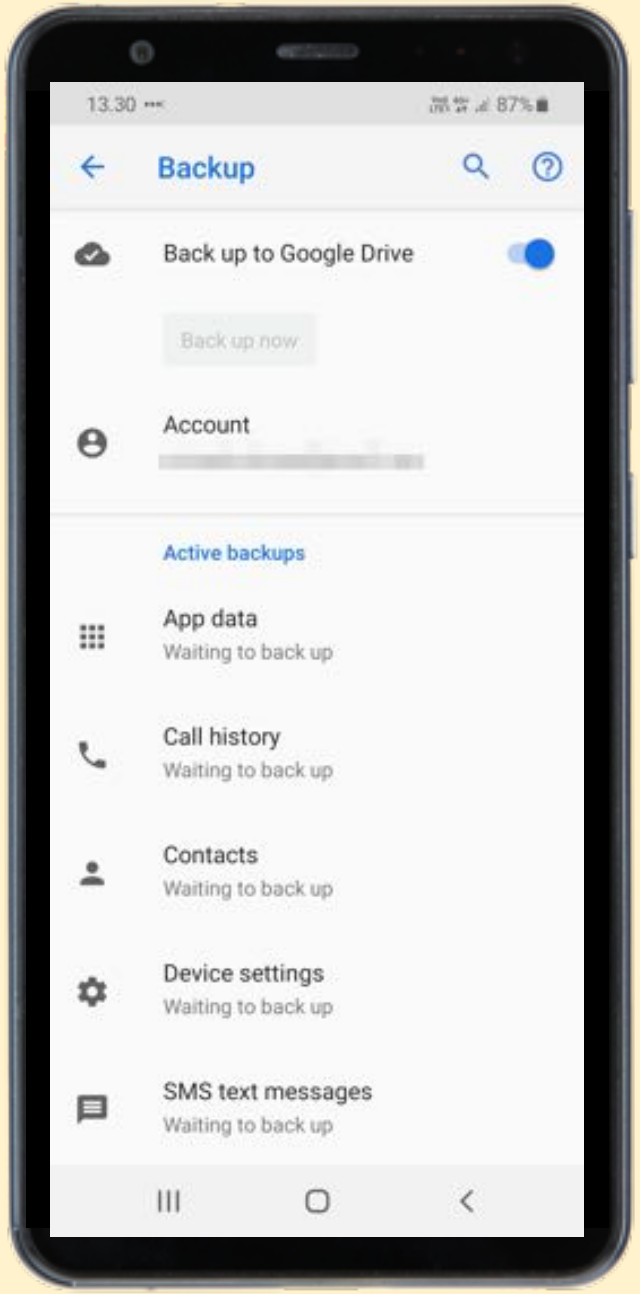
SafetyPin: Encrypted Backups with Human-Memorable Secrets

Emma Dauterman
UC Berkeley

Henry Corrigan-Gibbs
MIT CSAIL

David Mazières
Stanford

Appeared at OSDI 2020



Goal: Back up your data on Google's servers...

...without Google seeing your data

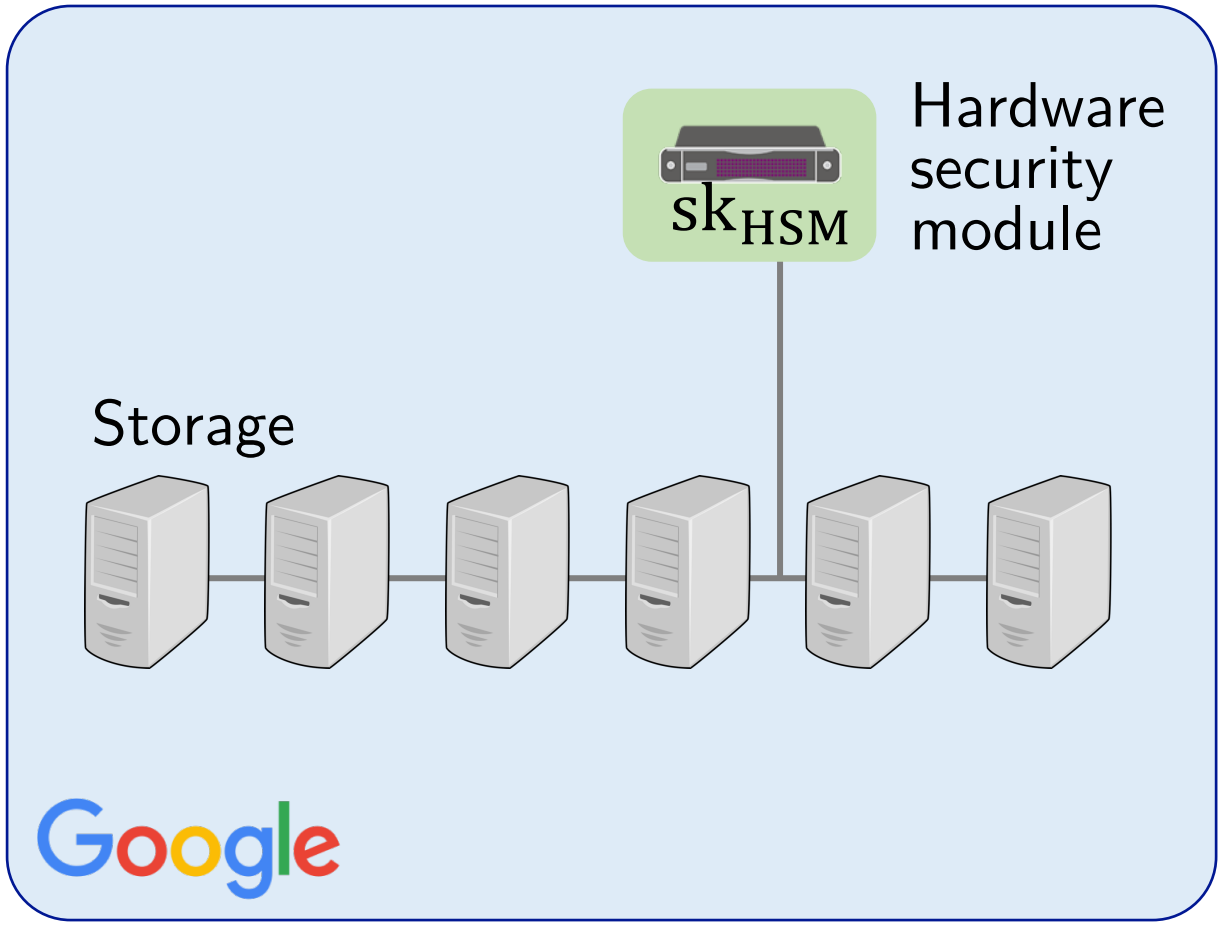
[Think: Intelligence or police agency in your surveillance state of choice.]

Mobile-device backups today [simplified]

Similar ideas used at Apple, Google, Signal, ...

1. Backup

2. Recovery



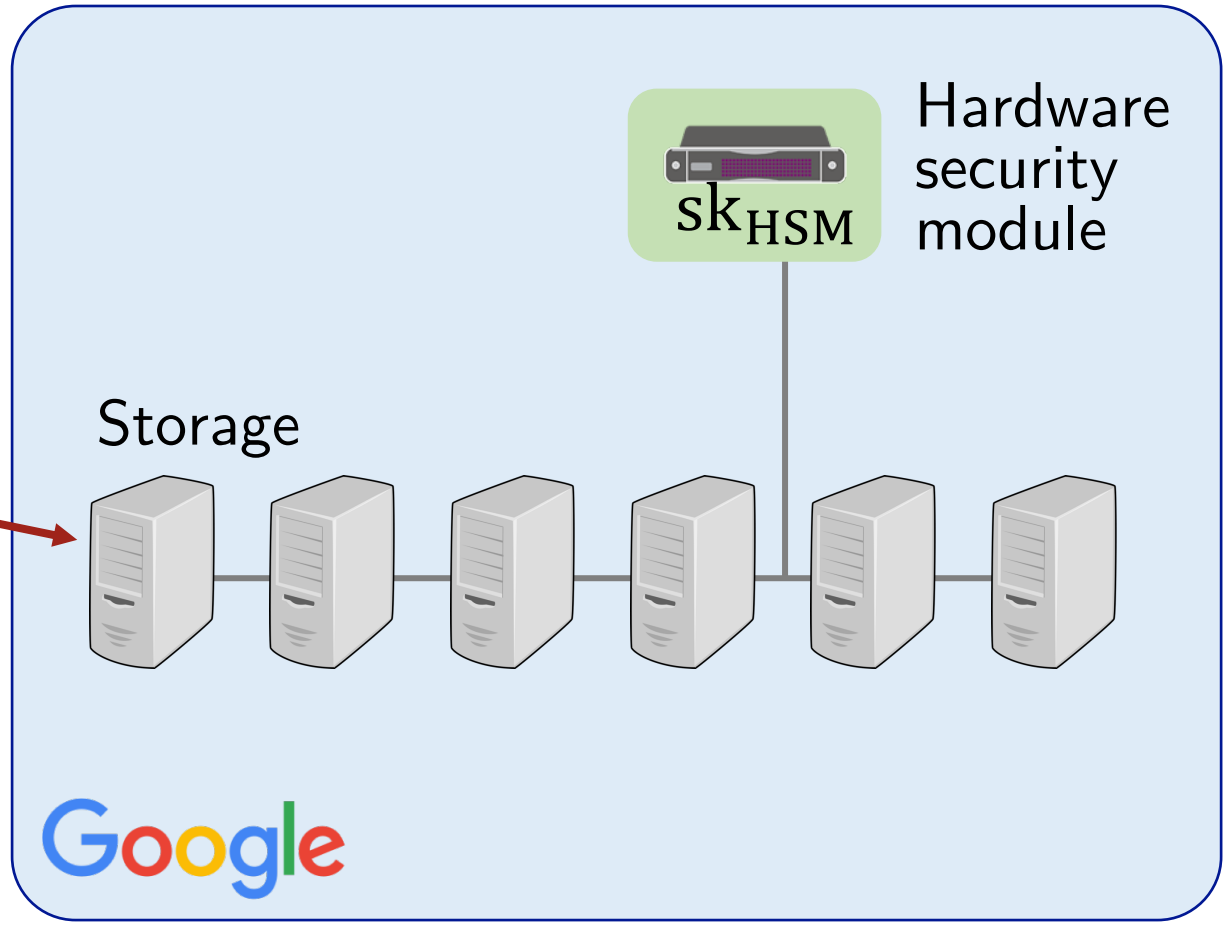
Mobile-device backups today [simplified]

1. Backup

$\mathrm{Enc}(\mathrm{pk}_{\mathrm{HSM}}, \mathrm{PIN}||k_{\mathrm{data}})$

$\mathrm{AES}(k_{\mathrm{data}}, \mathrm{data})$

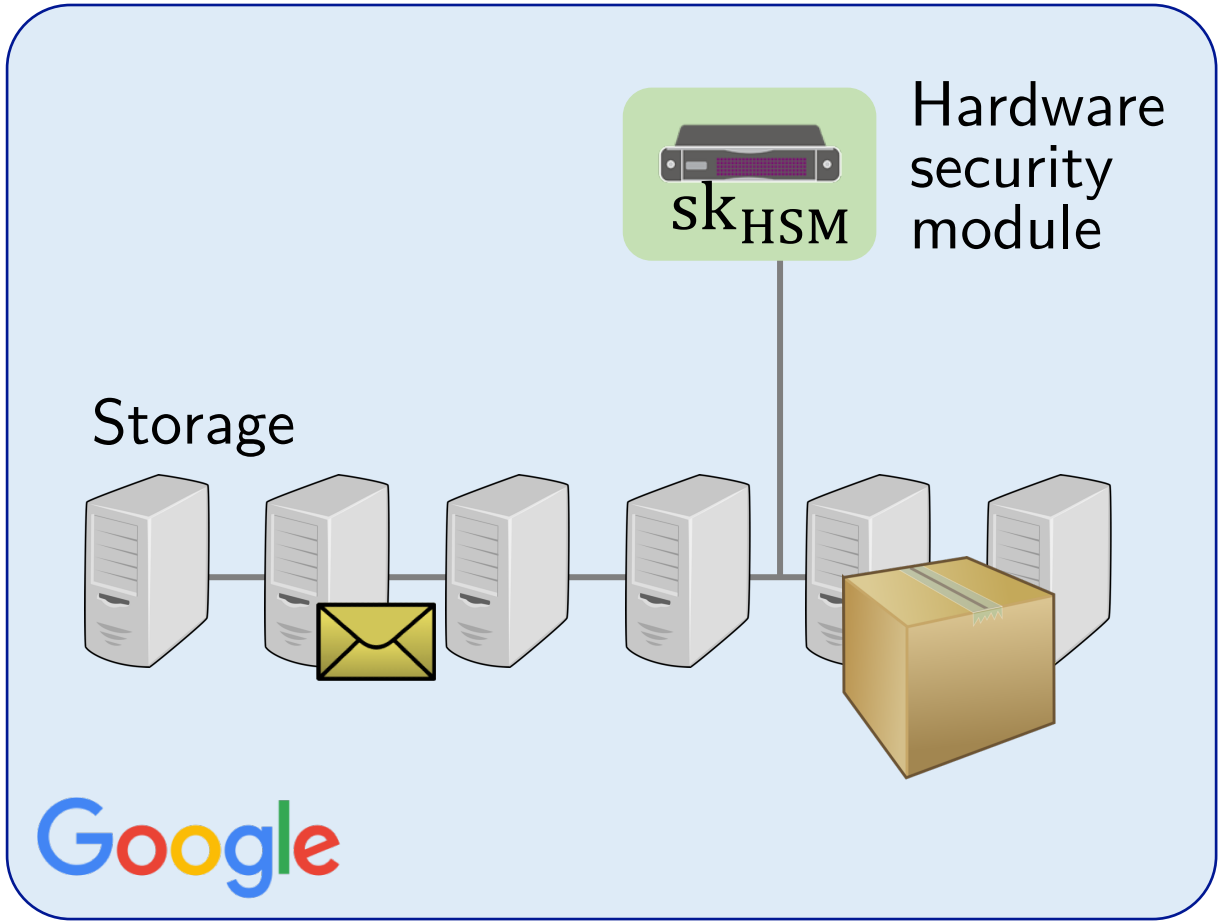
2. Recovery



Mobile-device backups today [simplified]

1. Backup

2. Recovery

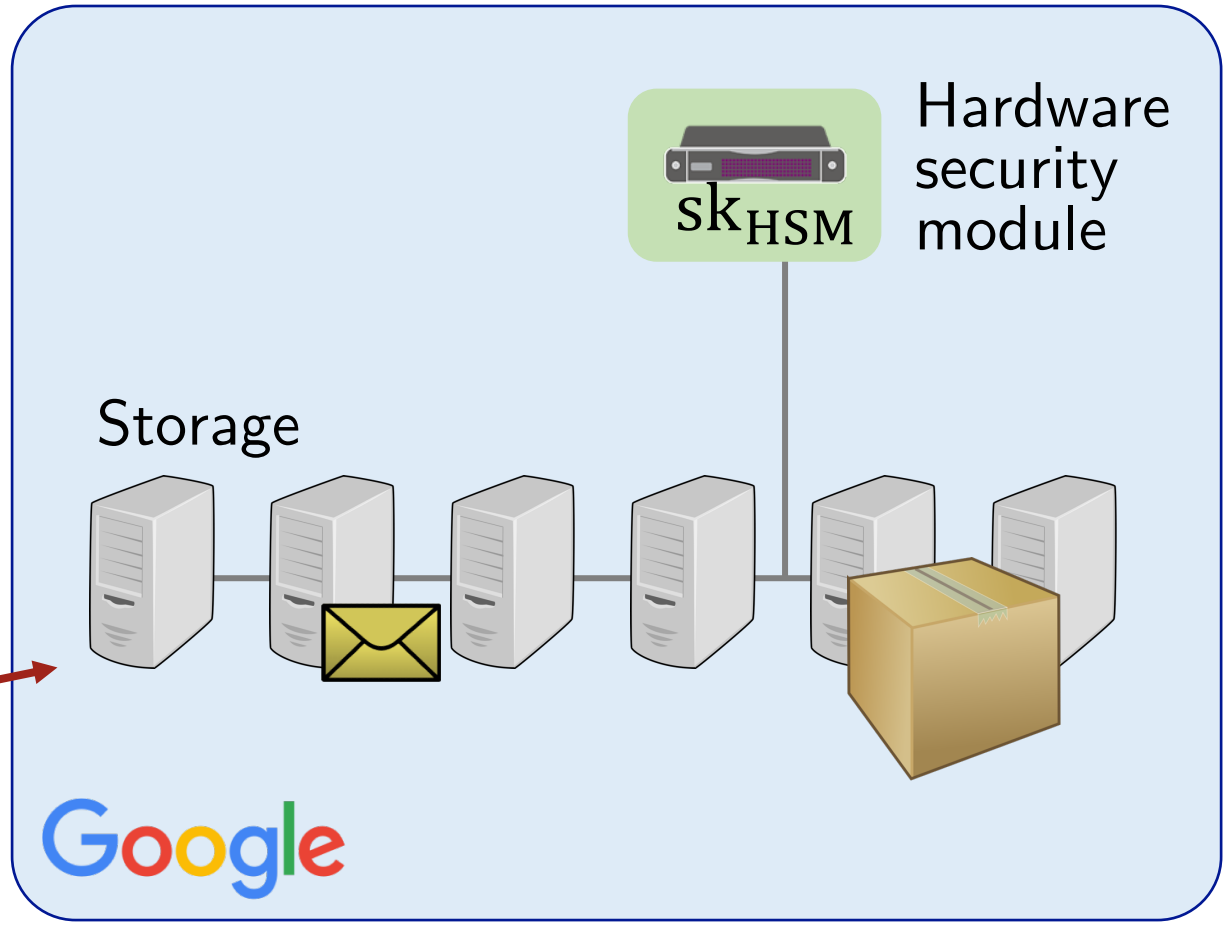


Mobile-device backups today [simplified]

1. Backup

2. **Recovery**

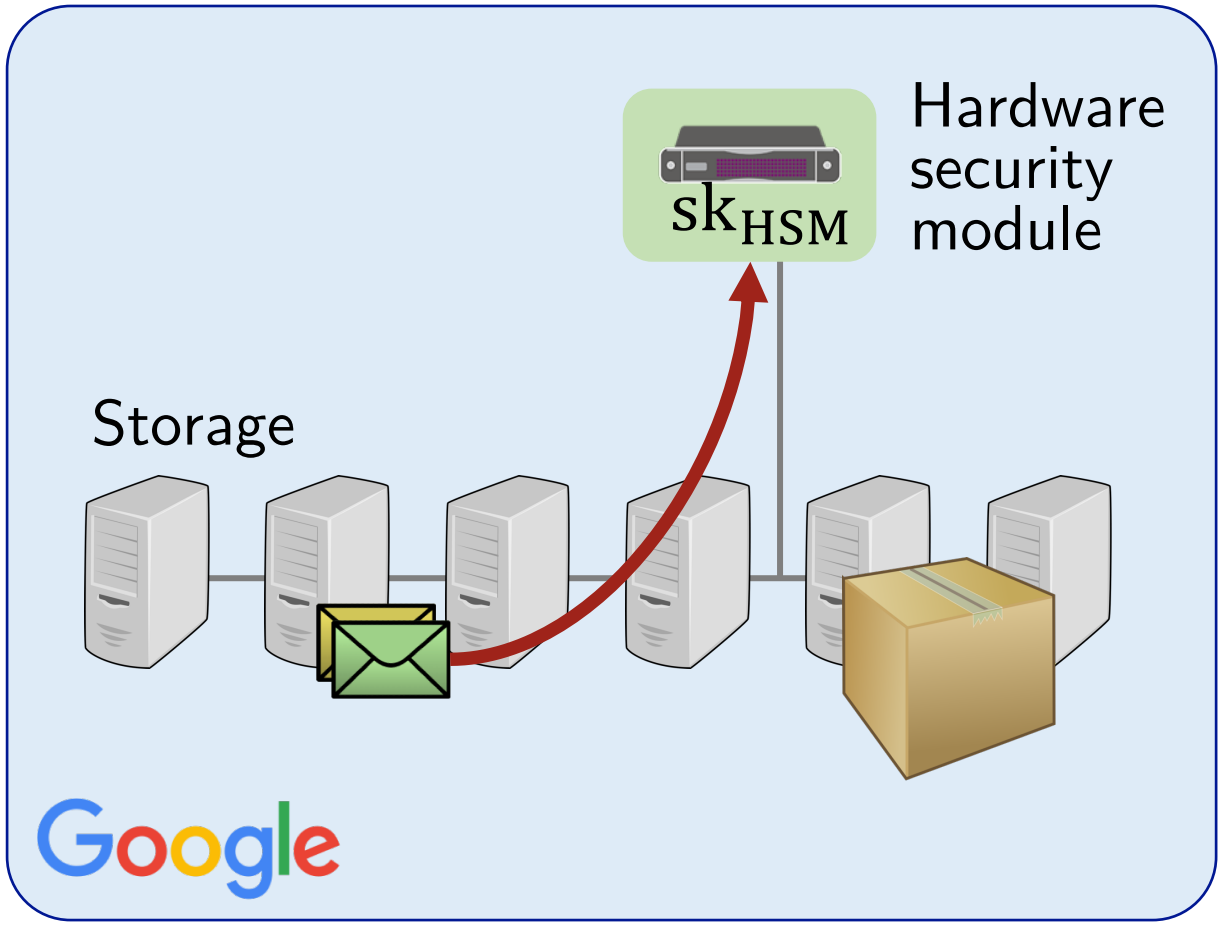
$\mathrm{Enc}(\mathrm{pk}_{\mathrm{HSM}}, \mathrm{PIN})$



Mobile-device backups today [simplified]

1. Backup

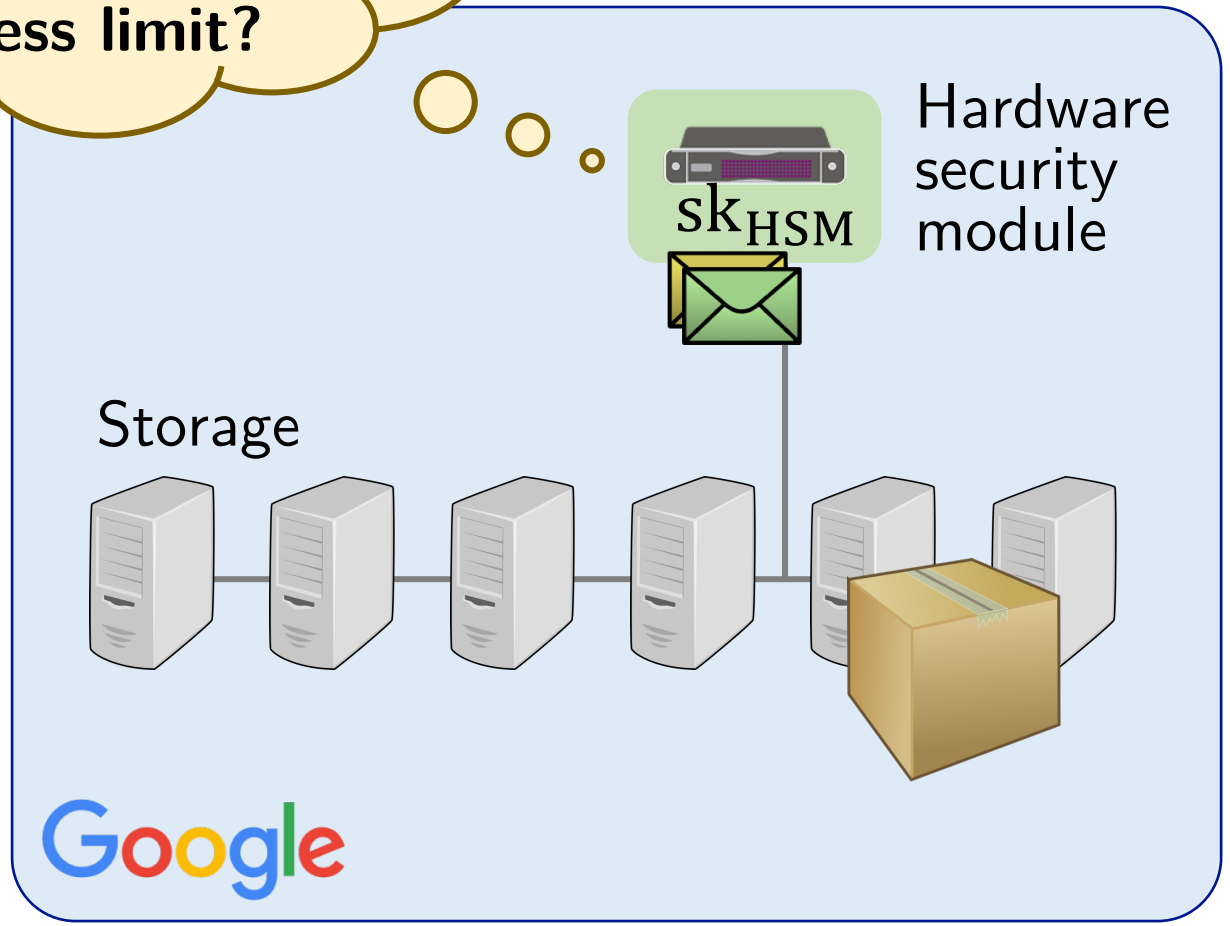
2. **Recovery**



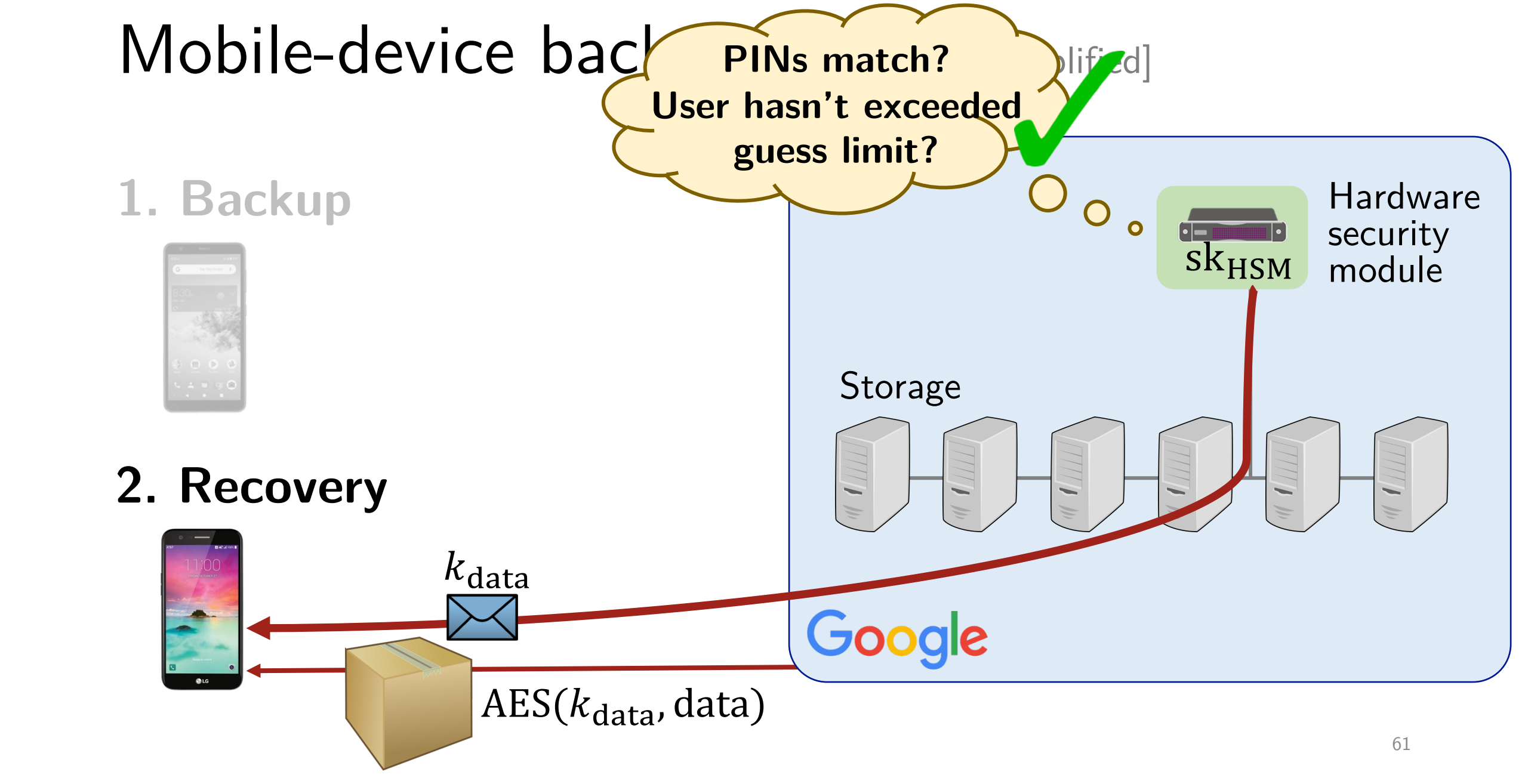
Mobile-device backup [simplified]

1. Backup

2. **Recovery**



Mobile-device backup [simplified]

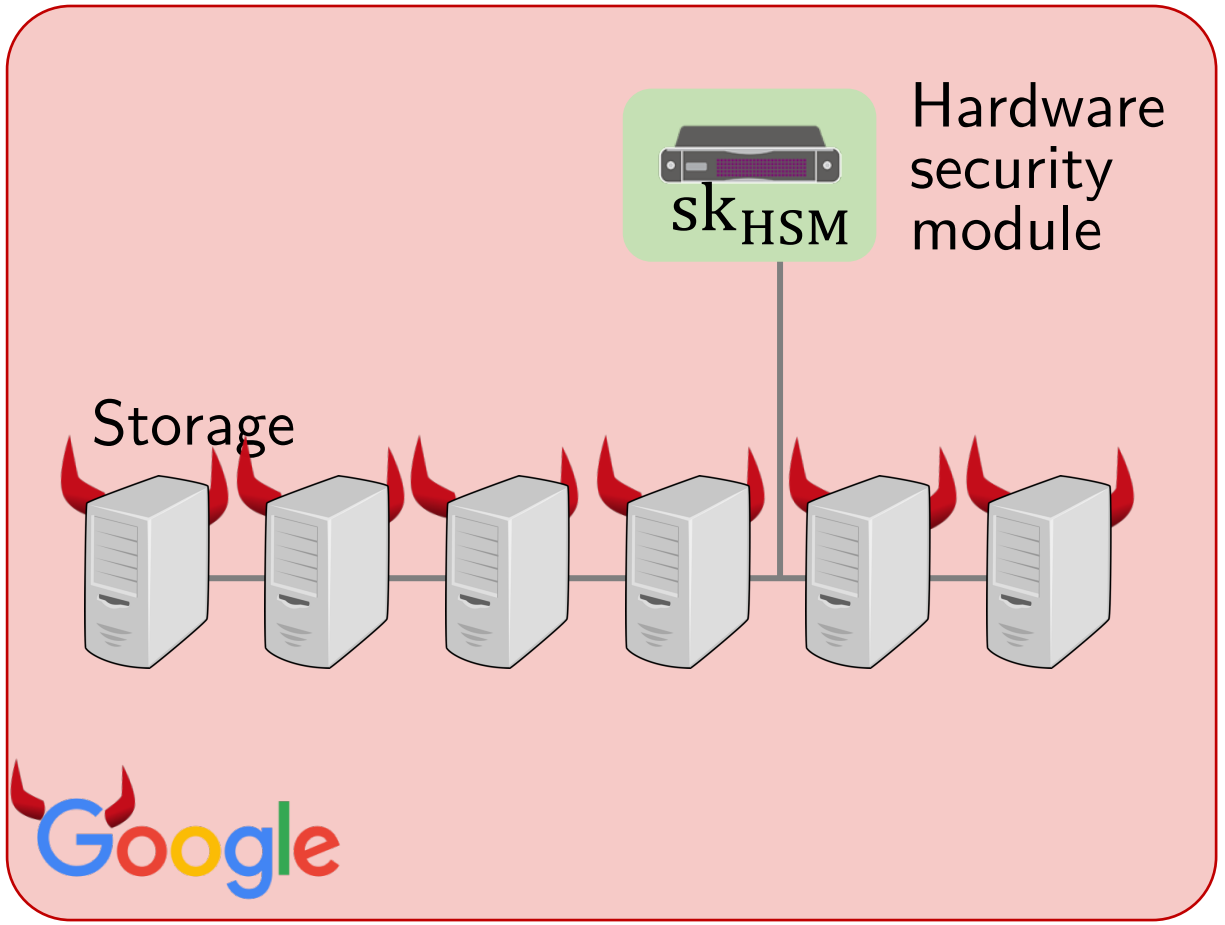


Today's systems: Benefits

+ Secure against compromise of Google's servers
  (HSM limits PIN guesses per user)
+ Convenient for user: just remember your PIN!

– Security of the entire system rests on the security of **ONE** HSM!

...Single point of security failure



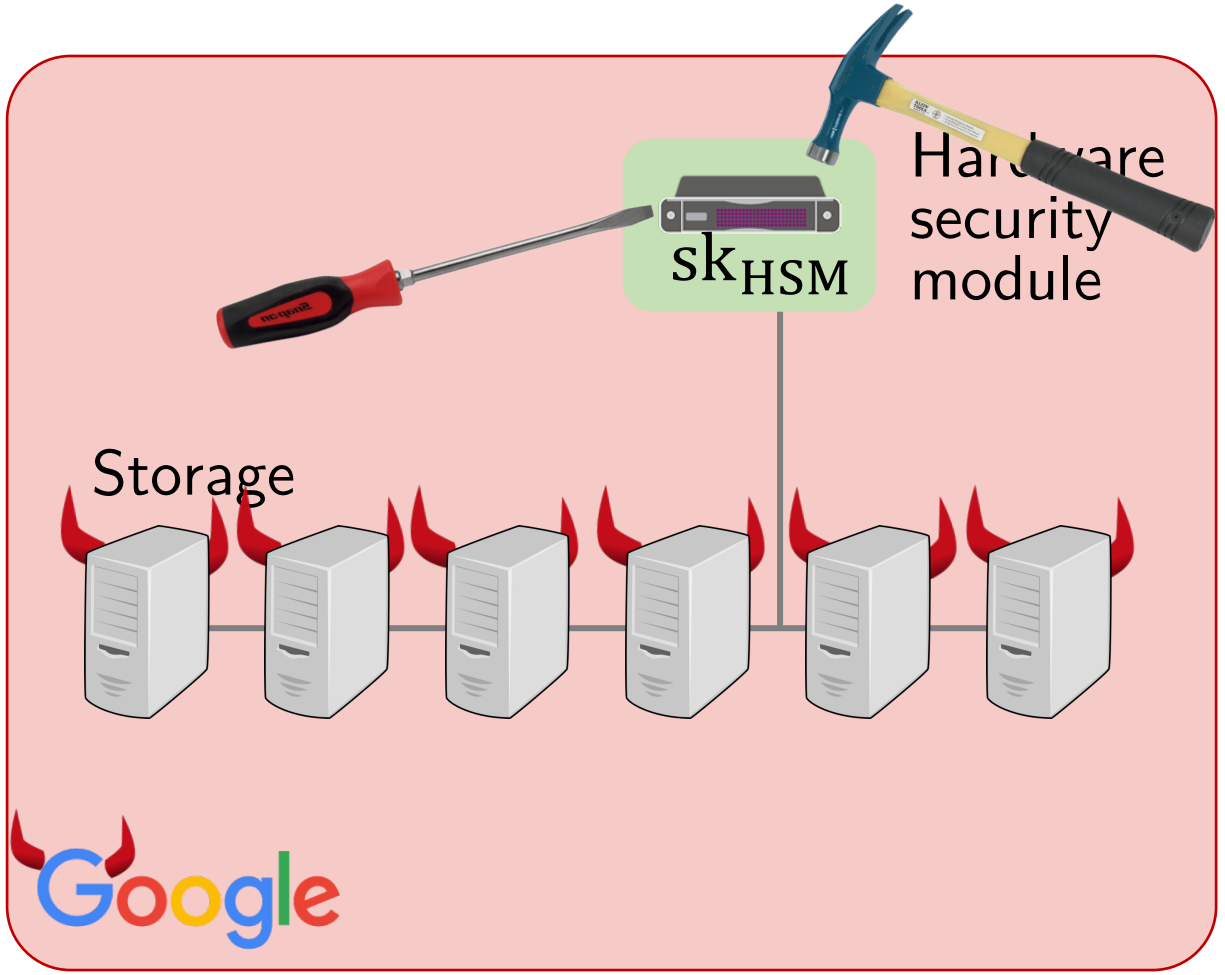
Today's systems: Risks

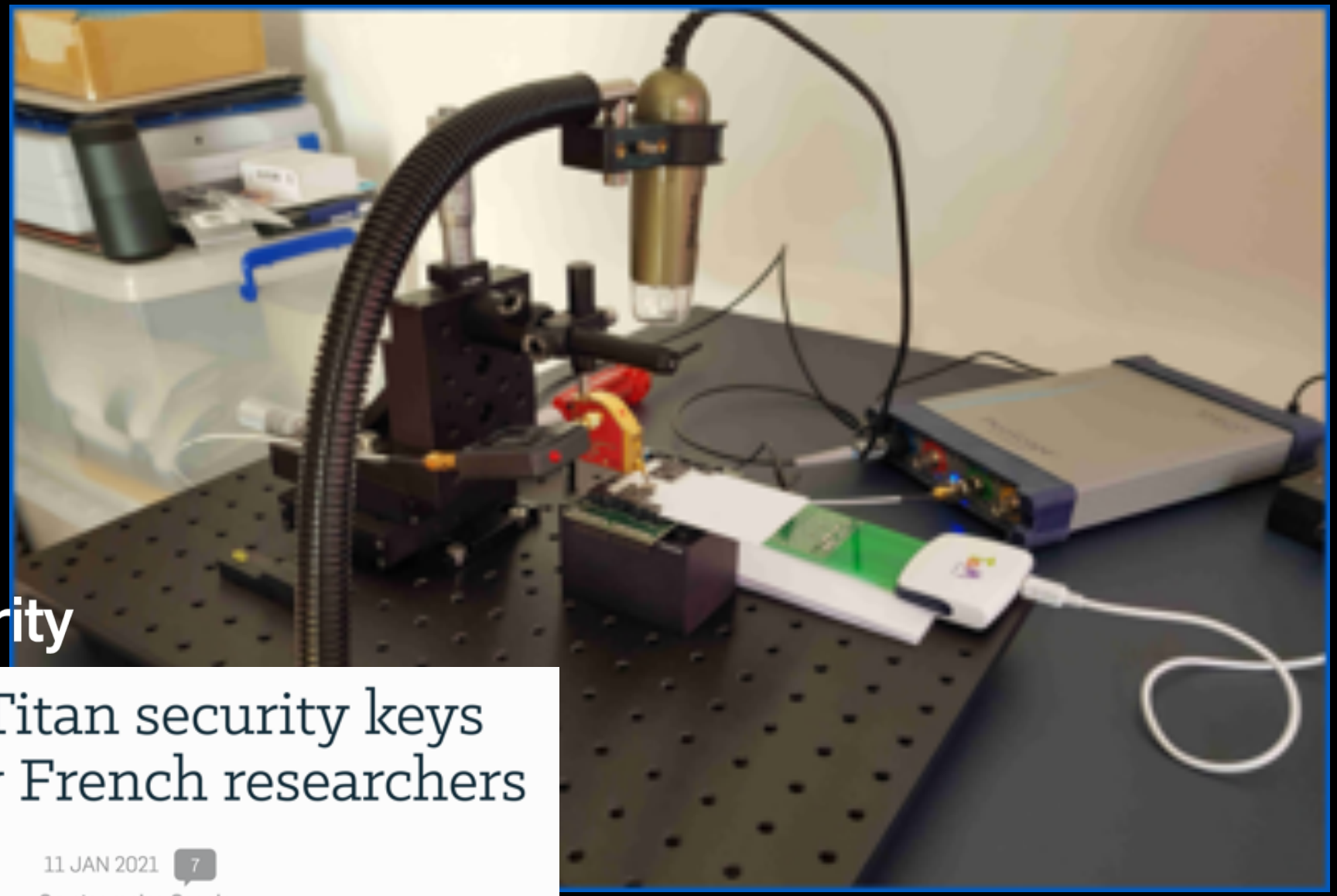
+ Secure against compromise of Google's servers (HSM limits PIN guesses)
+ Convenient for user: just remember your PIN!

– Security of the entire system rests on the security of **ONE** HSM!

...Single point of security failure

[NSSKM17], [HSPK18], ...





naked **security**

Google Titan security keys hacked by French researchers

11 JAN 2021 7

Cryptography, Google

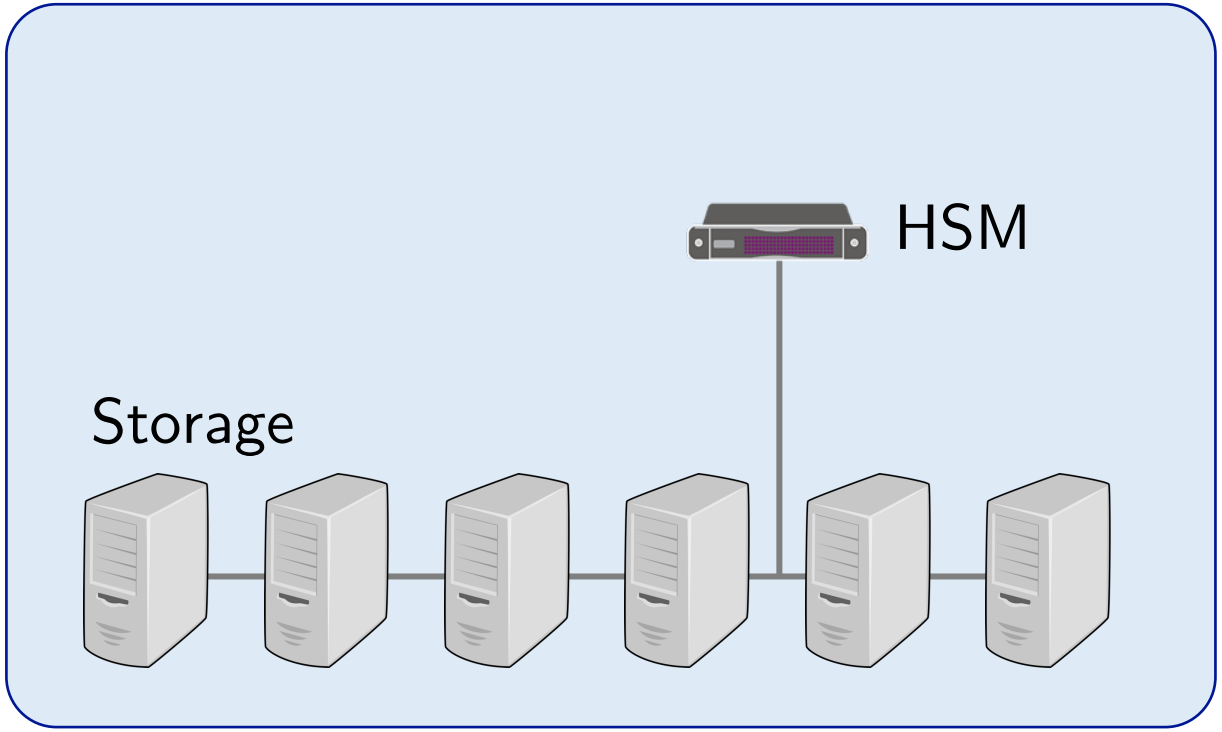
"A Side Journey to Titan" Victor Lomne and Thomas Roche (2021)

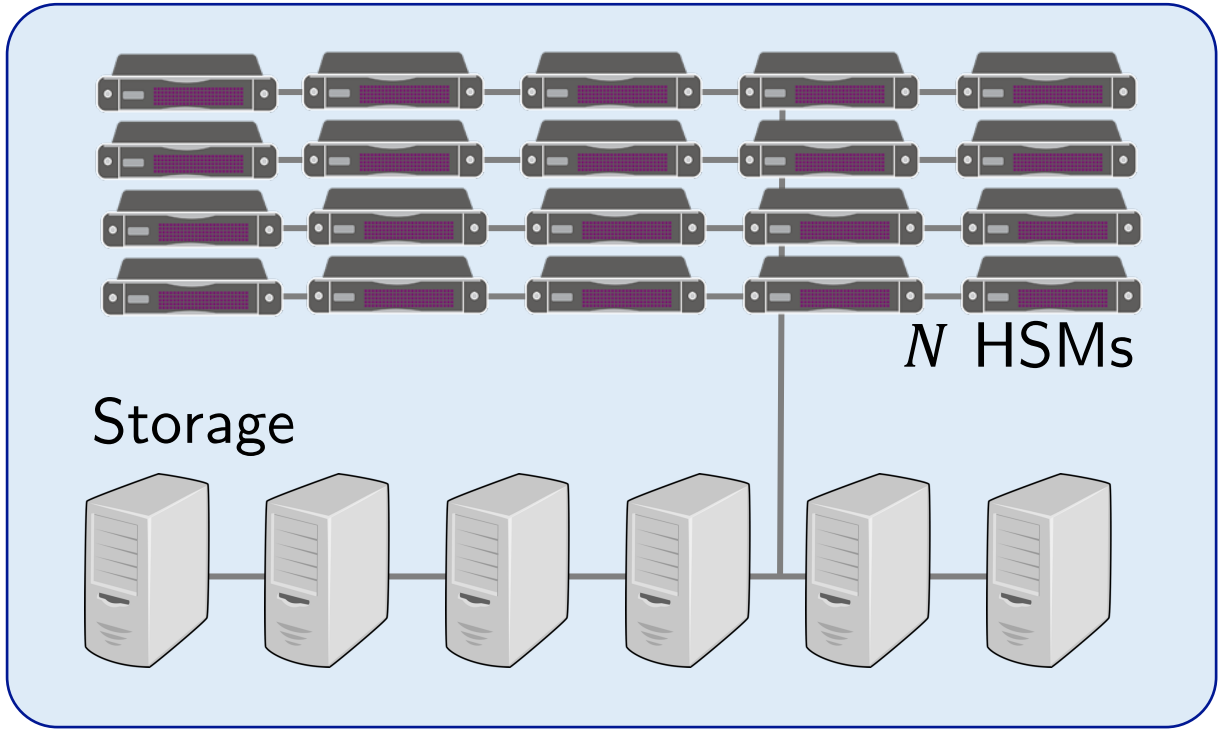
This talk: **SafetyPin**

Convenience and **scalability**
of today's PIN-based backup systems...

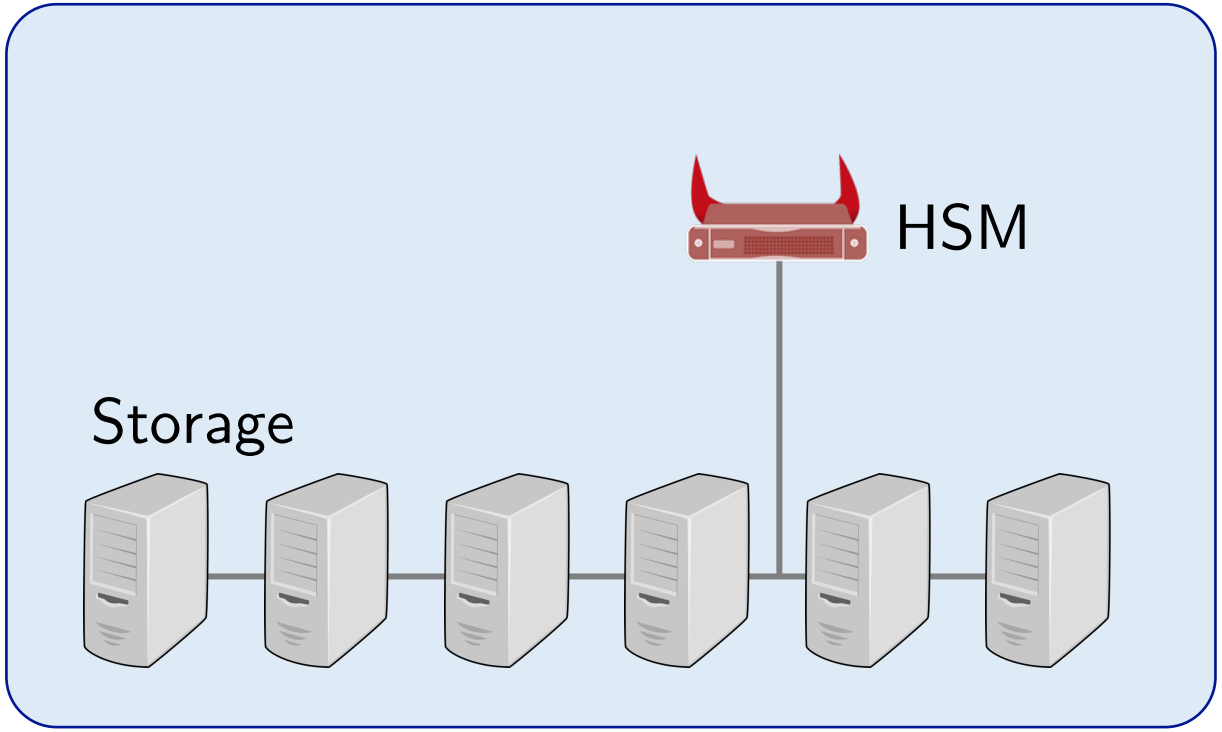
...with **stronger protection**
against HSM compromise.

Idea: Force attacker to compromise many HSMs

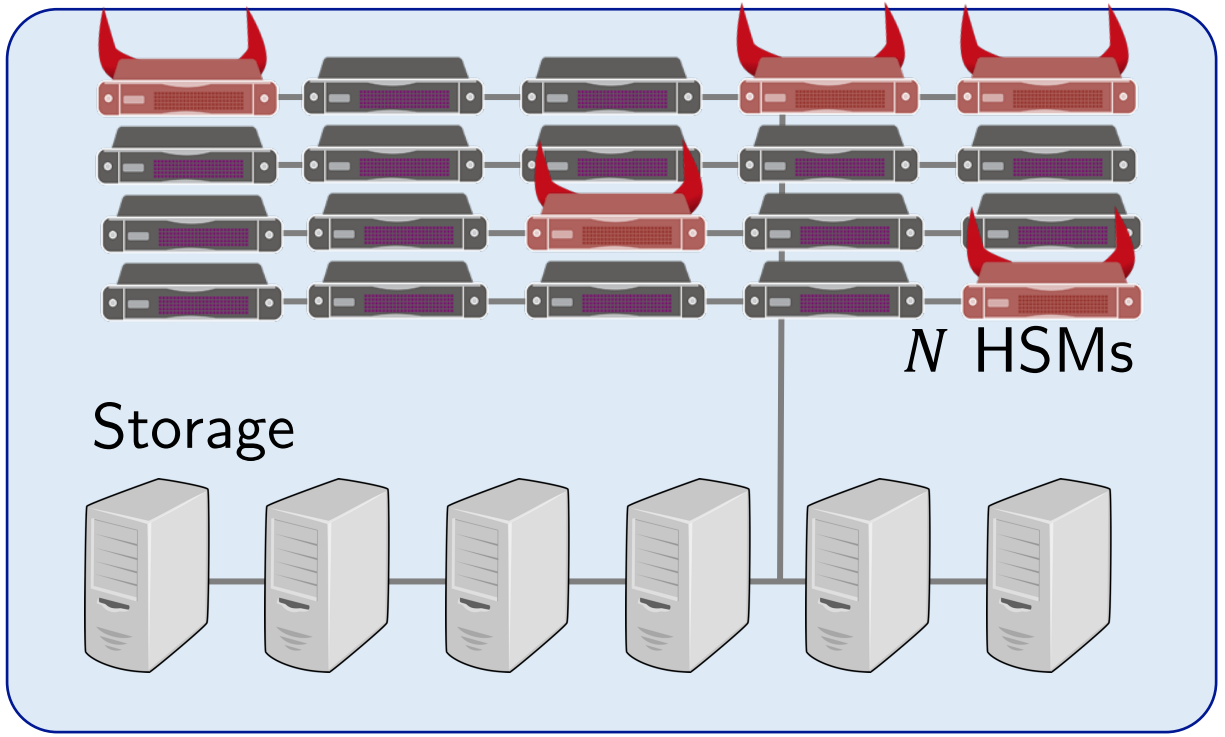




Today vs. SafetyPin: **Security**

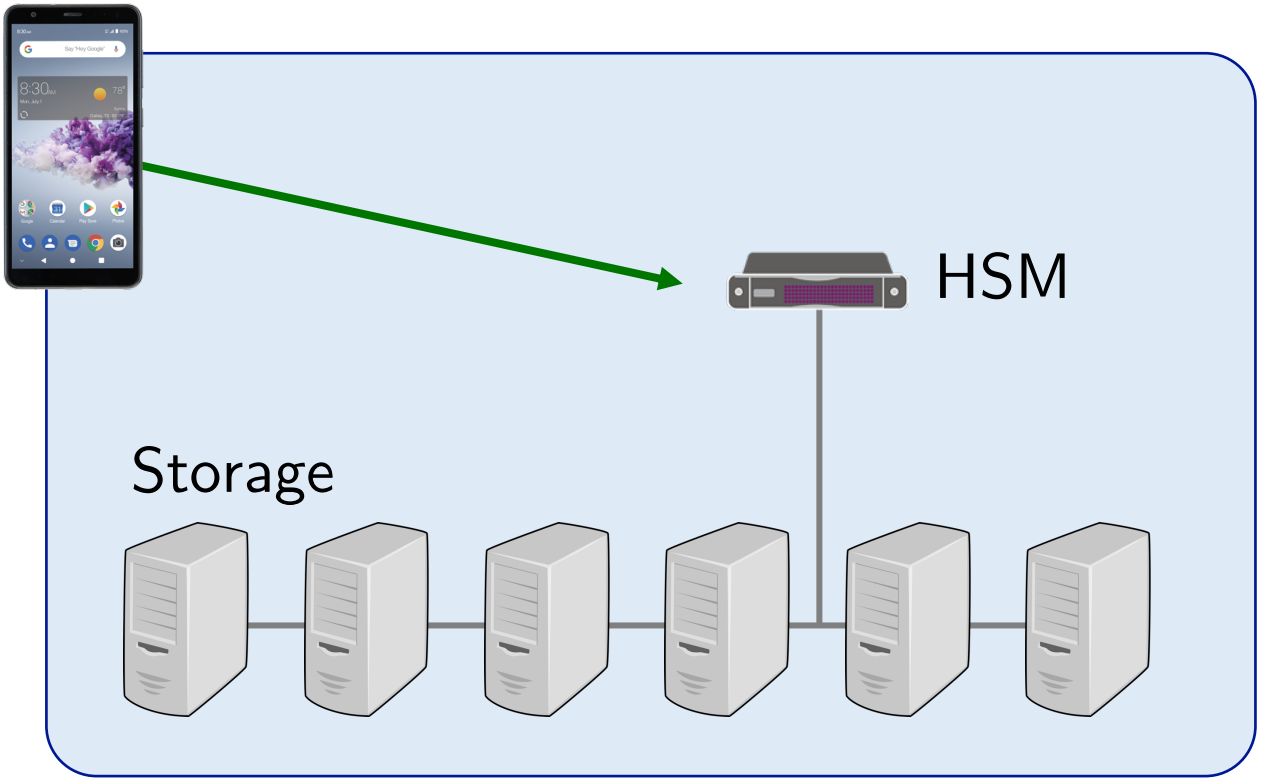


1 compromise = **millions of backups**

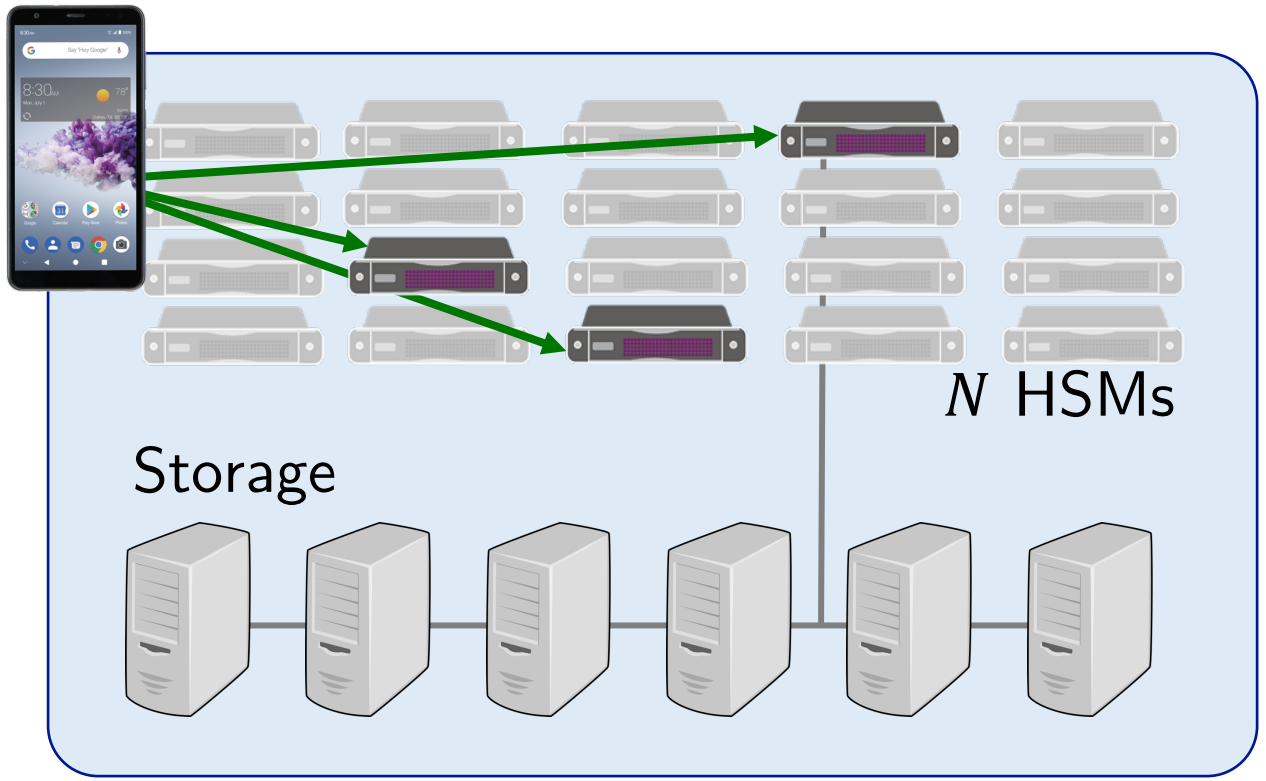


$< \frac{N}{16}$ compromises = **0 backups**

Today vs. SafetyPin: **Scalability**

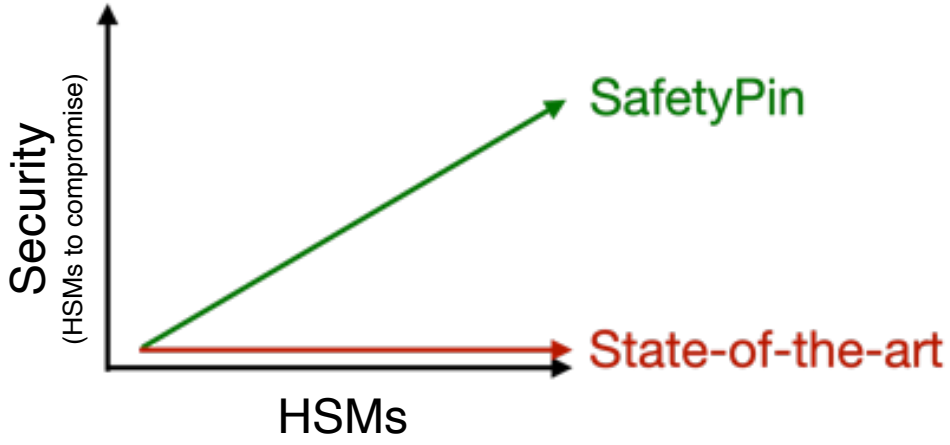


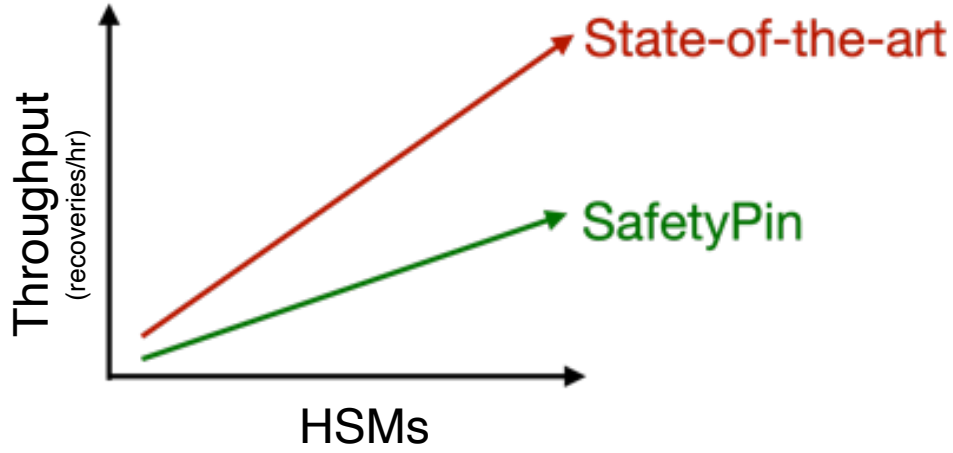
One HSM involved in recovery



"A few" HSMs involved in recovery

More HSMs ⇒ More security + higher throughput

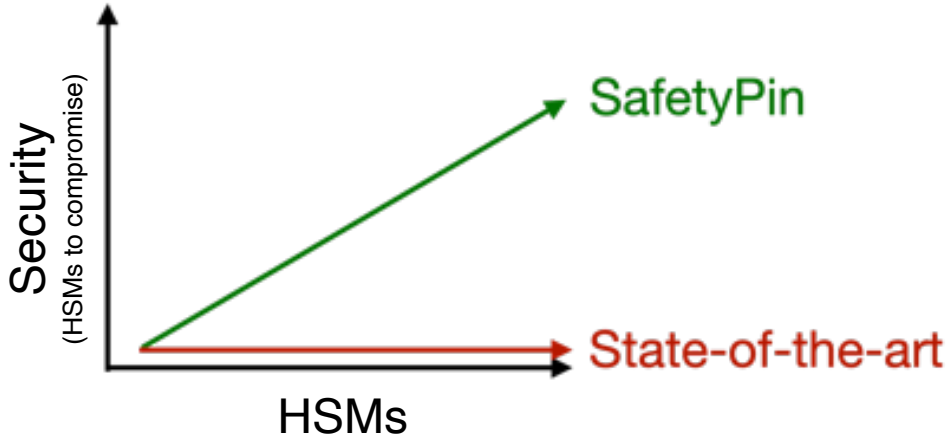


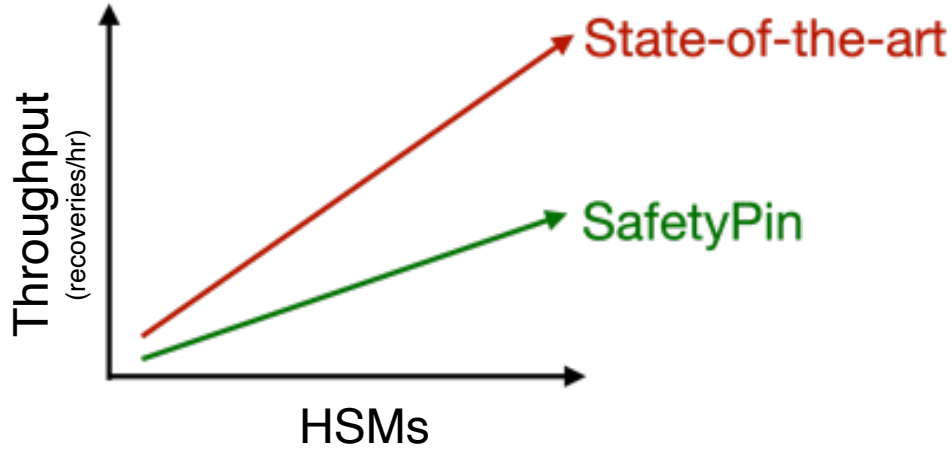


Claim: Compromising more HSMs is more expensive

- Cost of **physical attacks** scales linearly with the number of HSMs.
- Physically attacking more HSMs increases the **risk of exposure**.
- Can get some protection against software bugs with diverse HSMs.

More HSMs ⇒ More security + higher throughput





Claim: Compromising more HSMs is more expensive

- Cost of **physical attacks** scales linearly with the number of HSMs.
- Physically attacking more HSMs increases the **risk of exposure**.
- Can get some protection against software bugs with diverse HSMs.

This talk

- **Motivation**
- SafetyPin: Basic design
- Technical challenges
  - After-the-fact compromise
  - Rate limiting: Distributed log
- Evaluation

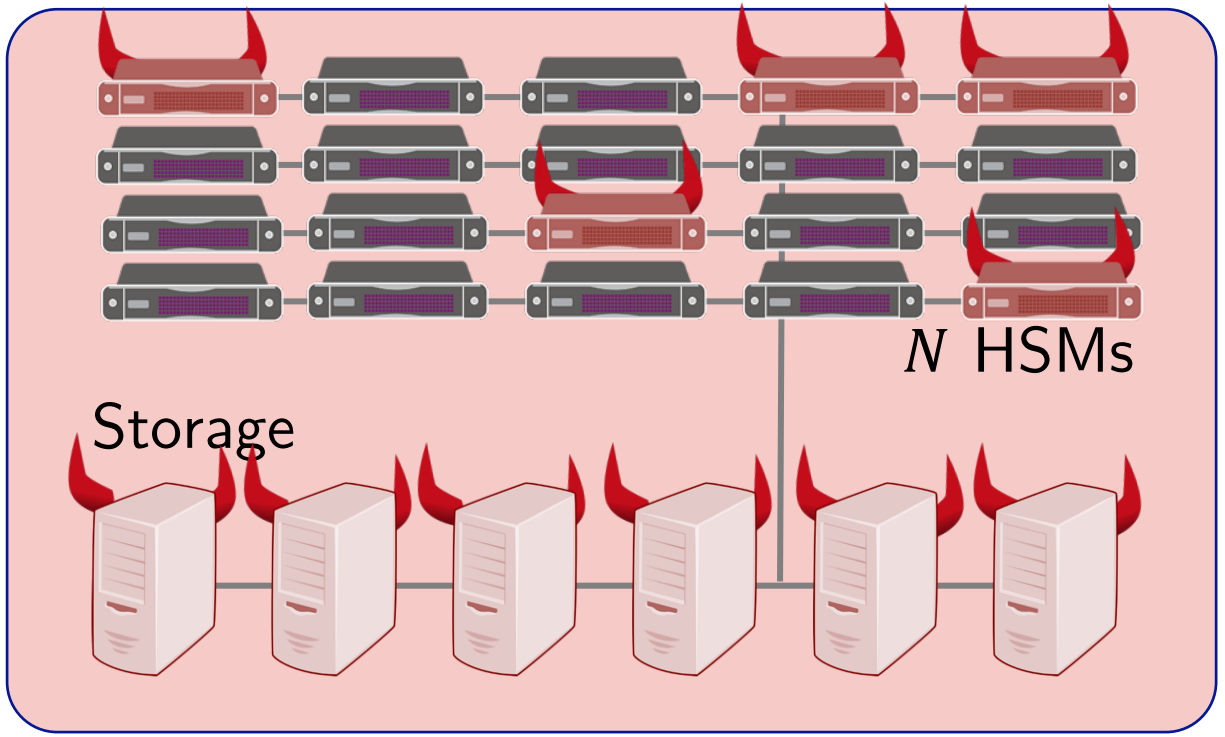
This talk

- Motivation
- **SafetyPin: Basic design**
- Technical challenges
  - After-the-fact compromise
  - Rate limiting: Distributed log
- Evaluation

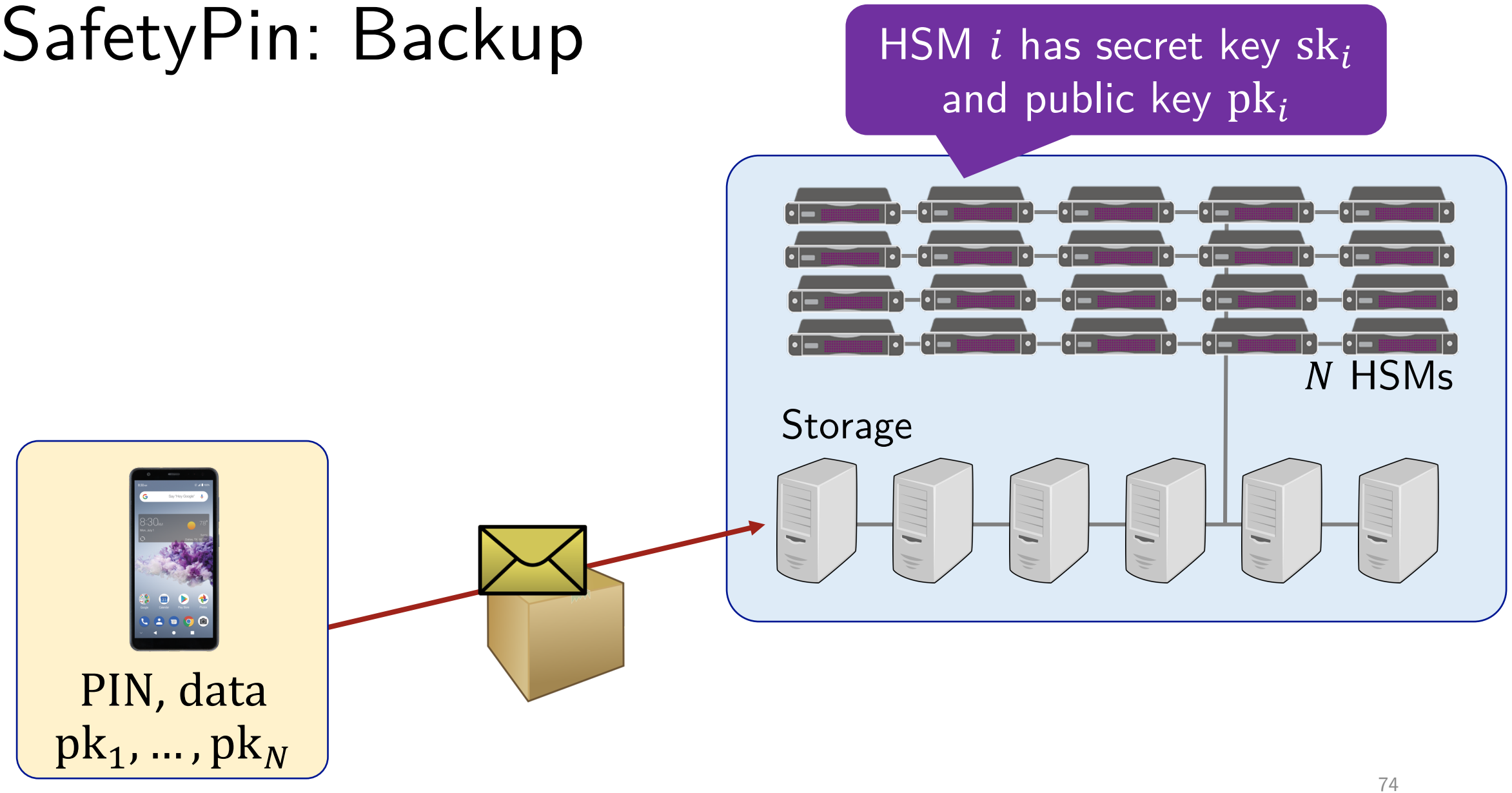
SafetyPin: Security goal

Attacker's chance of recovering honest client's data is "not much better" than guessing client's PIN, even if the attacker:

- Controls the data center,
- Adaptively compromises $N/16$ HSMs after client backs up and before client begins recovery, and
- Compromises all HSMs in the data center after the client recovers.



SafetyPin: Backup



SafetyPin: Backup [simplified]

1. Sample a random AES encryption key k_{data}
2. Split k_{data} into ~40 additive secret shares: $k_1, \ldots, k_{40}$
3. Sample a set of ~40 HSMs as
$$(i_1, \ldots, i_{40}) \leftarrow \text{Hash}(\text{userID}, \text{PIN})$$
4. Output ciphertext:
$$\langle \text{AES}(k_{\text{data}}, \text{data}), \text{Enc}(\text{pk}_{i_1}, k_1), \ldots, \text{Enc}(\text{pk}_{i_{40}}, k_{40}) \rangle$$

[Here Enc() is Hashed ElGamal encryption.]

SafetyPin: Backup [simplified]

1. Sample a random AES encryption key k_{data}
2. Split k_{data} into ~40 additive secret shares: $k_1, \ldots, k_{40}$
3. Sample a set of ~40 HSMs as
$$(i_1, \ldots, i_{40}) \leftarrow \text{Hash}(\text{userID}, \text{PIN})$$
4. Output ciphertext:
$$\langle \text{AES}(k_{\text{data}}, \text{data}), \text{Enc}(\text{pk}_{i_1}, k_1), \ldots, \text{Enc}(\text{pk}_{i_{40}}, k_{40}) \rangle$$

[Here Enc() is Hashed ElGamal encryption.]

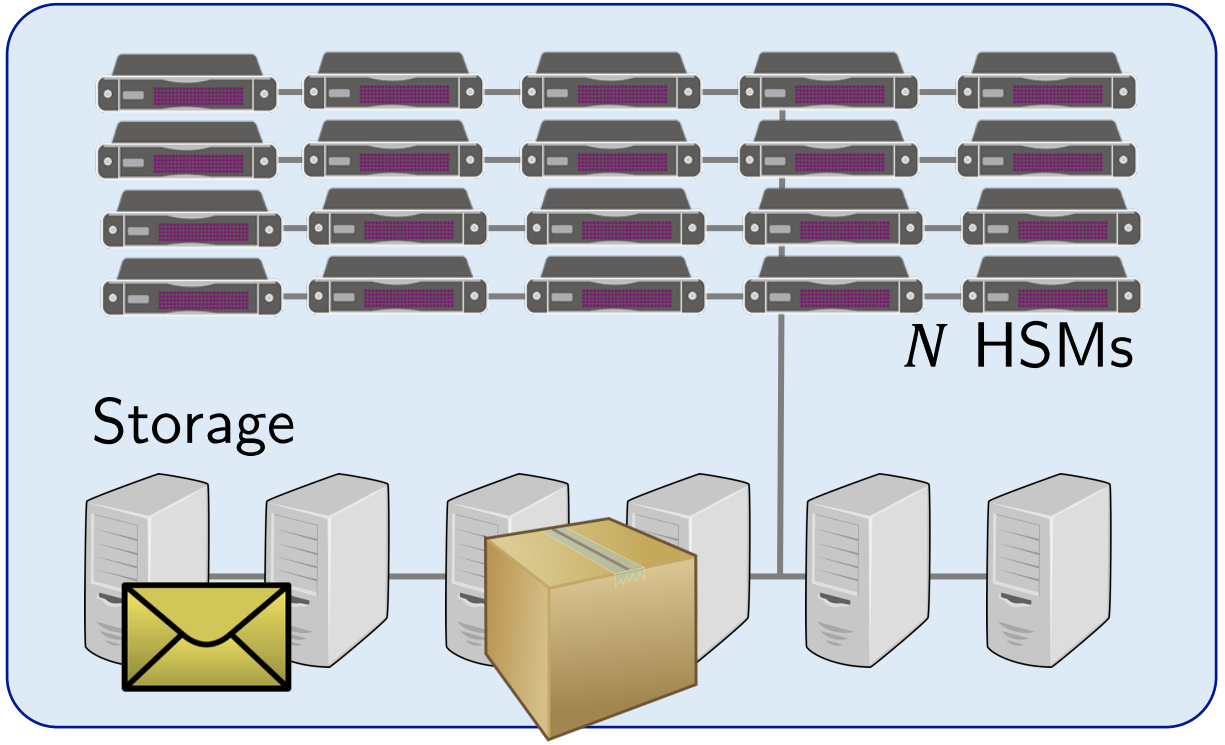
Need encryption scheme to be
(1) key-private (anonymous)
(2) secure under selective opening.

Key privacy: [BBDP01]
Selective opening: [CDNO97], [BHY09], [FHKW10], [HR14], ...

SafetyPin: Backup

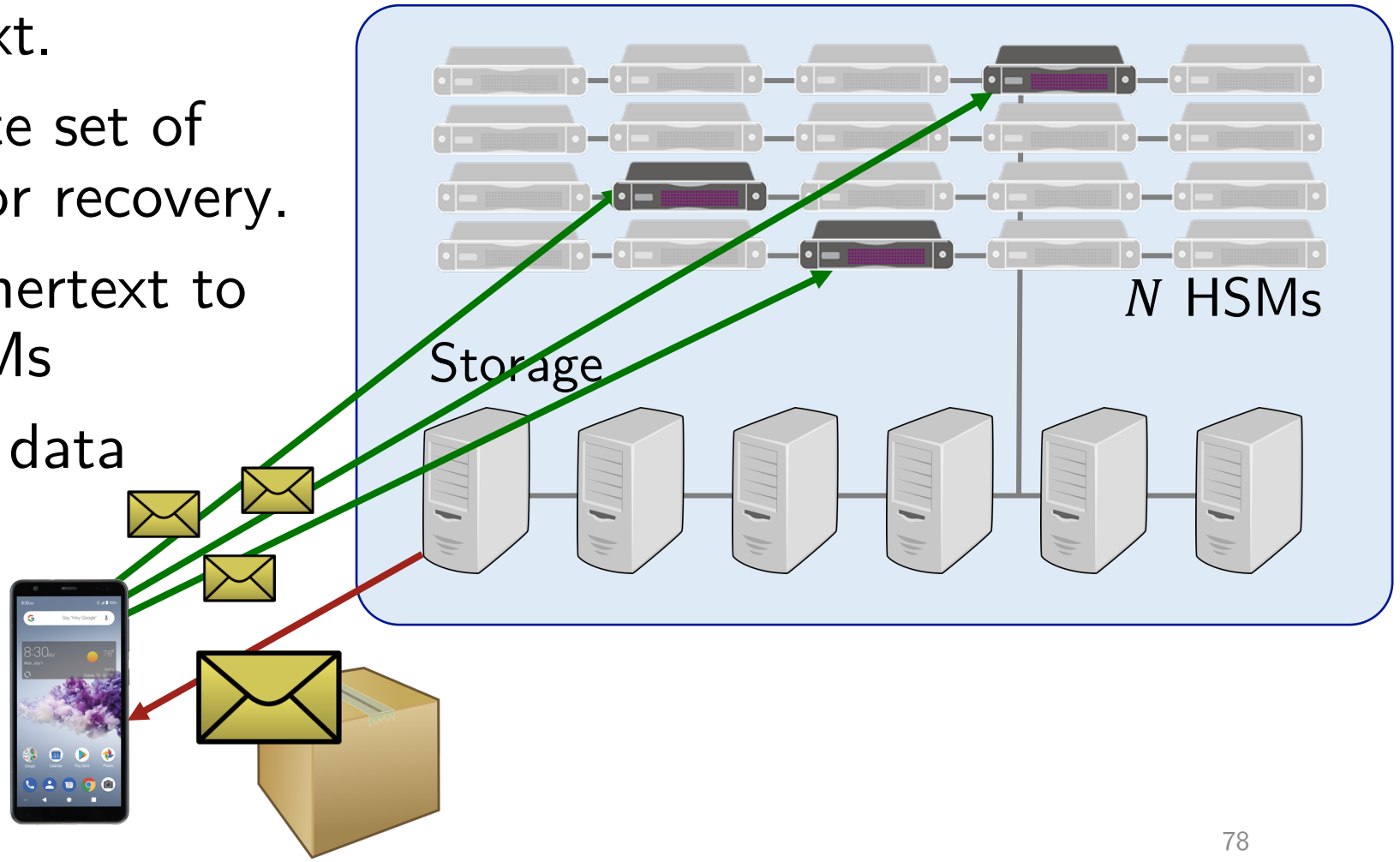
Security intuition.

- Attacker could get client data by compromising only 40 ($\ll \frac{N}{16}$) HSMs.
- Attacker doesn't know which HSMs to compromise.
  (Best strategy ≈ guess the PIN)
- Unless attacker guesses the right 40 HSMs to compromise, attacker gets nothing.



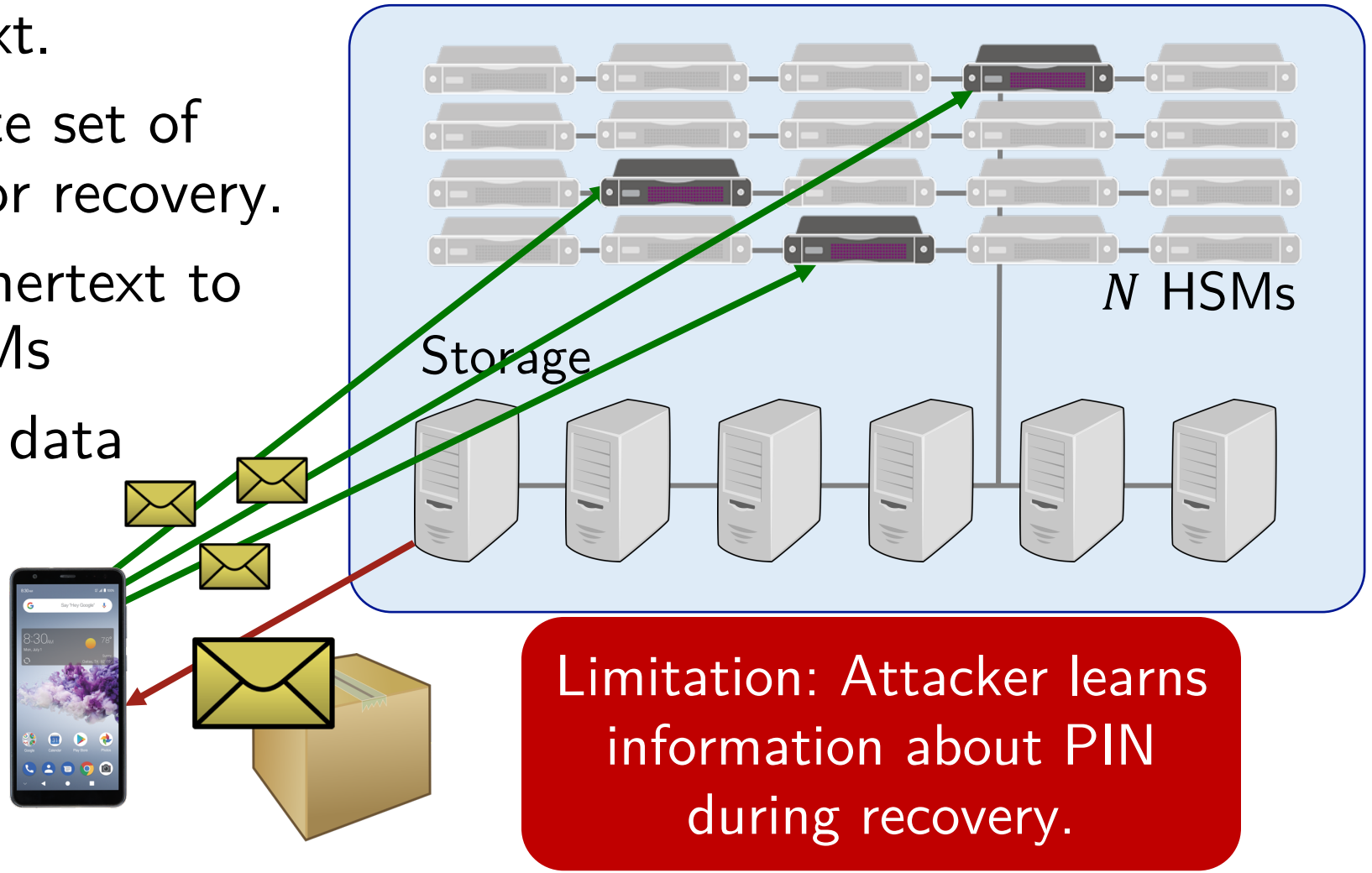
SafetyPin: Recovery [simplified]

1. Download ciphertext.
2. Use PIN to compute set of 40 HSMs needed for recovery.
3. Send key-share ciphertext to each of the 40 HSMs
4. Recover backed-up data



SafetyPin: Recovery [simplified]

1. Download ciphertext.
2. Use PIN to compute set of 40 HSMs needed for recovery.
3. Send key-share ciphertext to each of the 40 HSMs
4. Recover backed-up data



Limitation: Attacker learns information about PIN during recovery.

Fault Tolerance

Must be able to recover data even if some HSMs fail!

Today's systems: Replicate secret key at 10 HSMs
- Recover data if < 10 HSMs fail

SafetyPin: Replace additive secret sharing with Shamir
- Split key into 50 shares such that any 40 can recover
- Recover data if < 10 HSMs fail

This talk

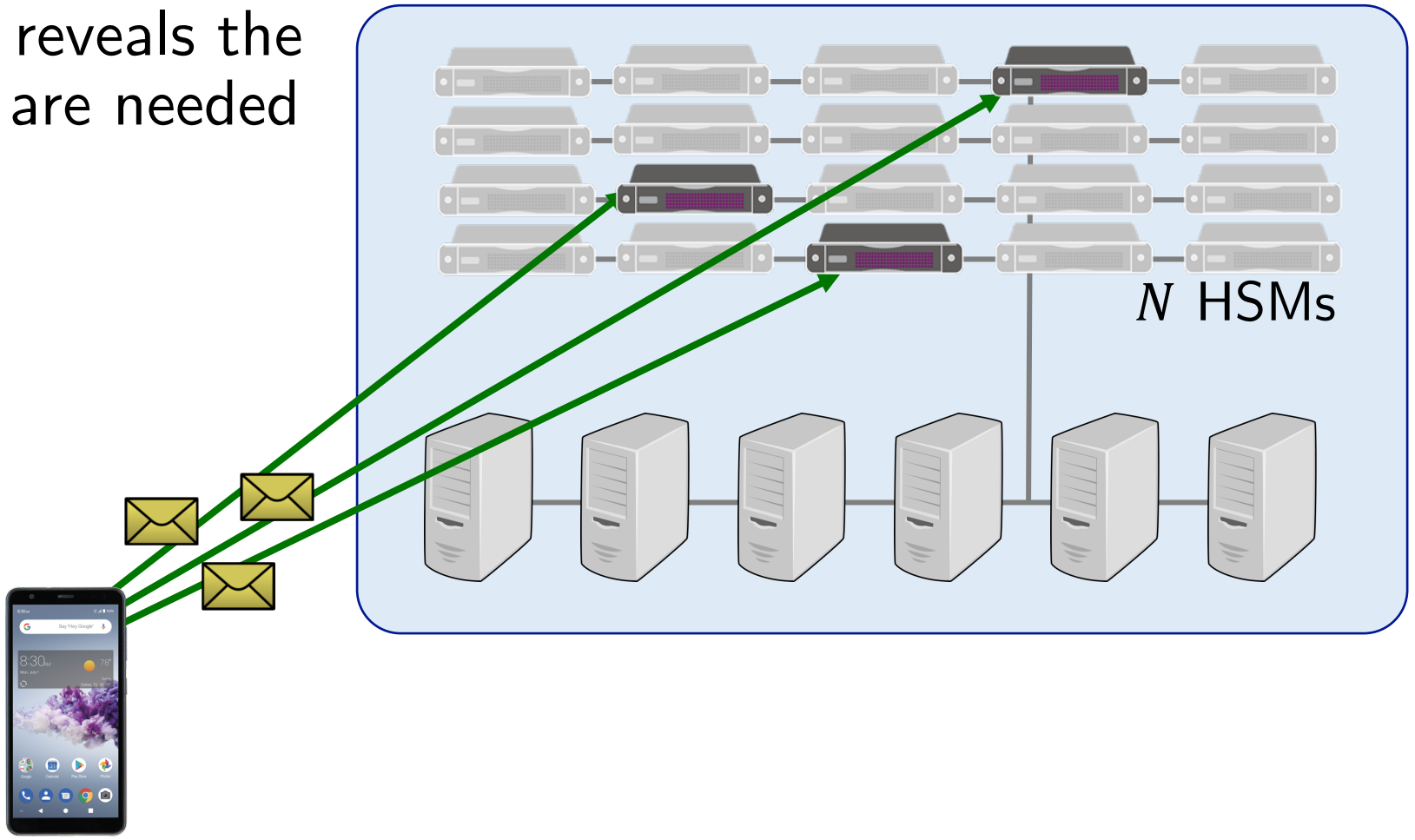
- Motivation
- **SafetyPin: Basic design**
- Technical challenges
  - After-the-fact compromise
  - Rate limiting: Distributed log
- Evaluation

This talk

- Motivation
- SafetyPin: Basic design
- Technical challenges
  - **After-the-fact compromise**
  - Rate limiting: Distributed log
- Evaluation

Problem: Post-recovery compromise

During recovery, client reveals the ~40 HSMs whose keys are needed to decrypt its backup.

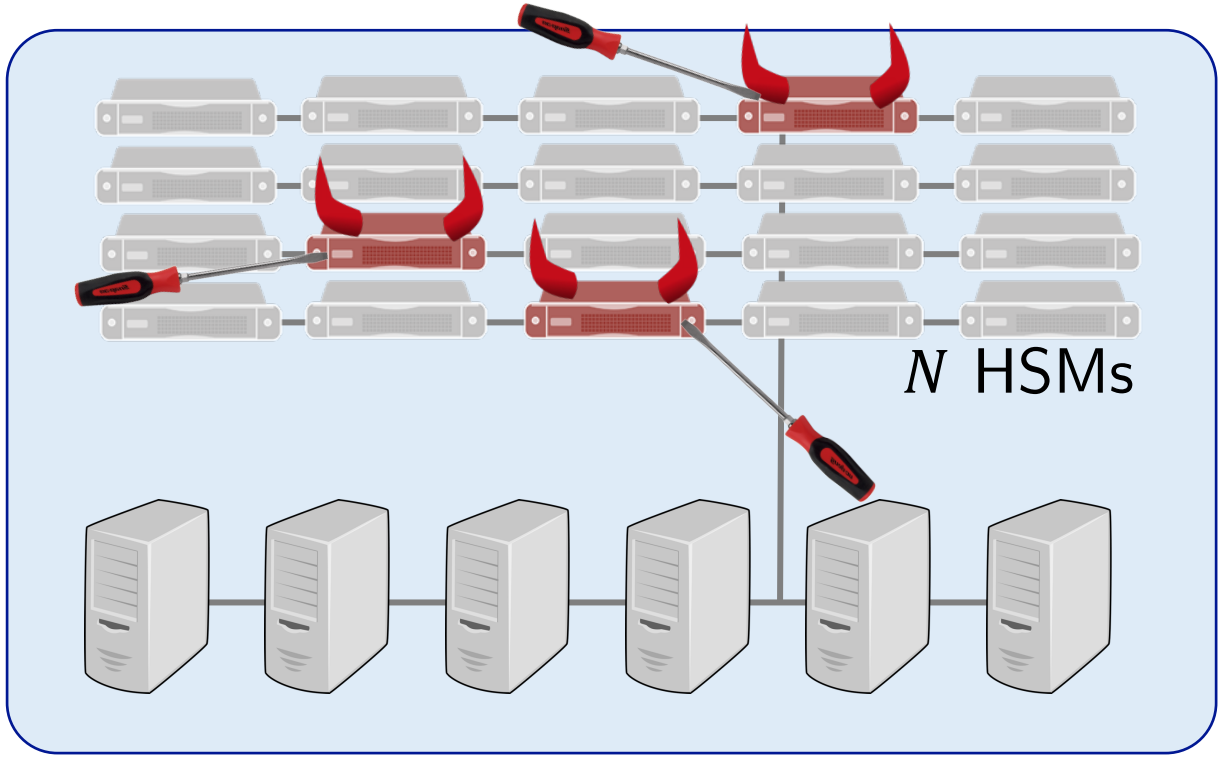


Problem: Post-recovery compromise

During recovery, client reveals the ~40 HSMs whose keys are needed to decrypt its backup.

Idea: After recovery, each HSM revokes its ability to decrypt the client's ciphertext.

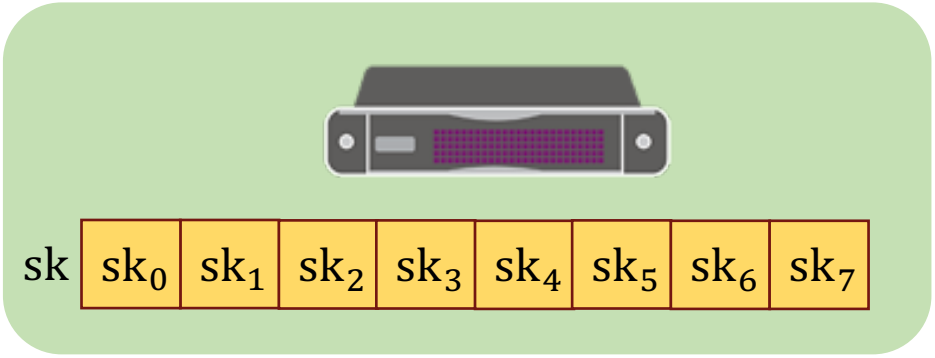
→ "Puncturable encryption" [GM15]



HSMs revoke their ability to decrypt using "puncturable encryption"

[GM15], [GHJL17], [DJSS18], …

- Secret key consists of a big array
  - 64 MB of data for our parameters
  - Can use crypto (HIBE) to compress pub key
- To revoke ability to decrypt a ciphertext, HSM deletes some elements of the array



BUT, HSMs has too little internal storage to store sk

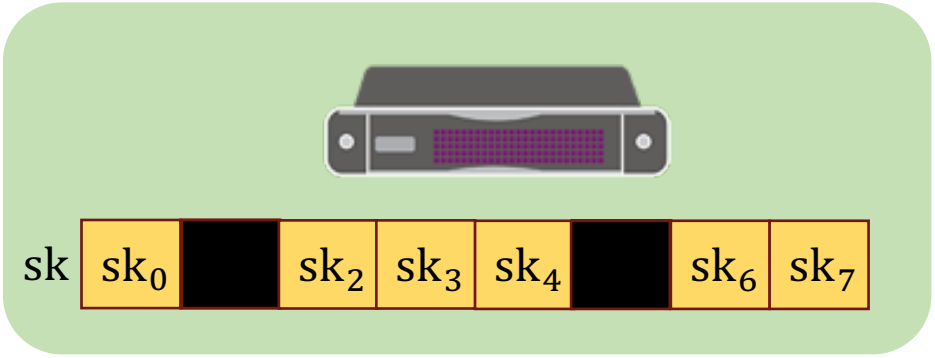
Our idea: HSM outsources storage to data center, while protecting against future compromise.

(I suspect that today's systems do some outsourcing too…)

HSMs revoke their ability to decrypt using "puncturable encryption"

[GM15], [GHJL17], [DJSS18], …

- Secret key consists of a big array
  - 64 MB of data for our parameters
  - Can use crypto (HIBE) to compress pub key
- To revoke ability to decrypt a ciphertext, HSM deletes some elements of the array



BUT, HSMs has too little internal storage to store sk

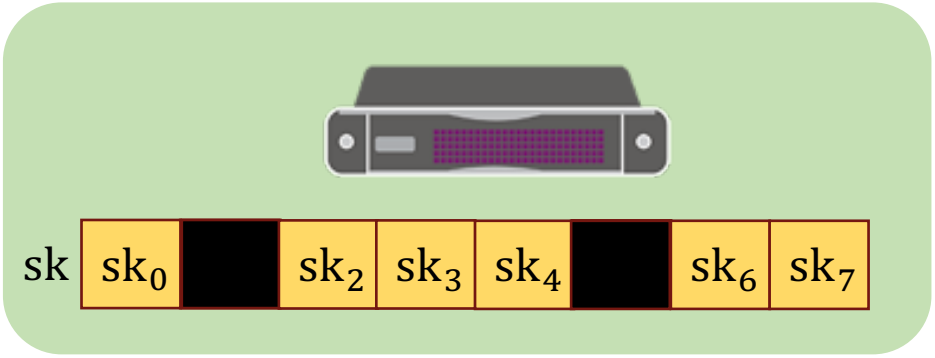
Our idea: HSM outsources storage to data center, while protecting against future compromise.

(I suspect that today's systems do some outsourcing too…)

HSMs revoke their ability to decrypt using "puncturable encryption"

[GM15], [GHJL17], [DJSS18], …

- Secret key consists of a big array
  - 64 MB of data for our parameters
  - Can use crypto (HIBE) to compress pub key



- To revoke ability to decrypt a ciphertext, HSM deletes some elements of the array

BUT, HSMs has too little internal storage to store sk

Our idea: HSM outsources storage to data center, while protecting against future compromise.

(I suspect that today's systems do some outsourcing too…)

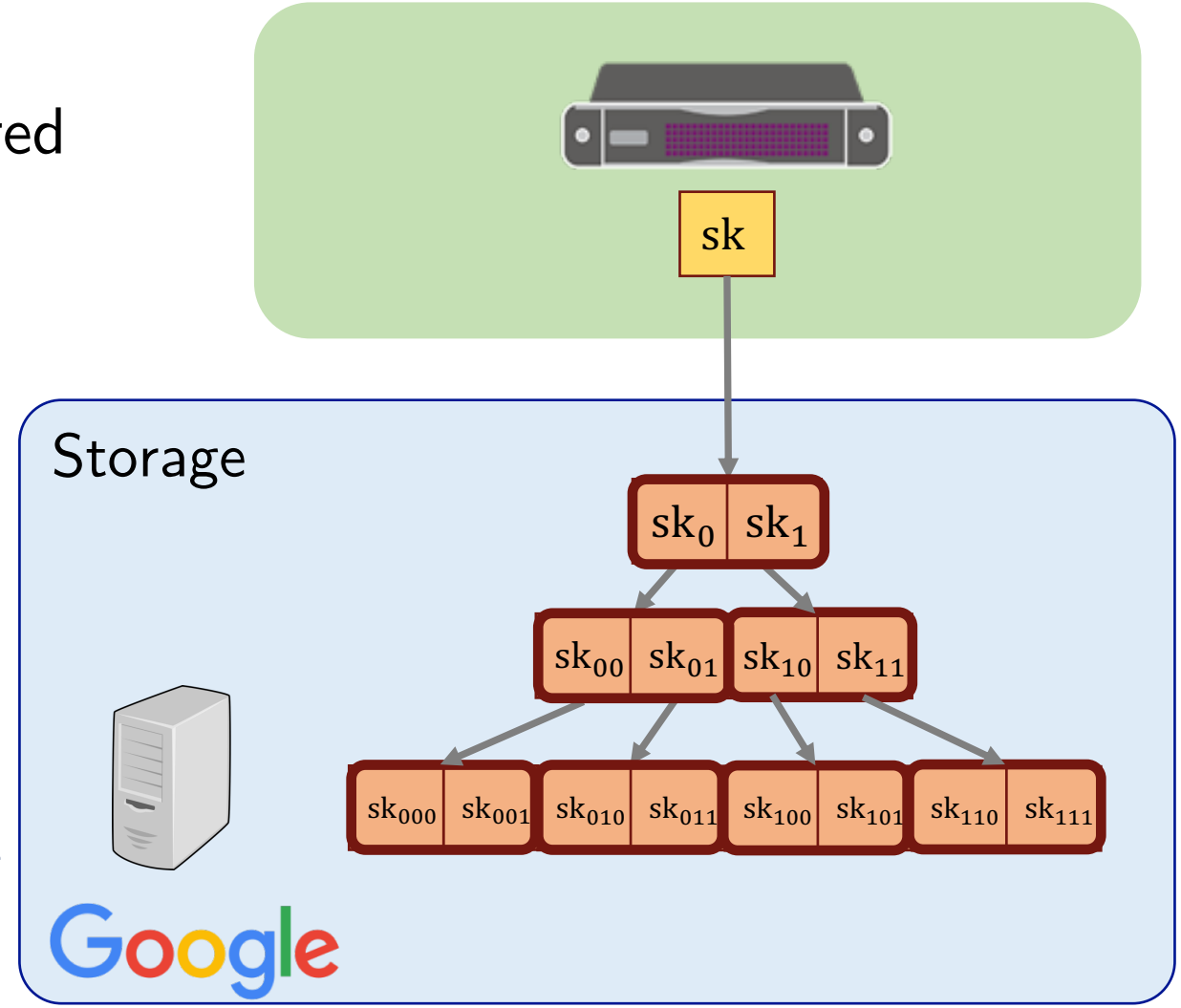
Forward-secure outsourced storage

- Each element of the key array is stored encrypted under its "parent" key
- HSM stores only the root key

To delete an element:

- Replace all keys on element's path to the root
- Replace HSM's root key

Attacker who compromises HSM state cannot recover deleted elements.



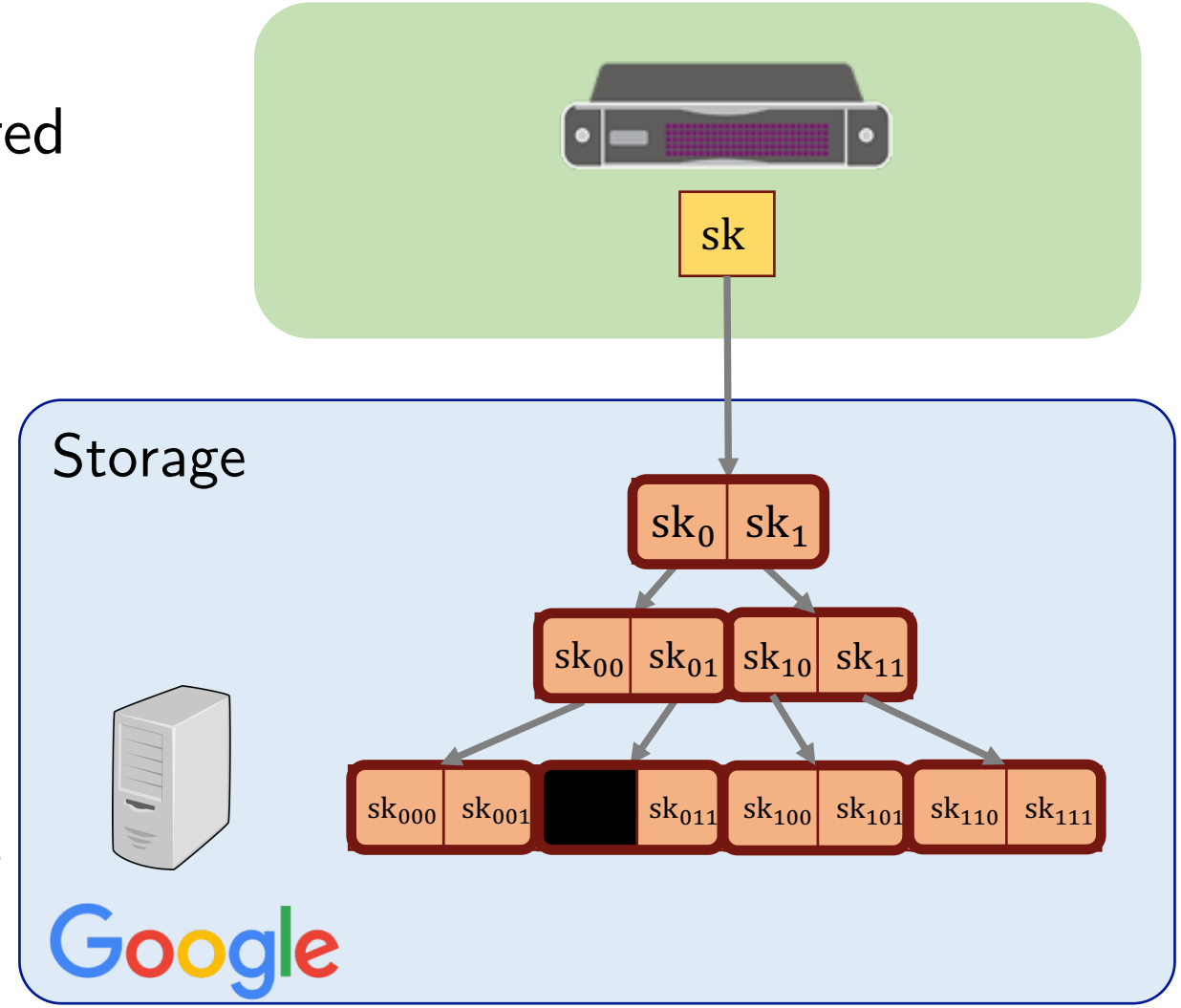
Forward-secure outsourced storage

- Each element of the key array is stored encrypted under its "parent" key
- HSM stores only the root key

To delete an element:

- Replace all keys on element's path to the root
- Replace HSM's root key

Attacker who compromises HSM state cannot recover deleted elements.



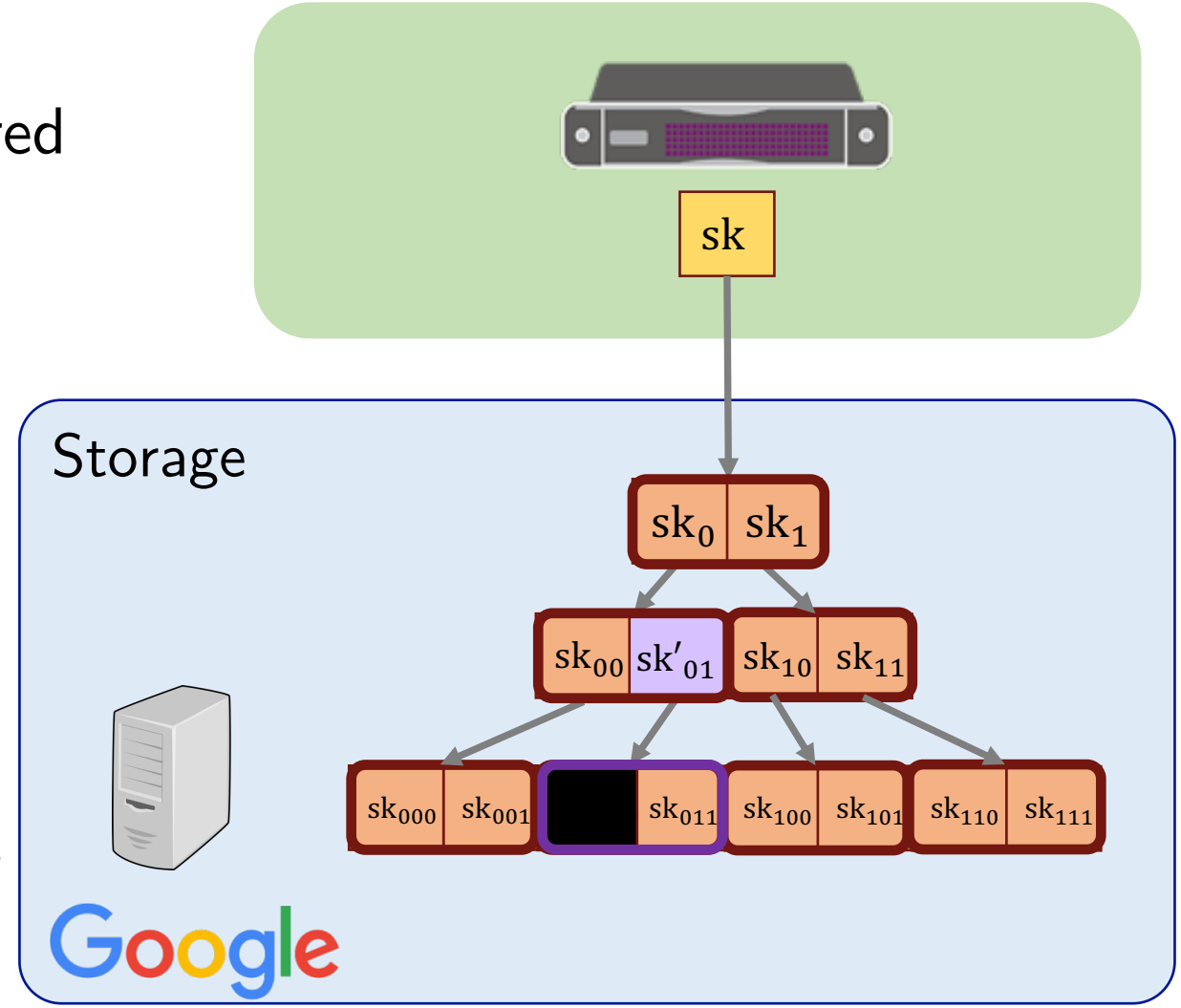
Forward-secure outsourced storage

- Each element of the key array is stored encrypted under its "parent" key
- HSM stores only the root key

To delete an element:

- Replace all keys on element's path to the root
- Replace HSM's root key

Attacker who compromises HSM state cannot recover deleted elements.



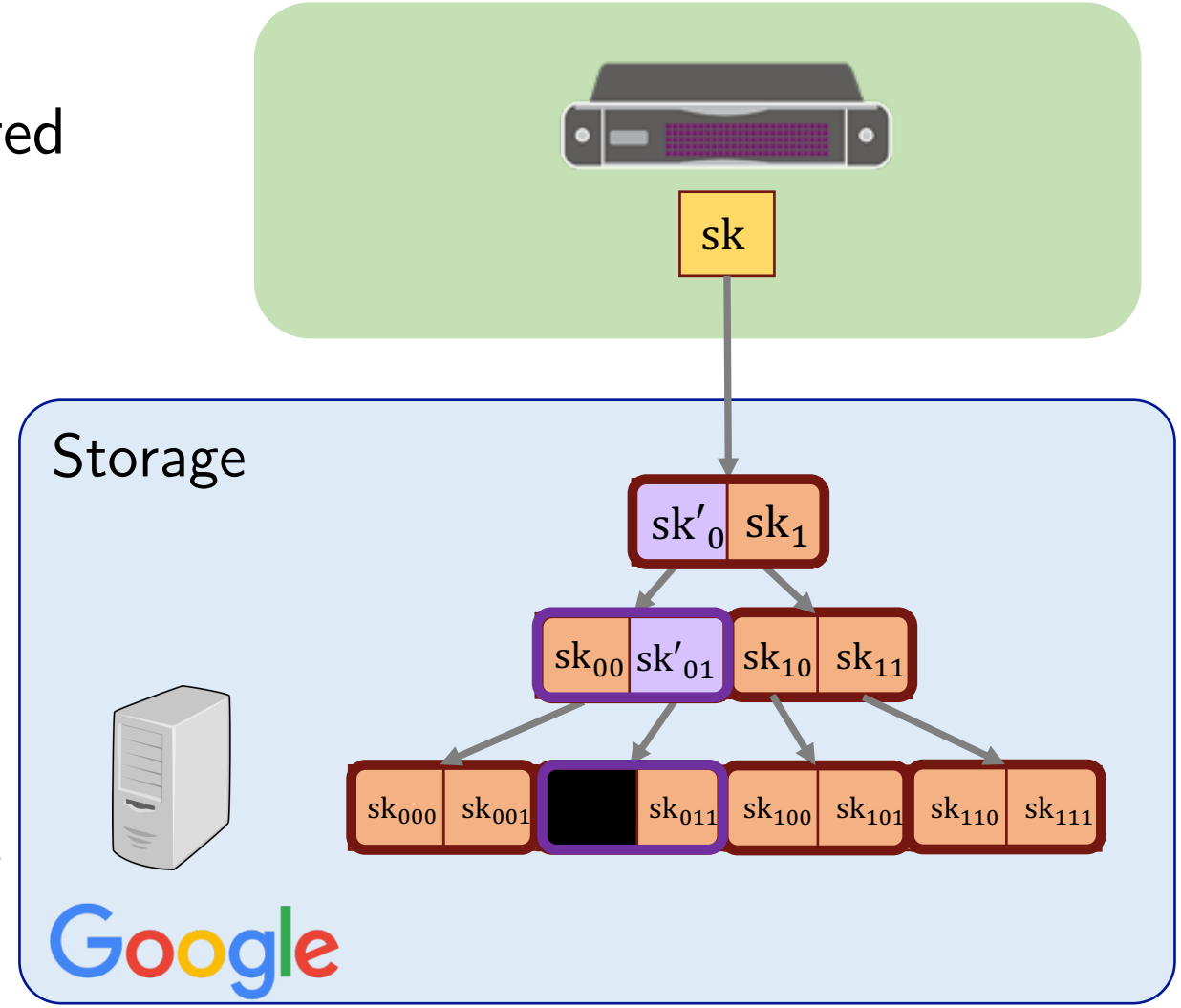
Forward-secure outsourced storage

- Each element of the key array is stored encrypted under its "parent" key
- HSM stores only the root key

To delete an element:

- Replace all keys on element's path to the root
- Replace HSM's root key

Attacker who compromises HSM state cannot recover deleted elements.



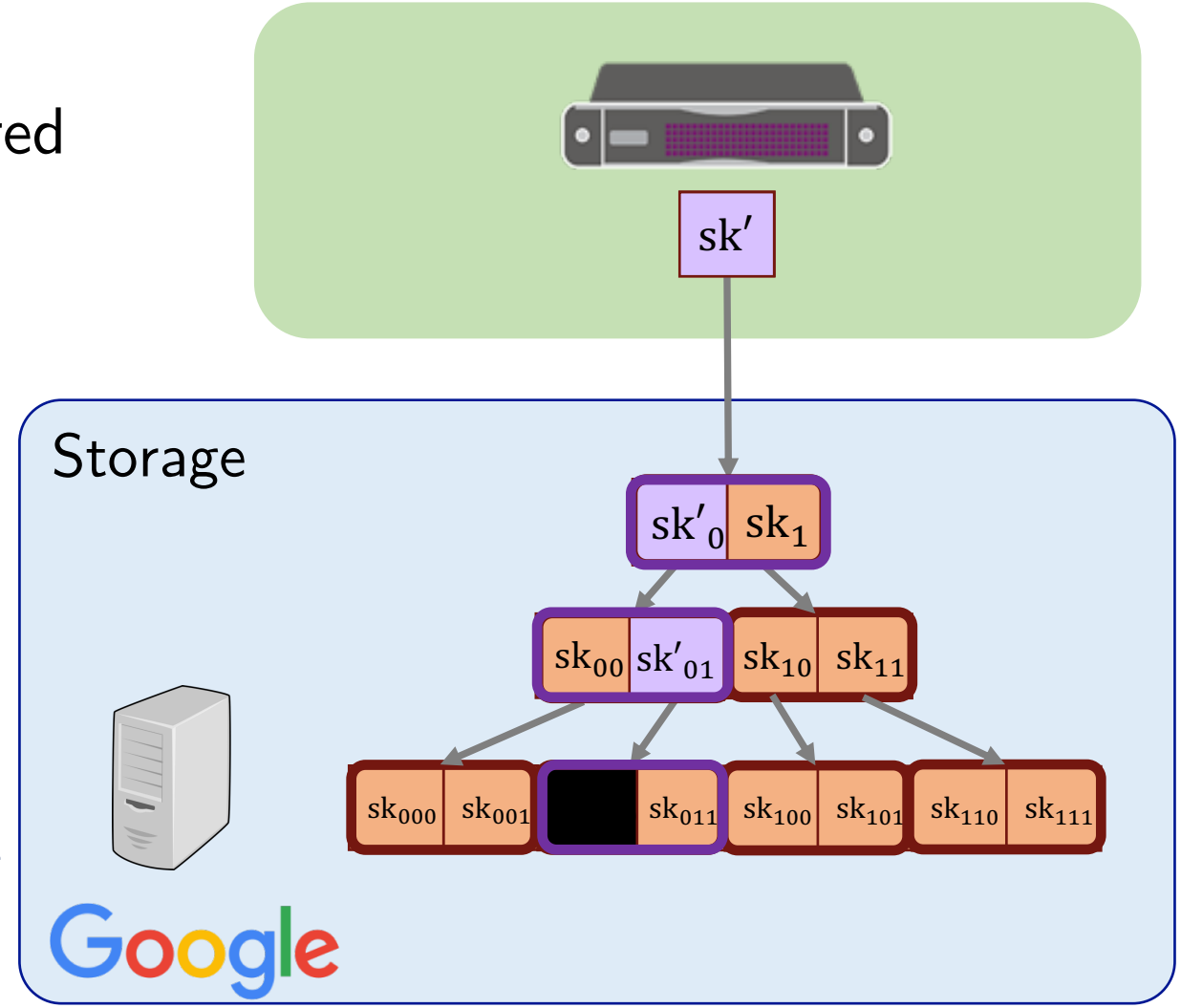
Forward-secure outsourced storage

- Each element of the key array is stored encrypted under its "parent" key
- HSM stores only the root key

To delete an element:

- Replace all keys on element's path to the root
- Replace HSM's root key

Attacker who compromises HSM state cannot recover deleted elements.

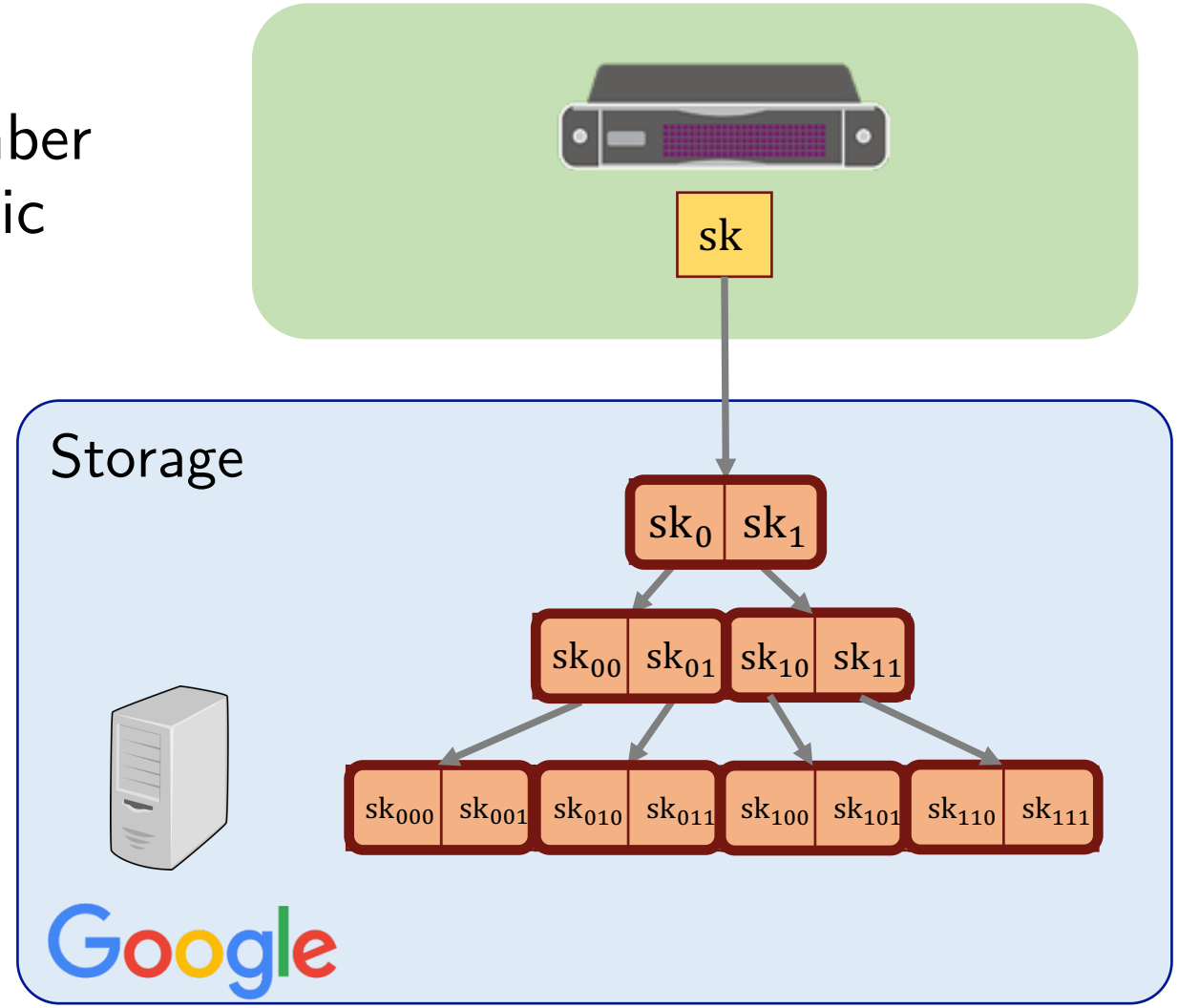


Forward-secure outsourced storage

Each read/write/delete requires a number of symmetric-key crypto ops logarithmic in the array size.

Concretely, for 64MB array:

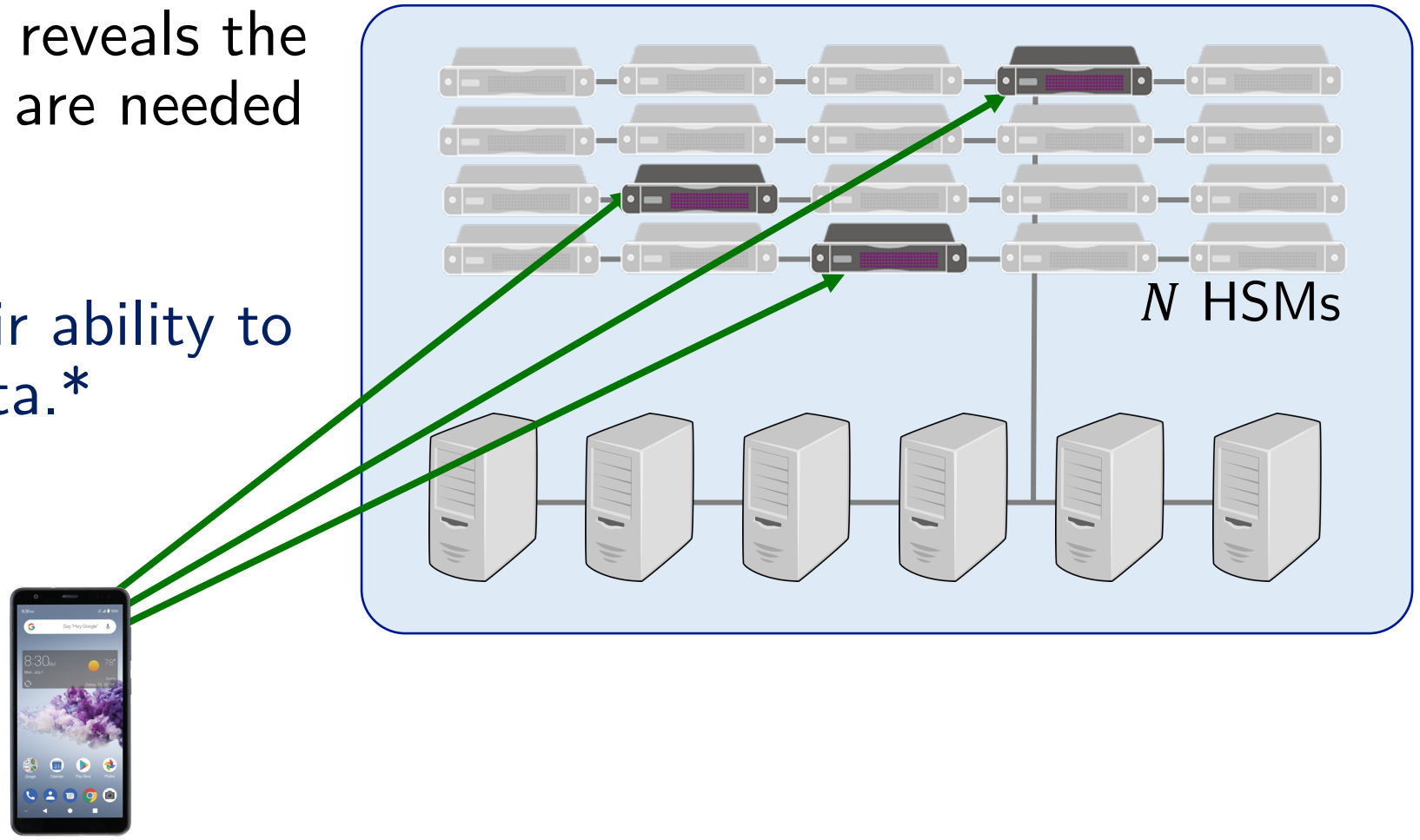
Our scheme:	**0.65**	sec
Naïve scheme:	**2,880.**	sec



Handling post-recovery compromise

During recovery, client reveals the ~40 HSMs whose keys are needed to decrypt its backup.

HSMs then revoke their ability to decrypt the client's data.*



This talk

- Motivation
- SafetyPin: Basic design
- Technical challenges
  - **After-the-fact compromise**
  - Rate limiting: Distributed log
- Evaluation

This talk

- Motivation
- SafetyPin: Basic design
- Technical challenges
  - After-the-fact compromise
  - **Rate limiting: Distributed log**
- Evaluation

Distributed rate limiting

- HSMs must limit the number of PIN guesses on each user's account
  → Prevent brute-force PIN-guessing attacks
- BUT no single HSM has a global view of users' recovery attempts

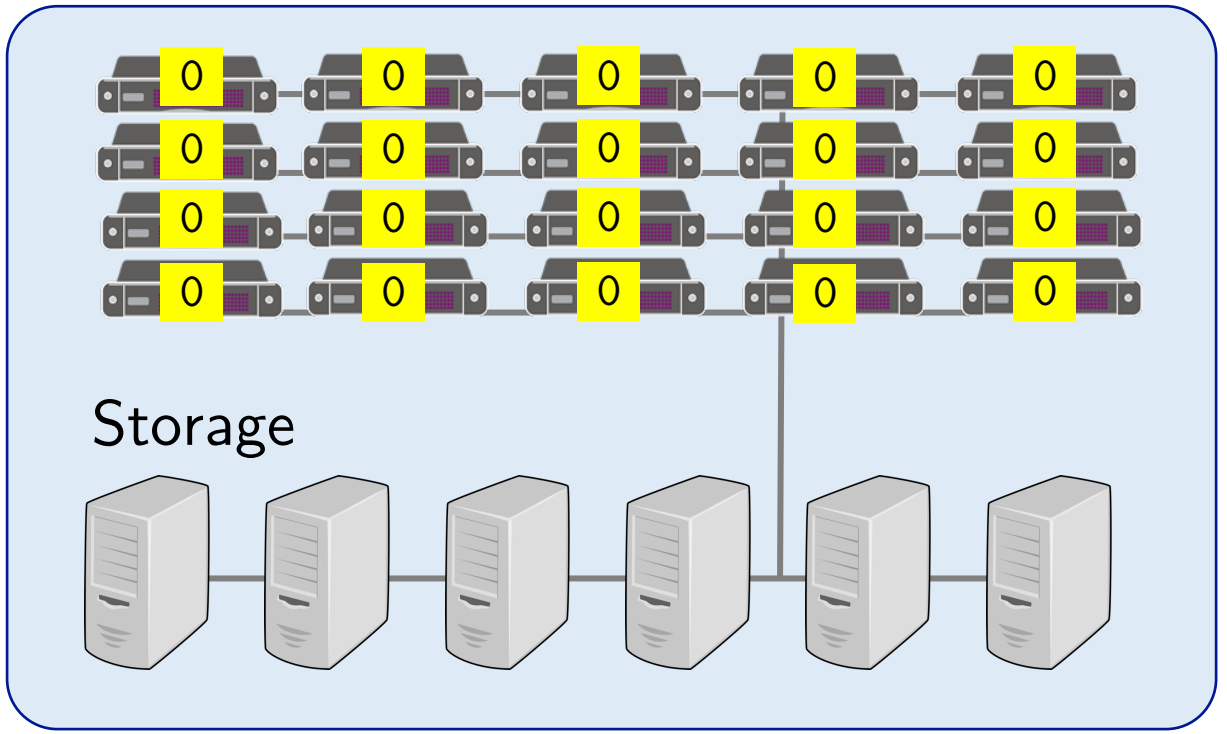
Our idea:

- Data center maintains a log of users' decryption requests
- HSMs collectively check the data center's work

Distributed rate-limiting is important

If each HSM keeps a local guess counter, can't prevent brute-force PIN-guessing attack

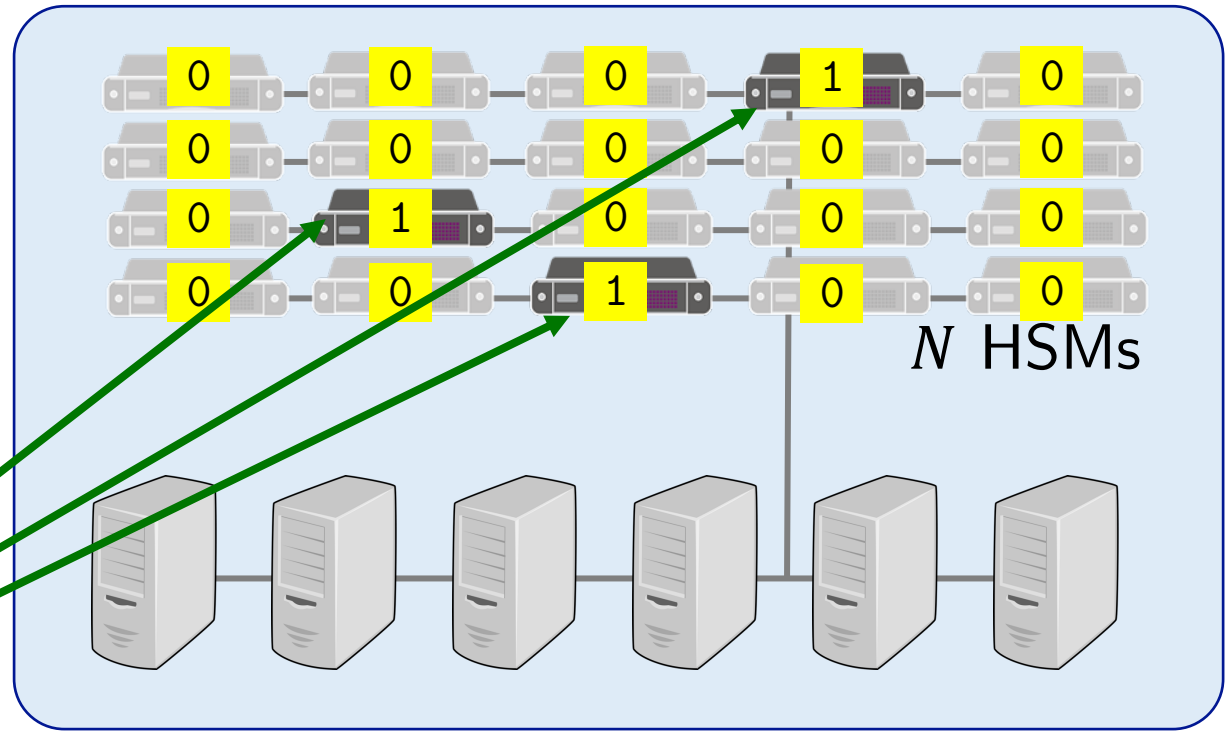
$(i_1, \ldots, i_{40}) \leftarrow \mathrm{Hash}(\text{“joe”}, 0000)$
$\mathrm{Hash}(\text{“joe”}, 0001)$
$\mathrm{Hash}(\text{“joe”}, 0002)$
...



Distributed rate-limiting is important

If each HSM keeps a local guess counter, can't prevent brute-force PIN-guessing attack

$(i_1, \ldots, i_{40}) \leftarrow$ **Hash("joe", 0000)**
Hash("joe", 0001)
Hash("joe", 0002)
...



Distributed rate-limiting is important

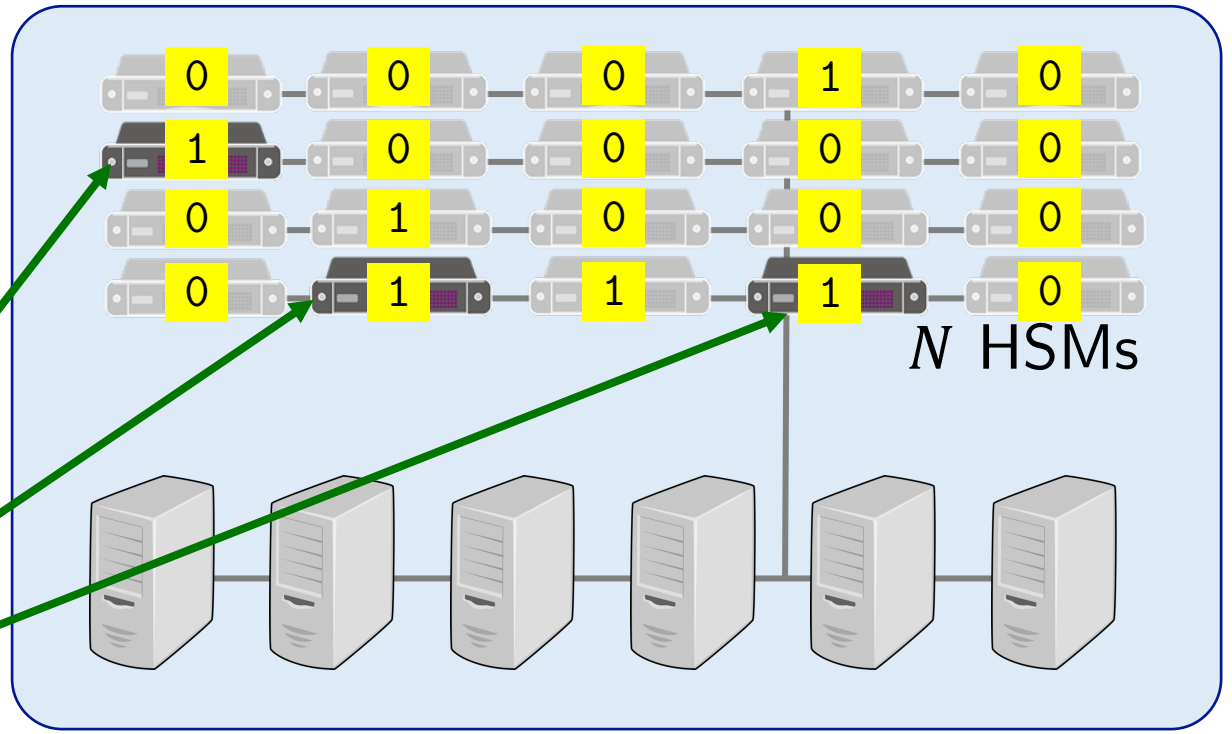
If each HSM keeps a local guess counter, can't prevent brute-force PIN-guessing attack

$(i_1, \ldots, i_{40}) \leftarrow \text{Hash}(\text{“joe”}, 0000)$

$\textbf{Hash}(\textbf{“joe”}, \textbf{0001})$

$\text{Hash}(\text{“joe”}, 0002)$

…



Distributed rate-limiting is important

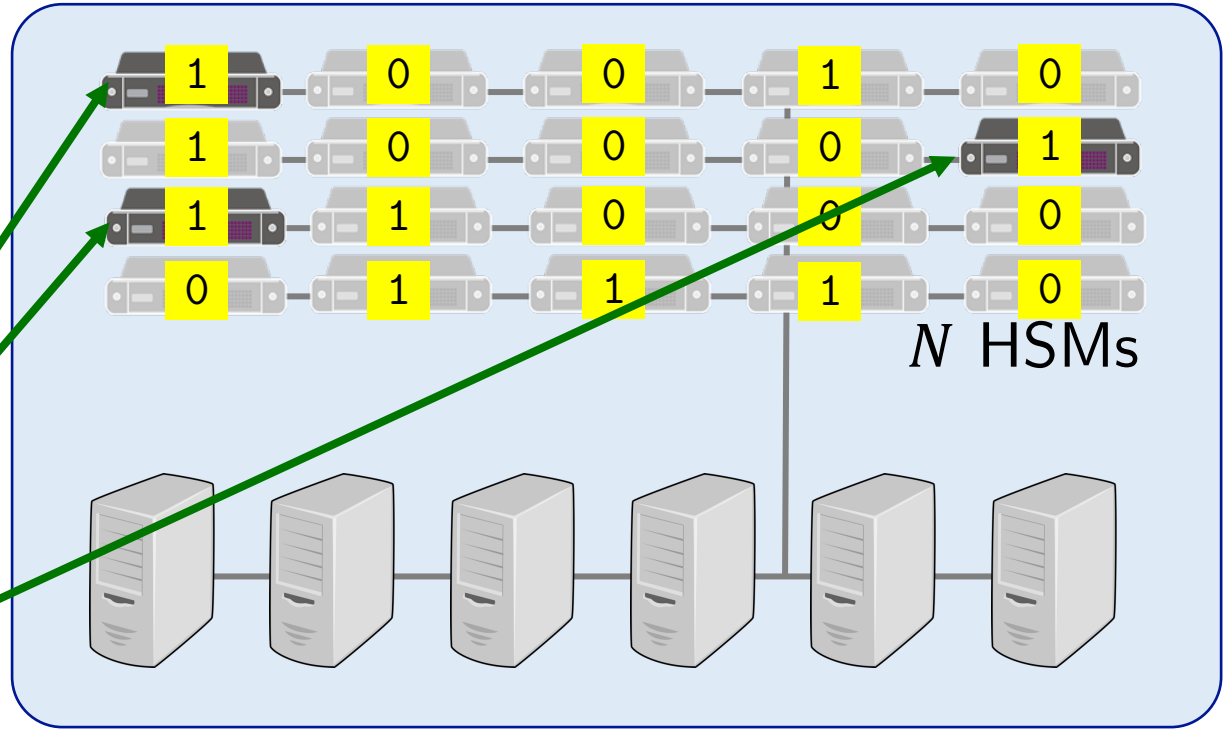
If each HSM keeps a local guess counter, can't prevent brute-force PIN-guessing attack

$(i_1, \ldots, i_{40}) \leftarrow \text{Hash}(\text{“joe”}, 0000)$

$\text{Hash}(\text{“joe”}, 0001)$

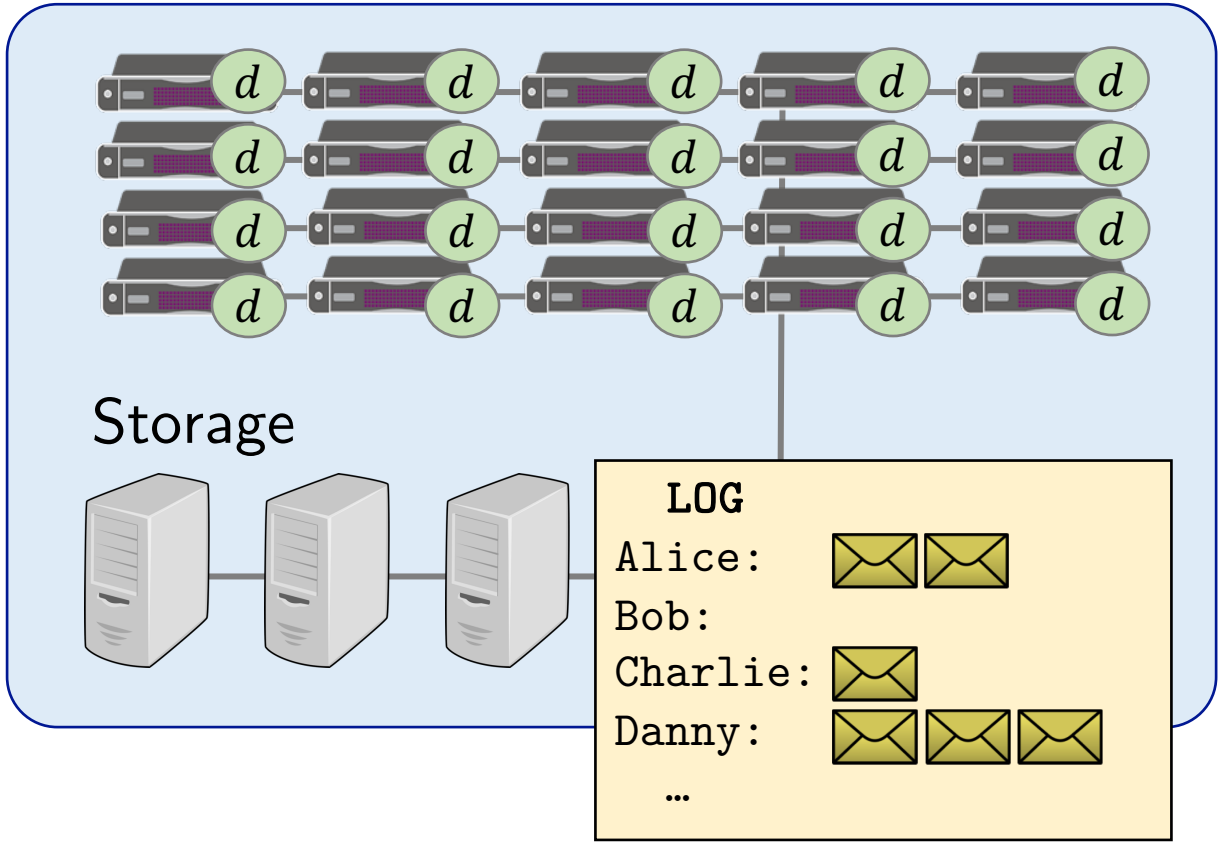
$\mathbf{\text{Hash}(\text{“joe”}, 0002)}$

...



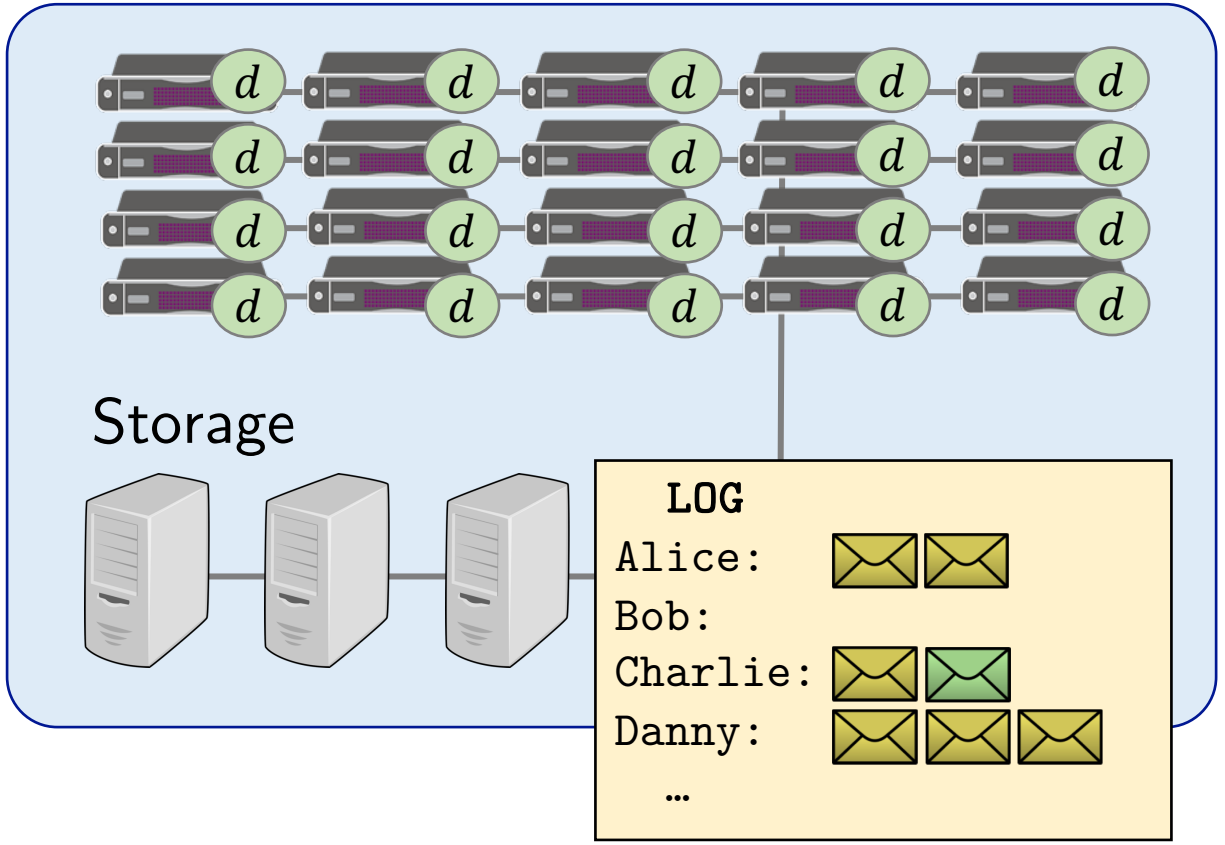
Using a log to enforce a global PIN-guess limit

- Each HSM holds root of a Merkle tree computed over log
- During recovery, client must **prove** that its decryption-request appears in the log
- Each client should be able to add ≤10 records to the log per month
  → Limits PIN guesses to 10 per month



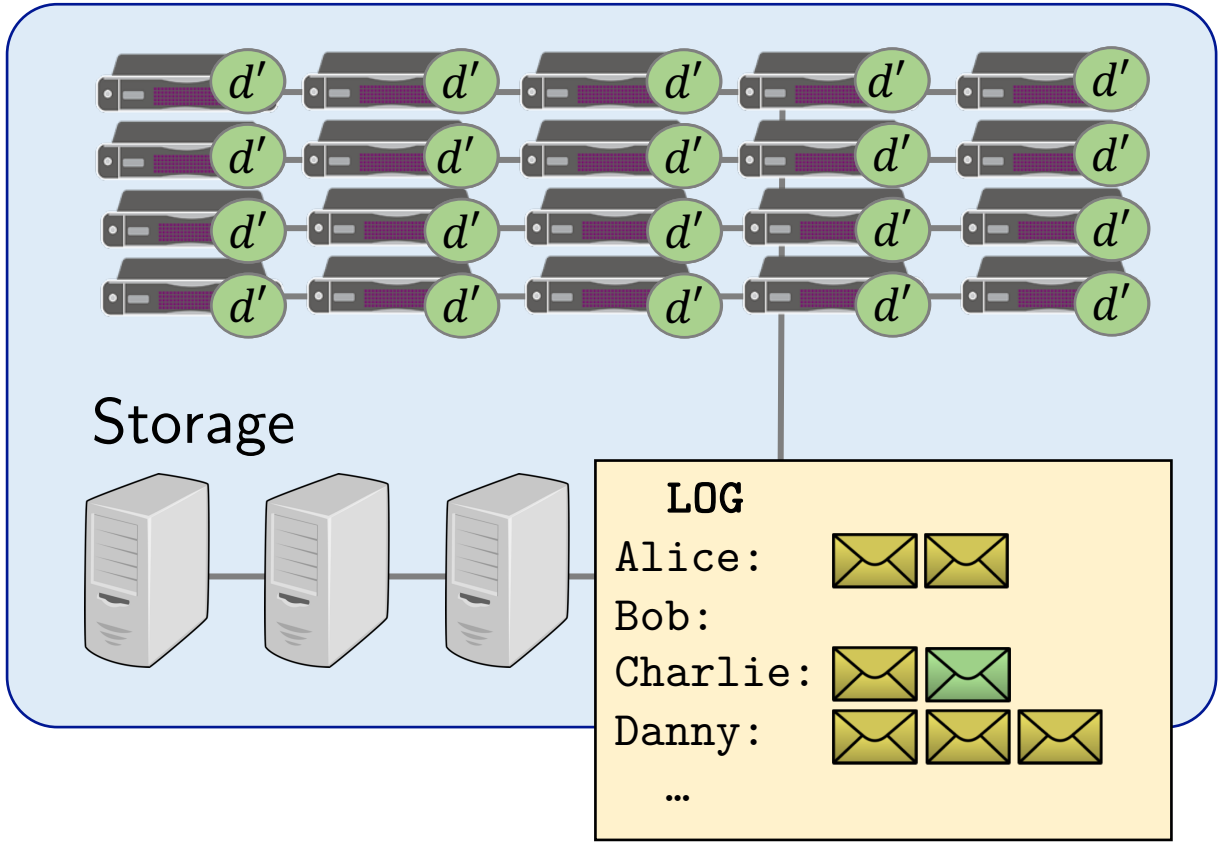
Maintaining the log

- When the data center inserts a record into the log, it proves to HSMs that the insertion was valid
  – Requires a pair of Merkle proofs
- For scalability, each HSM checks only $\approx \frac{128}{N}$ fraction of insertions
- Some tedious extra details to make this work (careful ordering, commitments, aggregate signatures, …)



Maintaining the log

- When the data center inserts a record into the log, it proves to HSMs that the insertion was valid
  – Requires a pair of Merkle proofs
- For scalability, each HSM checks only $\approx \frac{128}{N}$ fraction of insertions
- Some tedious extra details to make this work (careful ordering, commitments, aggregate signatures, …)



This talk

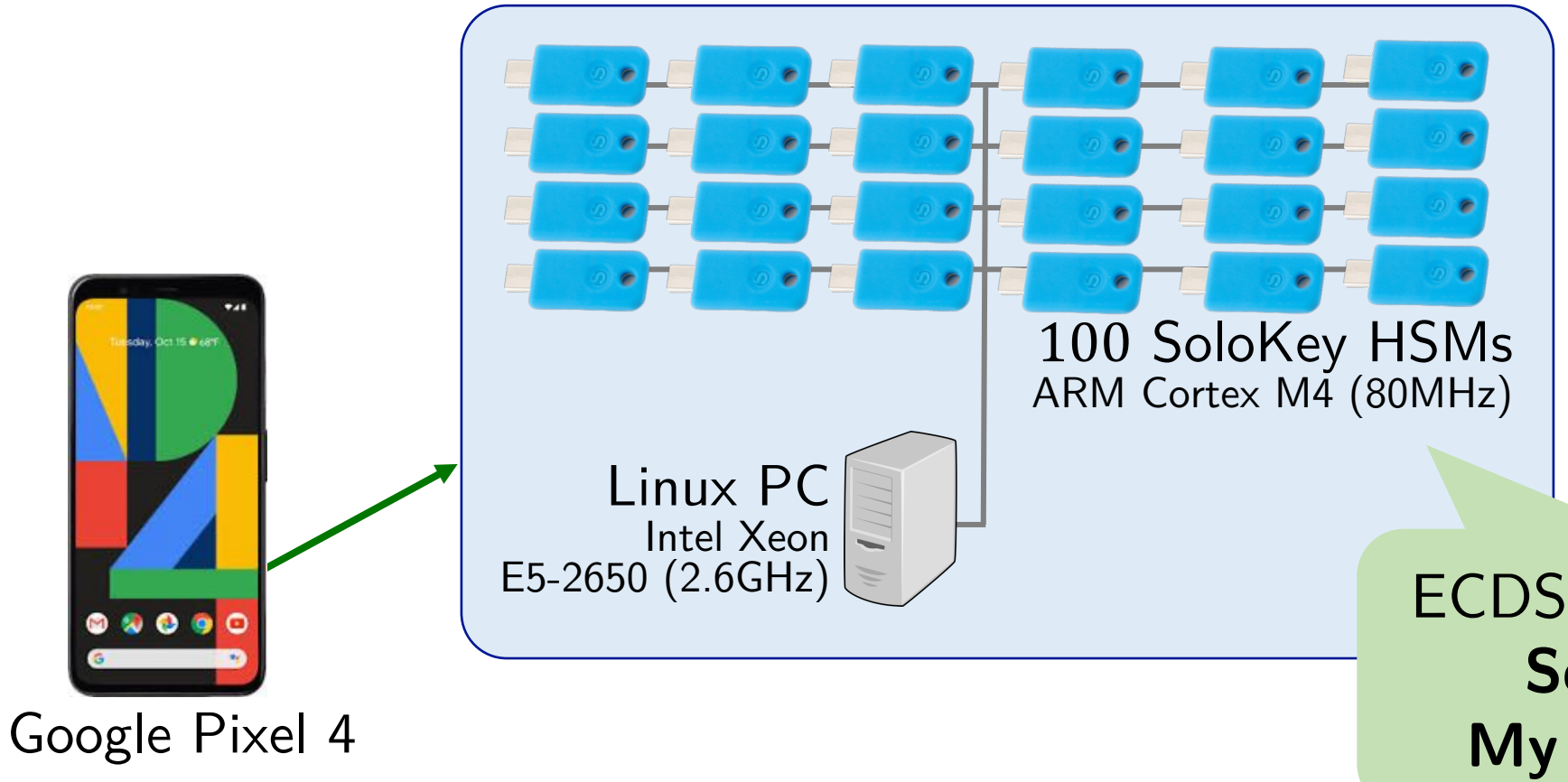
- Motivation
- SafetyPin: Basic design
- Technical challenges
  - After-the-fact compromise
  - **Rate limiting: Distributed log**
- Evaluation

This talk

- Motivation
- SafetyPin: Basic design
- Technical challenges
  - After-the-fact compromise
  - Rate limiting: Distributed log
- **Evaluation**

Experimental setup

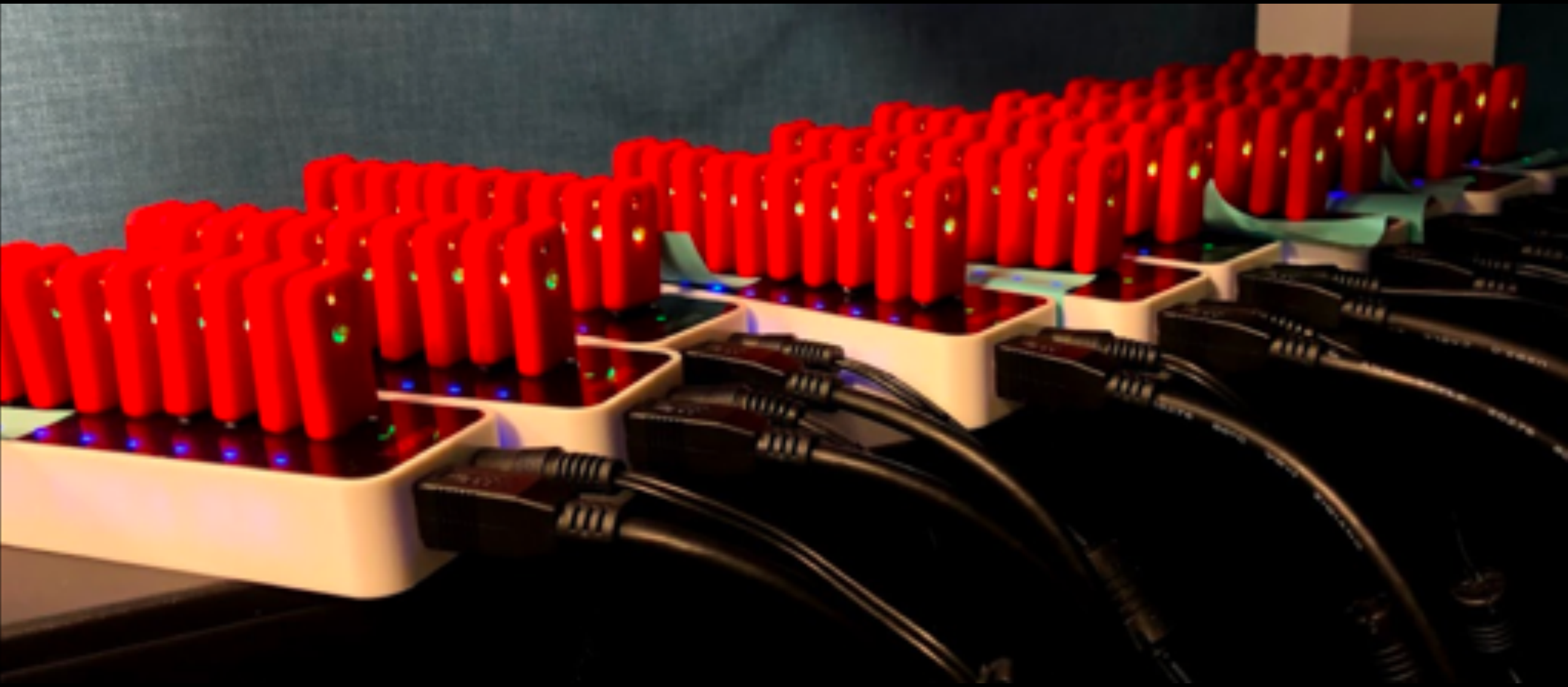
Code at: https://github.com/edauterman/SafetyPin



ECDSA256 verify/sec

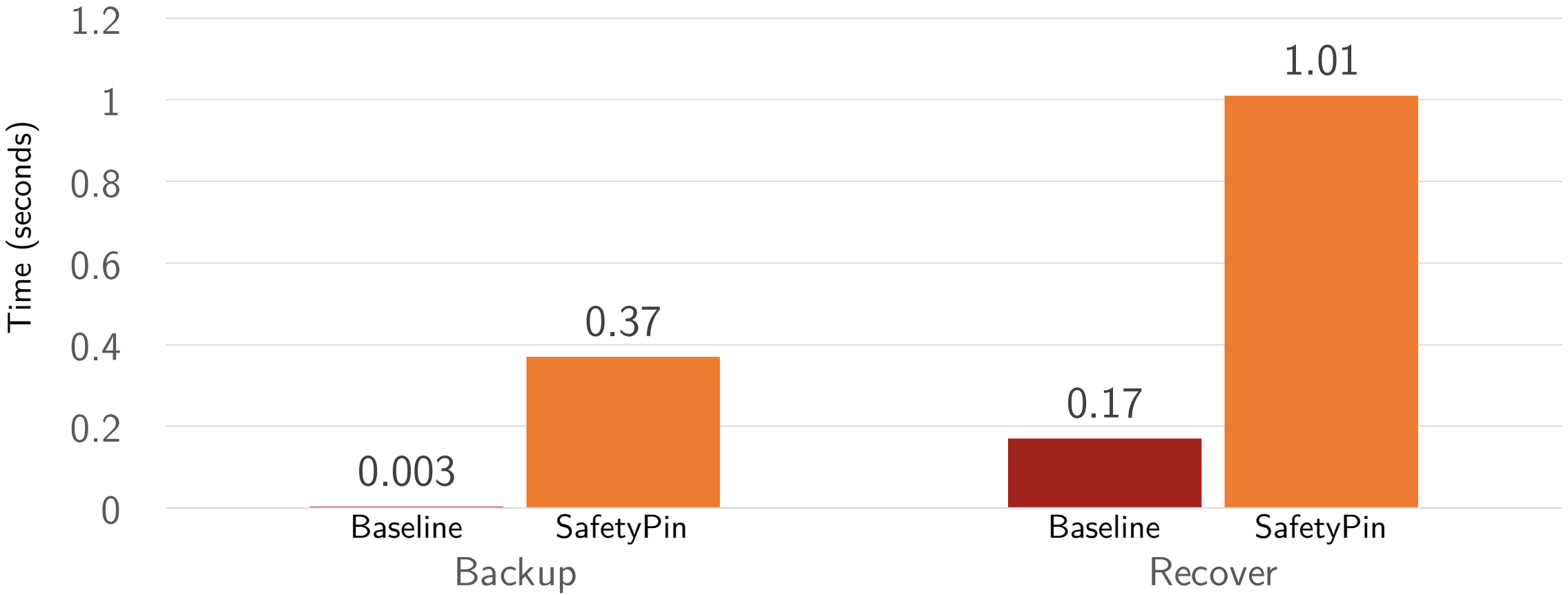
SoloKey: 7

My laptop: 14,566



End-to-end time is reasonable

(excludes time to encrypt disk image, unchanged)



Bandwidth cost

Each phone has to download **2MB of keying material per day**

- An artifact of the puncturable-encryption scheme we use
- Can happen overnight, while phone is plugged in

With fancier crypto, we can probably optimize this cost away...

Deployment-cost estimates

For a deployment supporting **one billion users**

Baseline

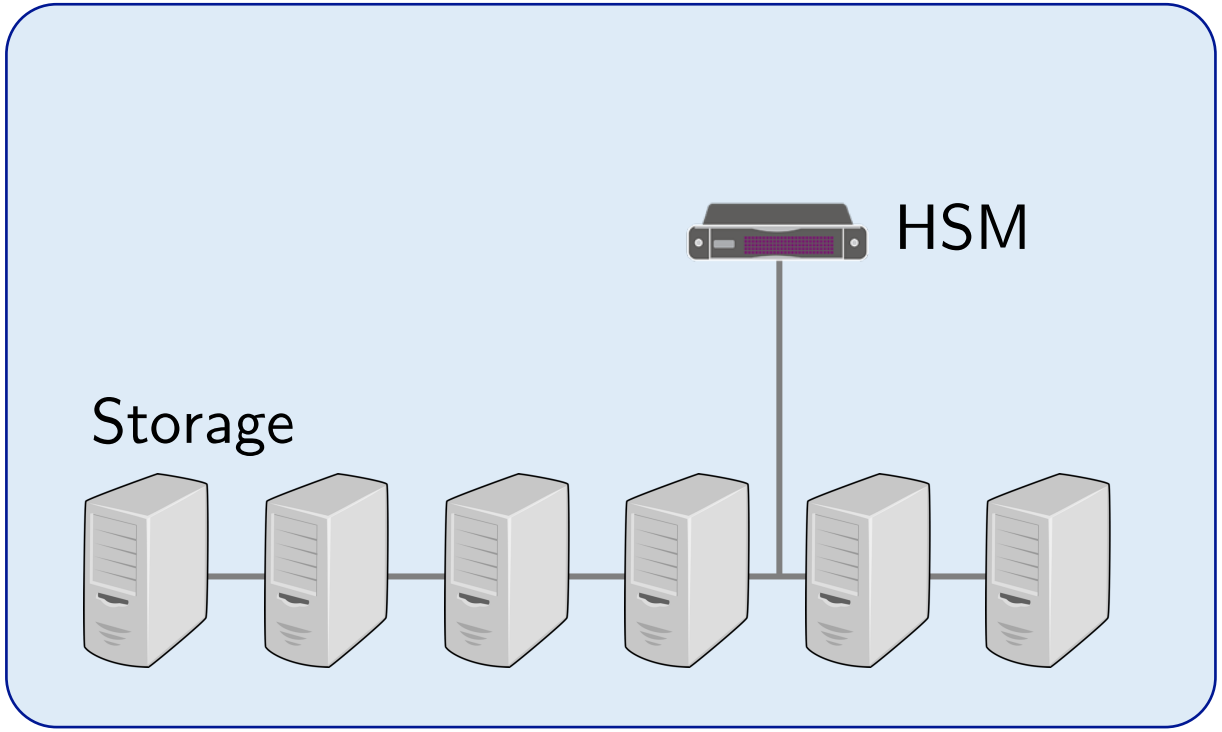
Cost of storing one 4GB backup per user per year: 600,000,000 USD

Additional SafetyPin costs

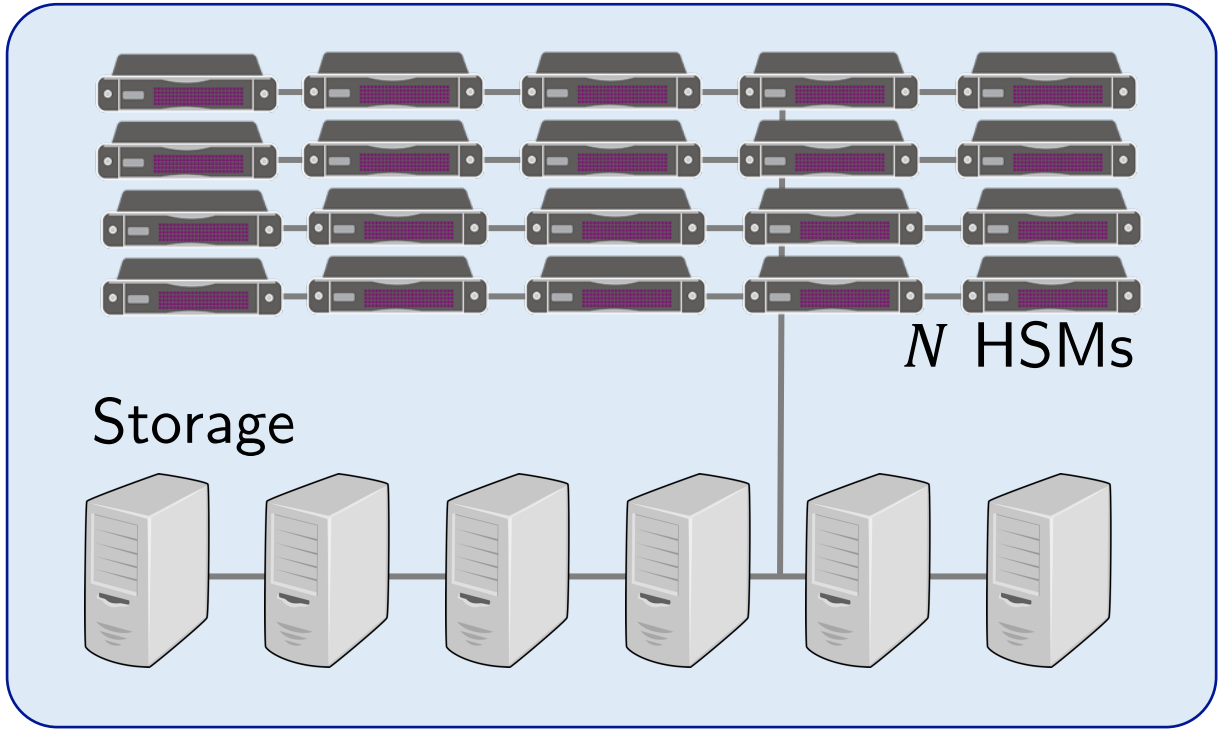
Using SoloKeys	60,700 USD
Using "industry-grade" HSMs (SafeNet A700)	14,800,000 USD

+2.5% increase

SafetyPin: Force attacker to compromise many HSMs



Today



SafetyPin

Conclusion

- Crypto hardware can help us build more trustworthy systems
  Example: HSM-based rate limiting for PIN-based encrypted backups
- BUT, we should remain strongly skeptical of "magic" hardware
  Implementation bugs? Hardware backdoors? Key-extraction attacks?
- Careful system design can give us the **benefits of secure hardware**, while protecting our our data from the **risks of hardware compromise**

Conclusion

- Crypto hardware can help us build more trustworthy systems
  Example: HSM-based rate limiting for PIN-based encrypted backups
- BUT, we should remain strongly skeptical of "magic" hardware
  Implementation bugs? Hardware backdoors? Key-extraction attacks?
- Careful system design can give us the **benefits of secure hardware**, while protecting our our data from the **risks of hardware compromise**

Emma Dauterman, HCG, David Mazières (OSDI 2020)
Paper: https://arxiv.org/abs/2010.06712
Code: https://github.com/edauterman/SafetyPin