WAVE: A Decentralized Authorization Framework with Transitive Delegation

[Andersen et al., USENIX Security 2019]
Representative authorization example

Building Owner

Tenant Company
CEO

Employees

BLDG2/Floor3
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Traditional approach

Building Owner

BLDG2/Floor3

E.g. OAuth, LDAP

Tenant Company
CEO

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS

Employees
Traditional approach

Building Owner

BLDG2/Floor3

Tenant Company
CEO

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS

Employees

Problems:
Central point of attack
Traditional approach

Problems:
Central point of attack
Can’t even trust operator

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

Building Owner
BLDG2/Floor3

Tenant Company
CEO
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS

Employees

Problems:
Central point of attack
Can’t even trust operator
Traditional approach

Problems:
Central point of attack
Can’t even trust operator

Building Owner

BLDG2/Floor3

Tenant Company CEO

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS

Employees
Traditional approach

**Problems:**
Central point of attack
Can’t even trust operator

Building Owner

Tenant Company CEO

Employees

BLDG2/Floor3
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Traditional approach

Problems:
Central point of attack
Can’t even trust operator

Building Owner → BLDG2/Floor3 → Tenant Company CEO

BLDG2/Floor3/HVAC → BLDG2/Floor3/LIGHT → BLDG2/Floor3/DOORS → Employees
Traditional approach

Problems:
Central point of attack
Can’t even trust operator
Sometimes delegation unsupported

Building Owner  Tenant Company  CEO  Employees

BLDG2/Floor3  BLDG2/Floor3/HVAC  BLDG2/Floor3/LIGHT  BLDG2/Floor3/DOORS
Traditional approach

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

Building Owner

Tenant Company
CEO

Employees

BLDG2/Floor3

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Lack of transitive delegation

Building Owner → Tenant Company CEO → Employees

BLDG2/Floor3
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Lack of transitive delegation
What we want:

Building Owner

Tenant Company
CEO

Employees

BLDG2/Floor3
BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
Existing work lacks some important features

<table>
<thead>
<tr>
<th>System / Work</th>
<th>Avoid central authority</th>
<th>Transitive Delegation</th>
<th>Permission Discovery</th>
<th>No ordering constraints</th>
<th>Offline participants</th>
<th>Protected permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP, AD</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>OAuth2</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Macaroons</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>SDSI/SPKI</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
**What is WAVE**

<table>
<thead>
<tr>
<th>System / Work</th>
<th>No central authority</th>
<th>Transitive Delegation</th>
<th>Permission Discovery</th>
<th>No ordering constraints</th>
<th>Offline participants</th>
<th>Protected permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WAVE</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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
- Reverse-discoverable encryption
- Scalable untrusted storage
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

Building Owner → BLDG2/Floor3

Tenant Company
CEO

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS

Employees
Graph Based Authorization

Participants: Entities
Collections of cryptographic keys: identifier is PK
Graph Based Authorization

Grants of permissions: Attestations
Signed certificates created by Entities

Building Owner

Tenant Company
CEO

Employees

BLDG2/Floor3

BLDG2/Floor3/HVAC
BLDG2/Floor3/LIGHT
BLDG2/Floor3/DOORS
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.

Permission: Read, Write

Resource: BldgOwner/BLDG2

Expires: 2019/04/05
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

Building Owner → Tenant Company CEO → Employees
Reverse Discoverable Encryption

Building Owner  <->  Tenant Company CEO  <->  F3 Manager

NS/Floor3  <->  NS/Floor4  <->  NS/Floor3
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

Janitorial Services
Reverse Discoverable Encryption

Building Owner

NS/Floor3

NS/Floor4

Tenant Company
CEO

F3 Manager

HVAC Controller

Janitorial Services

NS/Floor3
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services

Discovering permissions
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

Building Owner → NS/Floor3 → Tenant Company CEO → NS/Floor3 → F3 Manager → NS/Floor3 → HVAC Controller

Janitorial Services → NS/Floor3
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services
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

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services

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

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Janitorial Services

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 \(\rightarrow\) issuer \rightarrow\) subject

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

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)
)

WIBE protects keys
Reverse Discoverable Encryption

Building Owner

Tenant Company CEO

F3 Manager

HVAC Controller

Encrypted using:
ID: F(NS/Floor3)
Params: Controller
Contains secret key:
ID: F(NS/Floor3)
MSK: F3 Manager

Janitorial Services
Reverse Discoverable Encryption

Encrypted using: 
ID: F(NS/Floor3) 
Params: F3 Manager 
Contains secret key: 
ID: F(NS/Floor3) 
MSK: CEO
Reverse Discoverable Encryption

Encrypted using:
ID: F(NS/Floor3)
Params: CEO
Contains secret key: Not necessary
Reverse Discoverable Encryption

Encrypted using: ID: F(NS/Floor4)
Params: CEO
Cannot be decrypted: wrong resource
Reverse Discoverable Encryption

Encrypted using: ID: F(NS/Floor3)
Params: Janitorial Services

Cannot be decrypted:
No key from Janitorial Services
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

Graph based authorization

Reverse-discoverable encryption

Scalable untrusted storage
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

Storage servers

Clients
High Level Overview

Storage servers

Auditors

Clients
Constructed using three Merkle trees

Operation Log
- 0: Attestation
- 1: Attestation
- 2: Entity
- 3: Revocation

Object Map
- Hash -> Attestation
- Hash -> Entity
- Hash -> Revocation

Map Root Log
- 0: \( H(\text{Map}_0) \)
- 1: \( H(\text{Map}_1) \)
- 2: \( H(\text{Map}_2) \)
- 3: \( H(\text{Map}_3) \)

Merkle Tree Log
- Append-only
- Value exists in log

Merkle Tree Map
- Value does not exist
- Value exists

Merkle Tree Log
- 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

### Operation Log
- 0: Attestation
- 1: Attestation
- 2: Entity
- 3: Revocation
- ...

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

### Merkle Tree Map
- Hash -> Attestation
- Hash -> Entity
- Hash -> Revocation
- ...

### Object Map
- Hash -> Attestation
- Hash -> Entity
- Hash -> Revocation
- ...

### Map Root Log
- 0: H(Map_0)
- 1: H(Map_1)
- 2: H(Map_2)
- 3: H(Map_3)
- ...

### Map Root Log
- Root

### HOW?
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:

![Graph showing time vs. proof length](image)
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