DATA STORAGE SECURITY SUMMIT

01010011 01001110 01001001 01000001

SEPTEMBER 24, 2015
SANTA CLARA, CA

SNIA™

Reliable Expiration of Data from a Storage System

Radia Perlman
EMC

Note

- This is about storage in a cloud or file server
- Once the file is supposed to be gone (from the cloud/server) it will be unrecoverable (from the cloud/server or any backups that the cloud/server keeps)
- But this isn’t “DRM”. If an authorized client machine accesses the data and stores it in the clear, or prints a copy…this doesn’t solve that

This talk

- Two types of assured delete
  - Expiration date
  - On-demand, individual files
- Both are simple and practical
- But the on-demand is a bad idea! (I'll explain why, after explaining how to do it)

Expiration time

- When create data, put (optional) “expiration date” in metadata
- After expiration, data must be unrecoverable, even though backups will still exist

Obvious approach

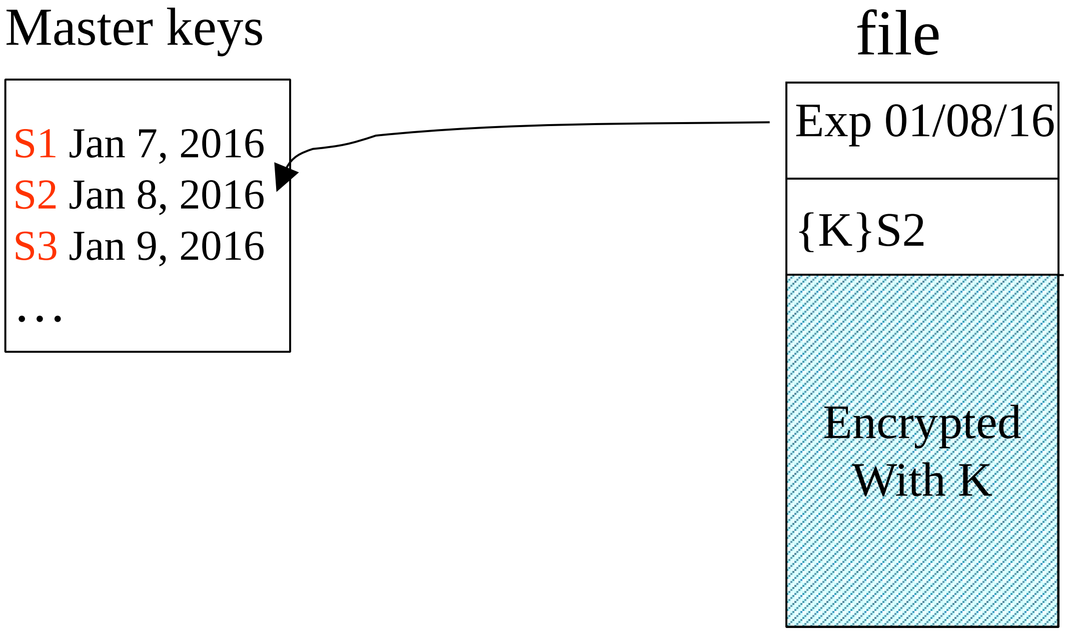
- Encrypt the data, and then destroy keys
- But to avoid prematurely losing data, you'd have to make lots of copies of the keys
- Which means it will be difficult to ensure all copies of backups of expired keys are destroyed

First concept: Encrypt all files with the same expiration date with the same key

File system with Master keys

Master keys: Secret keys (e.g., AES) generated by storage system

Delete key upon expiration



Master keys

S1 Jan 7, 2016

S2 Jan 8, 2016

S3 Jan 9, 2016

…

file

Exp 01/08/16

{K}S2

Encrypted With K

How many keys?

If granularity of one per day, and 30 years maximum expiration, 10,000 keys

So…how do you back up the master keys?

Imagine a service: An “ephemerizer”

- creates, advertises, protects, and deletes public keys
- Storage system “ephemerizes” each master key on backup, by encrypting with (same expiration date) ephemerizer public key
- To recover from backup: storage system asks ephemerizer to decrypt

Ephemerizer publicly posts

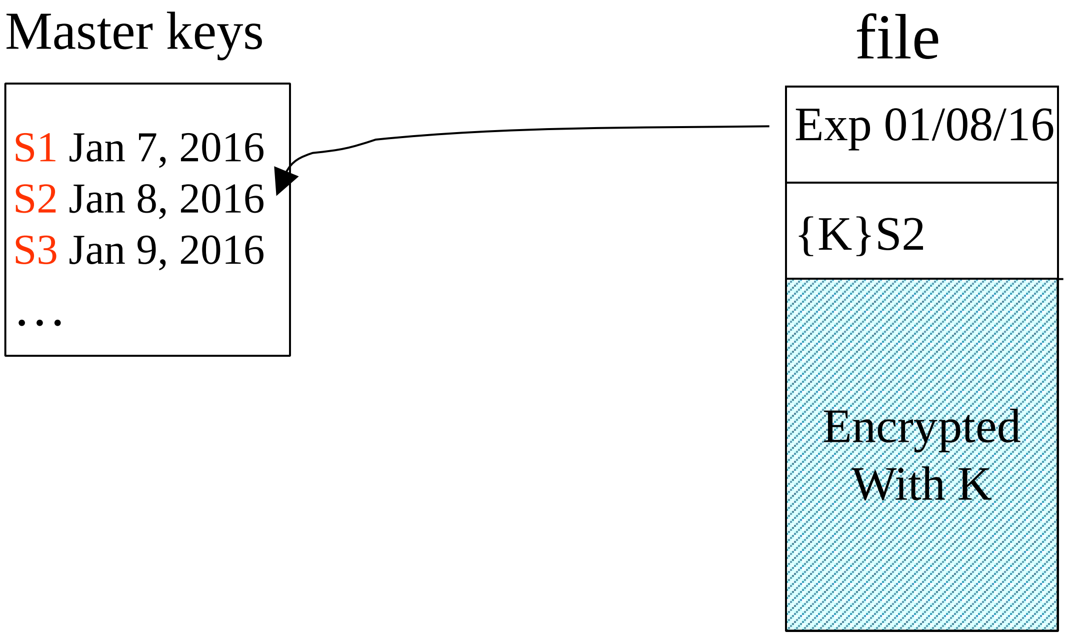
Jan 7, 2016: public key $P_{\text{Jan7of2016}}$
Jan 8, 2016: public key $P_{\text{Jan8of2016}}$
Jan 9, 2016: public key $P_{\text{Jan9of2016}}$
Jan 10, 2016: public key $P_{\text{Jan10of2016}}$

etc

Ephemerizer has permanent public key P certified through PKI
Signs the ephemeral keys with P

Storage system with Master keys

Master keys: Secret keys (e.g., AES) generated by storage system



Master keys

S1 Jan 7, 2016
S2 Jan 8, 2016
S3 Jan 9, 2016
…

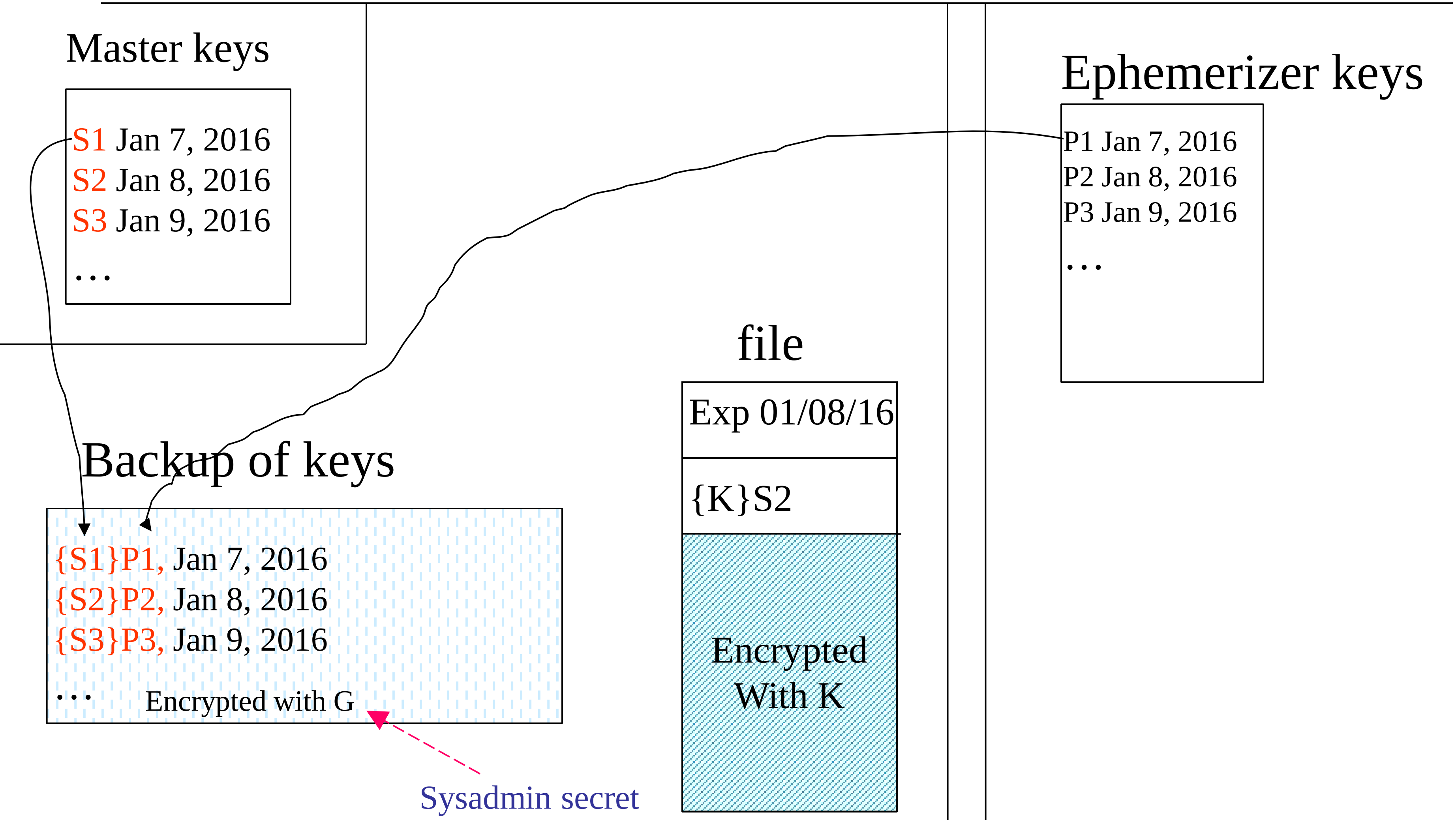
file

Exp 01/08/16

{K}S2

Encrypted With K

Backup of Master Keys



Master keys

S1 Jan 7, 2016
S2 Jan 8, 2016
S3 Jan 9, 2016
…

Ephemerizer keys

P1 Jan 7, 2016
P2 Jan 8, 2016
P3 Jan 9, 2016
…

file

Exp 01/08/16

{K}S2

Encrypted
With K

Backup of keys

{S1}P1, Jan 7, 2016
{S2}P2, Jan 8, 2016
{S3}P3, Jan 9, 2016

… Encrypted with G

Sysadmin secret

Notes

- Only talk to the ephemerizer if your hardware with master keys dies, and you need to retrieve master keys from backup
- Not every time you open a file!!
- Ephemerizer really scalable:
  - Same public keys for all customers (10,000 keys for 30 years, one per day)
  - Only talk to a customer perhaps every few years…to unwrap keys being recovered from backup

But you might be a bit annoyed at this point

But you might be a bit annoyed at this point

- Haven’t we simply pushed the problem onto the ephemerizer?
- It has to reliably keep private keys until expiration, and then reliably delete them

Two ways ephemerizer can “fail”

- Prematurely lose private keys
- Fail to forget private keys

Two ways ephemerizer can “fail”

- Prematurely lose private keys
- Fail to forget private keys
- Let’s worry about these one at a time…first worry about losing keys prematurely

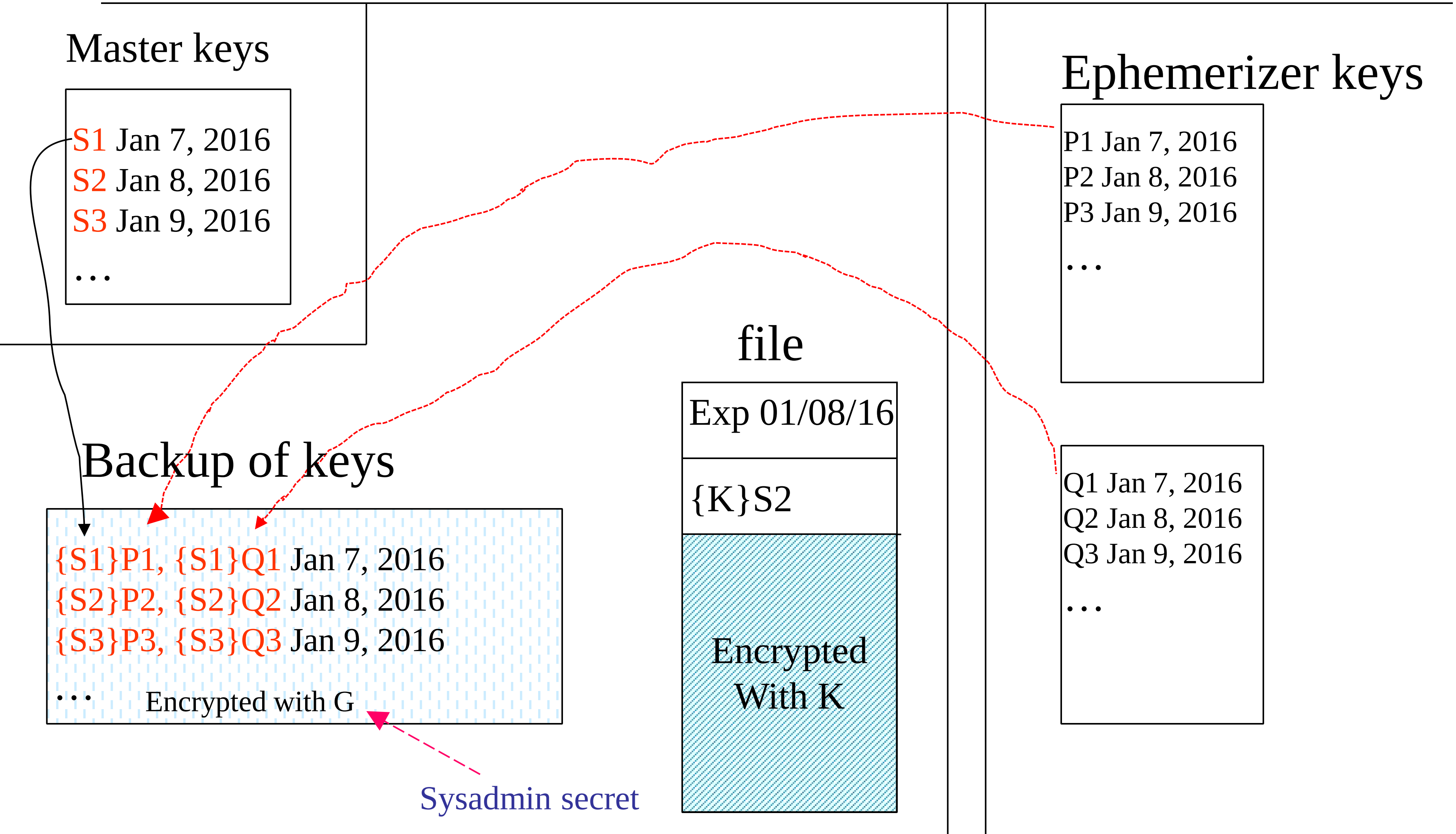
Losing keys prematurely

- We will allow an ephemerizer to be flaky, and lose keys
- Generate keys, and do decryption, on tamper-proof module
- An honest ephemerizer should not make copies of its ephemeral private keys
- So…wouldn’t it be a disaster if it lost its keys when a customer needs to recover from backup?

The reason why it's not just pushing the problem

- You can achieve arbitrary robustness by using enough "flaky" ephemerizers!
  - Independent ephemerizers
    - Different organizations
    - Different countries
    - Different continents
  - Independent public keys

Use multiple ephemerizers!



Master keys

S1 Jan 7, 2016
S2 Jan 8, 2016
S3 Jan 9, 2016
…

Ephemerizer keys

P1 Jan 7, 2016
P2 Jan 8, 2016
P3 Jan 9, 2016
…

Q1 Jan 7, 2016
Q2 Jan 8, 2016
Q3 Jan 9, 2016
…

file

Exp 01/08/16

{K}S2

Encrypted
With K

Backup of keys

{S1}P1, {S1}Q1 Jan 7, 2016
{S2}P2, {S2}Q2 Jan 8, 2016
{S3}P3, {S3}Q3 Jan 9, 2016
…

Encrypted with G

Sysadmin secret

Summarizing

- Only need ephemerizer after a disaster
- Ephemerizer really scalable: millions of customers; rarely talks to any of them
- Ephemerizer only needs 10,000 public keys (one per day, 30 years) regardless of number of customers
- Easy to build an ephemerizer; generate private keys and do decryptions in protected hardware, never making copies

General philosophy

- Achieve robustness by lots of can-be-flaky components
- Failures are truly independent
  - Different organizations
  - Different administrators
  - Independent clocks

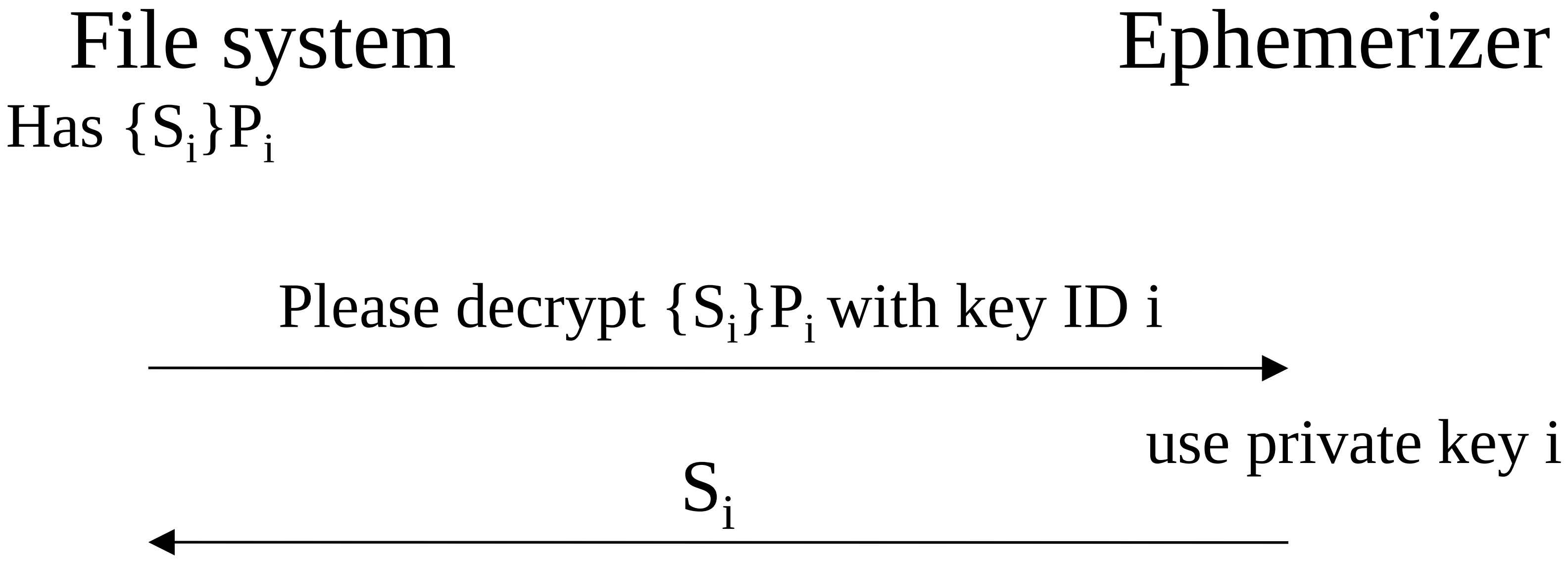
What if ephemerizer doesn’t forget expired private key?

- Then the storage system can use a quorum scheme (k out of n ephemerizers)
  - Break master key into n pieces, such that a quorum of k can recover it
  - Encrypt each piece with each of the n ephemerizers’ public keys

Asking ephemerizer to decrypt

- Cool protocol for asking the ephemerizer to decrypt
  - Which gives no information to the ephemerizer!

What we want to accomplish



File system

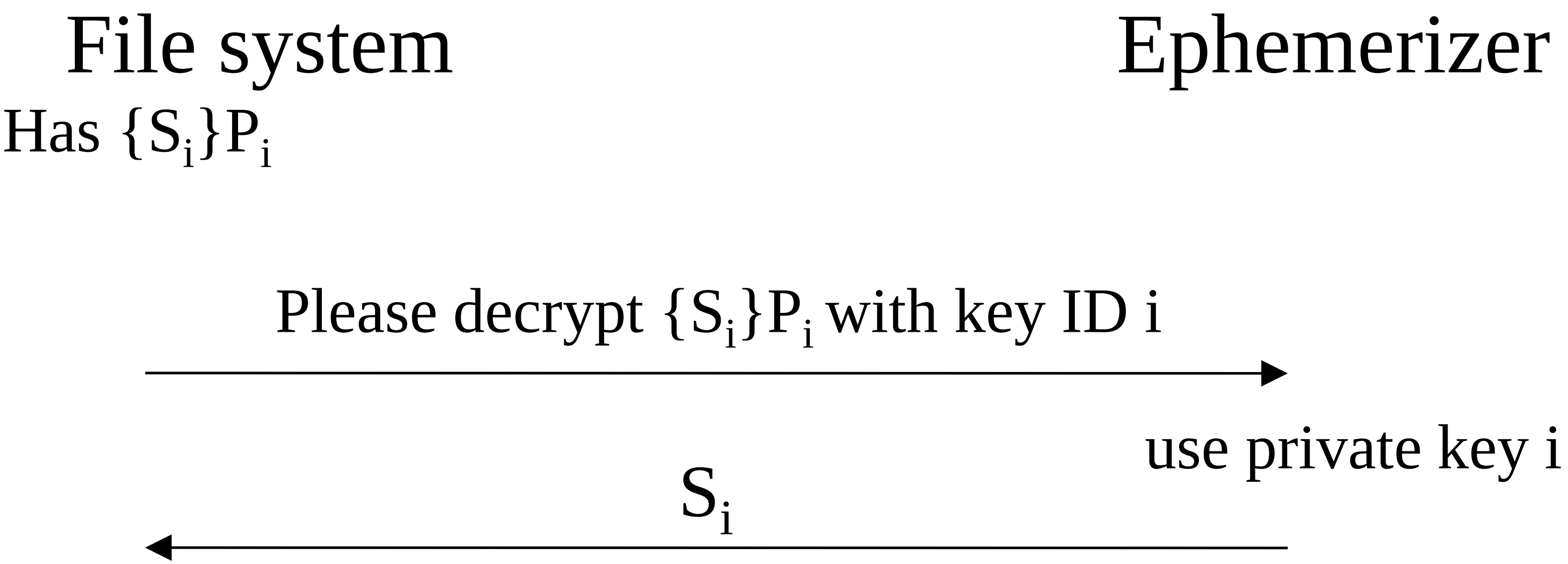
Ephemerizer

Has $\{S_i\}P_i$

Please decrypt $\{S_i\}P_i$ with key ID i

use private key i

S_i



File system

Ephemerizer

Has $\{S_i\}P_i$

Please decrypt $\{S_i\}P_i$ with key ID i

use private key i

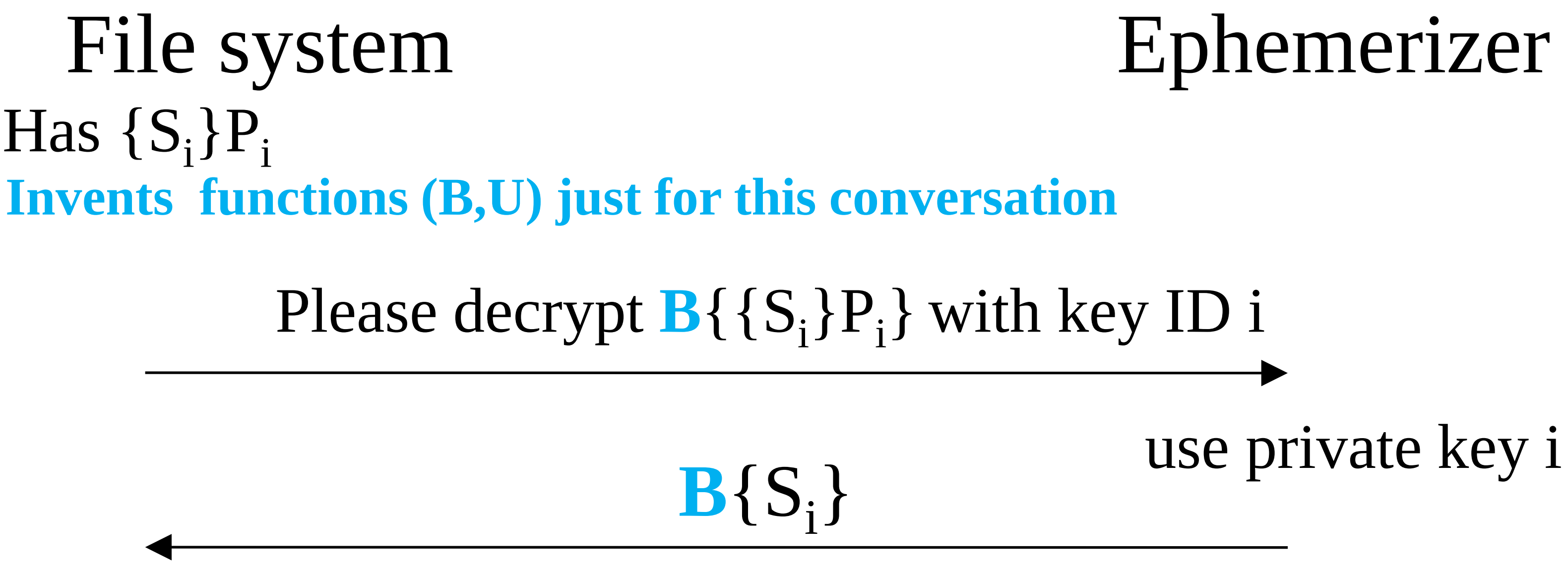
S_i

But we don’t want the Ephemerizer to see S_i

We’ll use “blind decryption”

- FS wants Eph to decrypt {Si}Pi with Eph’s private key #i
  - … Without Eph seeing what it is decrypting
- FS chooses inverse functions blind/unblind (B, U)
- encrypts (blinds) with Blind Function, which commutes with Eph’s crypto
- Then FS applies U to unblind

Using Blind Decryption



File system

Ephemerizer

Has $\{S_i\}P_i$

Invents functions (B,U) just for this conversation

Please decrypt **B**$\{\{S_i\}P_i\}$ with key ID i

use private key i

B$\{S_i\}$

File system applies U to get S_i

Ephemerizer only sees B$\{S_i\}$

Non-math fans can take a nap

For you math fans…

Quick review of RSA

- Public key is (e,n). Private key is (d,n), where e and d are “exponentiative inverses mod n”
- That means X^{ed} mod n=X
- Encrypt X with public key (e,n) means computing X^{e} mod n

Blind Decryption with RSA

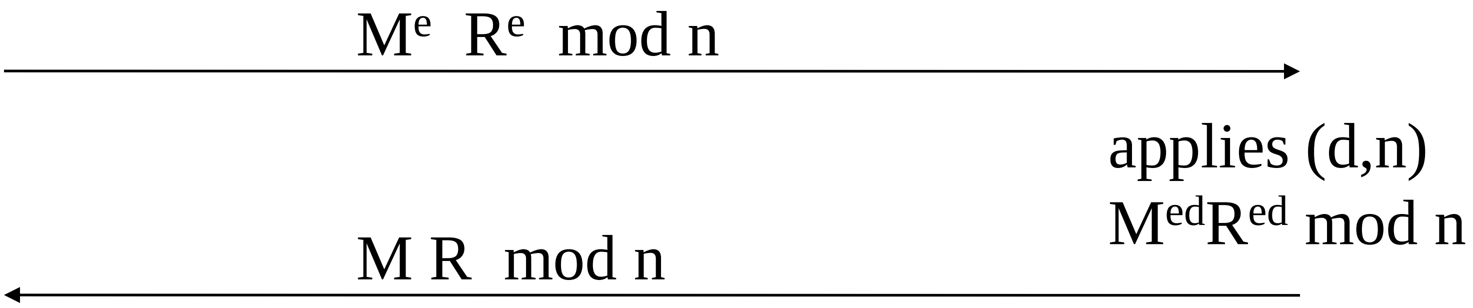
Ephemerizer’s RSA public key=(e,n), msg=M

Storage System

Ephemerizer

Knows encrypted M (M^e mod n). Wants M

Chooses random R, computes R^e mod n



$M^e \ R^e$ mod n

applies (d,n)

$M^{ed}R^{ed}$ mod n

M R mod n

divides by R mod n to get plaintext M

Other functions work

- For instance, there’s a Diffie-Hellman version that works with elliptic curves
- But for intuition, enough to see one function…

Properties of our protocol

- Ephemerizer gains no knowledge when it is asked to do a decryption
- Protocol is really efficient: one IP packet request, one IP packet response
- No need to authenticate either side
- Decryption can even be done anonymously

OK, non-math fans can wake up now

Because of blind decryption

- The customer does not need to run its own Ephemerizers, or really trust the Ephemerizers very much
- Ephemeral key management can be outsourced

Running an ephemerizer

- ❐ A customer *could* run some of its own ephemerizers—they should be fairly inexpensive and easy to manage
- ❐ But a customer might not be able to have enough of them in enough geographic locations for true robustness during disasters
- ❐ So it’s nice to use really remote ones if necessary

Outer encryption on ephemerized backup keys

❒ We need a global secret G

❒ Otherwise, anyone that got the encrypted backups could ask the Ephemerizer to decrypt

❒ G could be something like a sysadmin password, held in the head of multiple system administrators

To recover from backup

- To retrieve the state of the file system after a disaster you need:
  - G
  - The encrypted backups of the keys
  - The encrypted backups of the data
  - Help from the ephemerizer

Interaction with Ephemerizer

- Only need to bother Ephemerizer after a disaster, and do decryptions, one for each expiration date (i.e., 10,000 decryptions)
- But we can actually make it only one decryption after a disaster!

Another optimization

- Since the S’s are in a sequence…
- Make them derivable from each other, like with a one-way hash (or have file system store each successive S encrypted with previous S)
- That way, after a disaster, only have to talk to Ephemerizer once!

Variant: Custom Keys

Examples

- Company severs relations with a client; destroys all files
  - Keys can be nested; use time-based keys in that client’s folder
- Being sued; not allowed to delete anything; make makeup with custom key, then destroy custom key after suit done; key expirations will revert to time
- Spy: ship captured; tell ephemerizer not to decrypt with custom key

Custom Keys

- The ephemerizer's time-based key can be shared by many clients
- With custom keys, you need to keep a master key for each customer class
- You need to have the ephemerizers keep a key for each of your custom classes for you, so you can tell it when to delete it (or when to apply special authorization to use it for decryption)

Custom Keys

- Hopefully there would be a manageable number of these

Note: This isn’t DRM!

- DRM requires tamper-proof reader
- Ephemerization does not make DRM easier or harder

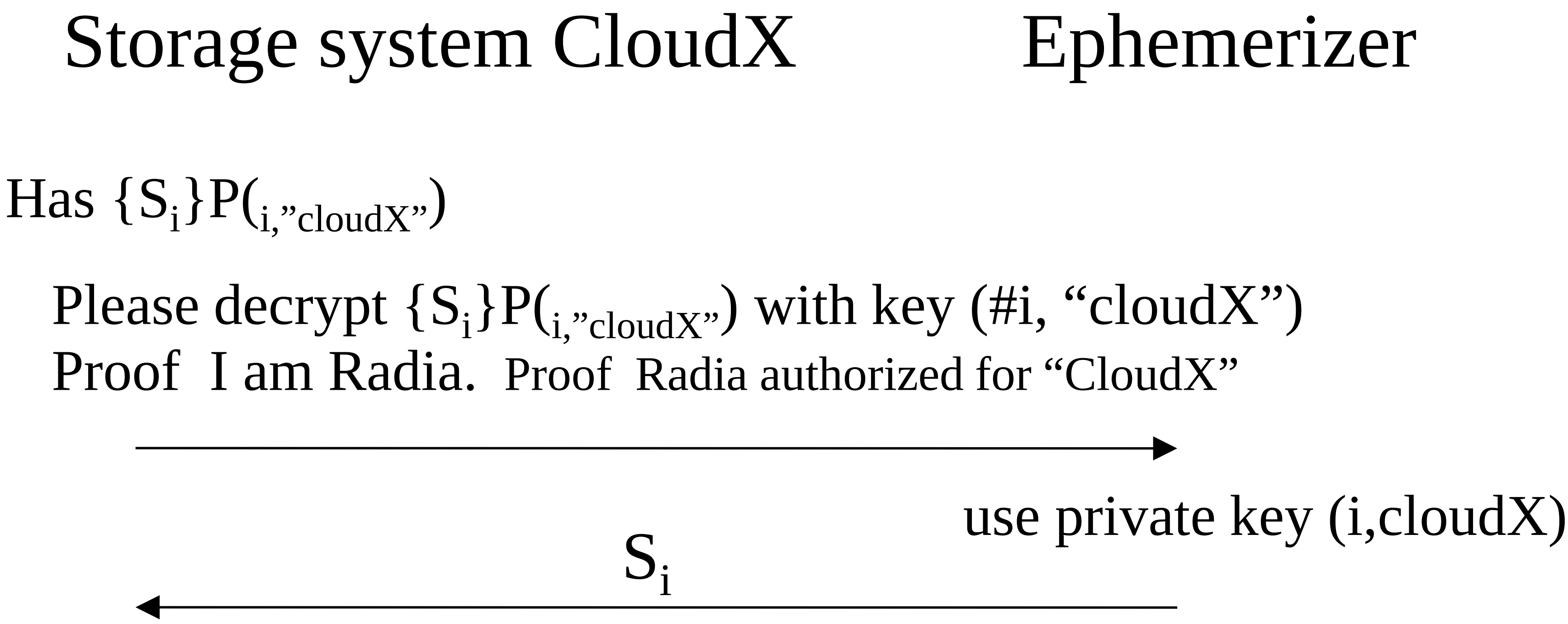
A subtle enhancement

- What if you fire system admin Fred?
- He might be able to find a backup of your keys, and he knows G
- Solution: Ephemerizer's key for Jan 1 isn't a single key, it's a family of keys, some function of
  - file system owner name
  - Advertised Ephemerizer parameters for that date's "public key"

Name-based public keys

- To “ephemerize” “CloudX”’s January 1 key, take the ephemerizer’s advertised value P, parameterize it with “CloudX” to get a public key $P_{CloudX/Jan1}$
- Encrypt with public key $P_{CloudX/Jan1}$
- To get ephemerizer to decrypt, you have to also prove you authorized to speak for “CloudX”
- That credential can be revoked through the PKI
  - [“Radia” is authorized to speak for CloudX] signed by CA

Name-based Public Keys



Storage system CloudX

Ephemerizer

Has $\{S_i\}P(_{\text{i,”cloudX”}})$

Please decrypt $\{S_i\}P(_{\text{i,”cloudX”}})$ with key (#i, “cloudX”)
Proof I am Radia. Proof Radia authorized for “CloudX”

use private key (i,cloudX)

S_i

To reduce slide clutter I left out blinding

- But of course you still need blinding
- In addition to being able to parameterize a key pair with a name

What functions work?

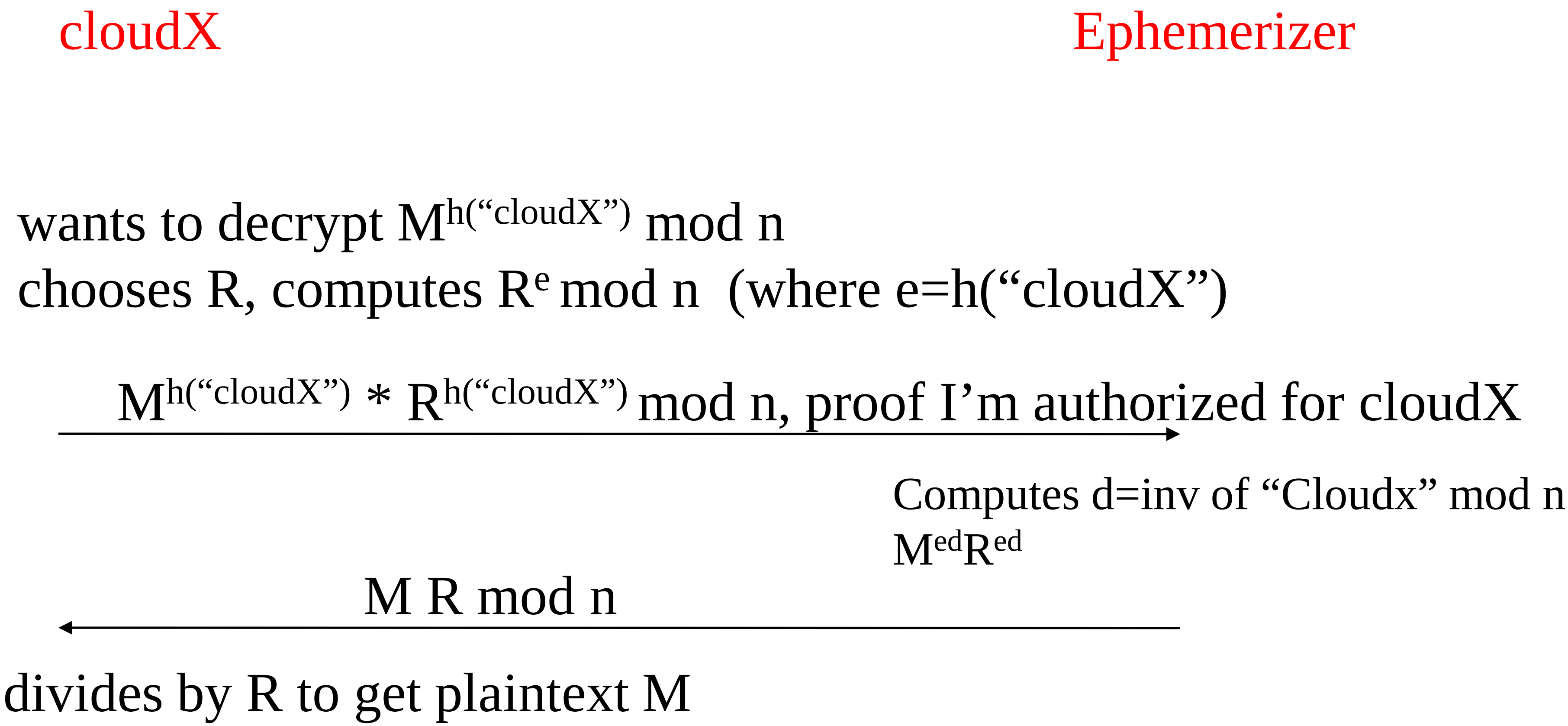
- The math of IBE (identity based encryption) (with separate “domain parameters” for each date)
- The Diffie-Hellman blindable math we omitted from these slides
- RSA variant which probably works…no known flaws (by me)…but easiest to explain…

Non-math fans can take a nap

RSA-based blindable parameterizable families

- Jan 1 “public key” isn’t (e,n): it’s just “n”
- Ephemerizer advertises: {(Jan 1, n1), (Jan 2, n2), …}
- The RSA encryption key “cloudx” uses for that date is (public exponent=h(“cloudx”), n)
  - $S^{h(\text{“cloudX”})}$ mod n
- The private key (known to the Ephemerizer) is knowledge of the factors of n (which enables the ephemerizer to compute exponentiative inverses mod n)

Blind Decryption with parameterized RSA



cloudX

Ephemerizer

wants to decrypt $M^{h(\text{“cloudX”})}$ mod n

chooses R, computes R^{e} mod n (where e=h(“cloudX”)

$M^{h(\text{“cloudX”})} * R^{h(\text{“cloudX”})}$ mod n, proof I’m authorized for cloudX

Computes d=inv of “Cloudx” mod n

$M^{ed}R^{ed}$

M R mod n

divides by R to get plaintext M

OK, non-math fans can wake up now

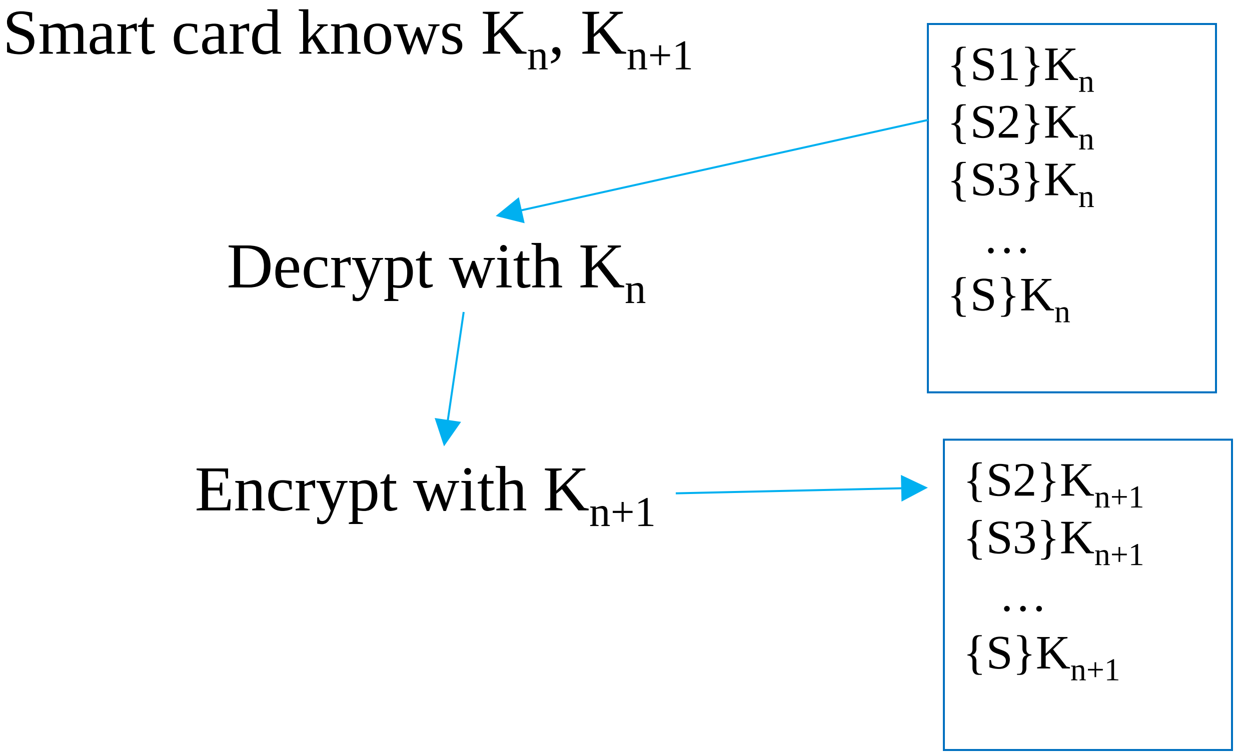
Another interesting (I hope) issue

- How to build an ephemerizer out of a dirt-cheap smart card
  - With limited storage, but attached to general purpose computer
- Smart card remembers two secret keys: current one and “next one”: K_n and K_{n+1}
- It generates public key pairs, encrypts the private key with K_n, and stores it on computer

How to forget one of the private keys

- Read in each private key (except the one you want to delete), one by one
- Decrypt with K_n
- Encrypt with K_{n+1}
- Store $\{S\}K_{n+1}$
- Forget K_n
- Generate K_{n+2}

Forgetting a key



Smart card knows K_n, K_{n+1}

$\{S1\}K_n$
$\{S2\}K_n$
$\{S3\}K_n$
…
$\{S\}K_n$

Decrypt with K_n

Encrypt with K_{n+1}

$\{S2\}K_{n+1}$
$\{S3\}K_{n+1}$
…
$\{S\}K_{n+1}$

New (small) topic

What if laws keep changing?

- ❐ Rather than file system keeping track of law that says this type of file must be retained for n years
- ❐ Have ephemerizer key based on (creation date, legal class) rather than expiration date
- ❐ Have ephemerizer destroy private key at the appropriate time

What if you get sued?

- You aren’t allowed to delete anything until the suit resolves
- So, you ask each ephemerizer to make you a (single) custom key (so if 3 ephemerizers, public keys P,Q,Z)
- You do a backup of all the (unexpired) master keys, “ephemerized” with P, Q, Z.
- After the suit is resolved, tell the ephemerizers to discard the private keys P, Q, Z.
- And the master keys go back to the original expirations

People kept wanting “on-demand” delete

- And I kept arguing that it was not useful, and wouldn’t be scalable

People kept wanting “on-demand” delete

- And I kept arguing that it was not useful, and wouldn’t be scalable
- But then I realized how to do it

People kept wanting “on-demand” delete

- And I kept arguing that it was not useful, and wouldn’t be scalable
- But then I realized how to do it
- And think it’s a really bad idea

People kept wanting “on-demand” delete

- And I kept arguing that it was not useful, and wouldn’t be scalable
- But then I realized how to do it
- And think it’s a really bad idea
- And it’s useful to see both how to do it, and why it’s a bad idea

On-demand delete

On-demand delete

- The previous design assumes
  - Key manager keeps one key for each expiration time
  - At file creation, you have to know its expiration
- What if you want to do on-demand delete?
- But then you wouldn’t be able to share keys…if you throw away a key, all files encrypted with the same key go away
- On the surface, seems much harder

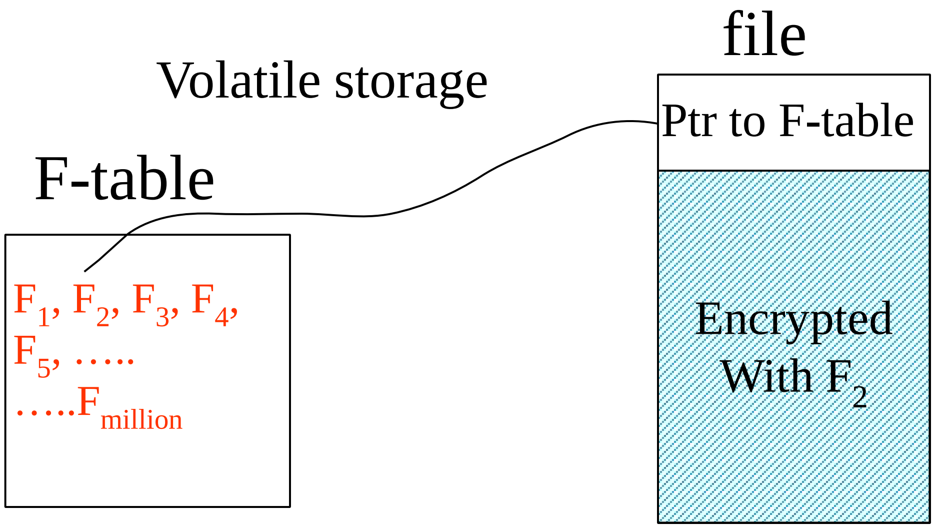
Instead of master keys

- ❒ Storage system keeps “F-table”, consisting of a secret key for each (expirable) piece of data
- ❒ Adds key to F-table when new (expirable) data stored
- ❒ Deletes key from F-table when (expirable) data is assuredly-deleted

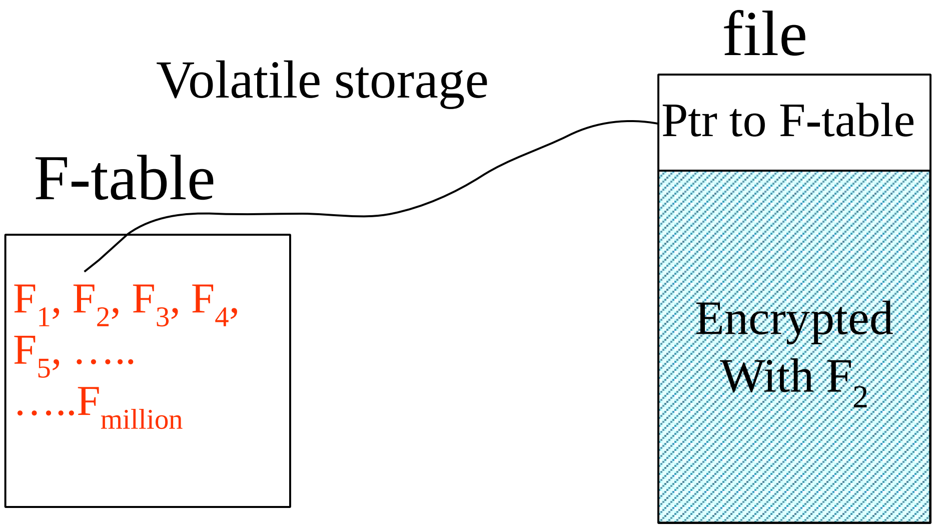
Ephemerizer state

- Needs to keep two public keys for each customer file system
  - current public key
  - previous public key

File system with F-table

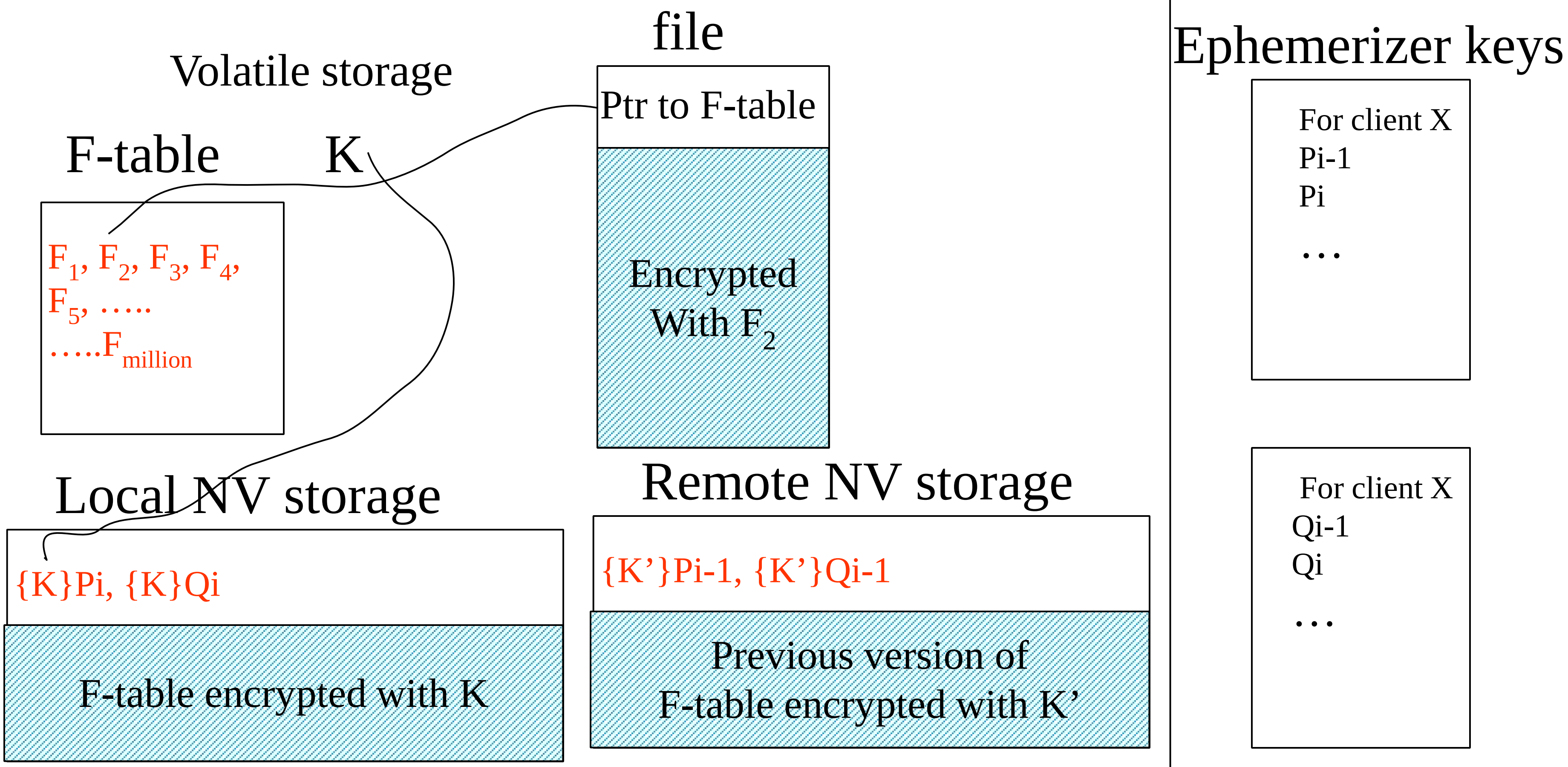


File system with F-table



Modify F-table when you assure-delete a file
Or create a new file
F-table has key for each file…if a million files, a million keys

File system with F-table



Volatile storage

F-table

K

F_1, F_2, F_3, F_4, F_5, …..

…..$F_{million}$

file

Ptr to F-table

Encrypted With F_2

Local NV storage

{K}Pi, {K}Qi

F-table encrypted with K

Remote NV storage

{K’}Pi-1, {K’}Qi-1

Previous version of F-table encrypted with K’

Ephemerizer keys

For client X
Pi-1
Pi
…

For client X
Qi-1
Qi
…

In theory…

- Ephemerizer could roll over keys on a schedule
- But then a week-long power failure could be a disaster

So what’s wrong?

My concern

- Suppose you change P's every week
- Suppose you find out that the file system was corrupted a month ago
- And that parts of the F-key database were corrupted, without your knowledge
- You can't go back

Why isn’t pre-determined expiration time as scary?

- If file system is not corrupted when a file is created, and the file is backed up, and the S-table is backed up, you can recover an unexpired file from backup
- Whereas with the on-demand scheme, if the file system gets corrupted, all data can get lost

Summary

- Use multiple independent ephemerizers with independent keys; ephemerizer keys need not be backed up
- Keys can be nested (expiring keys in a folder with custom keys)
- Not DRM (completely orthogonal to DRM)
- Time-based is probably the most useful

Questions?