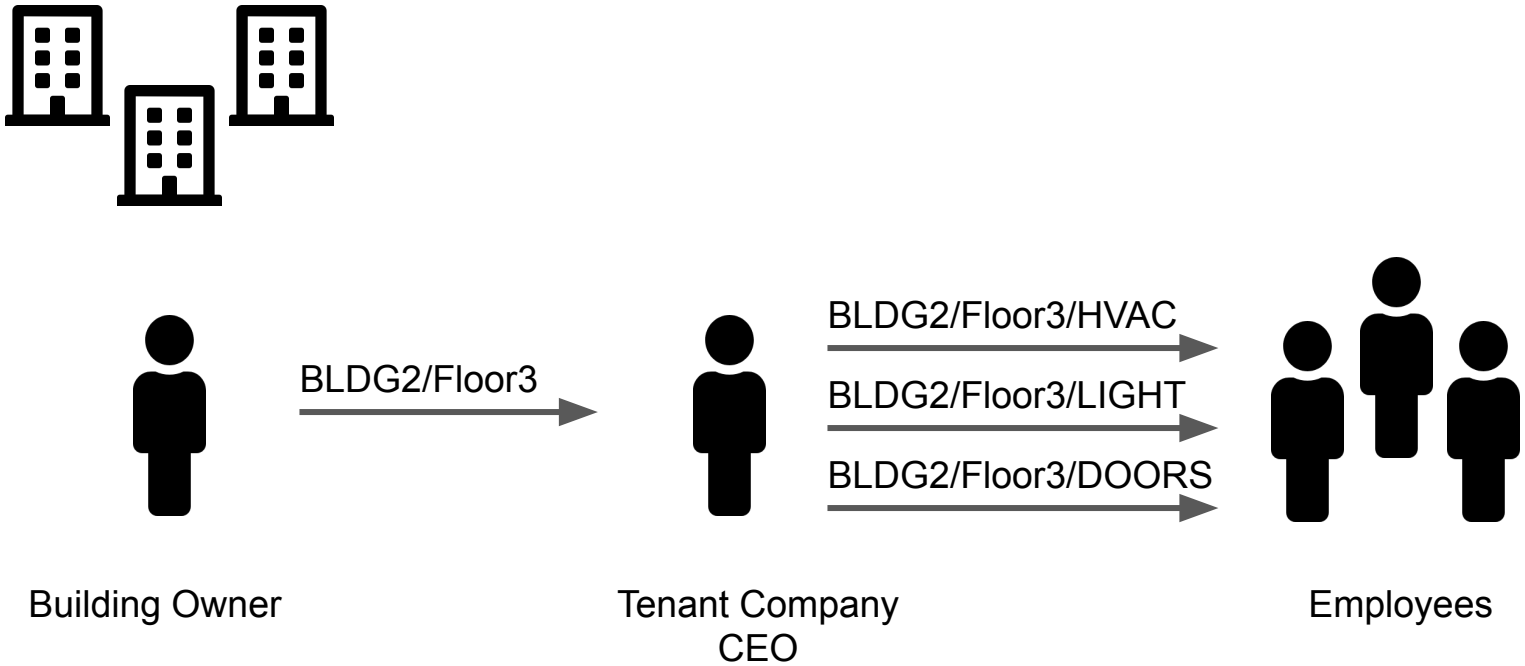
WAVE: A Decentralized Authorization Framework with Transitive Delegation

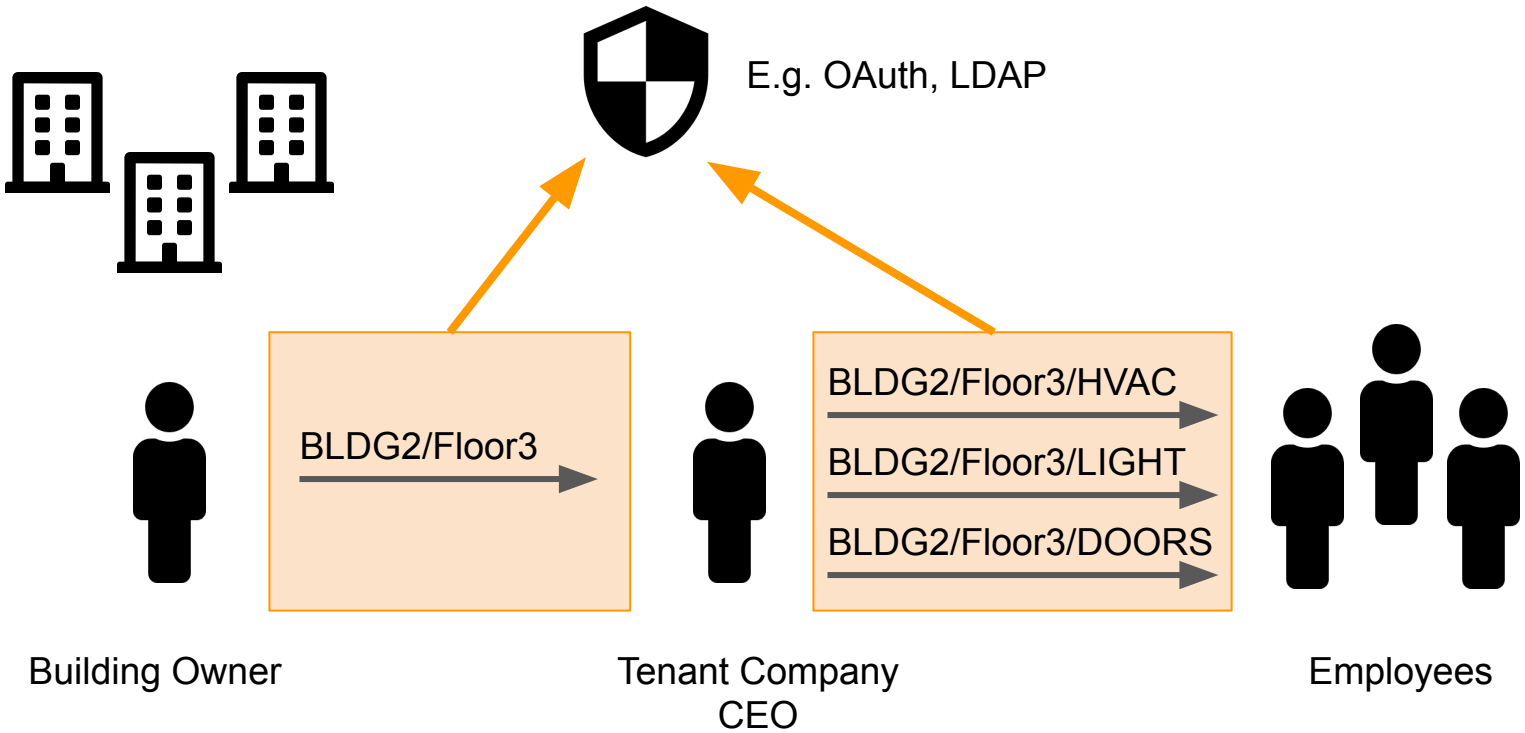
[Andersen et al., USENIX Security 2019]

Slides credit Michael Andersen

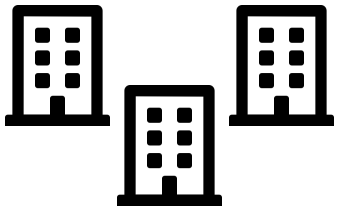
Representative authorization example

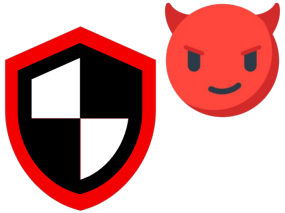


Traditional approach



Traditional approach





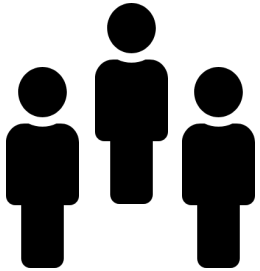
Problems:
Central point of attack

BLDG2/Floor3

BLDG2/Floor3/HVAC

BLDG2/Floor3/LIGHT

BLDG2/Floor3/DOORS

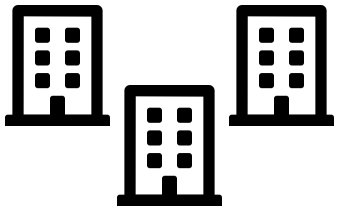


Building Owner

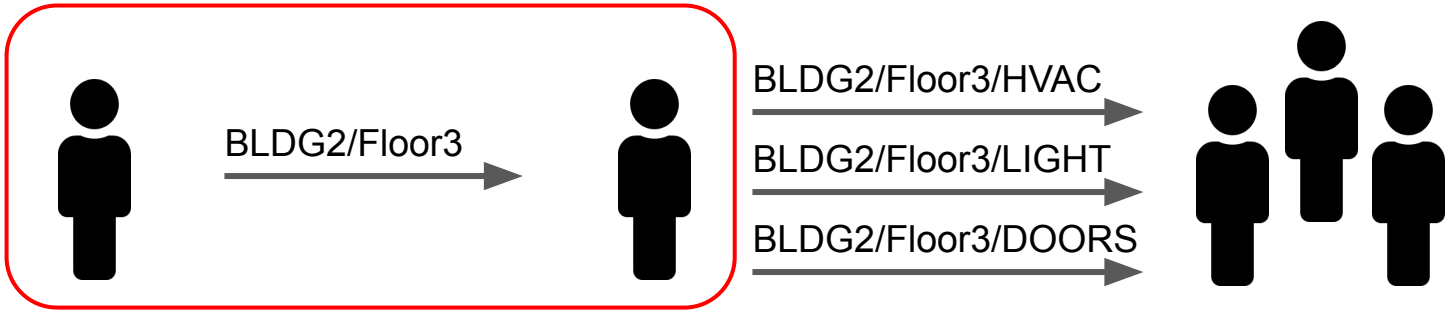
Tenant Company
CEO

Employees

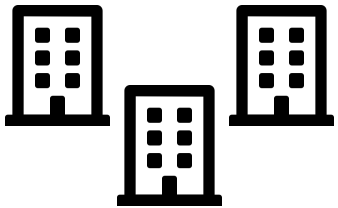
Traditional approach



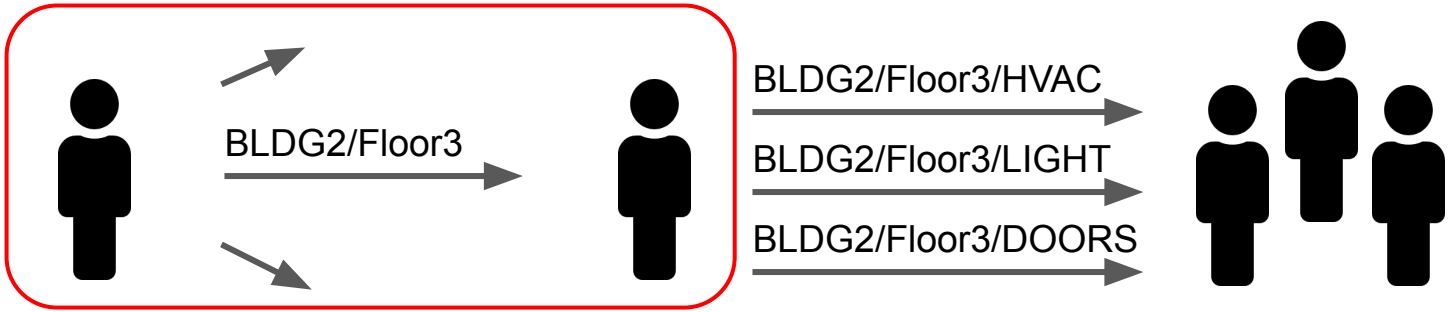
<u>Problems:</u>
Central point of attack
Can’t even trust operator



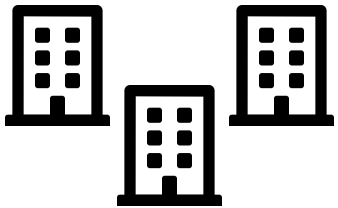
Traditional approach



<u>Problems:</u>
Central point of attack
Can’t even trust operator

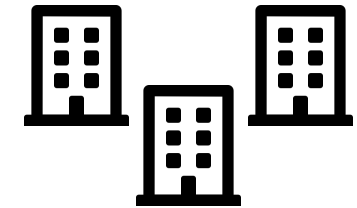


Traditional approach

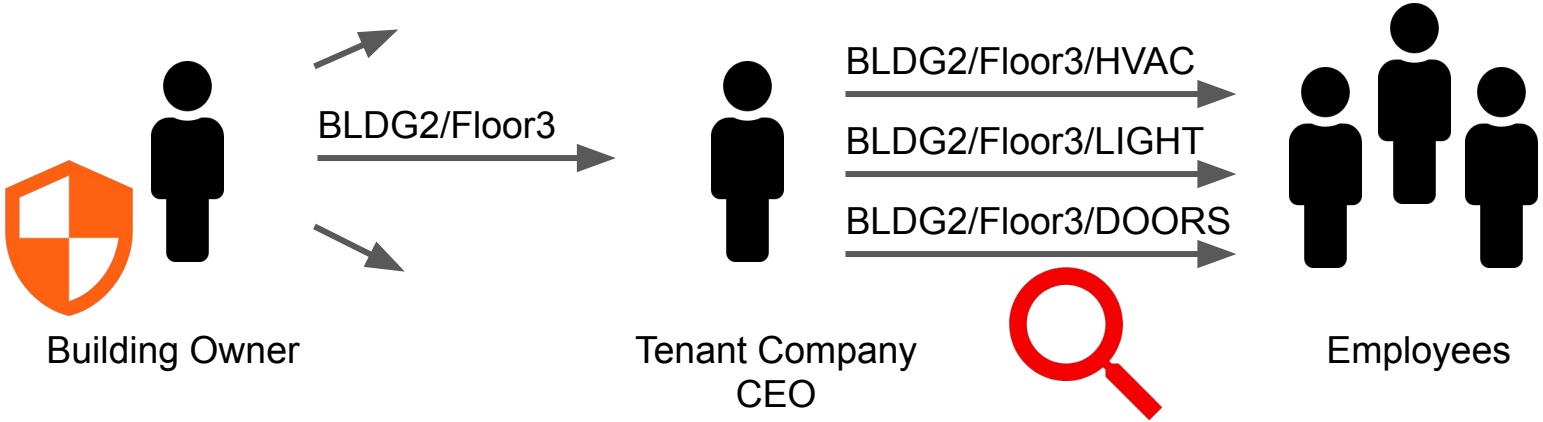


<u>Problems:</u>
Central point of attack
Can’t even trust operator

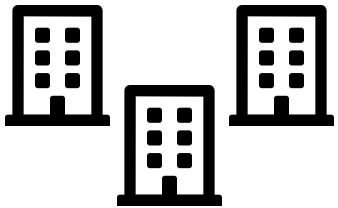
Traditional approach



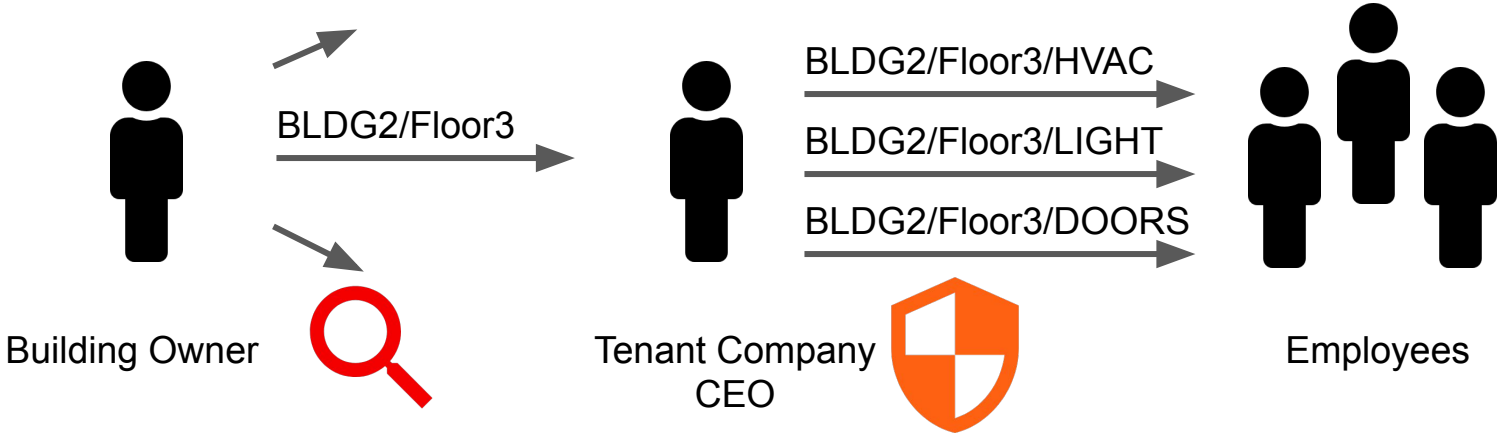
<u>Problems:</u>
Central point of attack
Can’t even trust operator



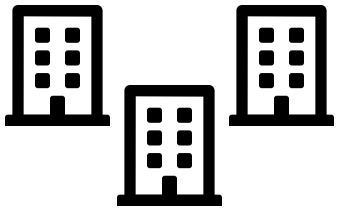
Traditional approach



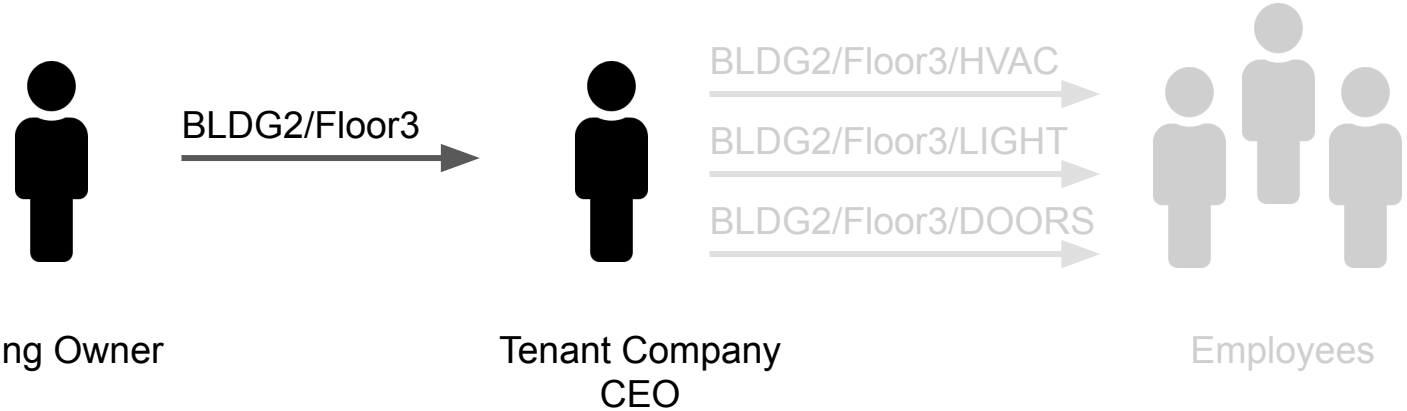
<u>Problems:</u>
Central point of attack
Can’t even trust operator



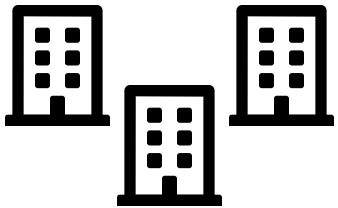
Traditional approach



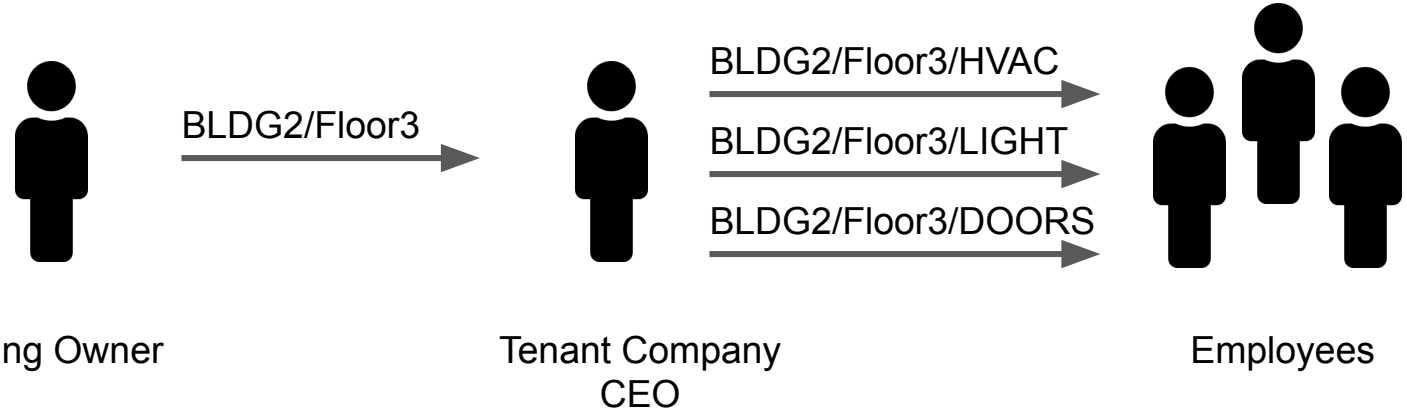
<u>Problems:</u>
Central point of attack
Can’t even trust operator
Sometimes delegation unsupported



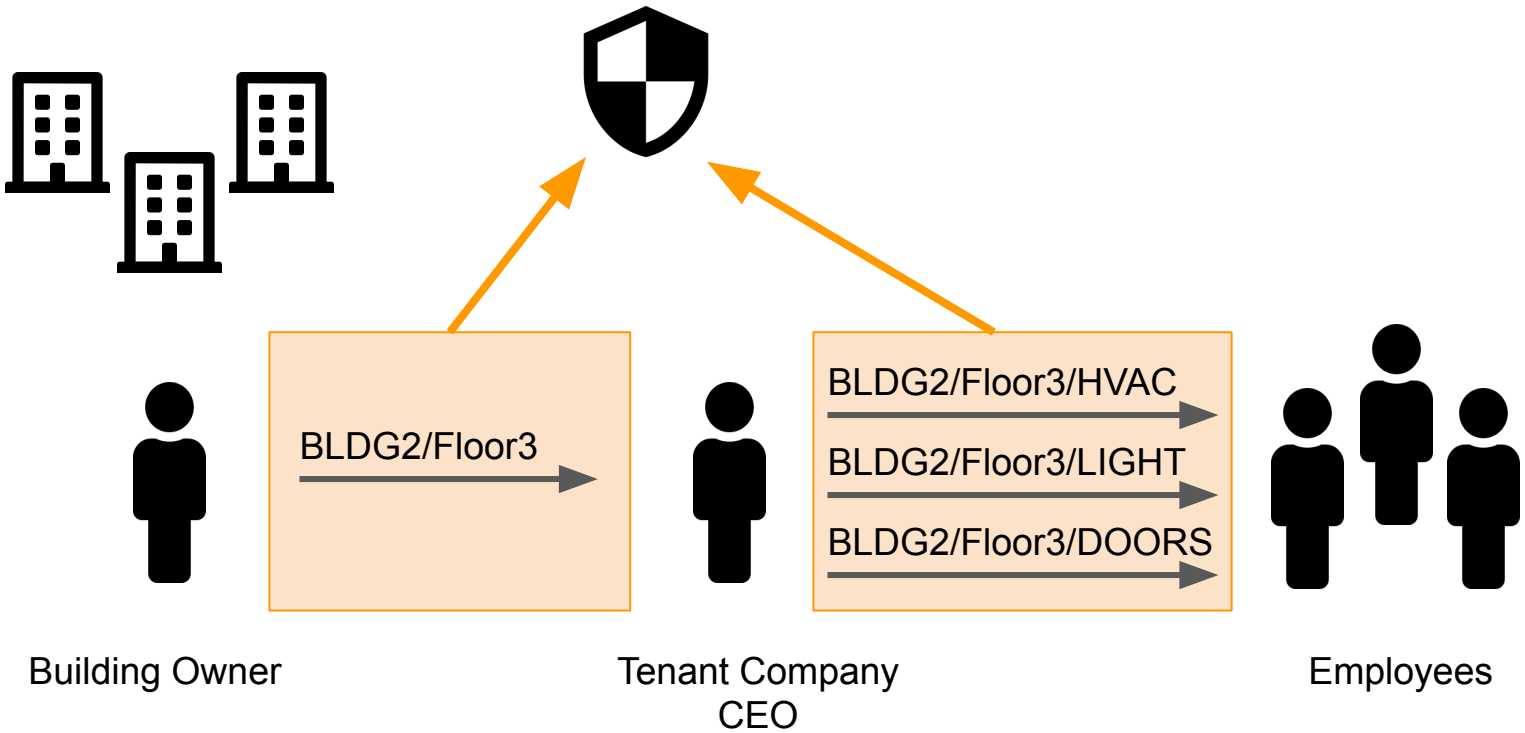
Traditional approach



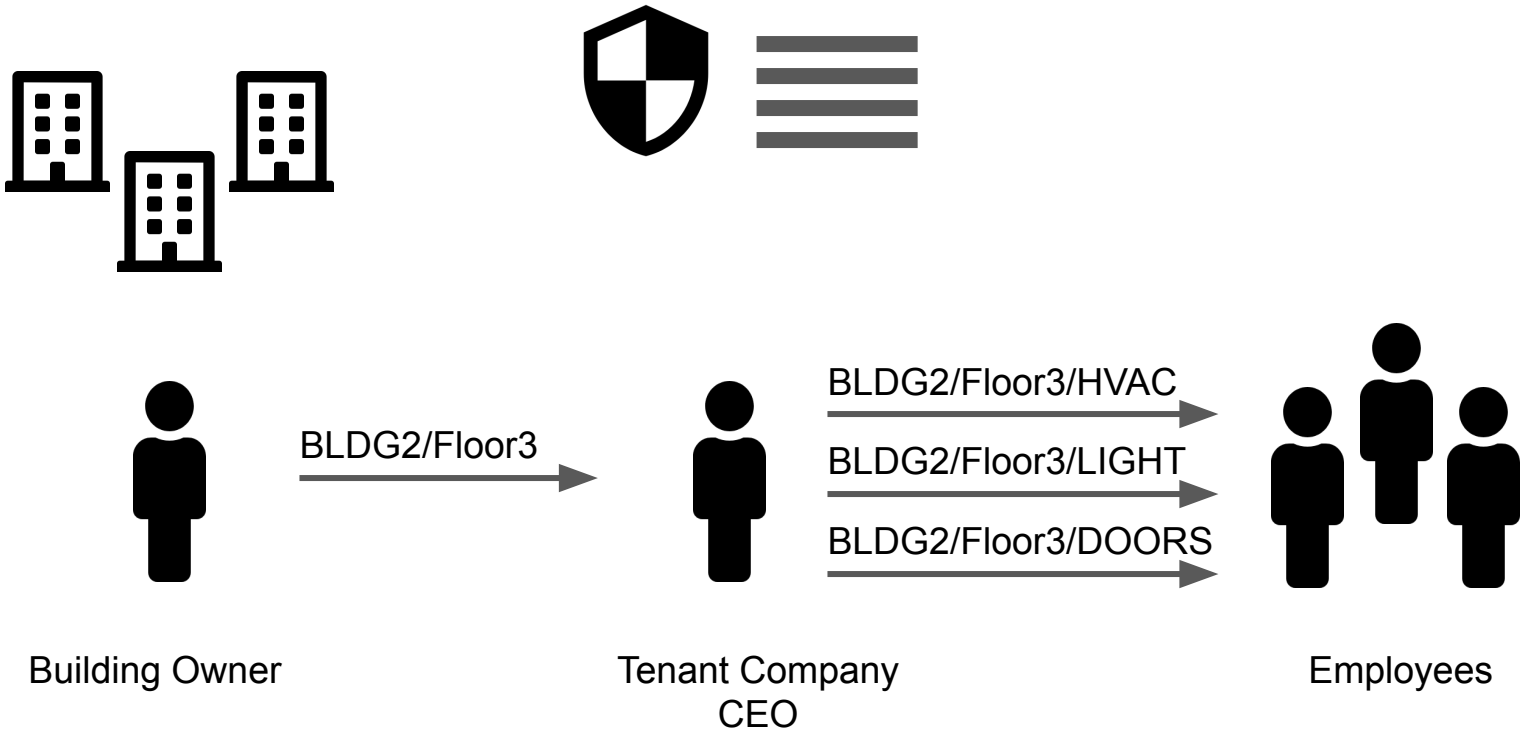
<u>Problems:</u>
Central point of attack
Can’t even trust operator
Sometimes delegation unsupported
When supported, not transitive



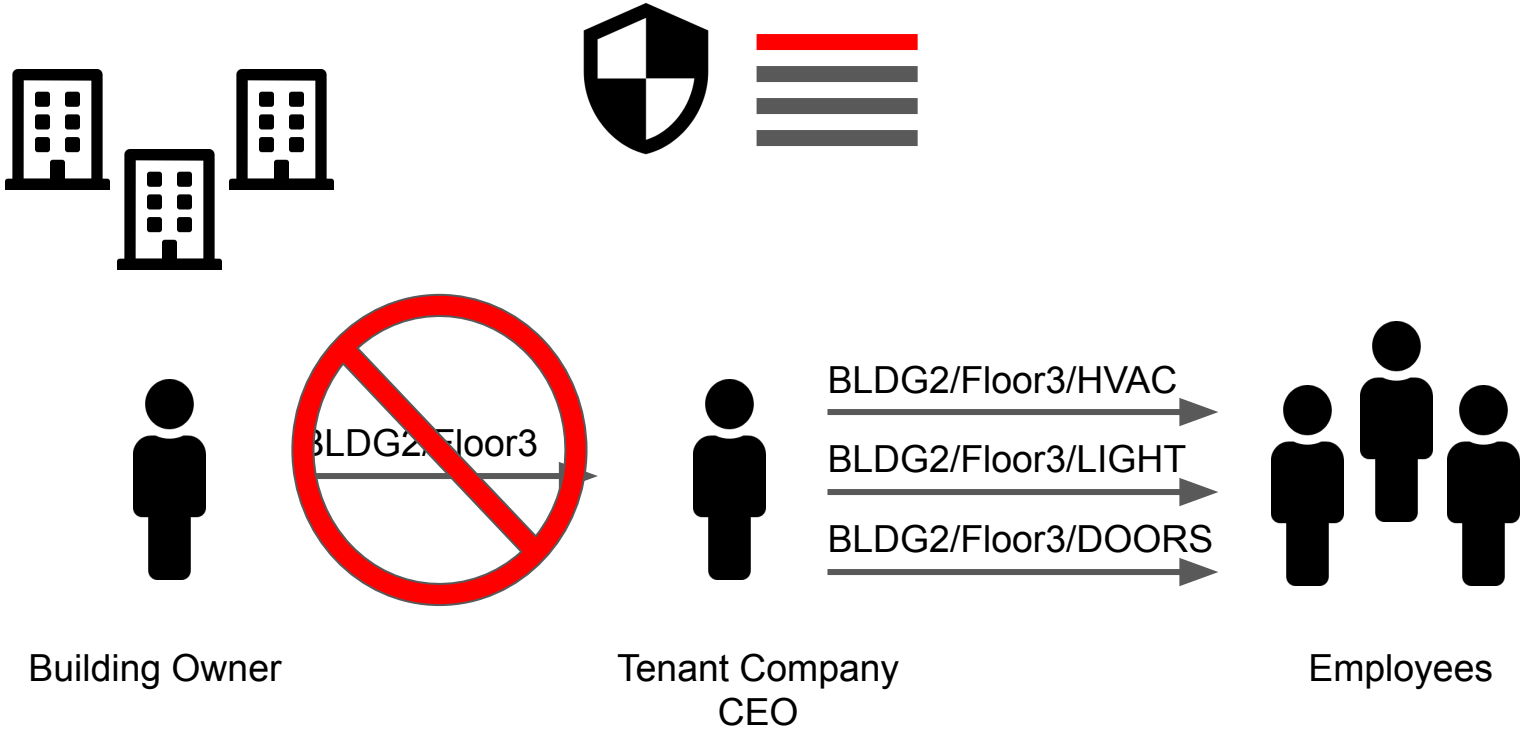
Lack of transitive delegation



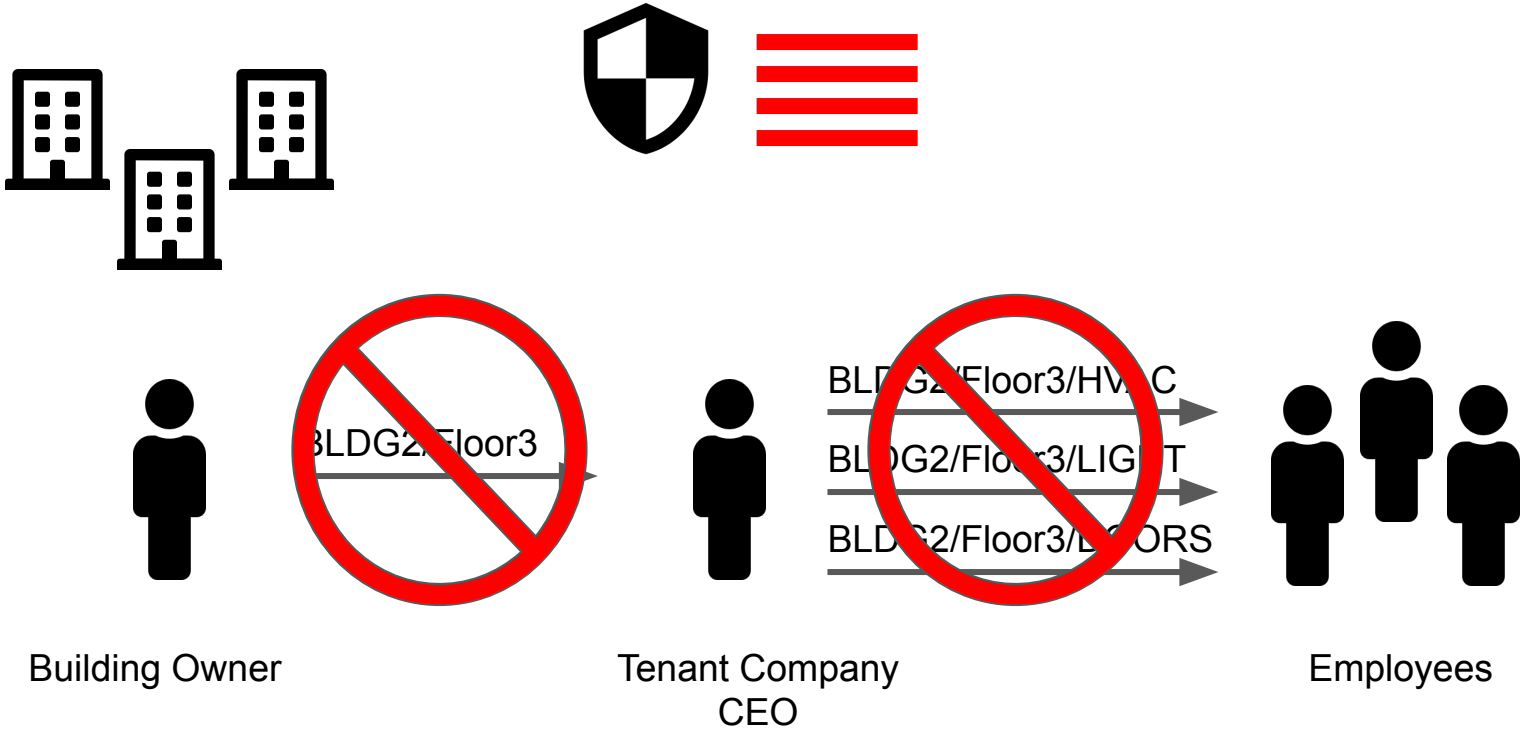
Lack of transitive delegation



Lack of transitive delegation



What we want:


BLDG2/Floor3
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Building Owner
Tenant Company
CEO
Employees

Existing work lacks some important features

System / Work	Avoid central authority	Transitive Delegation	Permission Discovery	No ordering constraints	Offline participants	Protected permissions
LDAP, AD	✗	✓	✓	✓	✓	✗
OAuth2	✗	✓	✓	✓	✓	✗
Macaroons	✗	✓	–	✗	–	–
SDSI/SPKI	✗	✓	–	✓	–	–

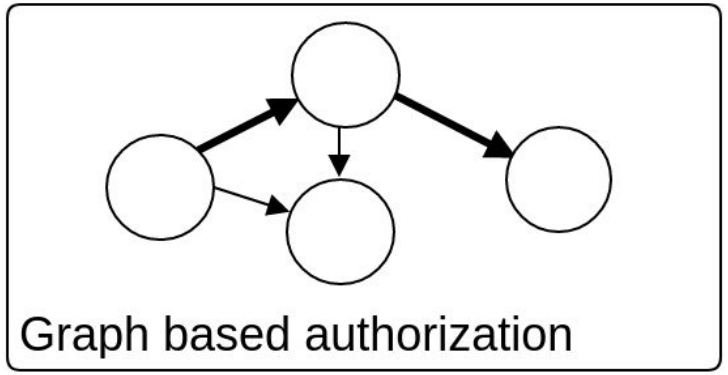
What is WAVE

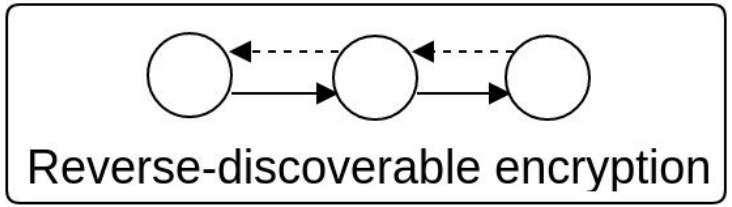
System / Work	No central authority	Transitive Delegation	Permission Discovery	No ordering constraints	Offline participants	Protected permissions
WAVE	✓	✓	✓	✓	✓	✓

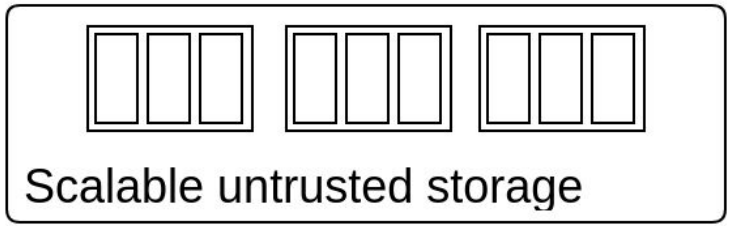
WAVE is a cryptographically enforced decentralized authorization system

- It can be used in place of most mainstream authorization systems
- Anyone can delegate permissions or revoke permissions they have delegated
- Anyone can discover their permissions and form a proof of authorization
- Anyone (even devices) can verify proofs of authorization

WAVE achieves this with three techniques:

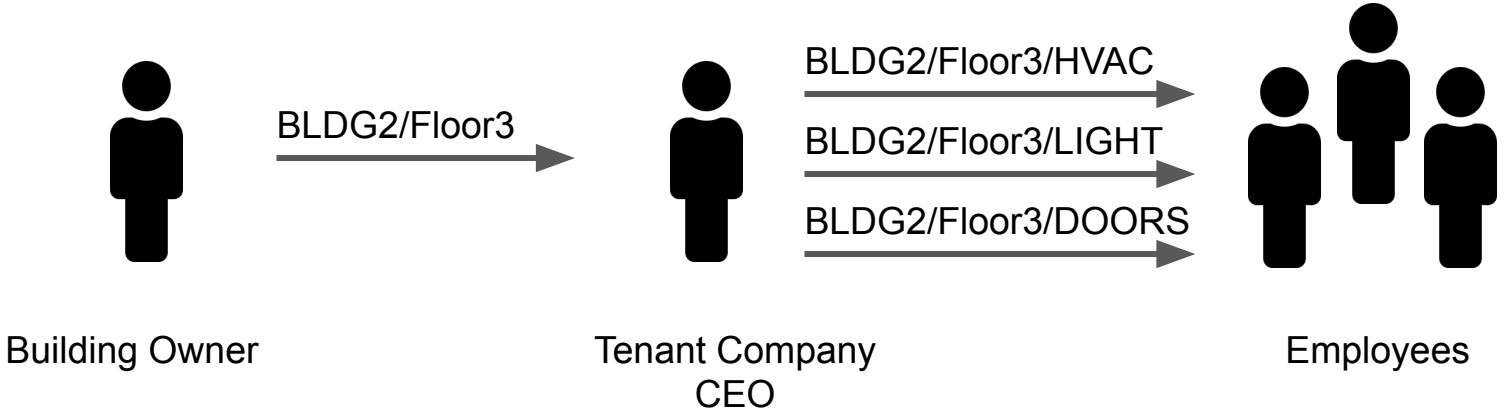




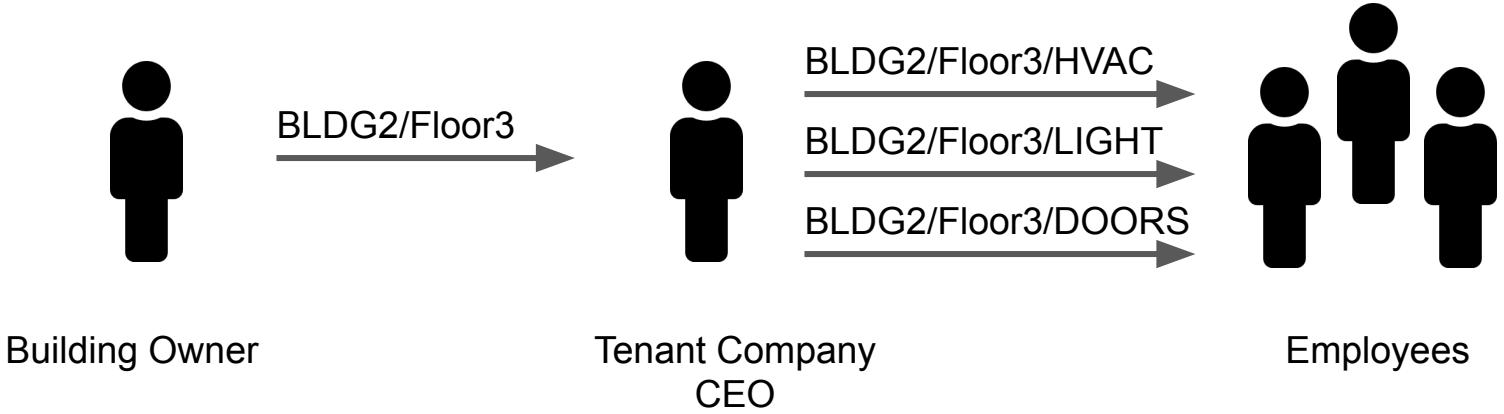


Graph Based Authorization

- Popularized by SDSI/SPKI [Rivest, Lampson, 1996]
- Represents permissions as a graph, rather than an ACL table
- Naturally represents transitive delegation

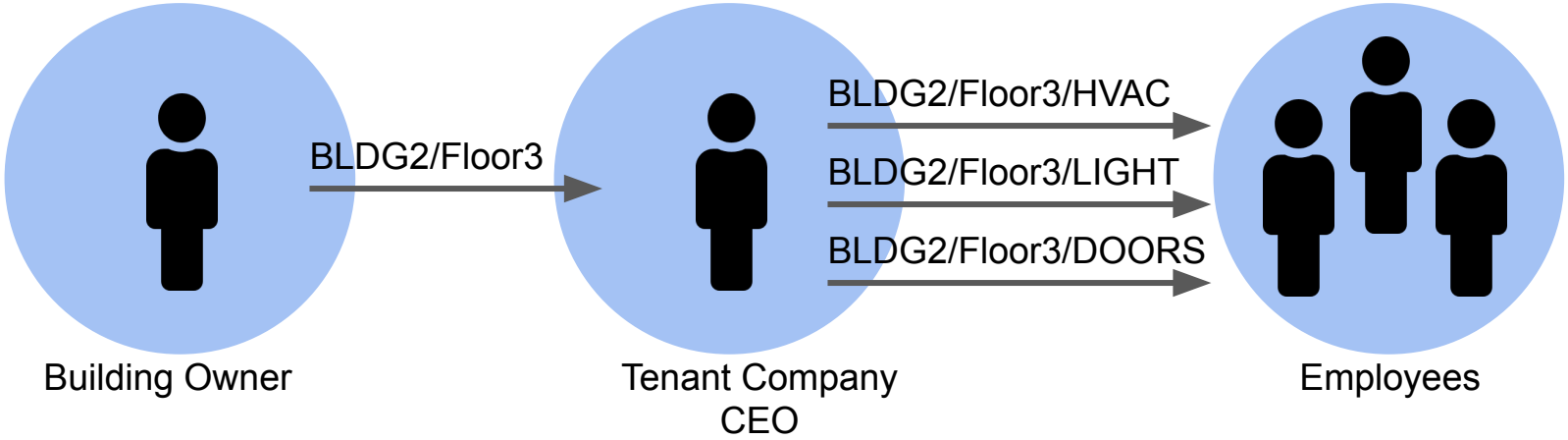


Graph Based Authorization



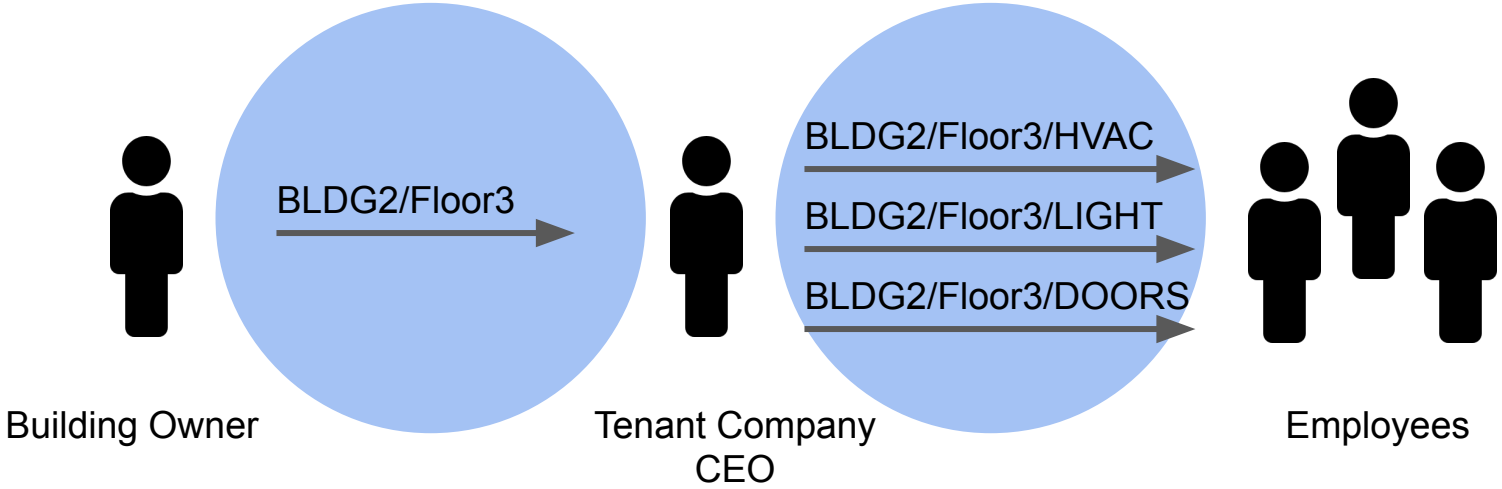
Graph Based Authorization

Participants: Entities
Collections of cryptographic keys: identifier is PK



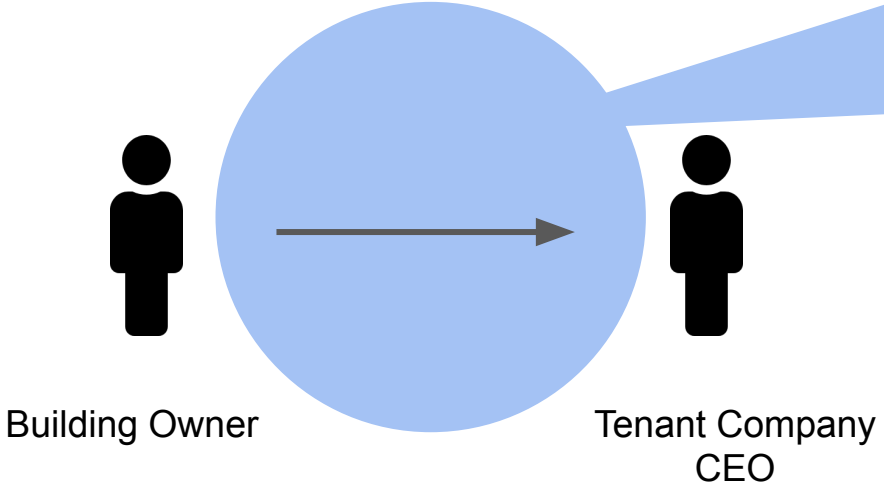
Graph Based Authorization

Grants of permissions: Attestations
Signed certificates created by Entities



Graph Based Authorization

Attestations grant permissions on a **resource**



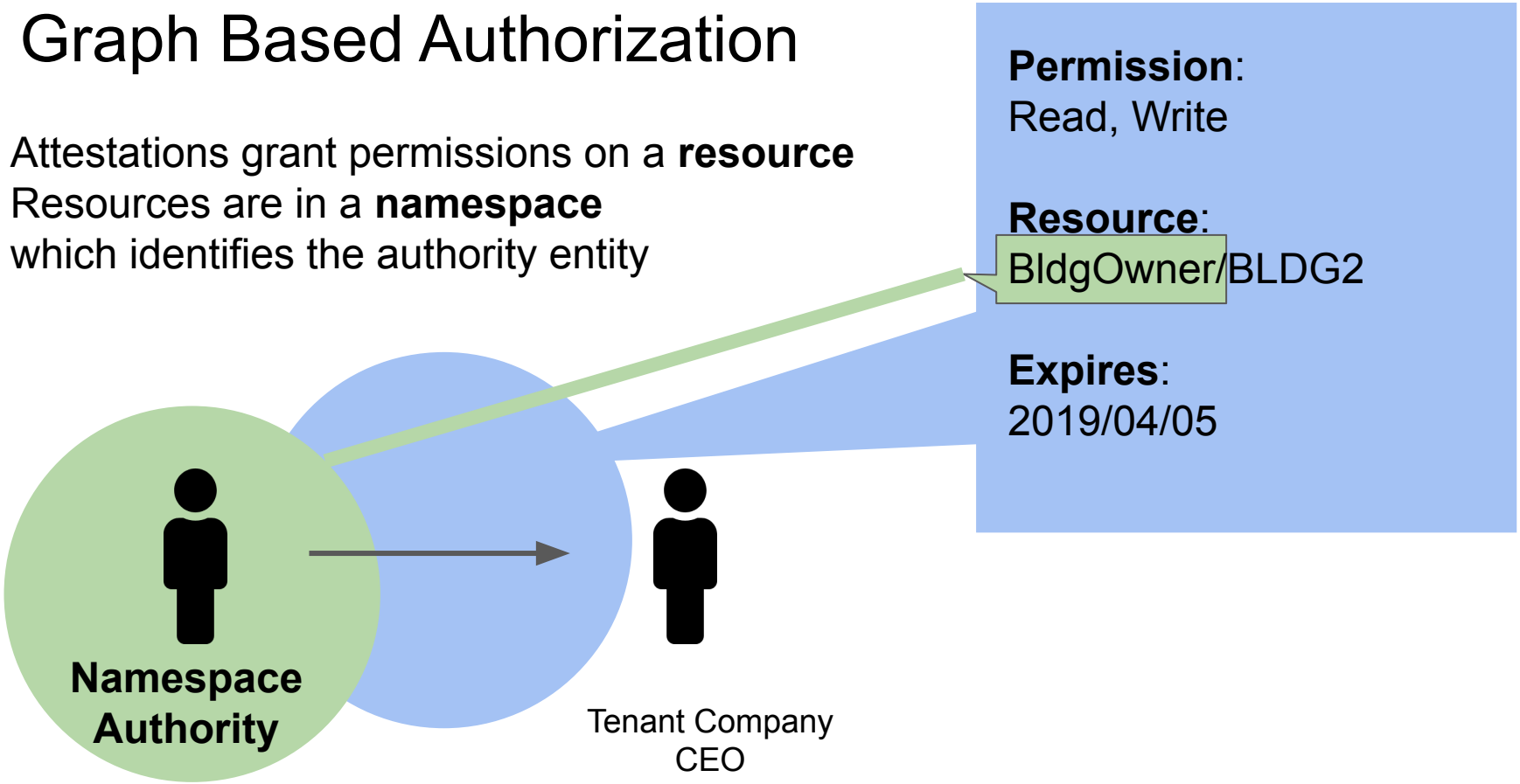
Permission:
Read, Write

Resource:
BldgOwner/BLDG2

Expires:
2019/04/05

Graph Based Authorization

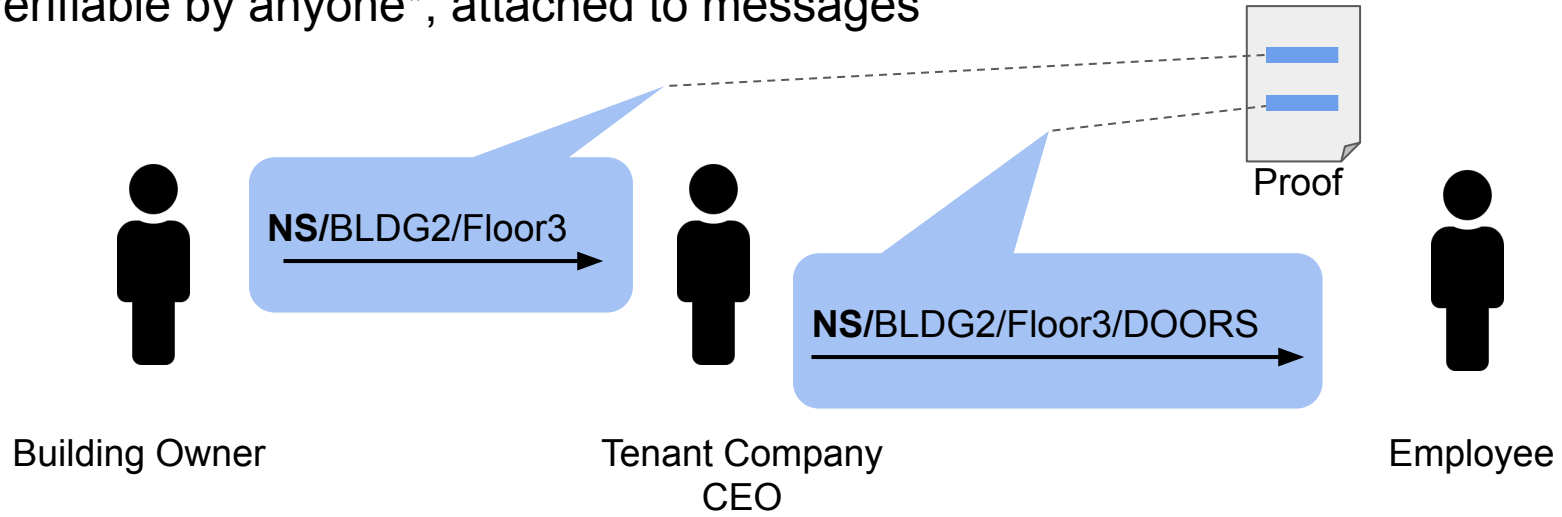
Attestations grant permissions on a **resource**
Resources are in a **namespace**
which identifies the authority entity



Graph Based Authorization

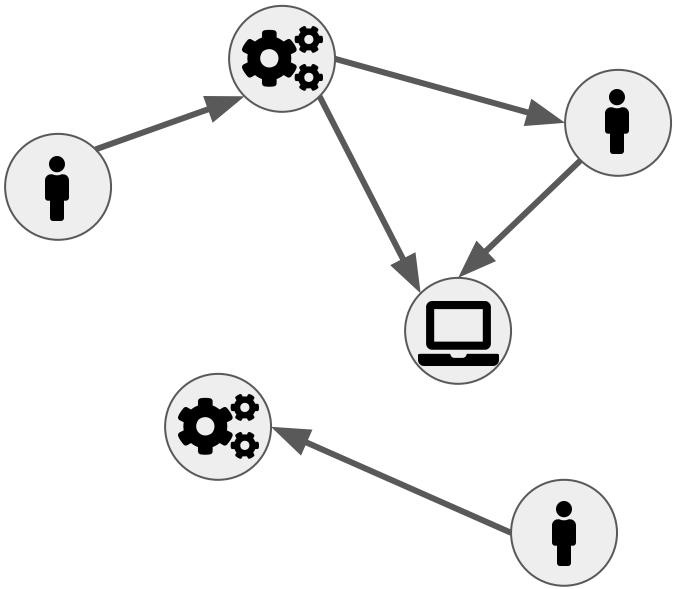
Proof of permissions: A path through the graph from Namespace Authority to the prover

Proof grants the intersection of the permissions of each attestation
Verifiable by anyone*, attached to messages

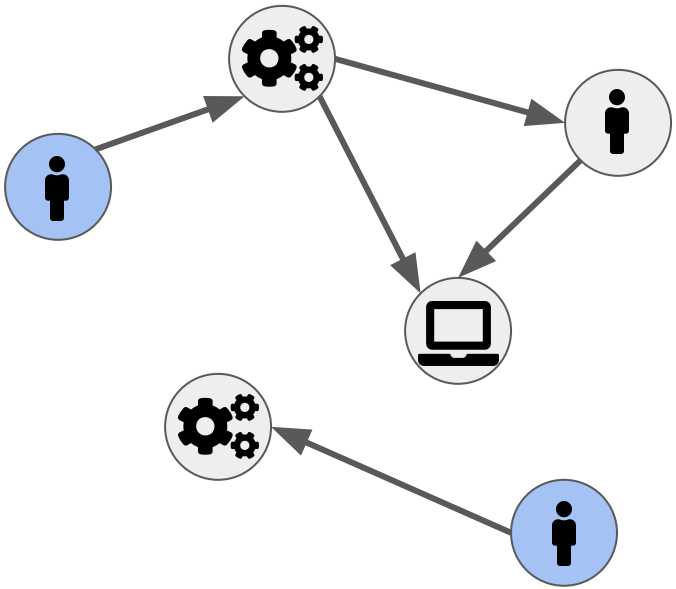


* In WAVE, not SDSI/SPKI

This forms a single global graph

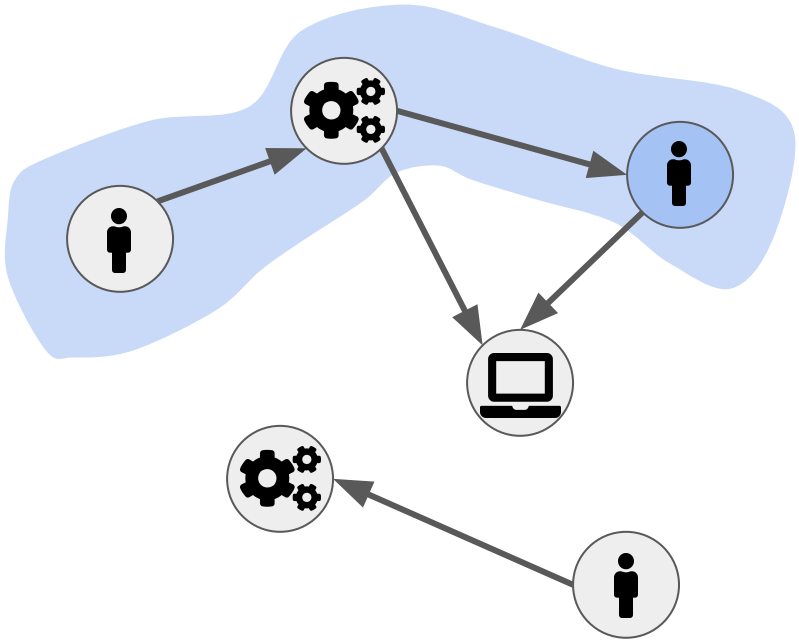


This forms a single global graph



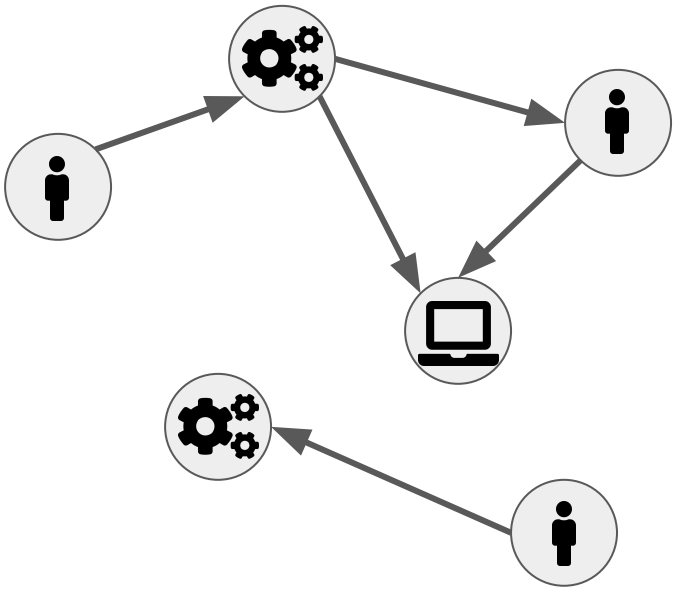
- Multiple namespace authorities in the graph

This forms a single global graph



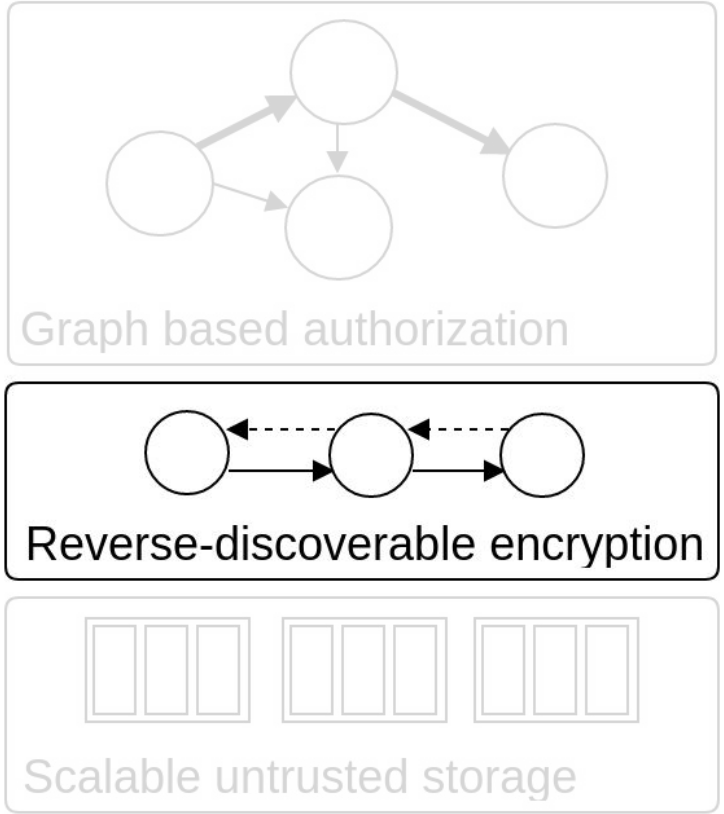
- Multiple namespace authorities in the graph
- Different entities will only see portions of the graph

This forms a single global graph

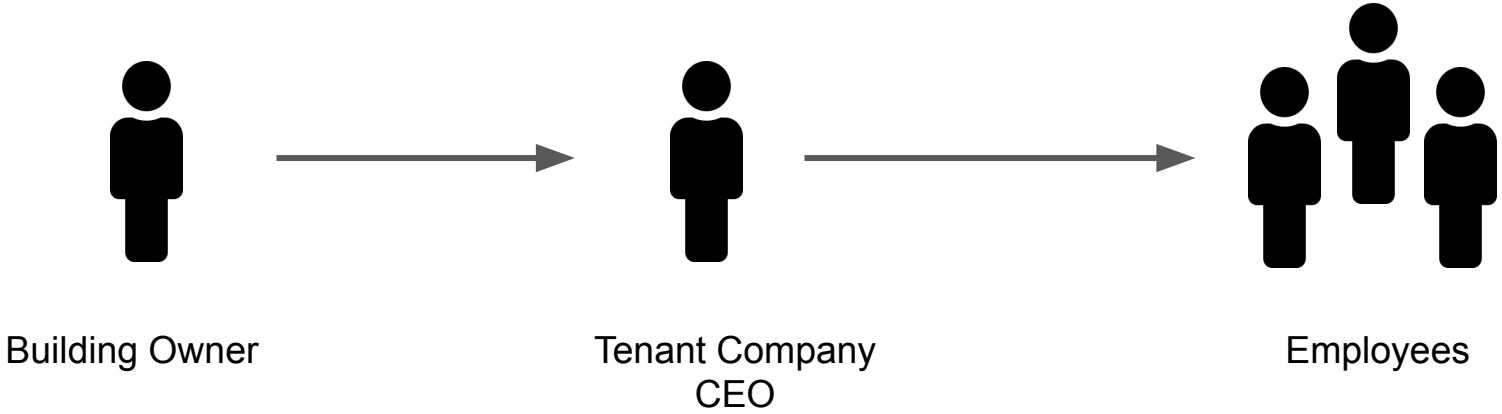


- Multiple namespace authorities in the graph
- Different entities will only see portions of the graph

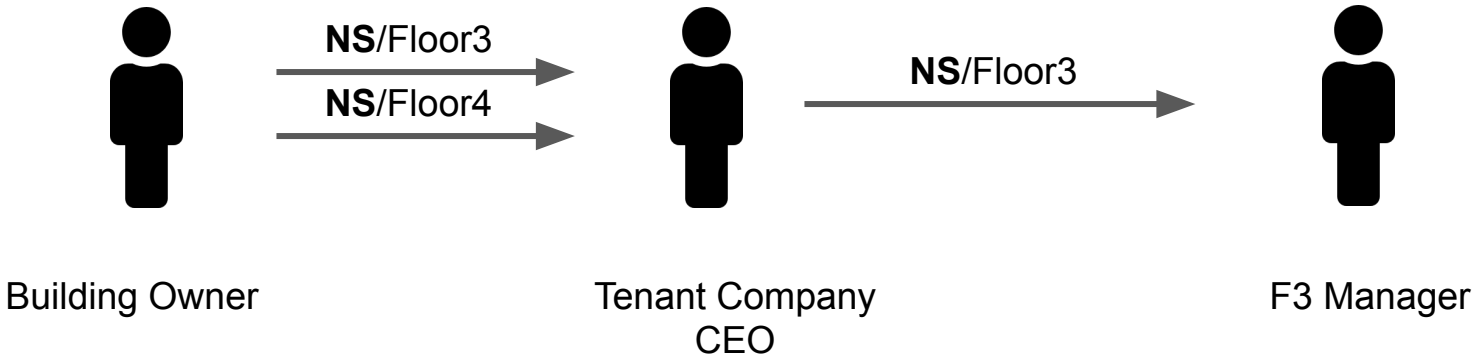
We need to hide portions of the graph



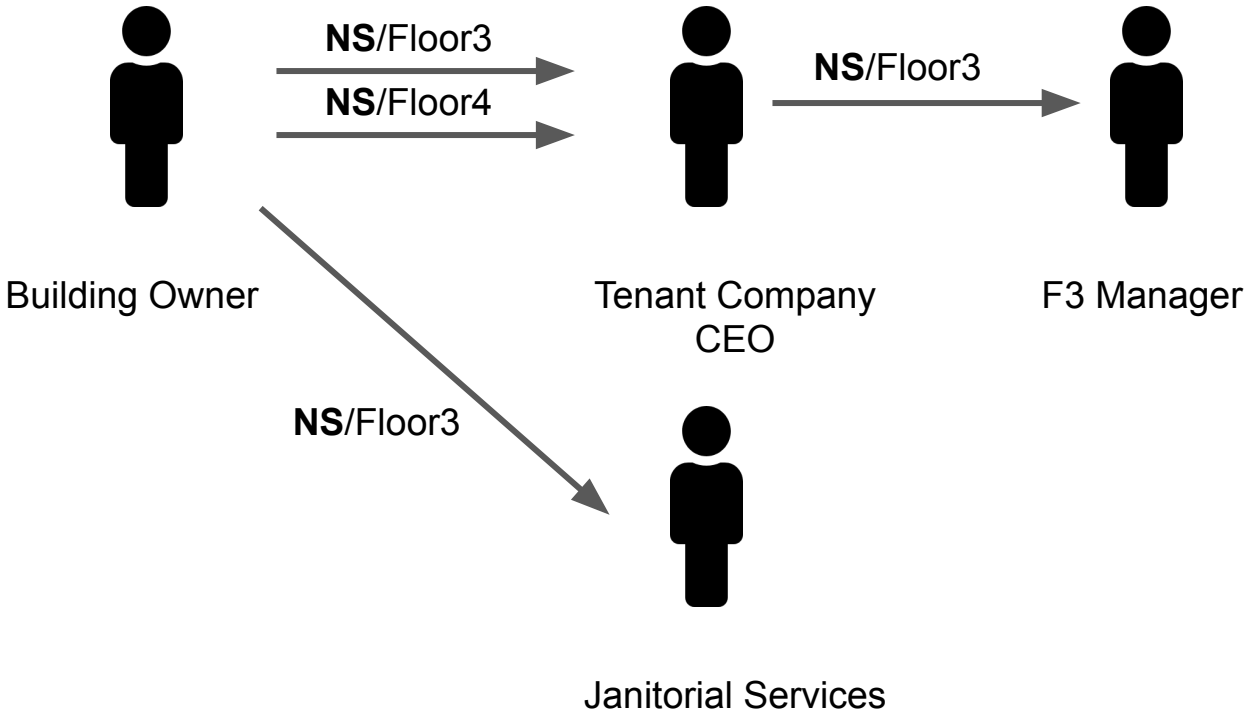
Reverse Discoverable Encryption



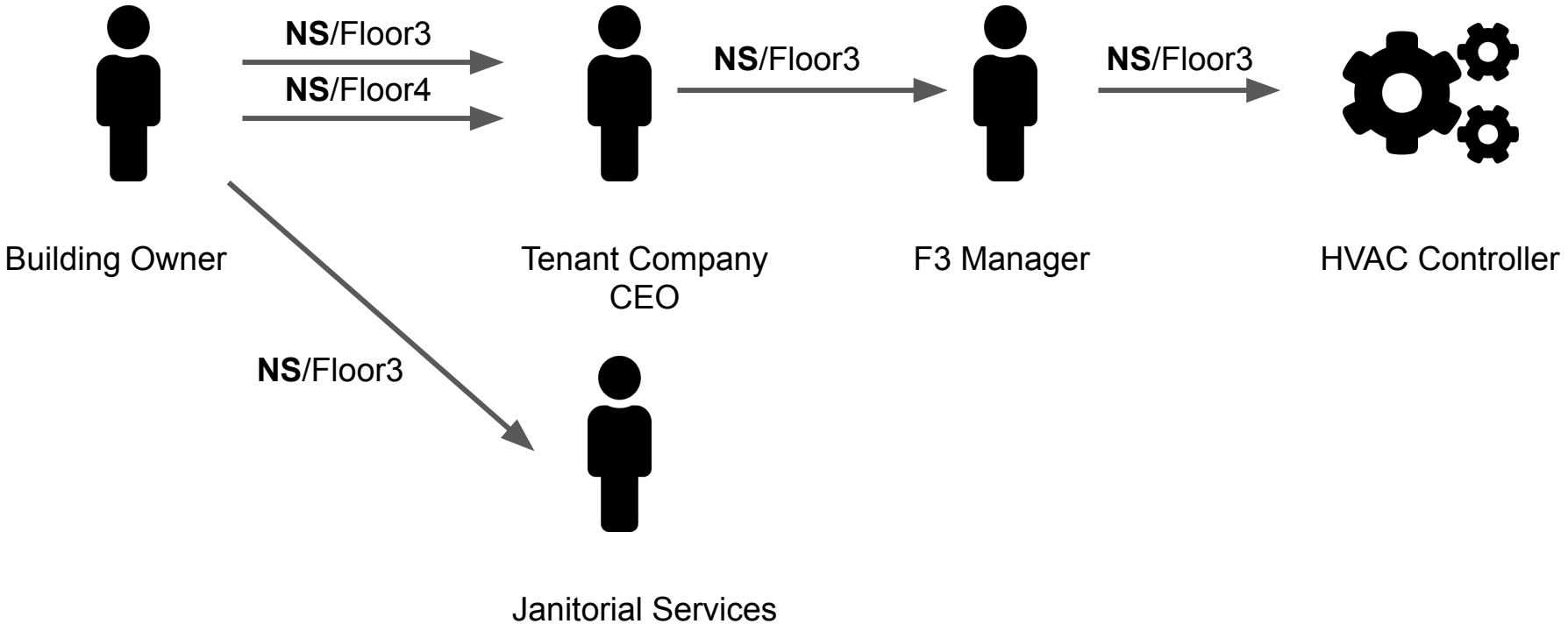
Reverse Discoverable Encryption



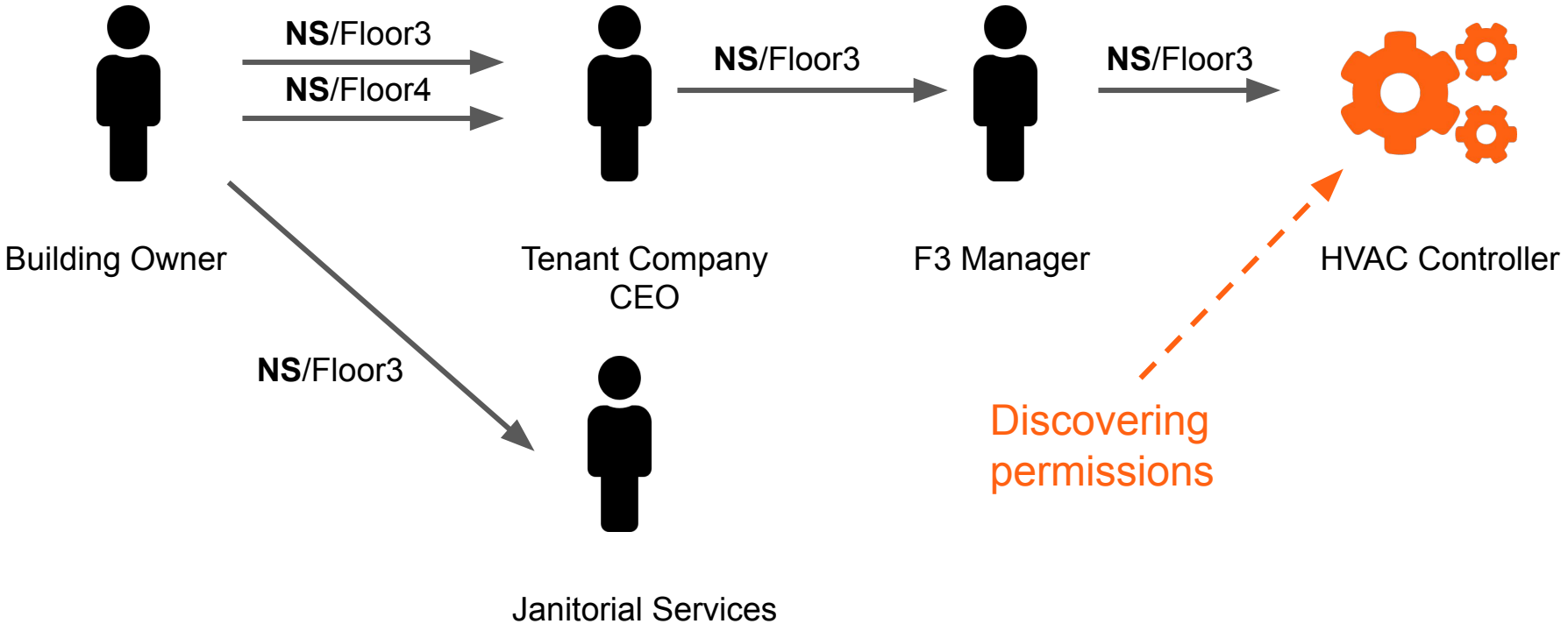
Reverse Discoverable Encryption



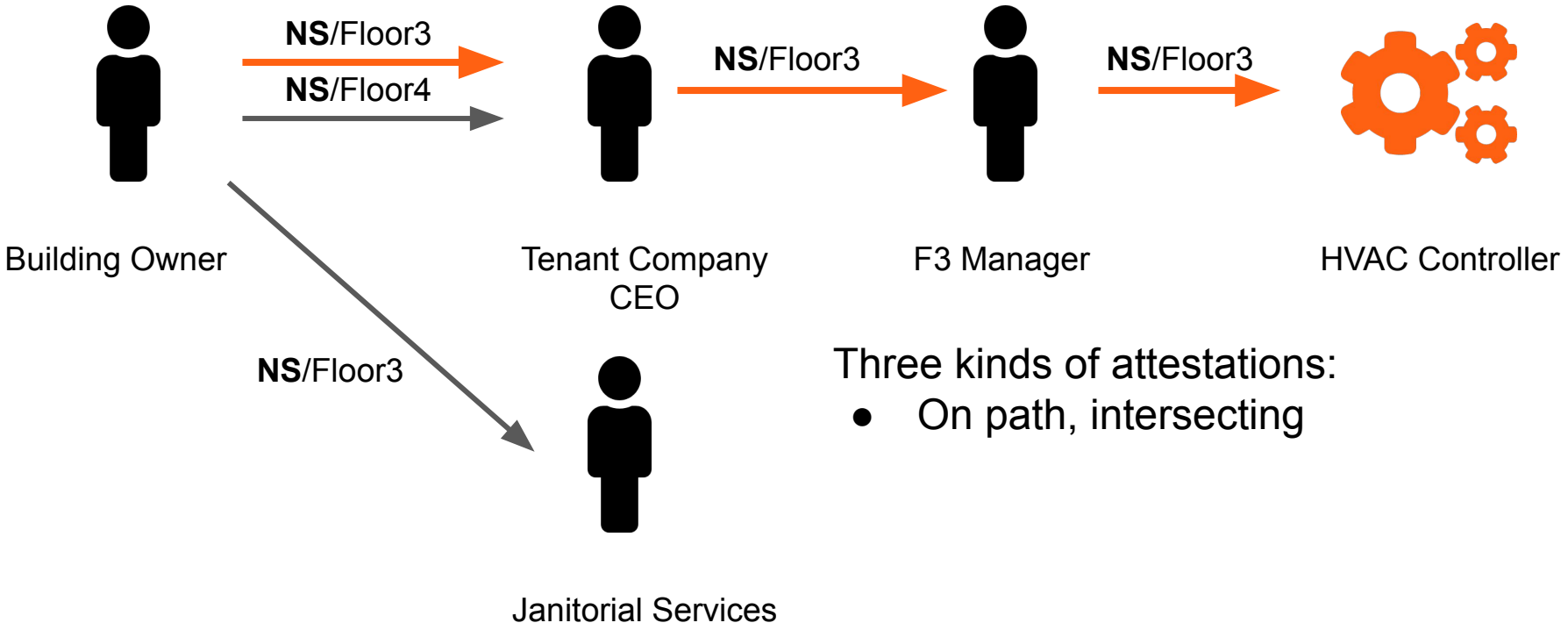
Reverse Discoverable Encryption



Reverse Discoverable Encryption



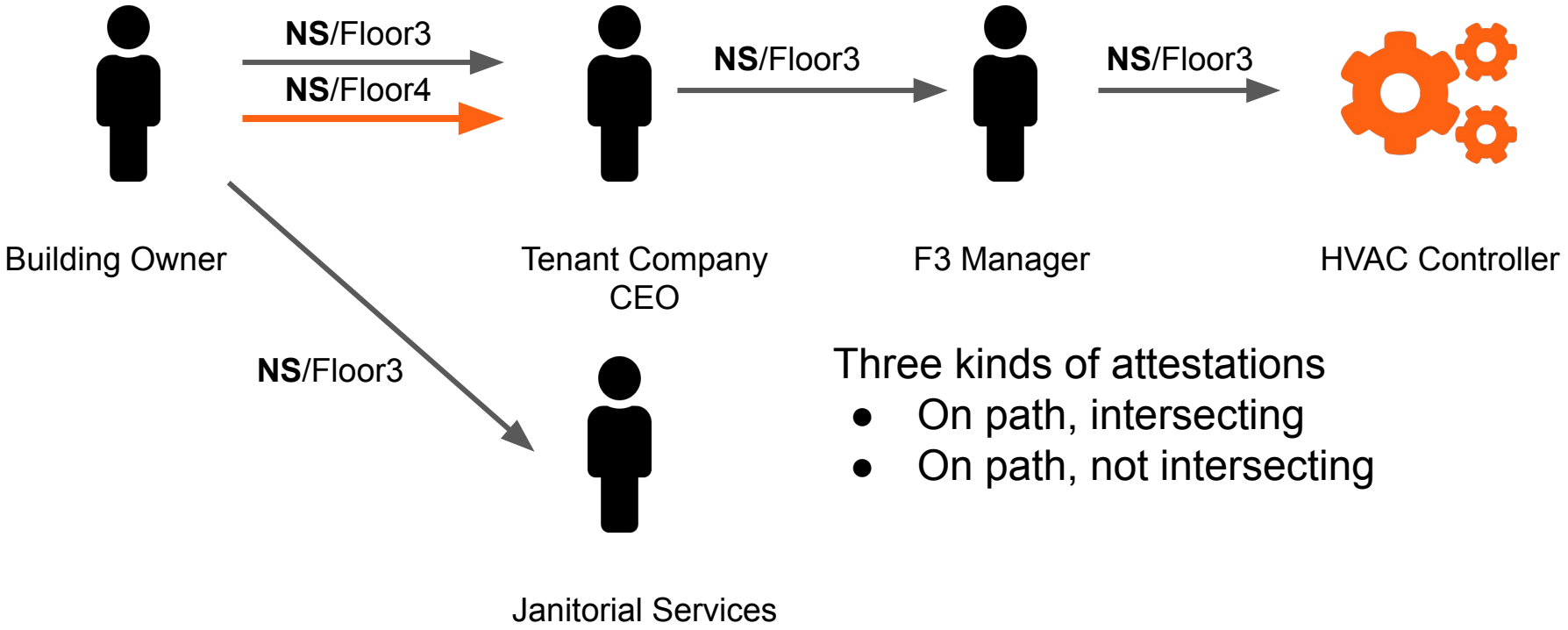
Reverse Discoverable Encryption



Three kinds of attestations:

- On path, intersecting

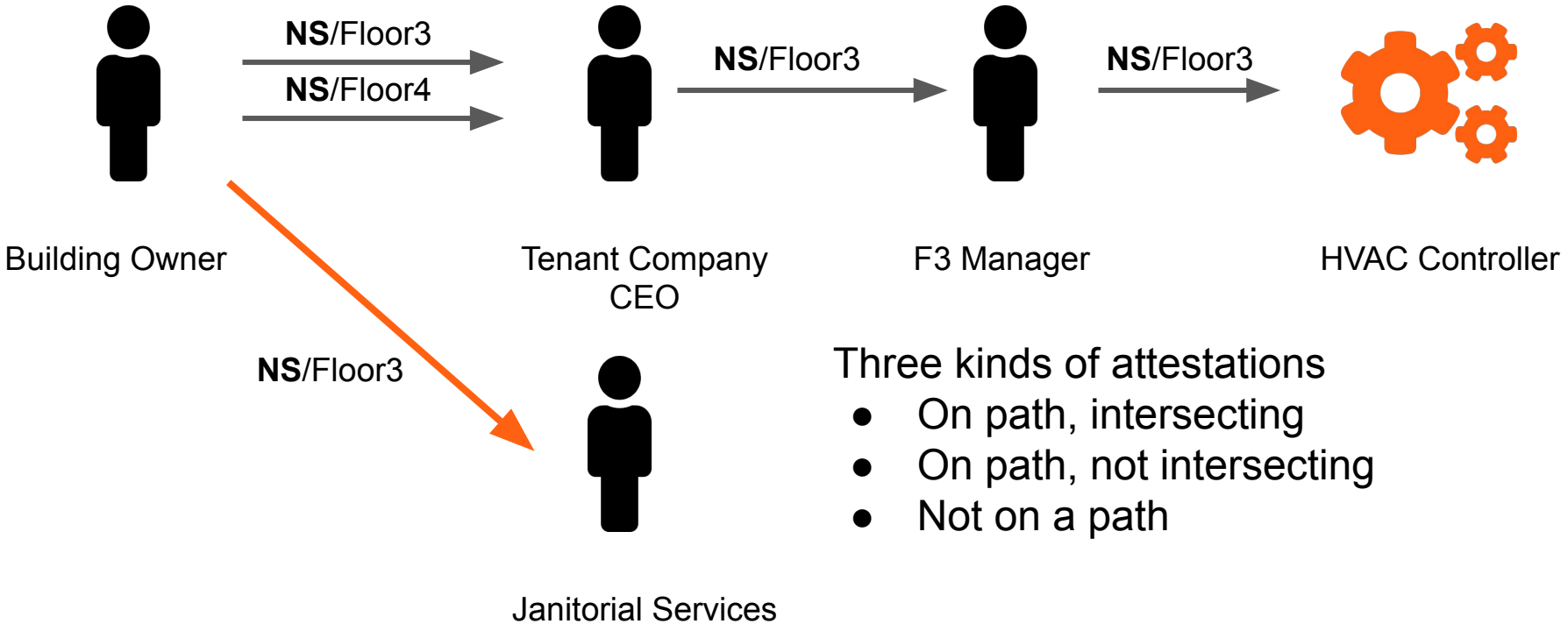
Reverse Discoverable Encryption



Three kinds of attestations

- On path, intersecting
- On path, not intersecting

Reverse Discoverable Encryption



Three kinds of attestations

- On path, intersecting
- On path, not intersecting
- Not on a path

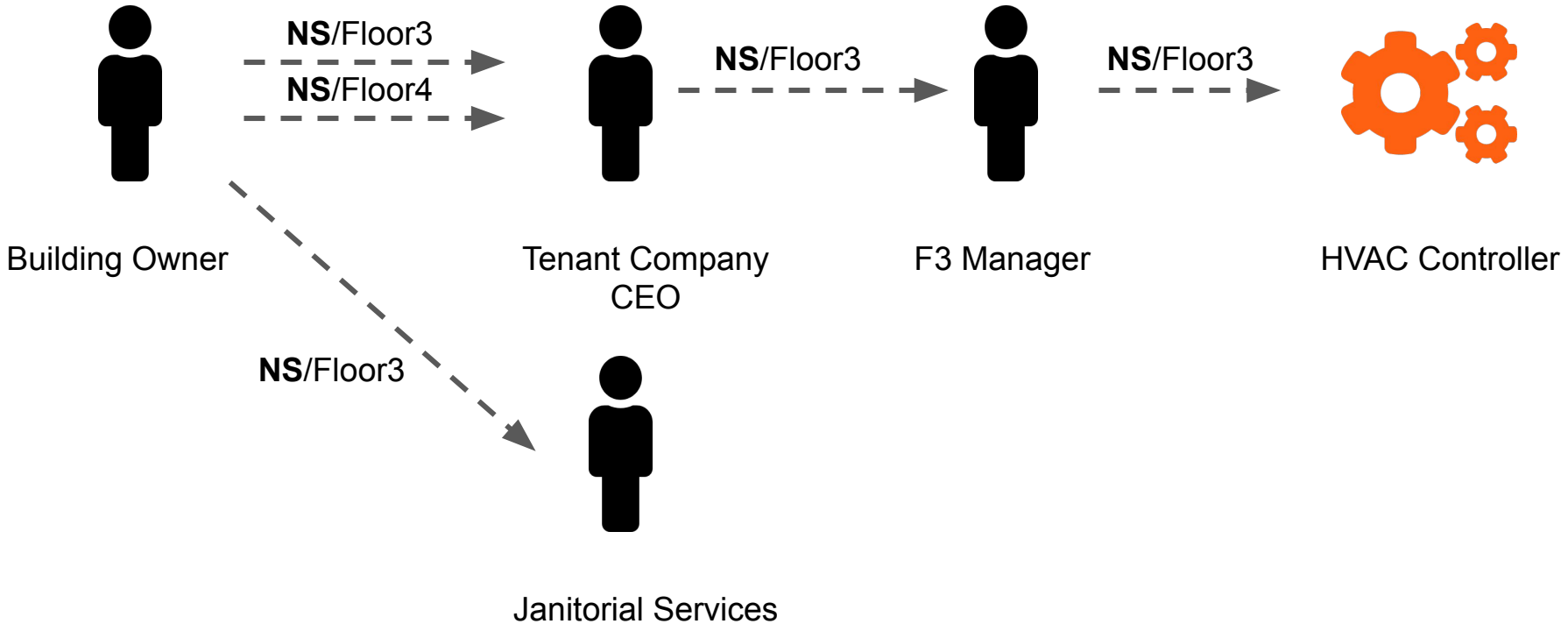
Technique in a nutshell

Encrypt attestations

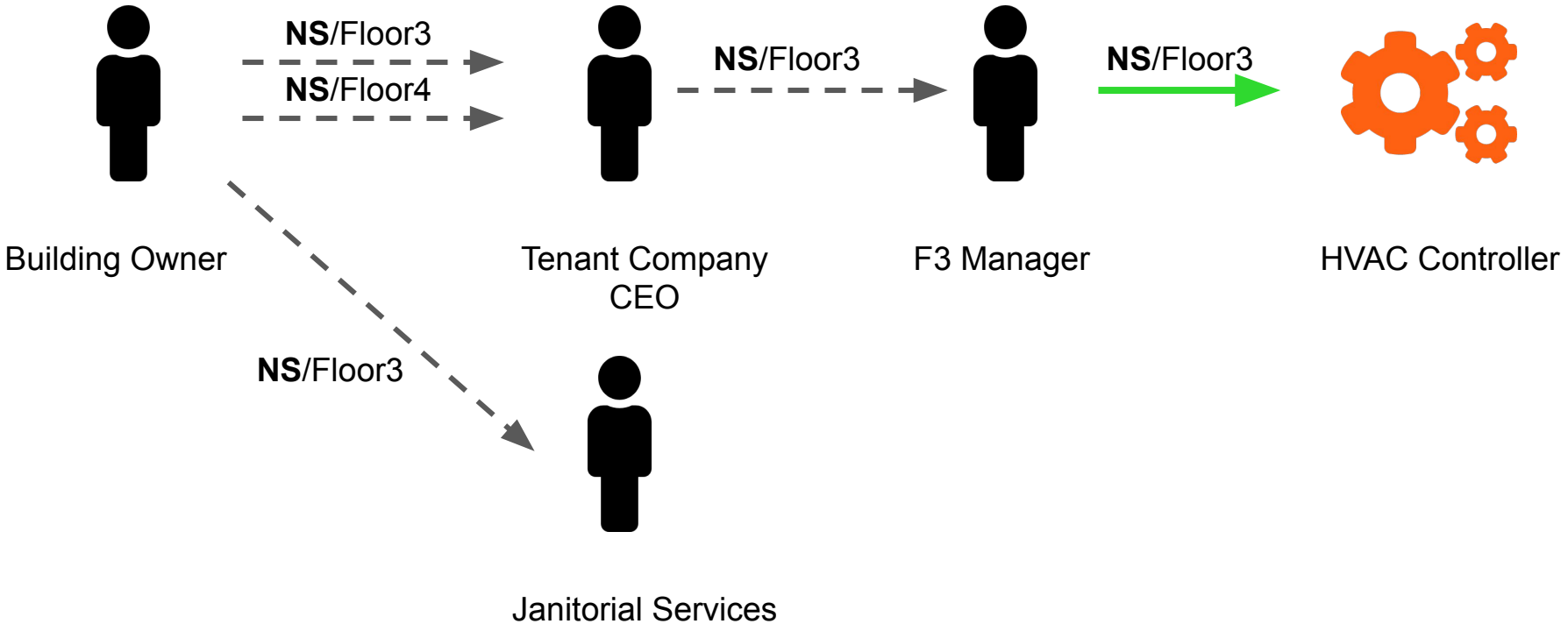
In each attestation, include a secret that allows you to decrypt upstream attestations that have intersecting permissions

(on path, intersecting)

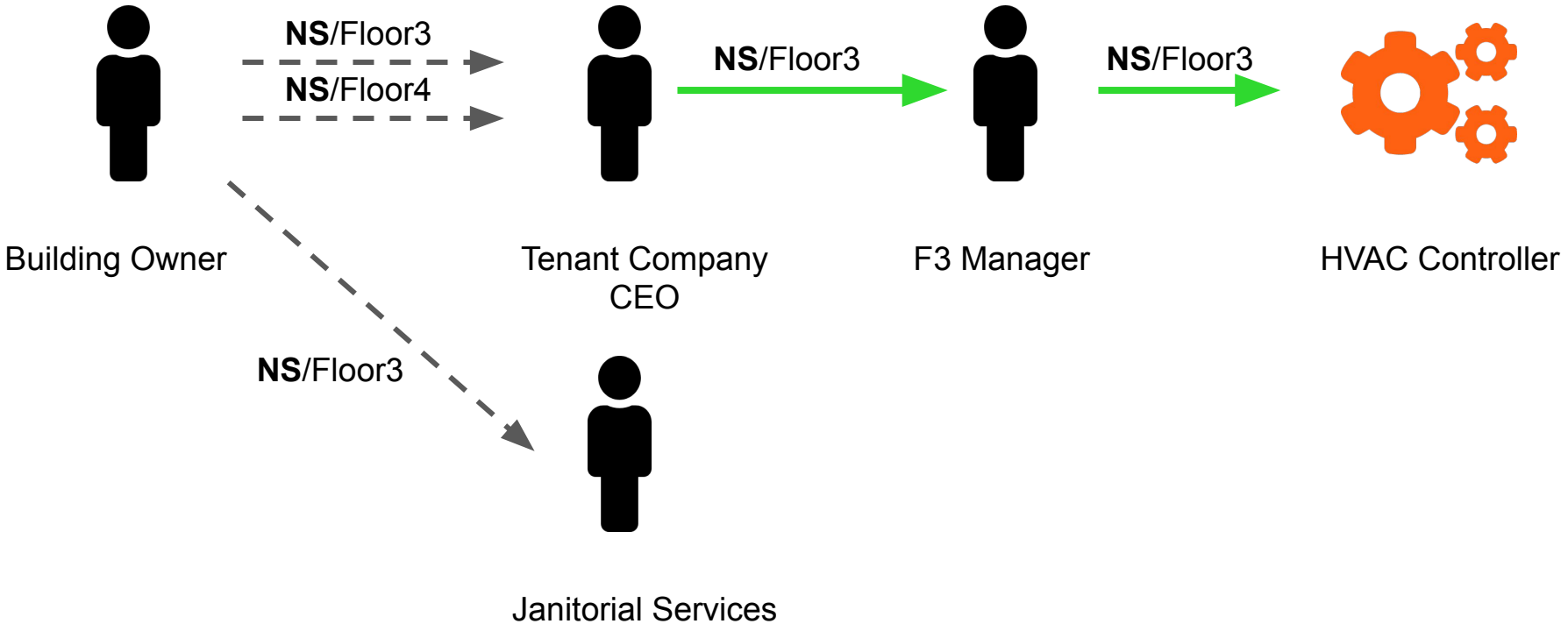
Reverse Discoverable Encryption



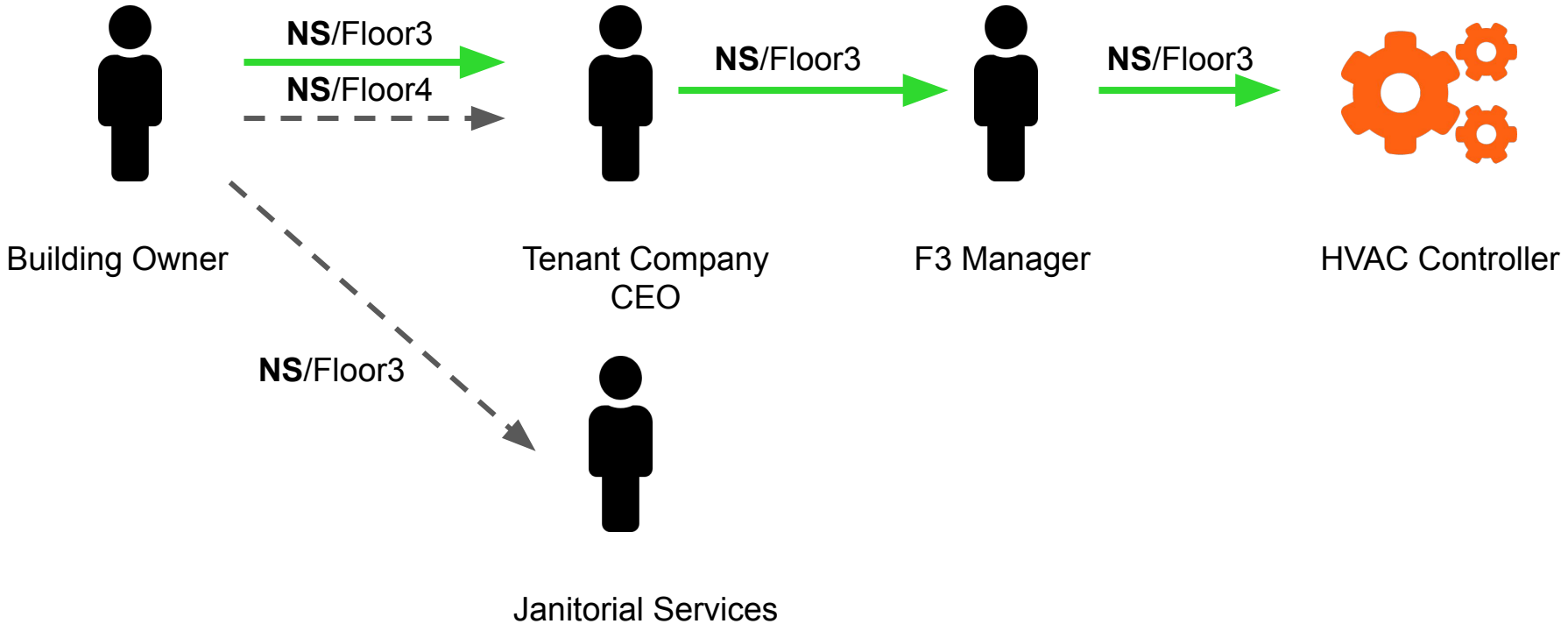
Reverse Discoverable Encryption



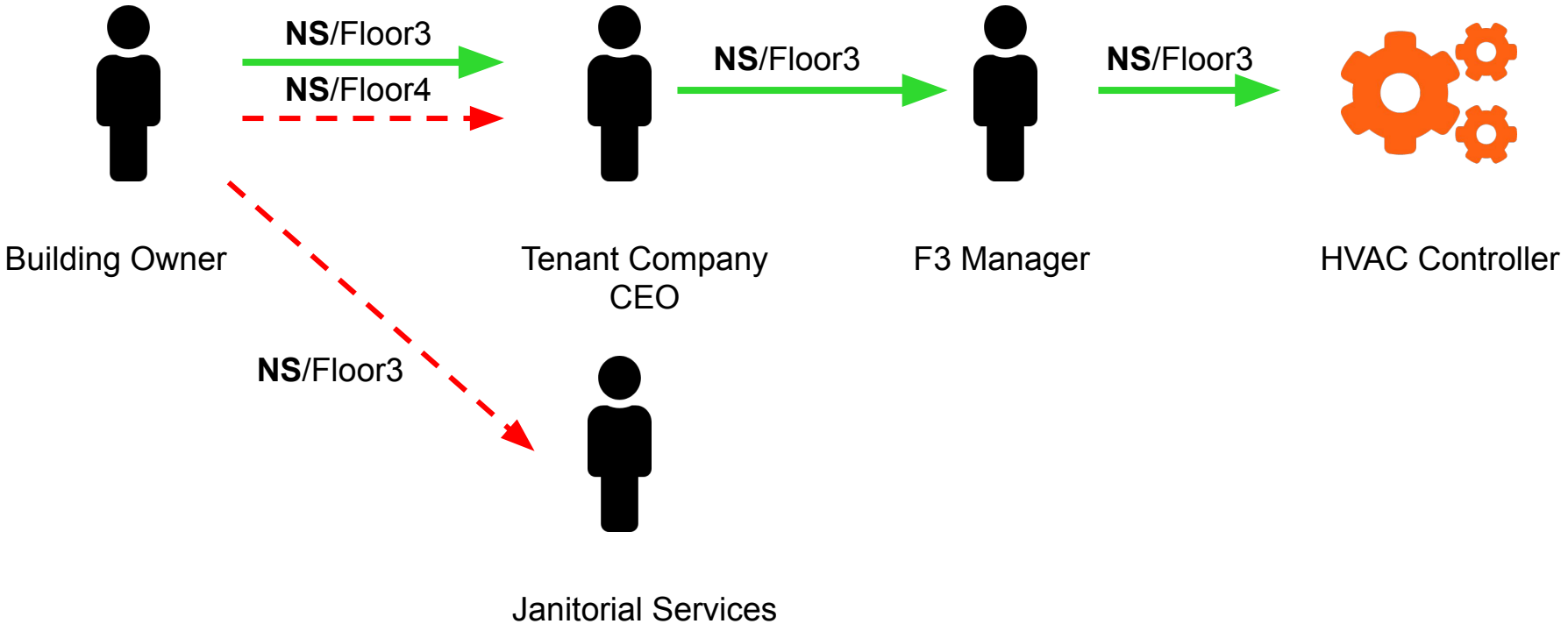
Reverse Discoverable Encryption



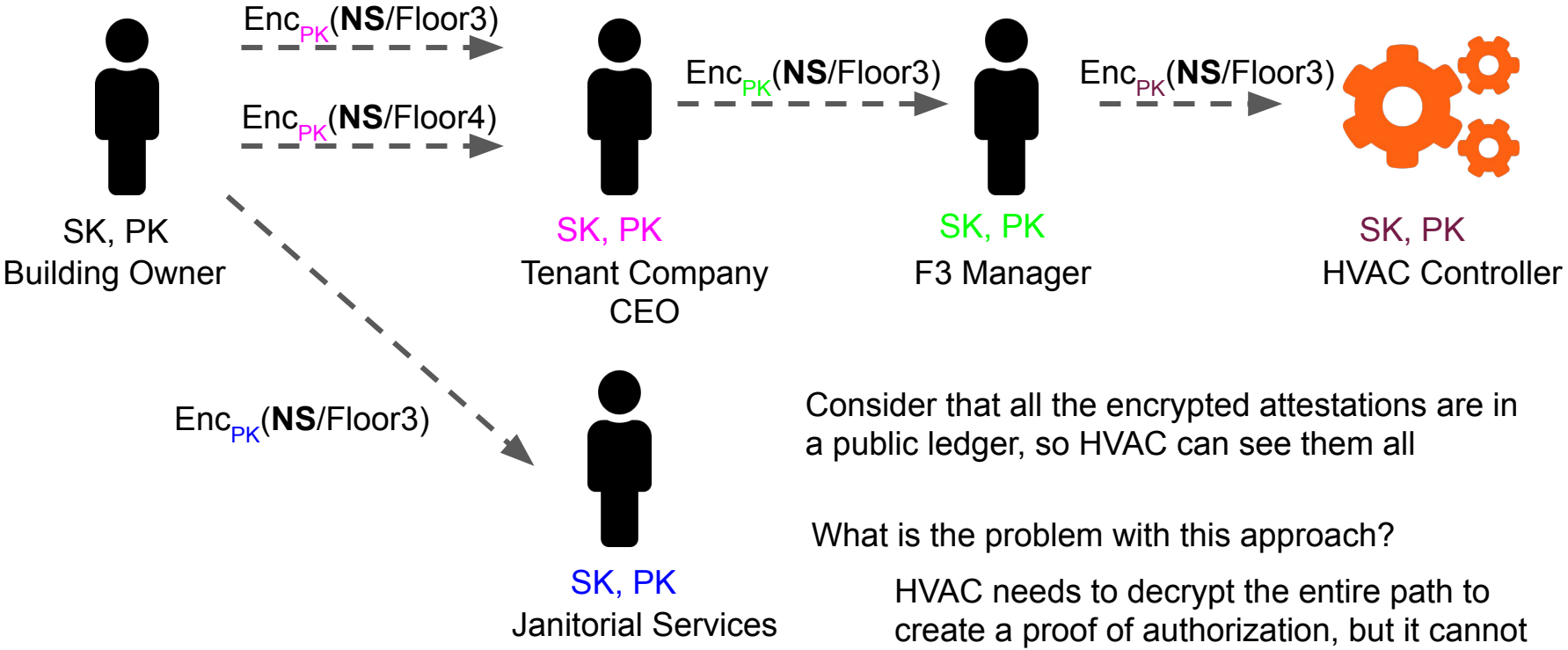
Reverse Discoverable Encryption



Reverse Discoverable Encryption



Attempt 1: public-key encryption

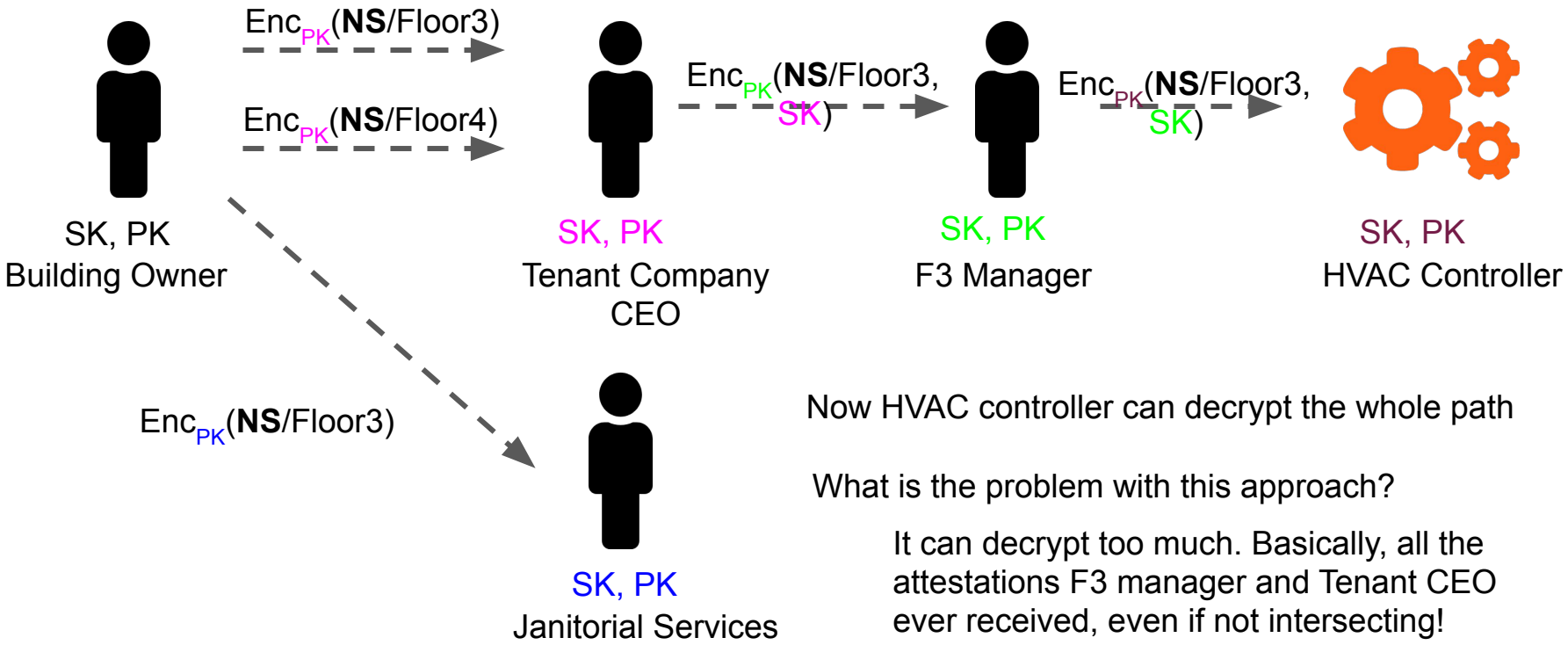


Consider that all the encrypted attestations are in a public ledger, so HVAC can see them all

What is the problem with this approach?

HVAC needs to decrypt the entire path to create a proof of authorization, but it cannot

Attempt 2: public-key encryption



Now HVAC controller can decrypt the whole path

What is the problem with this approach?

It can decrypt too much. Basically, all the attestations F3 manager and Tenant CEO ever received, even if not intersecting!

Insights

- The encryption & secret must capture the permissions
- It will not be tight, but the idea is to reduce the visibility of any entity only to permissions that are potentially relevant
- It is an example of encrypting on a public ledger while still allowing relevant entities to decrypt
- We use Identity Based Encryption (IBE) and Wildcard Identity Based Encryption (WIBE) [Abdalla, 2006]

IBE

IBE = identity based encryption

- Encrypt with someone's identity instead of PK
- Setup() -> msk, mpk by a trusted key generator
- Enc(mpk, ID; m) -> c, where ID is a string, the identity
- KeyGen(msk, ID) -> sk_{ID}
- Dec(c, sk_{ID})->m

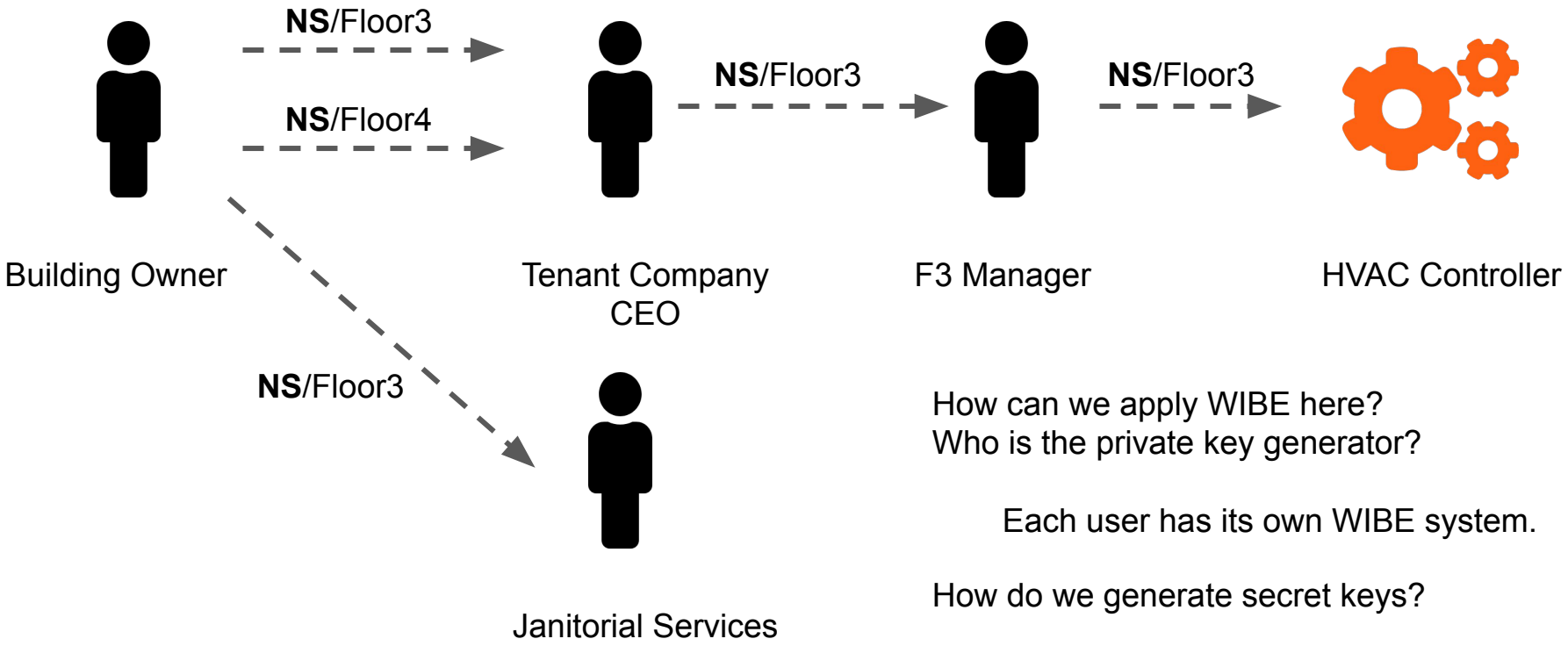
Security: semantic security for message m, but ID revealed

WIBE

- Setup() -> msk, mpk by a trusted key generator
- Enc(mpk, ID; m) -> c, where ID = (ID_1, …, ID_n)
- KeyGen(msk, ID*) -> sk_{ID^*}, where ID* is a vector of strings with some wildcards
- Dec(sk_{ID^*}, c) -> m, if ID* matches ID

Security: semantic security for message m, but all IDs revealed

Reverse Discoverable Encryption



How can we apply WIBE here?
Who is the private key generator?

Each user has its own WIBE system.

How do we generate secret keys?

The encryption & secret must capture the permissions

- Every entity has a WIBE master key
  - No PKG, **every entity has their own system**

- When you create an attestation from A to B with permissions
  - Form WIBE ID = P(permissions)
  - KeyGen(msk, ID) -> sk
  - Include sk in attestation
  - Encrypt attestation using WIBE params for recipient using same ID

This is simplified, please see paper for more details

RDE = Nesting IBE and WIBE

Highly technical. You don't need to understand details, just get a sense.

preissuer → issuer

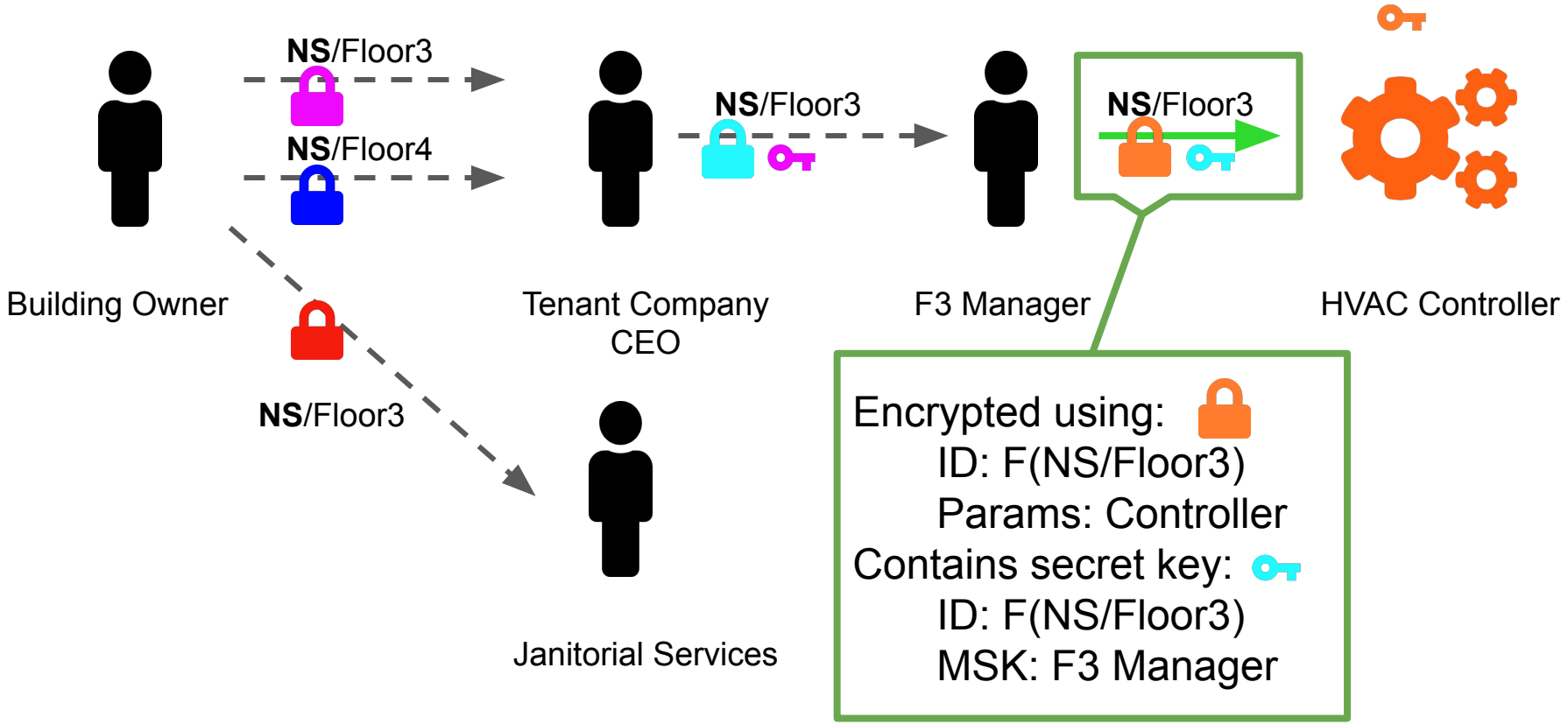
Attestation A
(read, NS/floor3/*, 2019/Jan/*) → subject

IBE.Enc(IBE.mpk$_{subject}$, NS;
[read; NS, floor3, *, *; 2019, Jan, *, *]=P,
WIBE.Enc(WIBE.mpk$_{subject}$,P;
WIBE.KeyGen(WIBE.msk$_{issuer}$, ID*$_i$) for ID$_i$* that could decrypt current attestation,
Attestation A,
IBE.KeyGen(IBE.msk$_{issuer}$; NS)
)
)

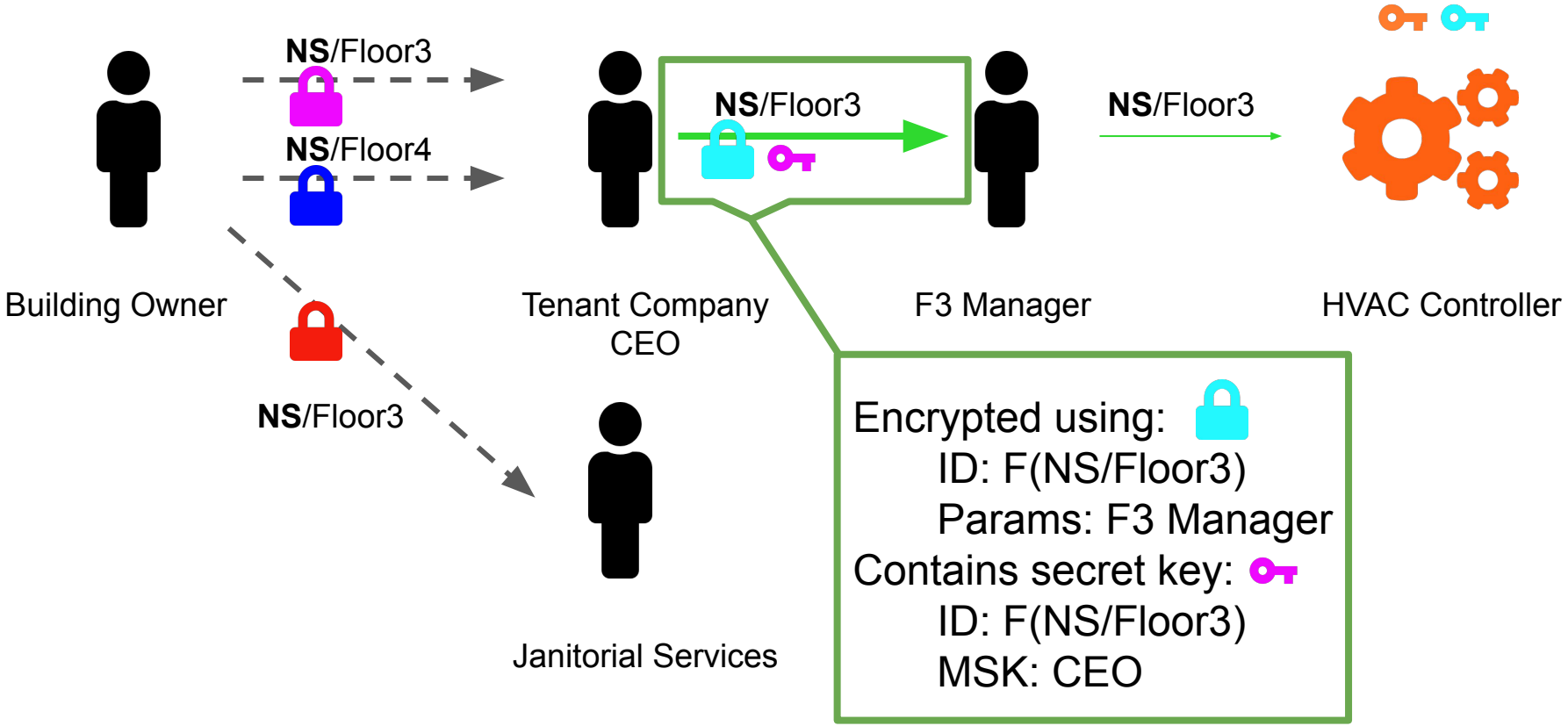
IBE protects URLs

WIBE protects keys

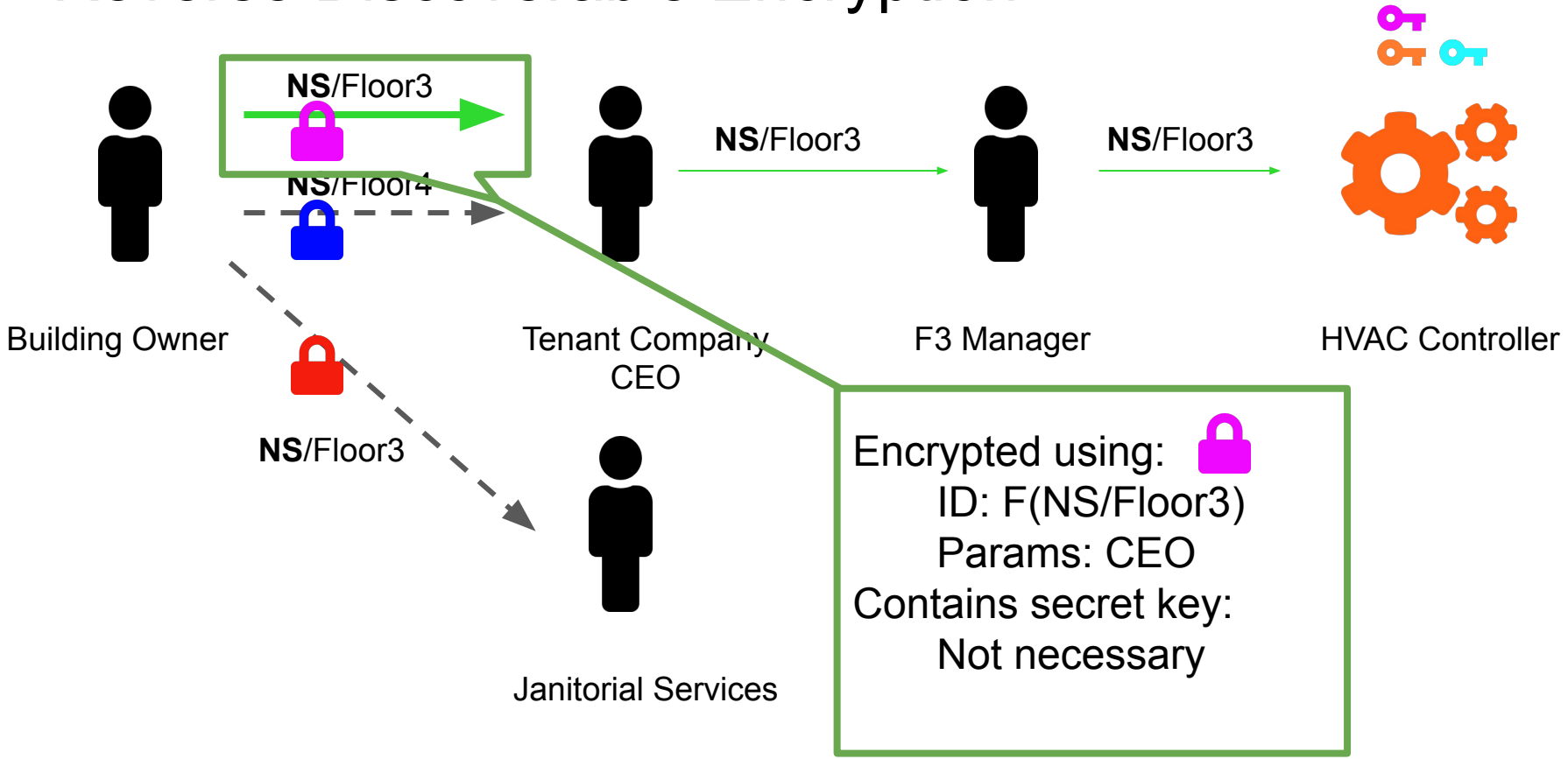
Reverse Discoverable Encryption



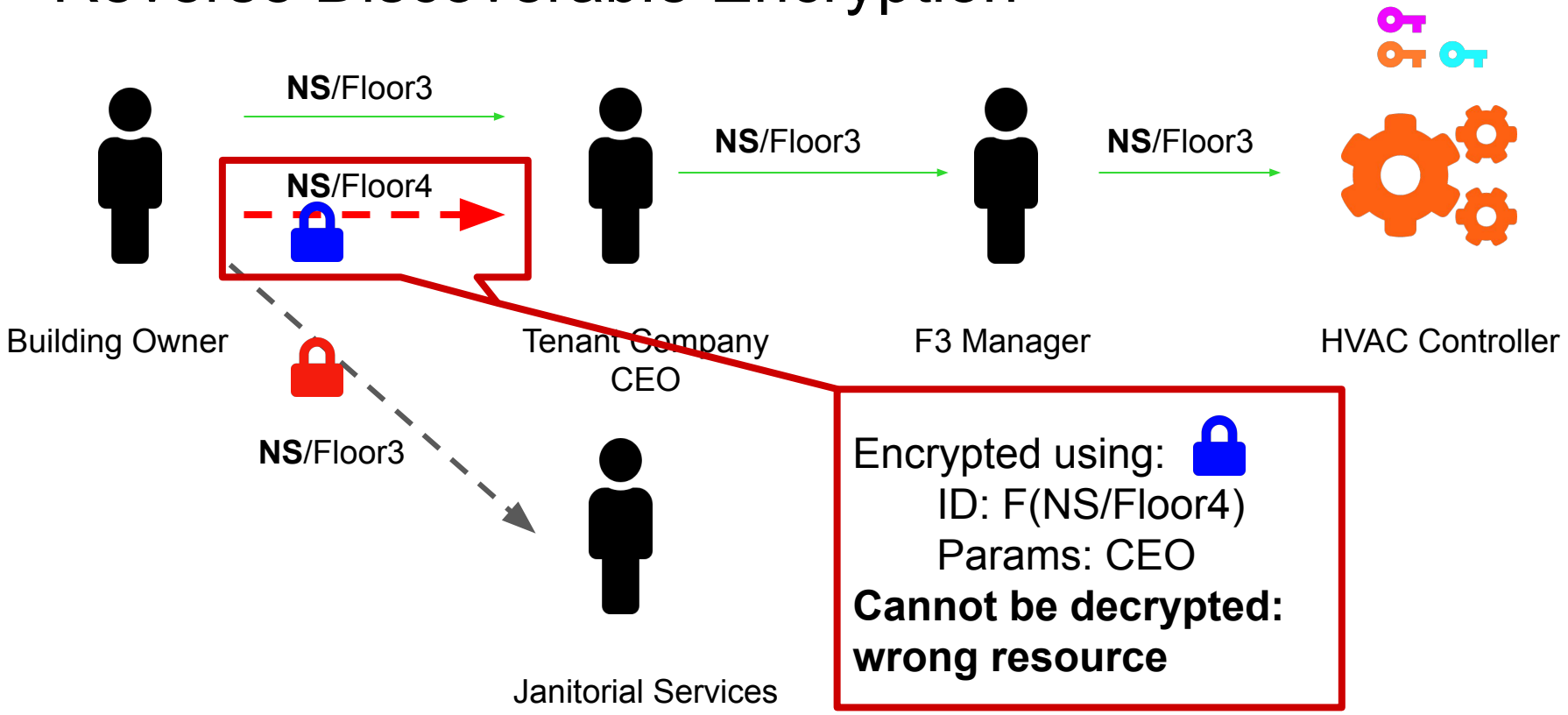
Reverse Discoverable Encryption



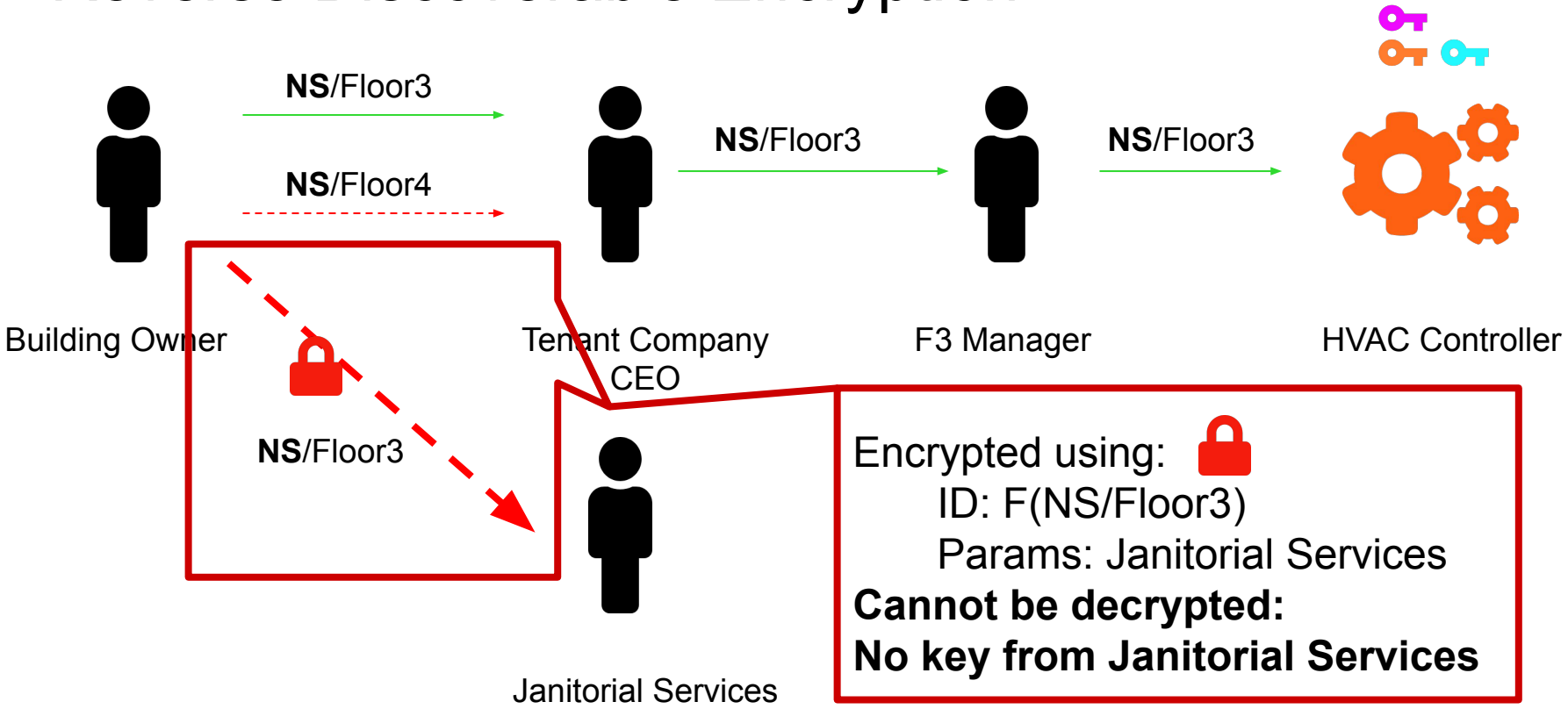
Reverse Discoverable Encryption



Reverse Discoverable Encryption



Reverse Discoverable Encryption

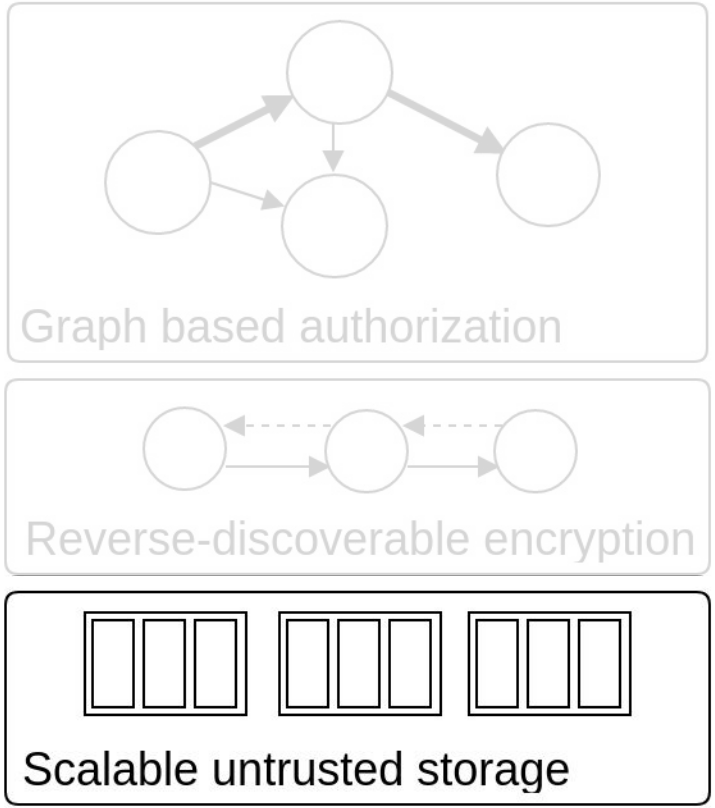


Reverse Discoverable Encryption Summary

- Allows entities to decrypt attestations that they can use in a proof
- Does not require out of band communication
- Works when attestations are granted in any order

Full version (in paper) supports expiry of attestations

We need a place to store the encrypted attestations



A “classical” blockchain (e.g. Ethereum) nearly works

- Our earlier work used a blockchain
  - Cryptographically proven integrity
  - No central authorities

- Unfortunately it didn’t scale well
  - Blockchains don’t really go past a few tens of transactions per second
  - Especially if transactions are large (attestation objects)

Unequivocable Log Derived Map (ULDM)

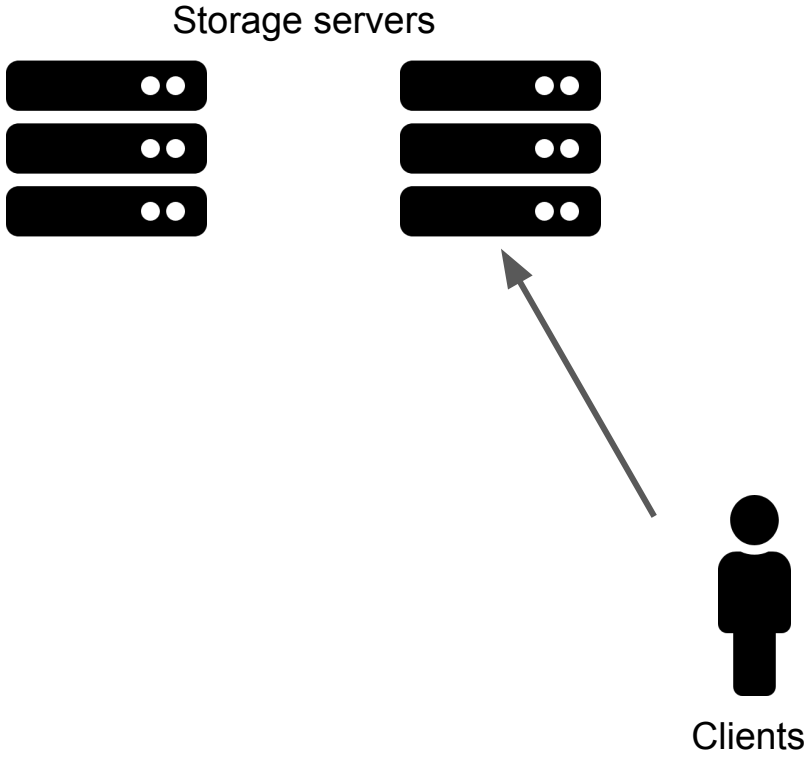
ULDM aims to provide similar guarantees to a blockchain, when only storing objects

Horizontally scalable public ledger with cryptographic integrity proofs

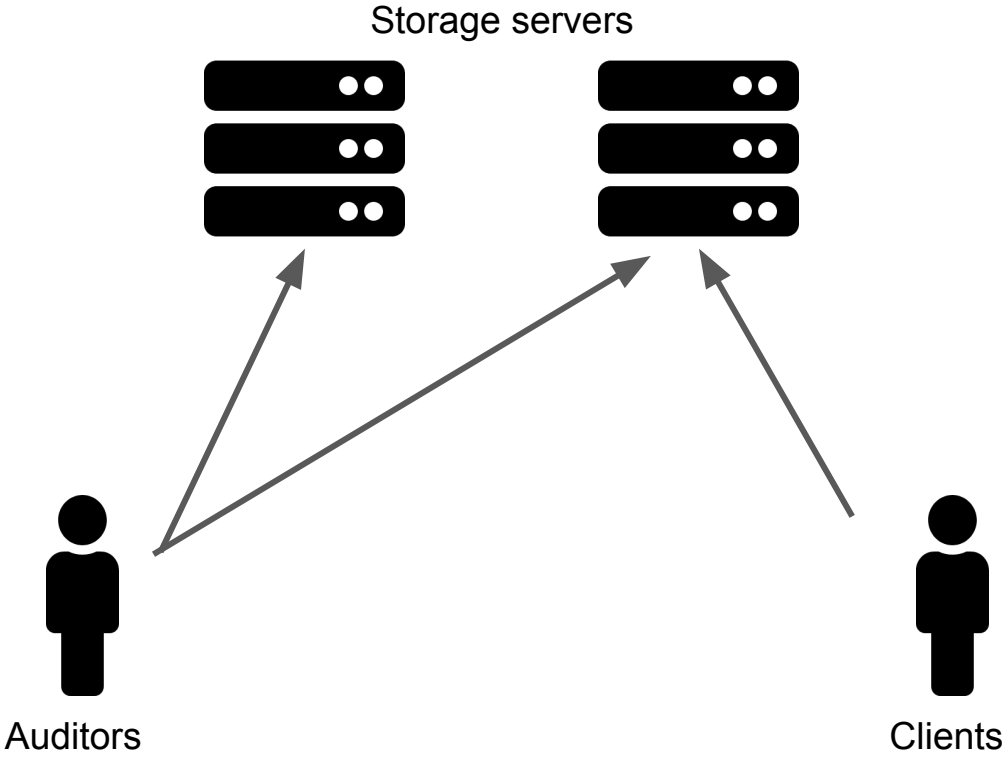
Similar to Certificate Transparency or Key Transparency, except:

1) Over CT, it supports efficient proof of non-existence, which allows revocation
2) Over KT, no need for monitoring by clients in every epoch

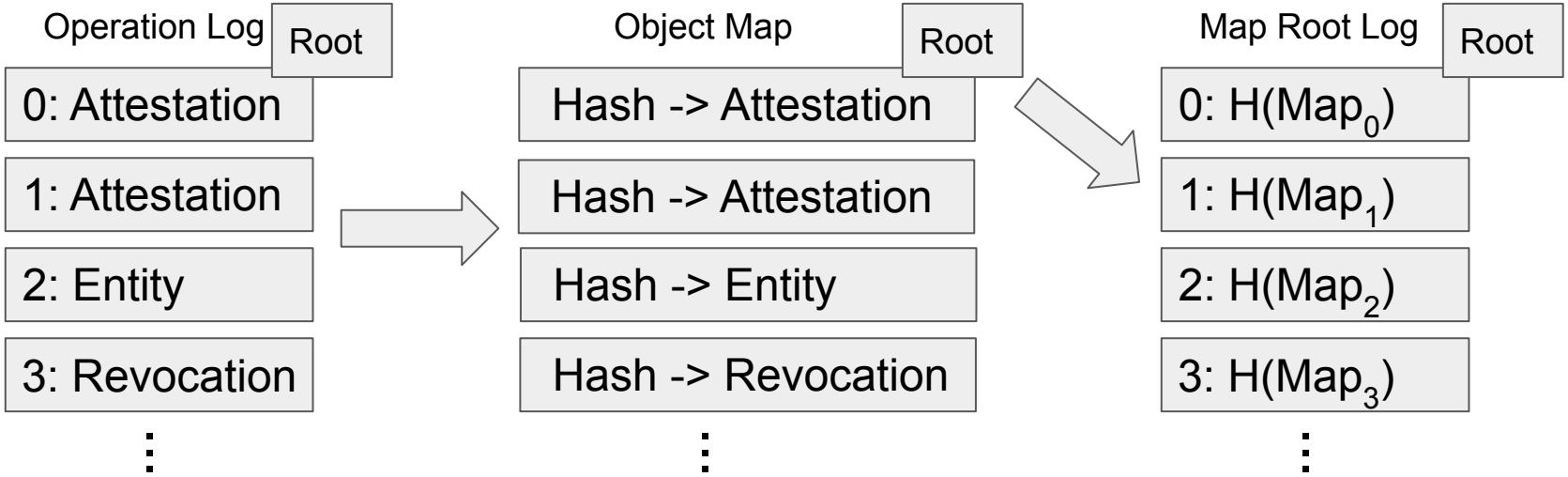
High Level Overview



High Level Overview



Constructed using three Merkle trees



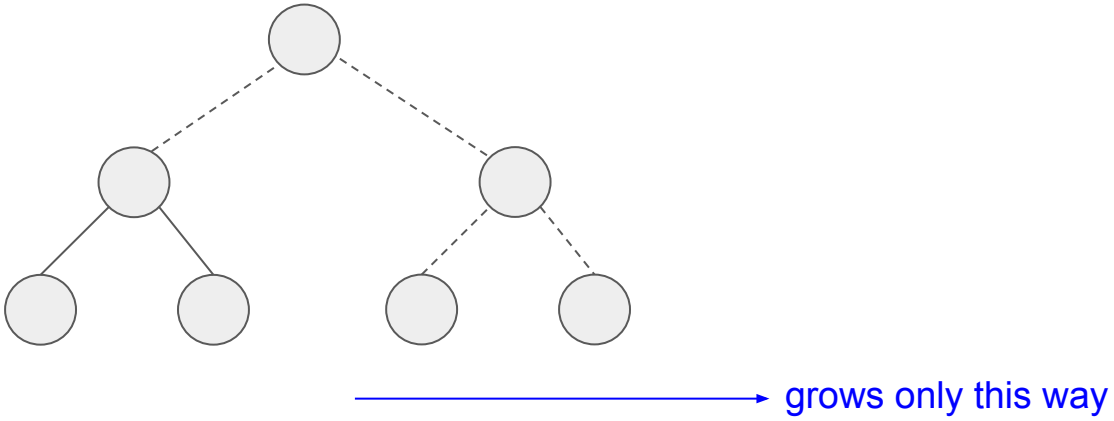
Merkle Tree Log
Can prove:
- Append-only
- Value exists in log

Merkle Tree Map
Can prove:
- Value does not exist
- Value exists

Merkle Tree Log
Can prove:
- Append-only
- Value exists in log

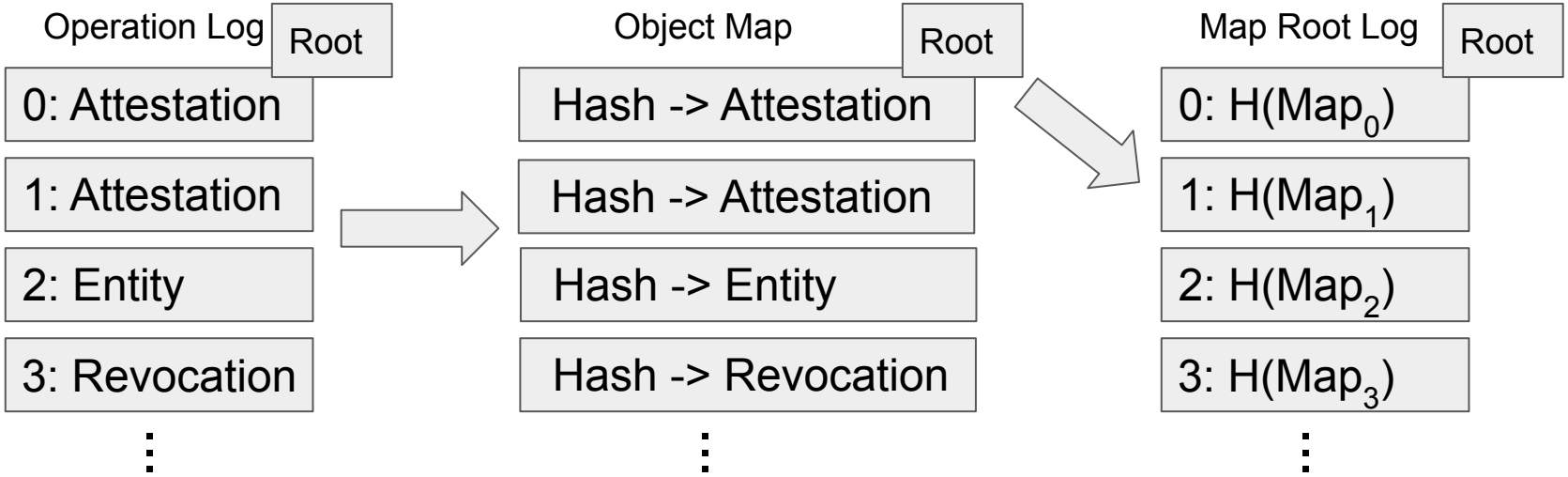
Operation Log ~ Certificate Transparency

Storage server stores Merkle tree, which acts like append-only log



Server publishes signed MH Root on every epoch to auditors
Auditors check that current tree is an extension of the previous tree against the two Merkle roots. How?

Constructed using three Merkle trees

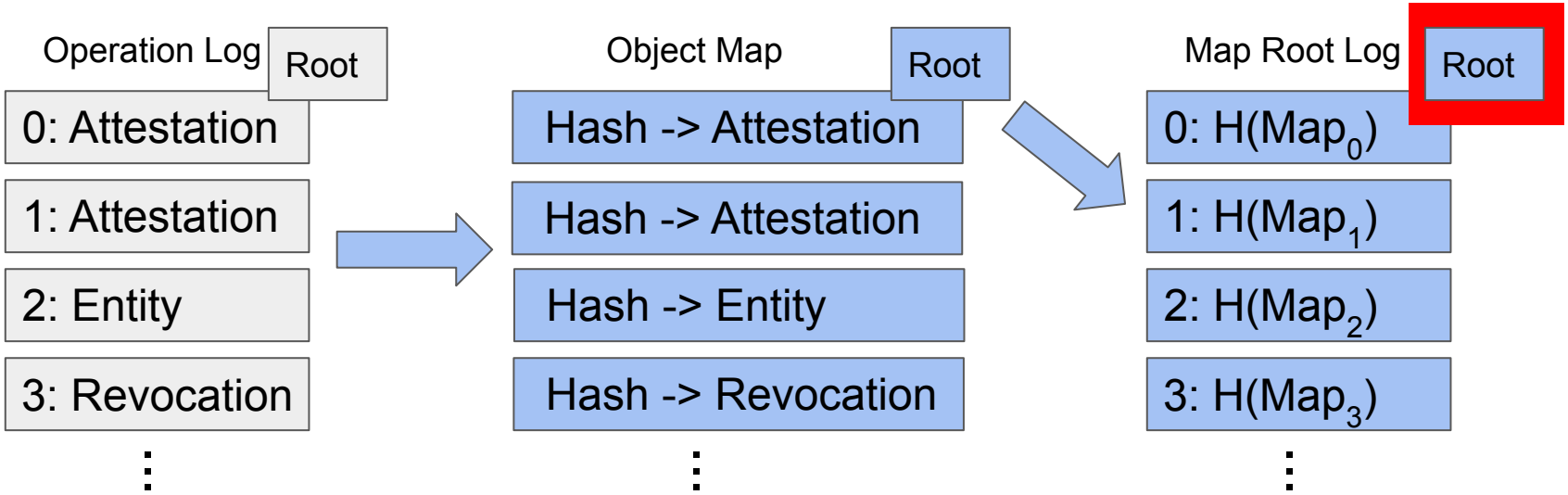


Merkle Tree Log
Can prove:
- Append-only
- Value exists in log

Merkle Tree Map
Can prove:
- Value does not exist
- Value exists
(Lookup is by hash of value)
HOW?

Merkle Tree Log
Can prove:
- Append-only
- Value exists in log

Auditor replays operation log to construct replica



Ensures Object Map is properly derived from operation log

Clients send Root Hash of Map Root Log to auditors periodically (daily)

- Ensures every client is seeing the same data structure

Revocation = the ability to revoke an attestation

Ideas?

We can place the revoked attestation on the ledger with the message ``revoked''
The ledger supports proofs of non-inclusion which helps
Issues with this?

- Need to protect the attestation still, so some sort of RDE?
- Need to make sure that only permitted entity revokes it.

Another idea is for the issuer of the attestation to include a nonce in the attestation, and sign that nonce with the revocation message and put in the ledger
Issues with this?

- The authorization verifier needs to lookup the signature in the ledger, but does not know what it is

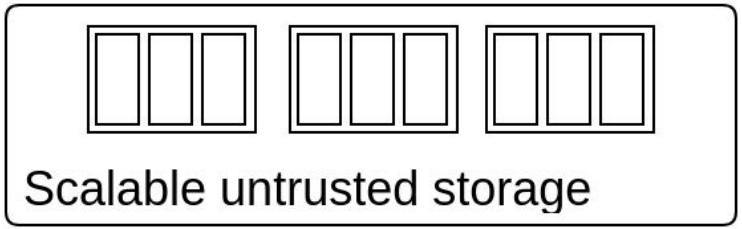
Revocation in WAVE

Each attestation contains h(s) for s a random seed

When revoking, place s in storage indexed by h(s)

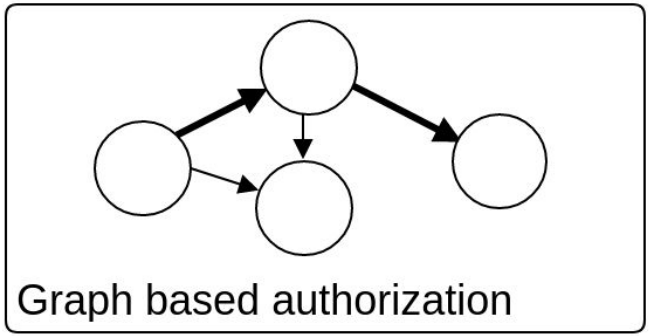
Anyone wishing to check that the attestation was not revoked will query by h(s) and check that s is the hash preimage

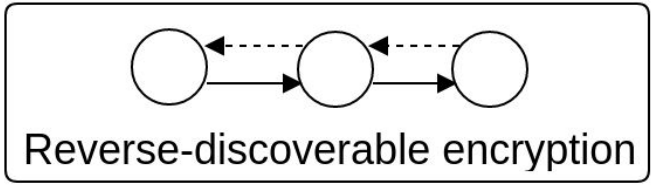
Unequivocable Log Derived Map Summary

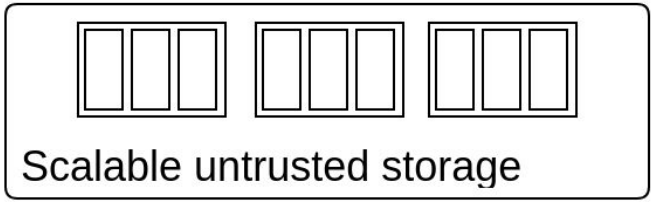


- Stores encrypted attestations, public entity objects, revocations
- Uses cryptographic proofs of integrity
- Forces operators to be honest, or be detected as dishonest
- Auditing requires infrequent communication between clients and auditors

WAVE is implemented







It's written in Go, with some crypto in C++

github.com/immesys/wave

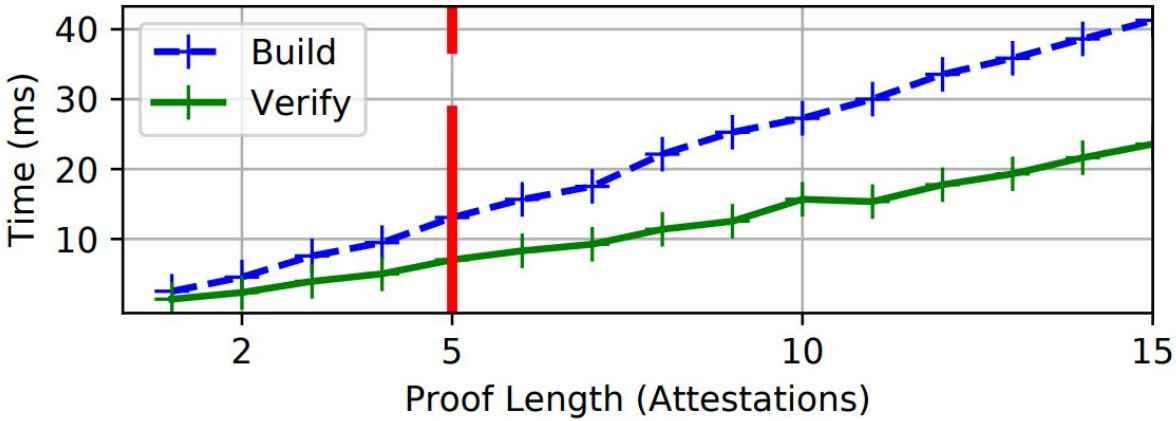
We've used various versions of WAVE over the course of three years:

>200 devices, 20 buildings, multiple namespaces and organizations

It’s pretty fast

- Graph-changing operations - very fast by UI standards:
  - Creating an entity takes 9ms
  - Creating an attestation takes 43 ms
  - Decrypting an attestation takes 6ms

- Proof building / verification:



Conclusion

WAVE is a decentralized authorization system that
offers transitive delegation by using graph based authorization

- Stores the graph in global storage with cryptographically enforced integrity
- Encrypts attestations, hiding the graph
- It can be used in place of most traditional authorization systems