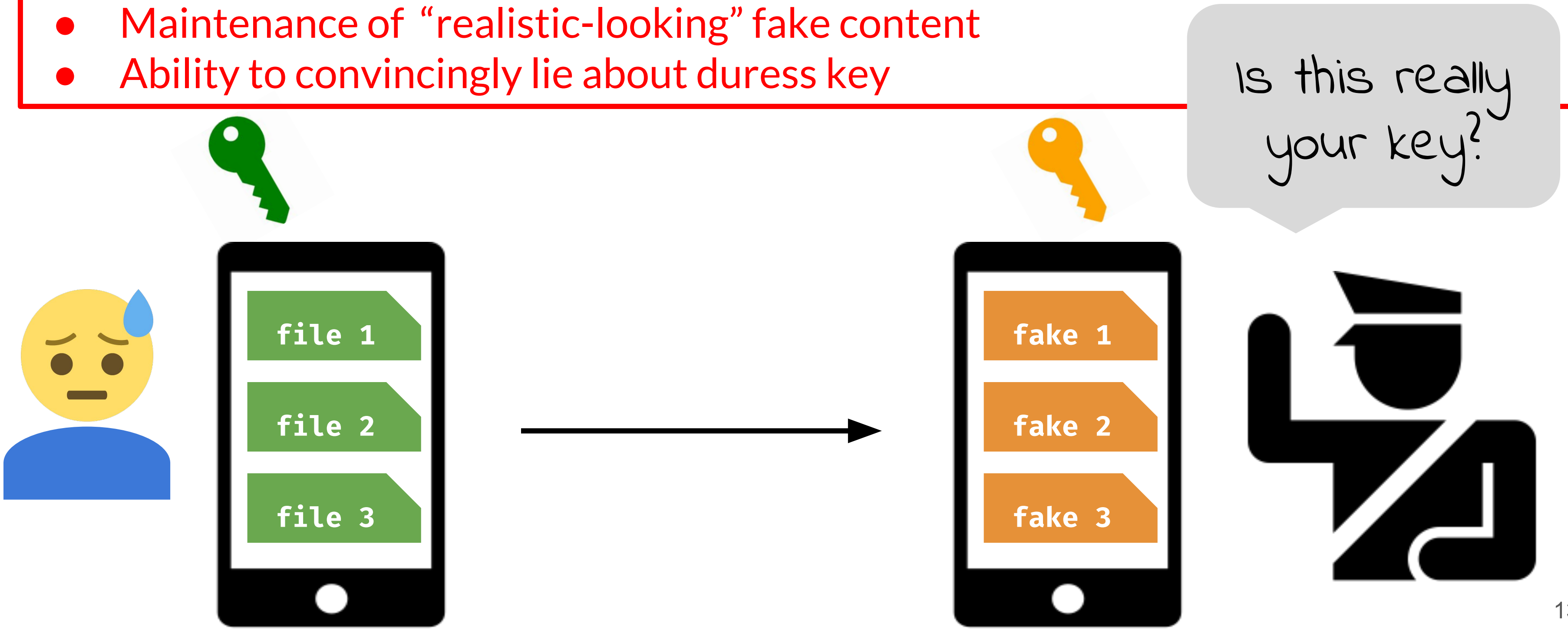


Deniable/Steganographic file systems hide files by deceiving authority

[CDNO96, ANS98, ADW97, Truecrypt]

Limitation: High usability burden where deception is inherent to security

- Maintenance of "realistic-looking" fake content
- Ability to convincingly lie about duress key



A Different Approach

1. Allow users to honestly comply at compelled access checkpoints

A Different Approach

1. Allow users to honestly comply at compelled access checkpoints

    Strawman: burner device or full wipe of device

A Different Approach

1. Allow users to honestly comply at compelled access checkpoints

    Strawman: burner device or full wipe of device

    BurnBox: selective temporary self-revocation of sensitive files

A Different Approach

1. Allow users to honestly comply at compelled access checkpoints

    Strawman: burner device or full wipe of device

    BurnBox: selective temporary self-revocation of sensitive files

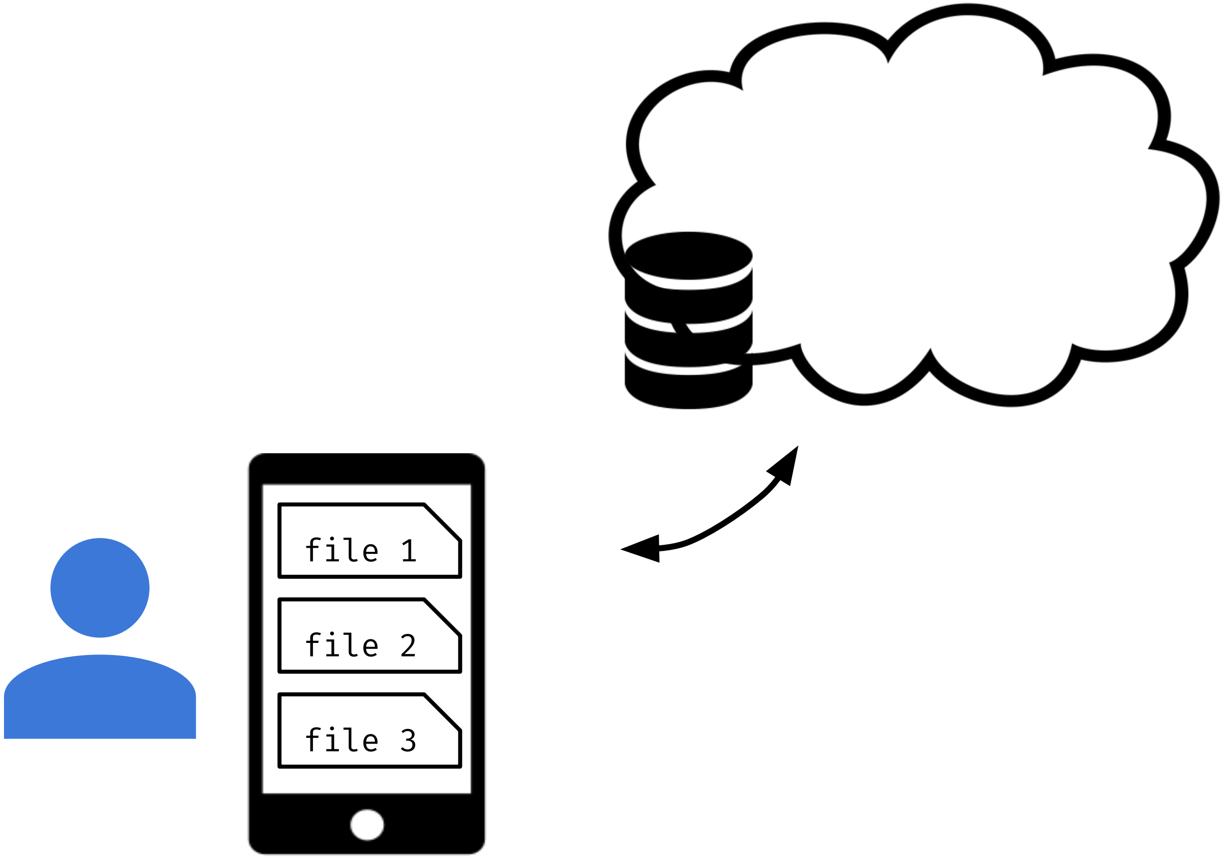
2. Designed specifically for use with the cloud

    BurnBox: secure against passive cloud adversaries

Threat Model

Untrusted Cloud Storage

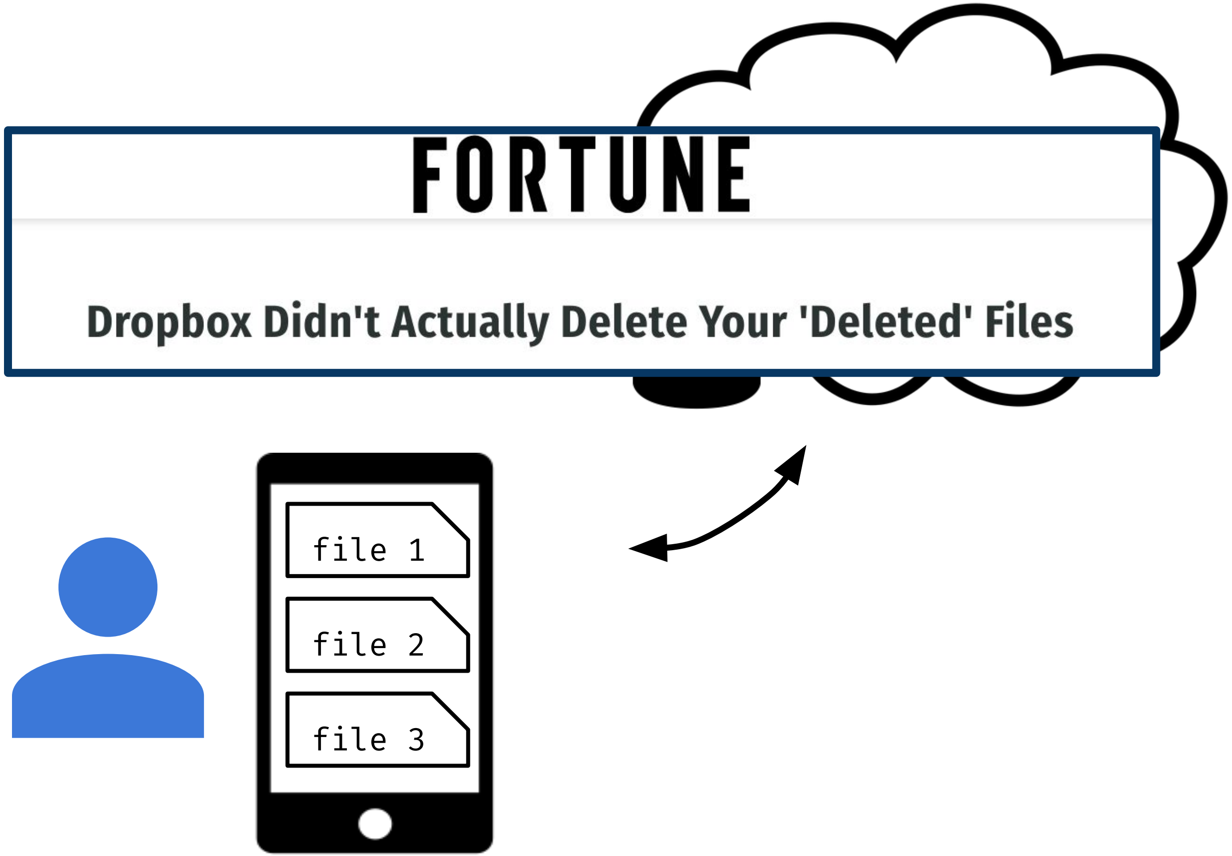
- Write-only store
- Passive attacker



Threat Model

Untrusted Cloud Storage

- Write-only store
- Passive attacker



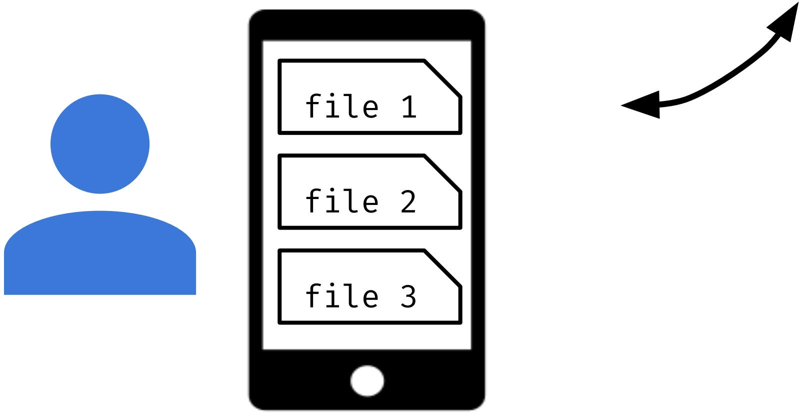
Threat Model

Untrusted Cloud Storage

- Write-only store
- Passive attacker

FORTUNE

Dropbox Didn't Actually Delete Your 'Deleted' Files



The Washington Post

Investigations

U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program

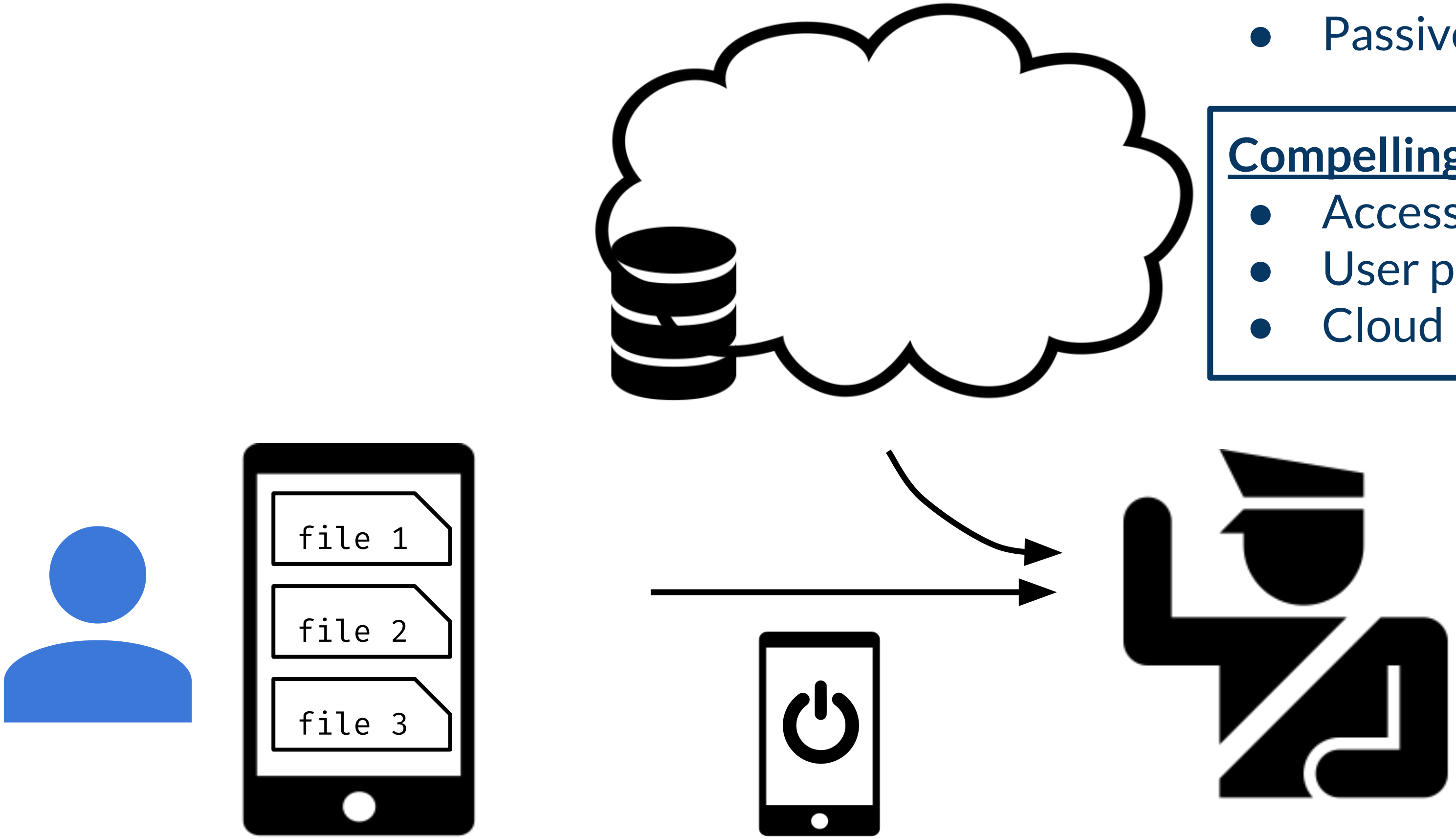
Threat Model

Untrusted Cloud Storage

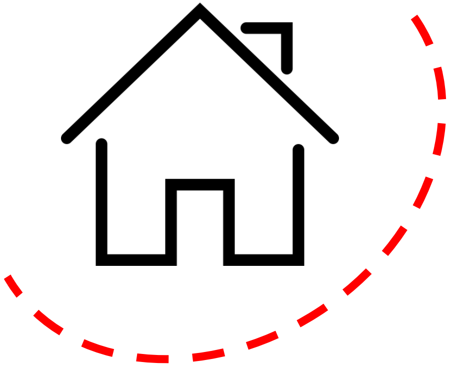
- Write-only store
- Passive attacker

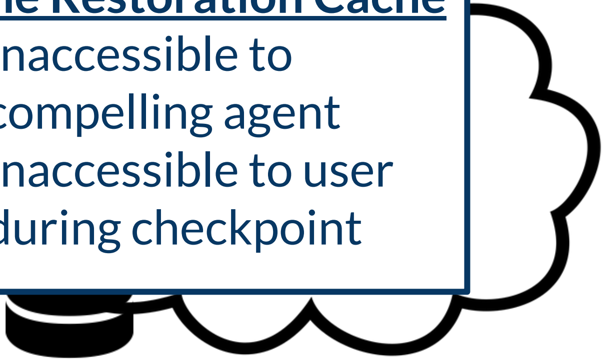
Compelling Agent

- Access to local device
- User passwords
- Cloud history



Threat Model





Offline Restoration Cache

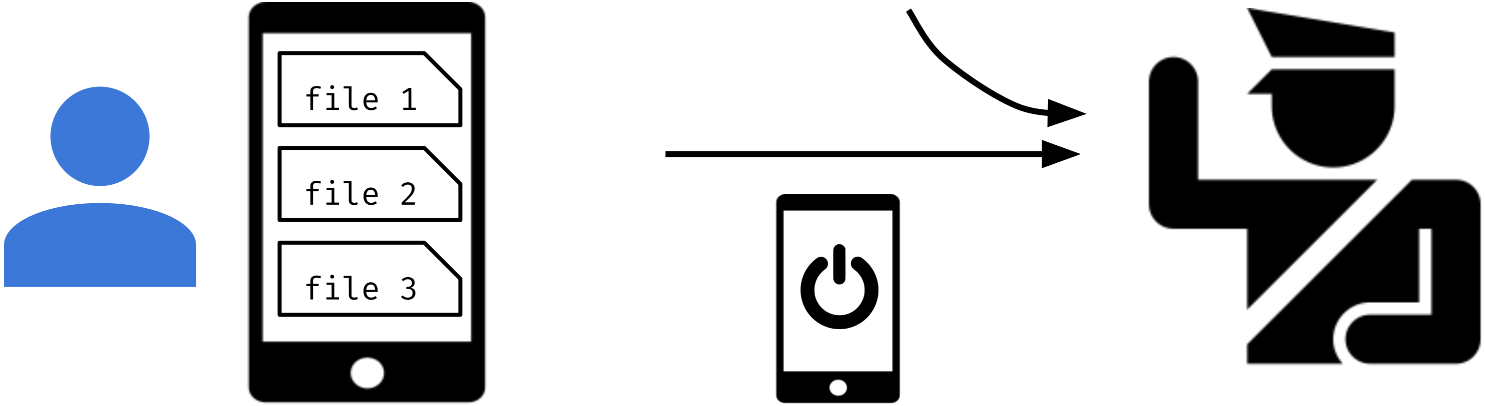
- Inaccessible to compelling agent
- Inaccessible to user during checkpoint

Untrusted Cloud Storage

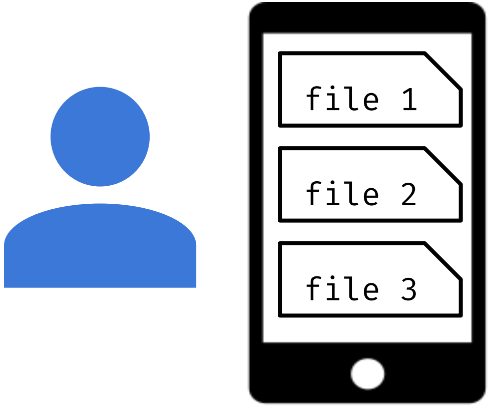
- Write-only store
- Passive attacker

Compelling Agent

- Access to local device
- User passwords
- Cloud history

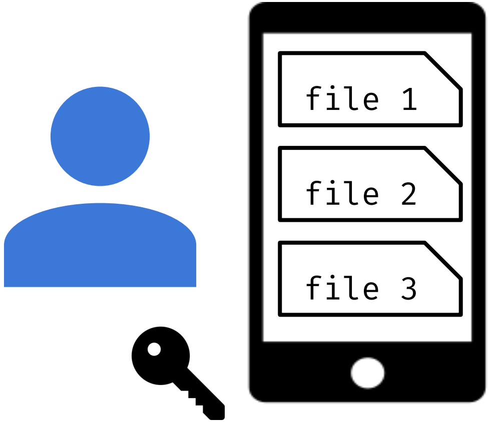


BurnBox Overview



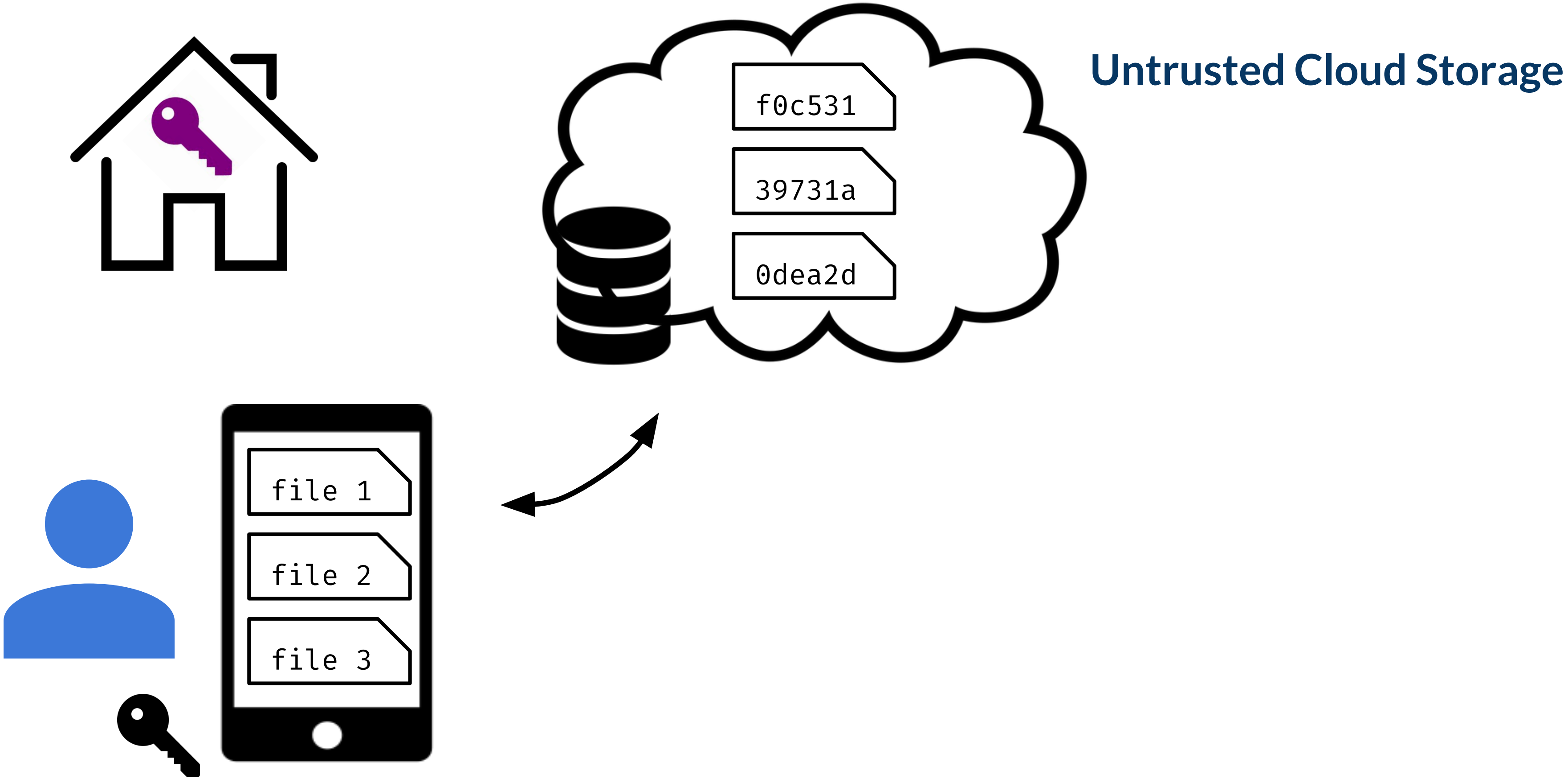
BurnBox Overview

Offline Restoration Cache



Local Device

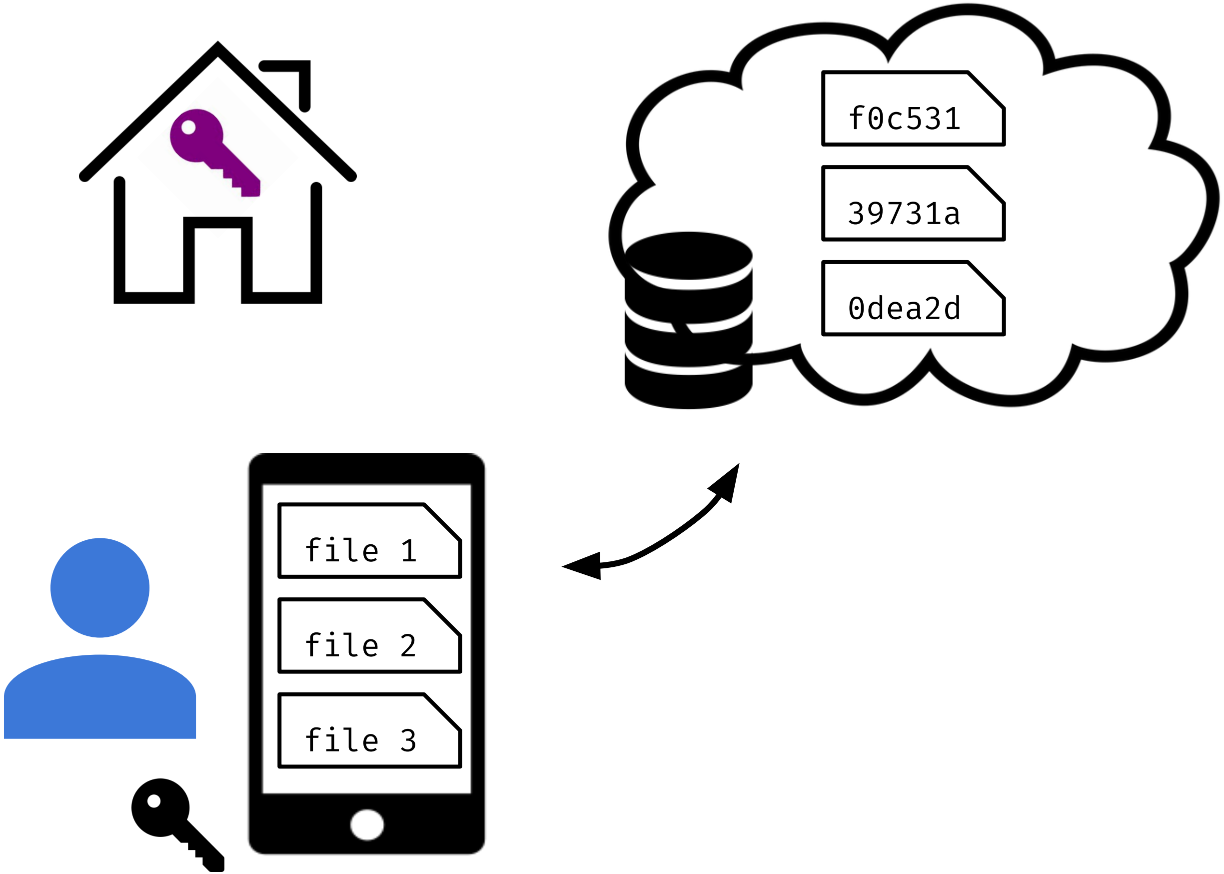
BurnBox Overview



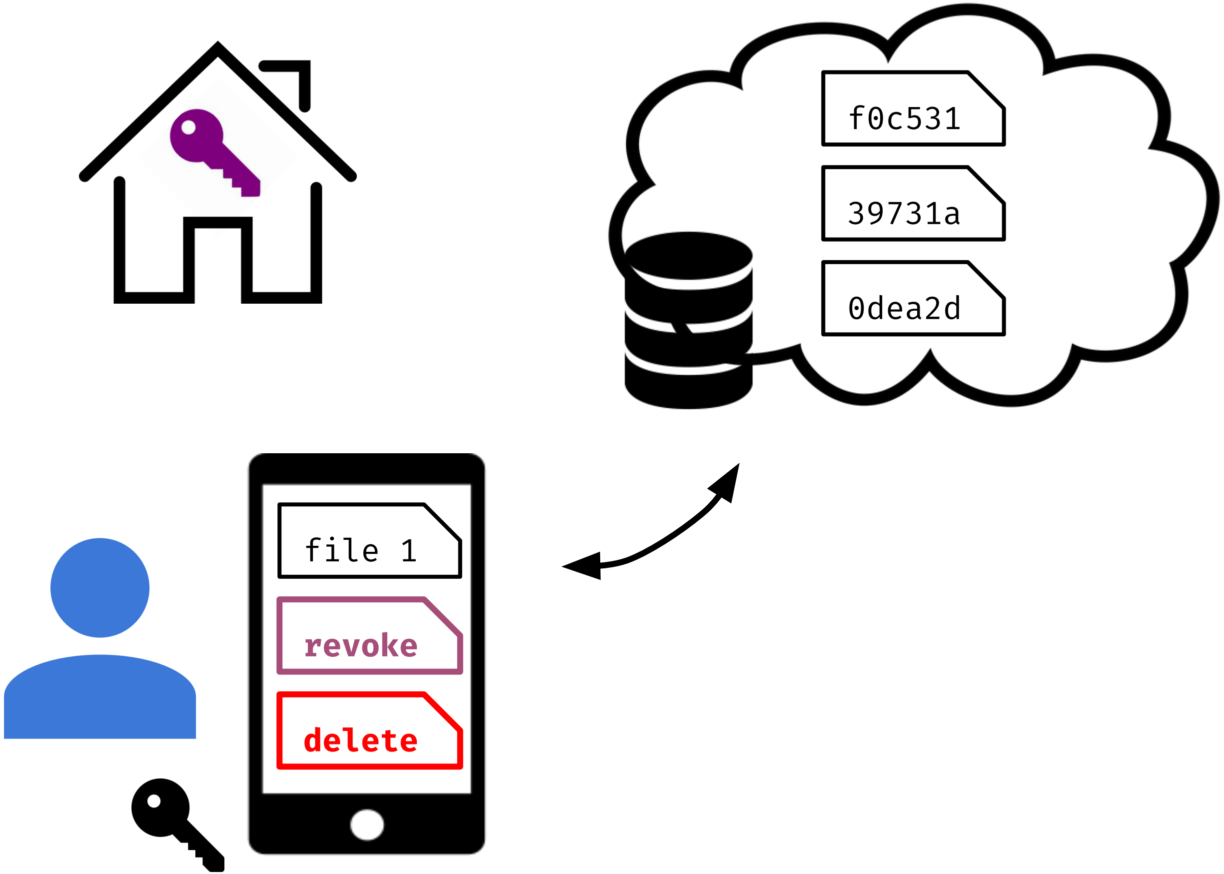
BurnBox Overview

Before Compelled Access

User selectively deletes and revokes sensitive files



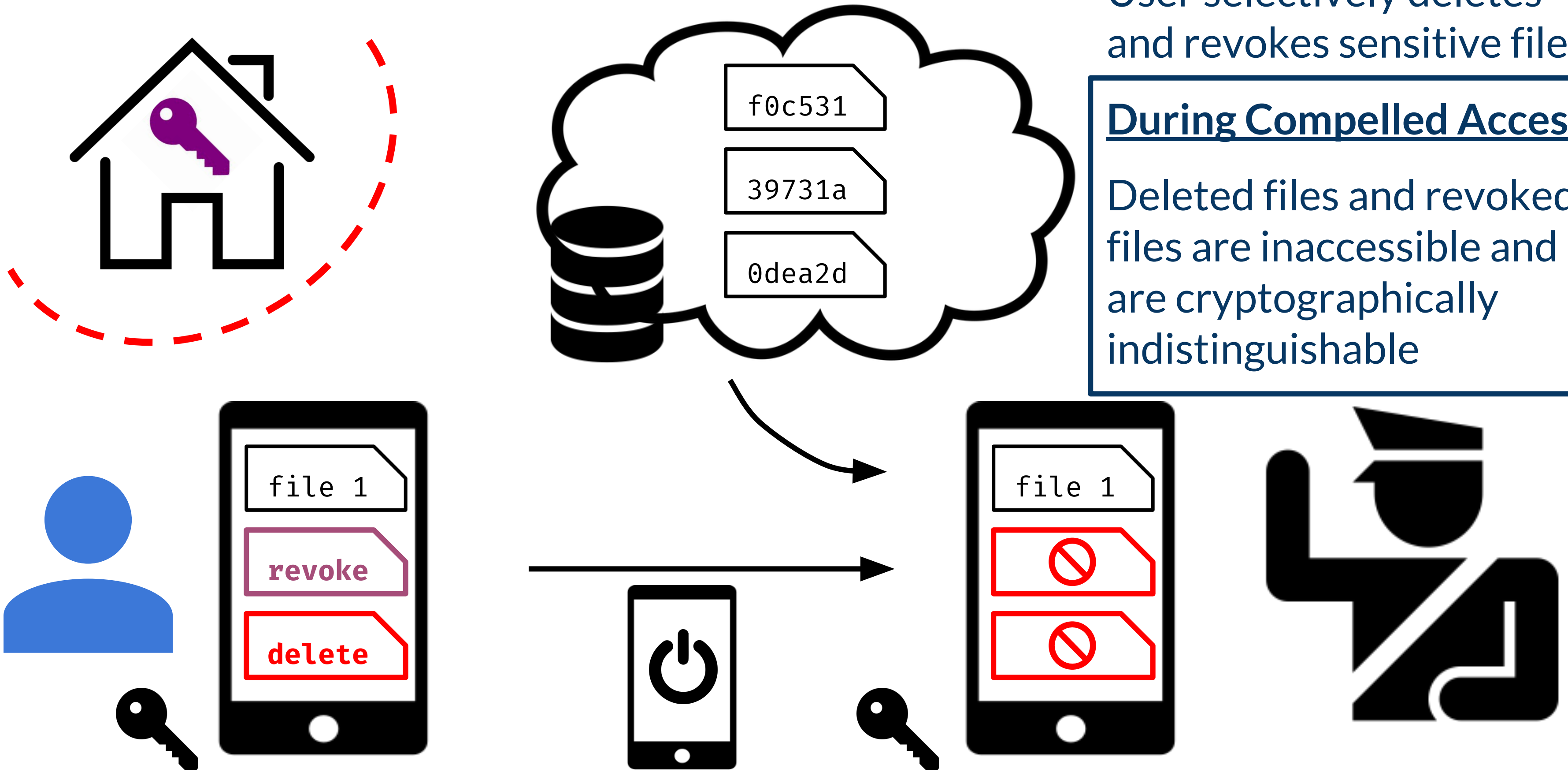
BurnBox Overview



Before Compelled Access

User selectively deletes and revokes sensitive files

BurnBox Overview



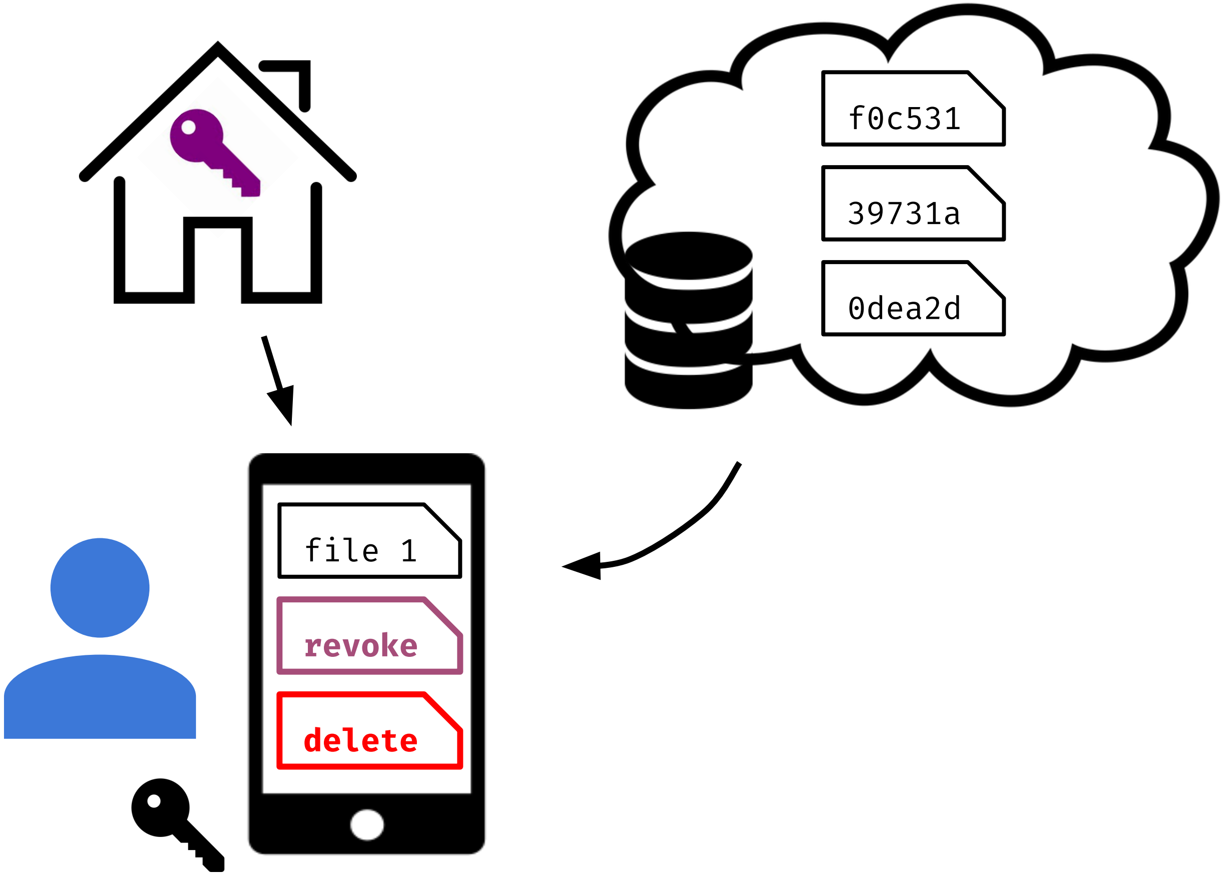
Before Compelled Access

User selectively deletes and revokes sensitive files

During Compelled Access

Deleted files and revoked files are inaccessible and are cryptographically indistinguishable

BurnBox Overview



Before Compelled Access

User selectively deletes and revokes sensitive files

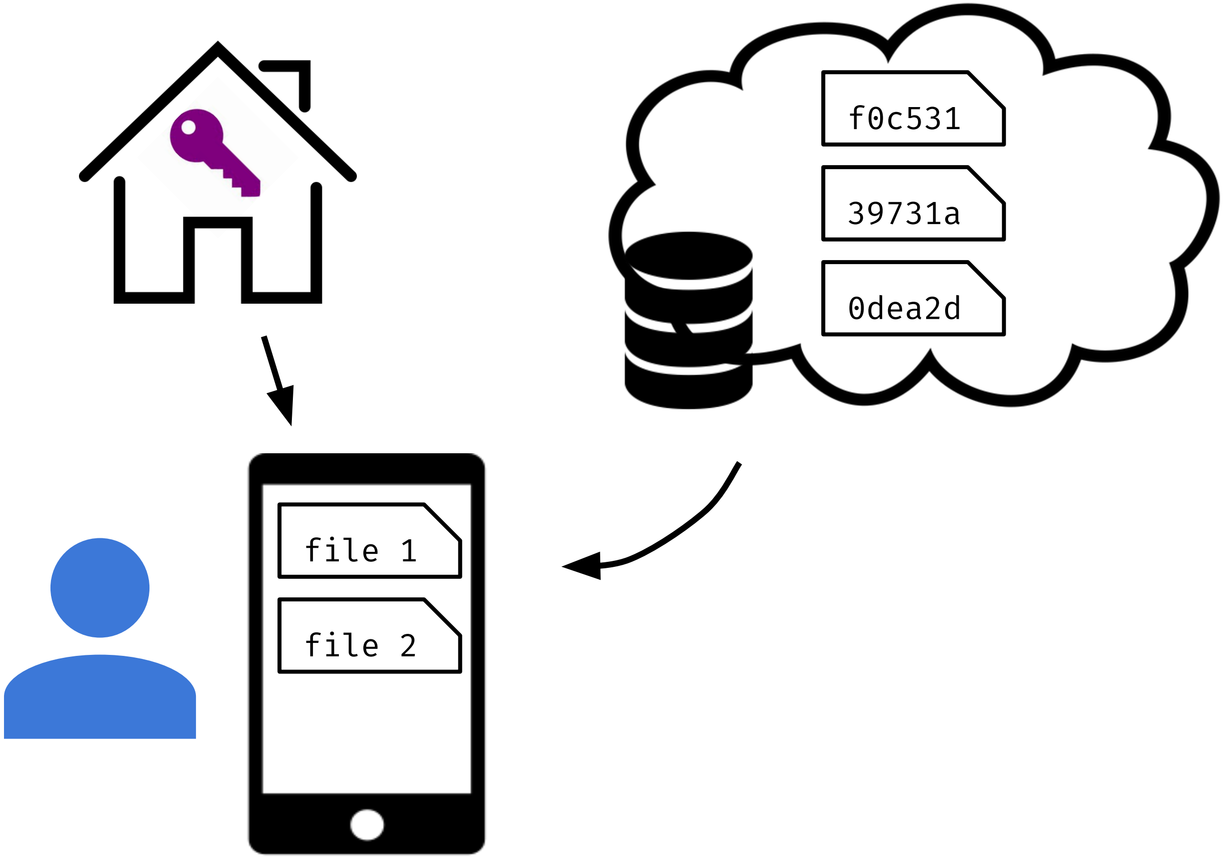
During Compelled Access

Deleted files and revoked files are inaccessible and are cryptographically indistinguishable

After Compelled Access

User restores access to revoked files with access to restoration key

BurnBox Overview



Before Compelled Access

User selectively deletes and revokes sensitive files

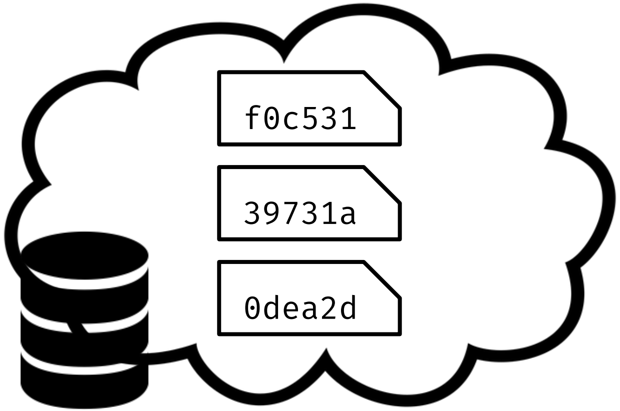
During Compelled Access

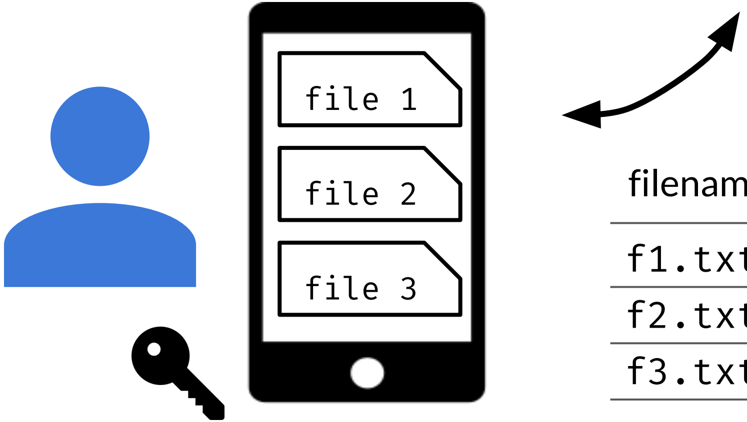
Deleted files and revoked files are inaccessible and are cryptographically indistinguishable

After Compelled Access

User restores access to revoked files with access to restoration key

Conventional client-side encryption

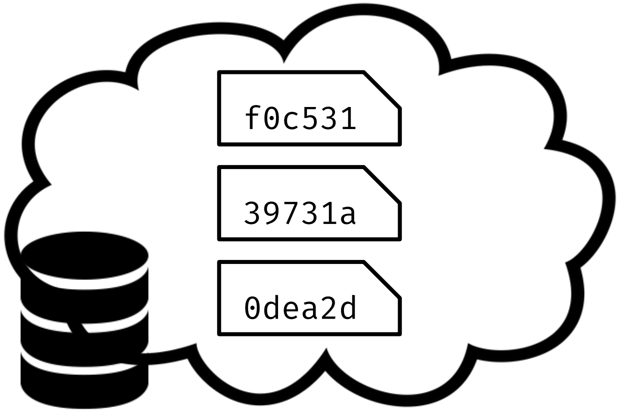


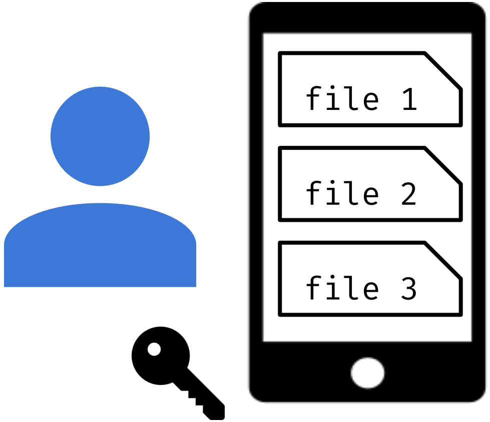


Device State

filename	encryption key	encrypted file
f1.txt	cc64c3	f0c531
f2.txt	5707dd	39731a
f3.txt	1be052	0dea2d

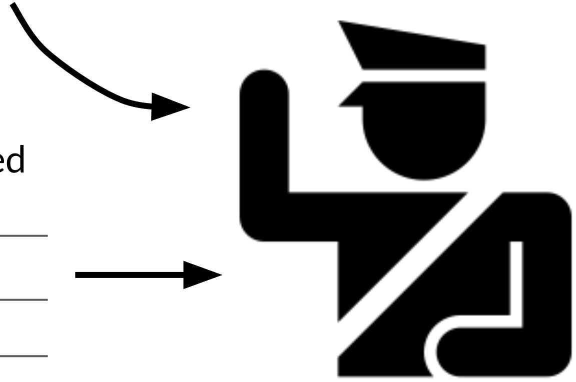
Compelled access reveals local keys



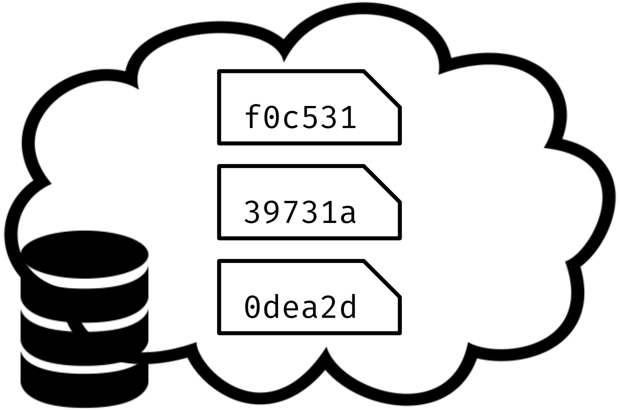


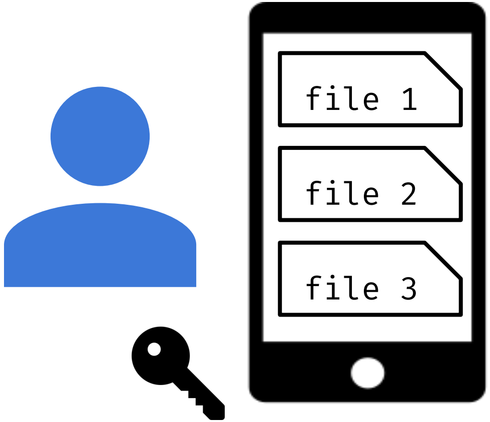
Device State

filename	encryption key	encrypted file
f1.txt	cc64c3	f0c531
f2.txt	5707dd	39731a
f3.txt	1be052	0dea2d



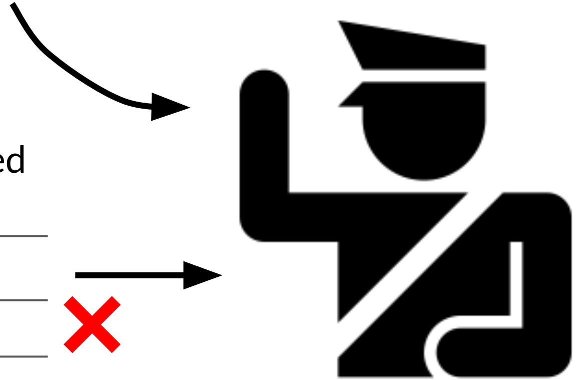
Delete rows of sensitive files





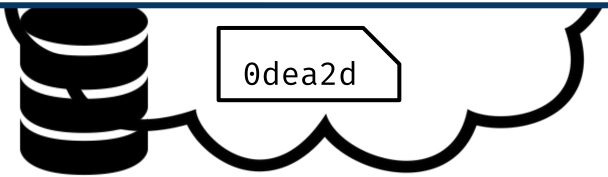
Device State

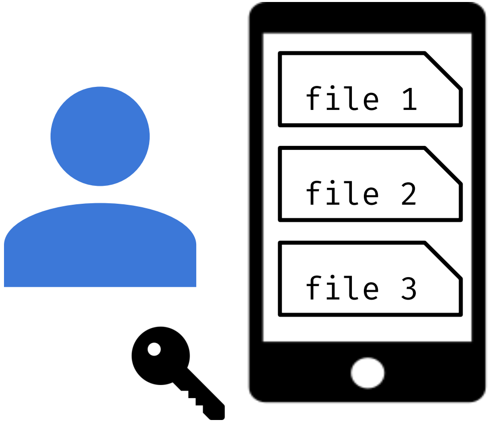
filename	encryption key	encrypted file
f1.txt	cc64c3	f0c531
f2.txt	5707dd	39731a
f3.txt	1be052	0dea2d



Delete rows of sensitive files

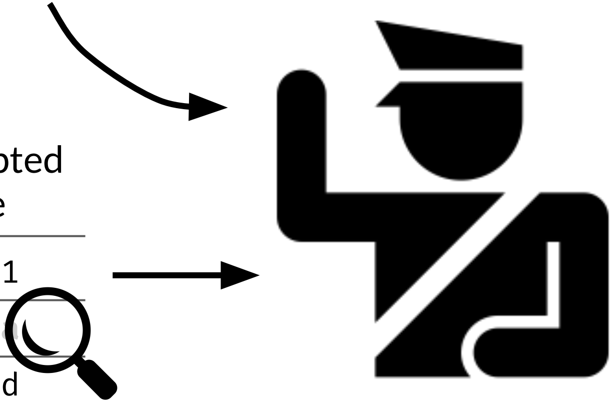
Problem 1: How to support revocation?
Problem 2: Secure deletion of persistent state is ***hard***.





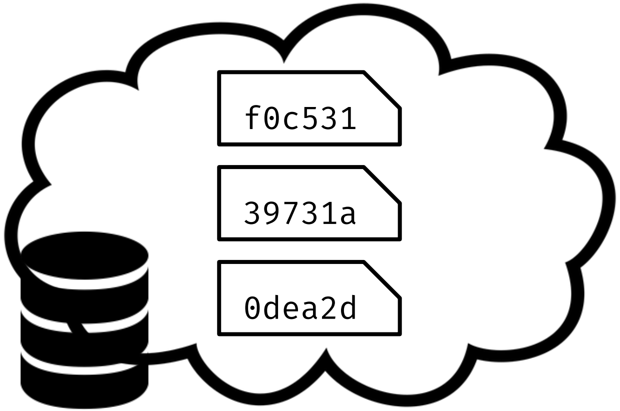
Device State

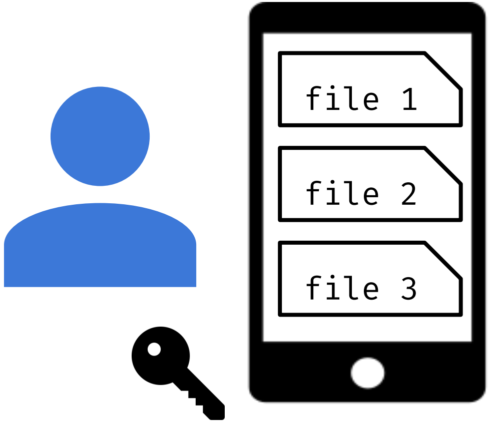
filename	encryption key	encrypted file
f1.txt	cc64c3	f0c531
f2.txt	5707dd	39731
f3.txt	1be052	0dea2d



Forensic analysis

Revocation: use public-key encryption

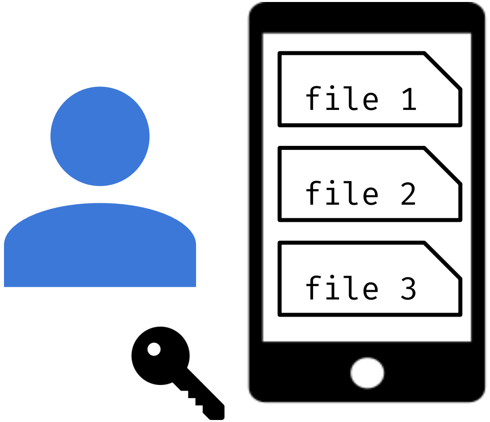




Device State

filename	encryption key	encrypted file	restoration ciphertext
f1.txt	cc64c3	f0c531	**E(pk**,cc64c3**)**
f2.txt	5707dd	39731a	**E(pk**,39731a**)**
f3.txt	1be052	0dea2d	**E(pk**,1be052**)**

Revocation: use public-key encryption

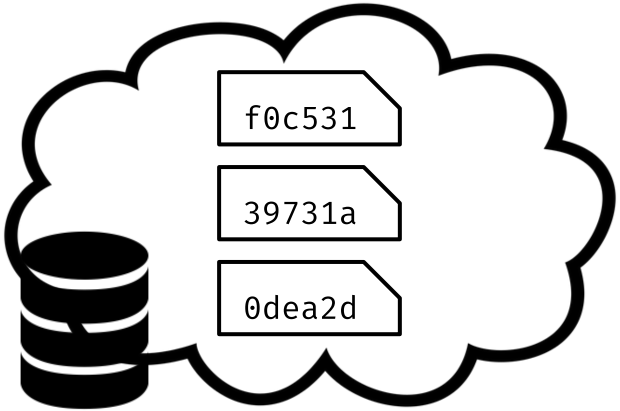


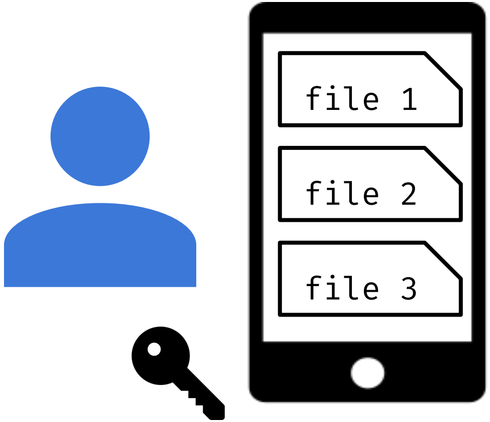
Device State

filename	encryption key	encrypted file	restoration ciphertext
f1.txt	cc64c3	f0c531	E(pk,cc64c3)
~~f2.txt~~	~~5707dd~~	~~39731a~~	E(pk,39731a)
f3.txt	1be052	0dea2d	E(pk,1be052)

Revoke

Revocation: use public-key encryption





Device State

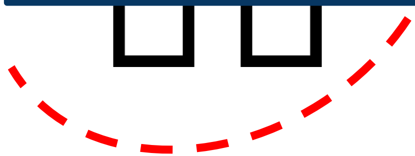
filename	encryption key	encrypted file	restoration ciphertext
f1.txt	cc64c3	f0c531	E(pk,cc64c3)
f2.txt	5707dd	39731a	E(pk,39731a)
f3.txt	1be052	0dea2d	E(pk,000000)

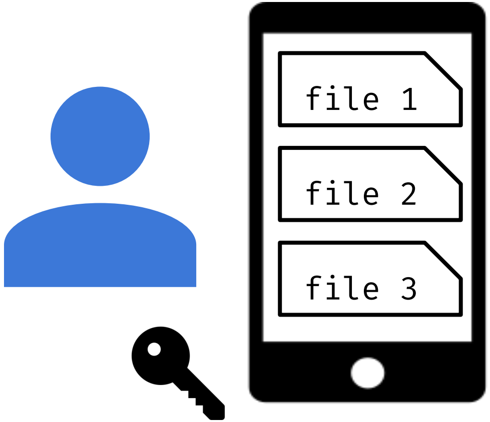
Revoke

Delete

Revocation: use public-key encryption

Problem 1: How to support revocation? ✓
Problem 2: Secure deletion of persistent state is ***hard***.





Device State

filename	encryption key	encrypted file	restoration ciphertext
f1.txt	cc64c3	f0c531	E(pk,cc64c3)
f2.txt	5707dd	39731a	E(pk,39731a)
f3.txt	1be052	0dea2d	E(pk,000000)

Revoke

Delete

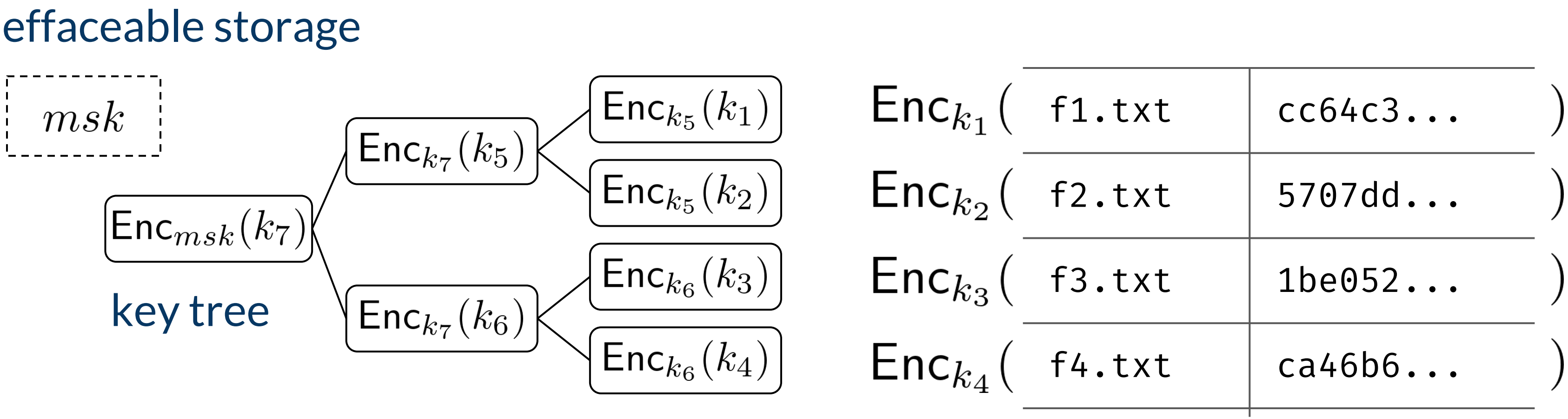
Erasable Index

- File keys stored in append-only table

$\mathsf{Enc}_{k_1}($	f1.txt	cc64c3...	$)$
$\mathsf{Enc}_{k_2}($	f2.txt	5707dd...	$)$
$\mathsf{Enc}_{k_3}($	f3.txt	1be052...	$)$
$\mathsf{Enc}_{k_4}($	f4.txt	ca46b6...	$)$

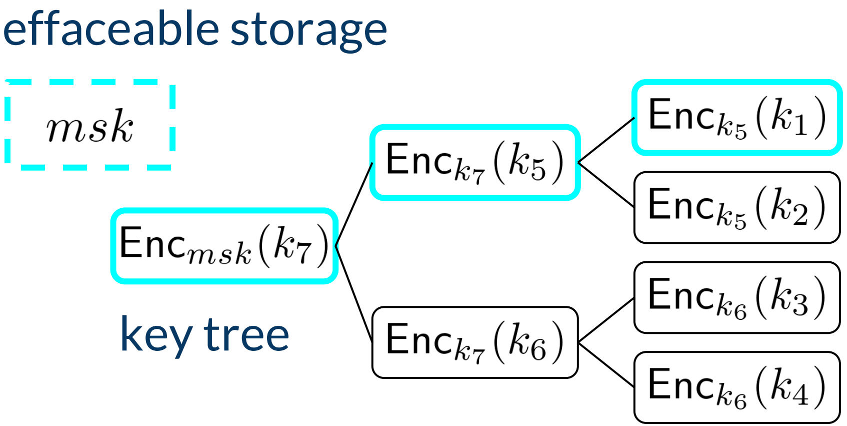
Erasable Index

- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
  - Trusted hardware assumed to manage small "effaceable" storage
  - E.g., TPM, iOS/Android keystore APIs



Erasable Index

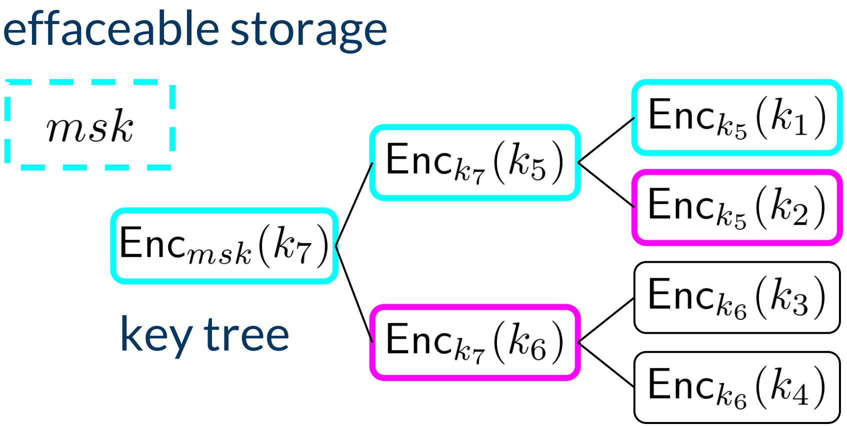
- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
    - Trusted hardware assumed to manage small “effaceable” storage
    - E.g., TPM, iOS/Android keystore APIs



$\mathsf{Enc}_{k_1}($	`f1.txt`	`cc64c3...`	$)$
$\mathsf{Enc}_{k_2}($	`f2.txt`	`5707dd...`	$)$
$\mathsf{Enc}_{k_3}($	`f3.txt`	`1be052...`	$)$
$\mathsf{Enc}_{k_4}($	`f4.txt`	`ca46b6...`	$)$

Erasable Index

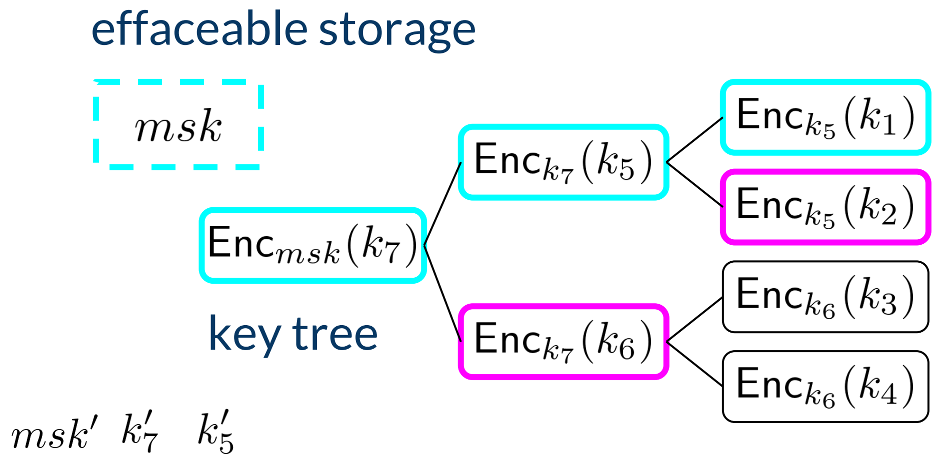
- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
  - Trusted hardware assumed to manage small “effaceable” storage
  - E.g., TPM, iOS/Android keystore APIs



$\mathsf{Enc}_{k_1}($	`f1.txt`	`cc64c3...` $)$
$\mathsf{Enc}_{k_2}($	`f2.txt`	`5707dd...` $)$
$\mathsf{Enc}_{k_3}($	`f3.txt`	`1be052...` $)$
$\mathsf{Enc}_{k_4}($	`f4.txt`	`ca46b6...` $)$

Erasable Index

- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
  - Trusted hardware assumed to manage small “effaceable” storage
  - E.g., TPM, iOS/Android keystore APIs

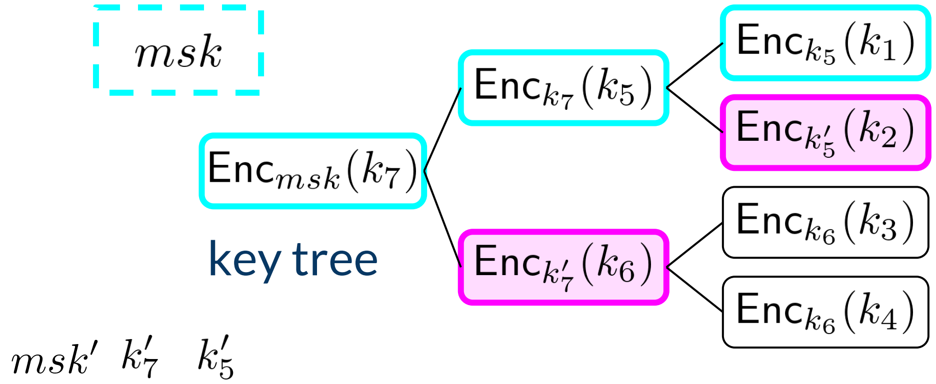


$\mathsf{Enc}_{k_1}($	f1.txt	cc64c3... $)$
$\mathsf{Enc}_{k_2}($	f2.txt	5707dd... $)$
$\mathsf{Enc}_{k_3}($	f3.txt	1be052... $)$
$\mathsf{Enc}_{k_4}($	f4.txt	ca46b6... $)$

Erasable Index

- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
  - Trusted hardware assumed to manage small "effaceable" storage
  - E.g., TPM, iOS/Android keystore APIs

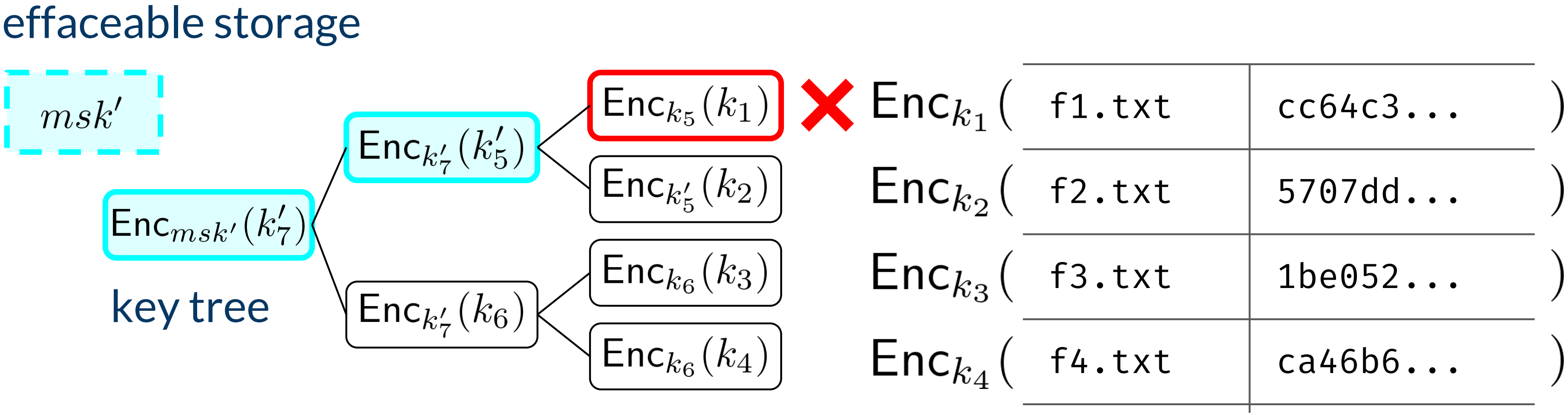
effaceable storage



$\mathsf{Enc}_{k_1}($	`f1.txt`	`cc64c3...` $)$
$\mathsf{Enc}_{k_2}($	`f2.txt`	`5707dd...` $)$
$\mathsf{Enc}_{k_3}($	`f3.txt`	`1be052...` $)$
$\mathsf{Enc}_{k_4}($	`f4.txt`	`ca46b6...` $)$

Erasable Index

- File keys stored in append-only table
- Secure deletion of row keys with trusted hardware [RRBC13]
  - Trusted hardware assumed to manage small "effaceable" storage
  - E.g., TPM, iOS/Android keystore APIs



Efficiency and other approaches?

Erasable index uses:

- Storage on the order of number of files
- Linear time search by filename

In practice, this is actually fine

Related asymptotically better approaches not secure against threat model

- Puncturable pseudorandom functions [GMM86]
- History-independent data structures [NT01]

Security Analysis

- Provide formal security models
- Limit leakage to well-specified access pattern history
    - Pseudonymous operation history

Adversary observing:

Cloud communication history
Encrypted cloud contents
Erasable index on local device

=

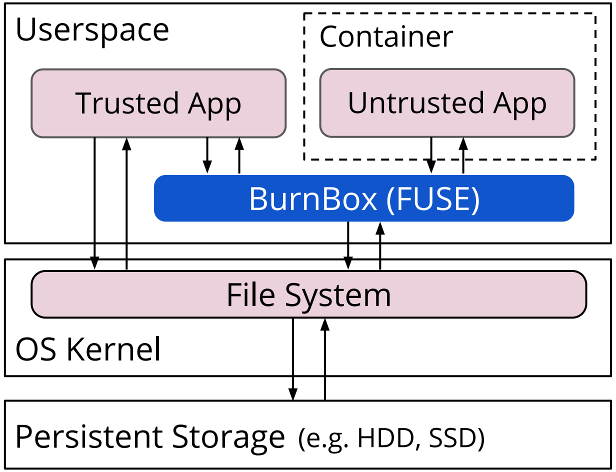
Pseudonymous operation history
E.g.,
Add file A at 1:00
Access file A at 4:30

Open question: Inference attacks on file accesses?

[CGPR15,NKW15]

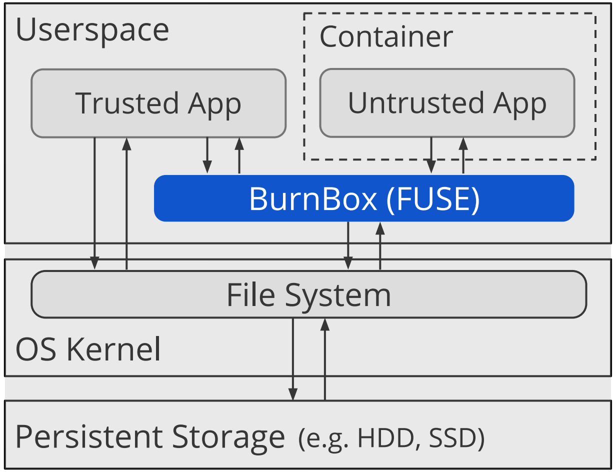
Prototype

- Implemented as file system in userspace (FUSE)
  - Available at github.com/mhmughees/burnbox
  - About as efficient as standard client-side encryption



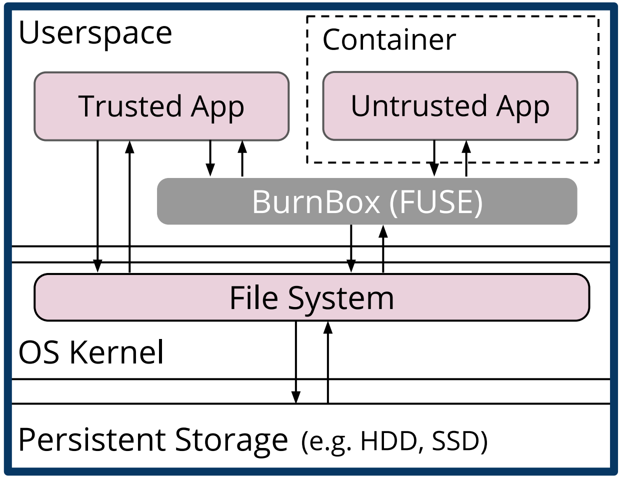
Prototype

- Implemented as file system in userspace (FUSE)
  - Available at github.com/mhmughees/burnbox
  - About as efficient as standard client-side encryption



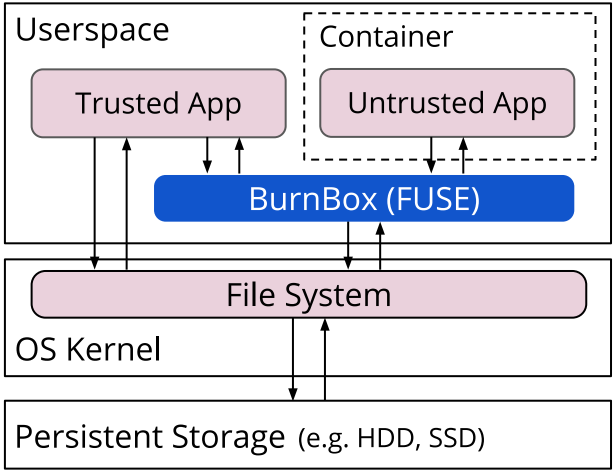
Prototype

- Implemented as file system in userspace (FUSE)
  - Available at github.com/mhmughees/burnbox
  - About as efficient as standard client-side encryption



Prototype

- Implemented as file system in userspace (FUSE)
  - Available at github.com/mhmughees/burnbox
  - About as efficient as standard client-side encryption
- Best effort to address application and OS leakage [CHKGKS08,DLJKSXSW12]
  - Memory-locked pages
  - Containers for untrusted applications
  - Guidelines for off-the-shelf OS configurations



Contributions

- BurnBox: Cloud storage secure in compelled access setting
  - Allow users to honestly comply with authorities while preserving confidentiality
  - Secure deletion: permanently delete files
  - Temporary revocation: self-revoke access to files temporarily
- Formal compelled access security notions and analysis
- Proof-of-concept prototype
  - github.com/mhmughees/burnbox