Hewlett Packard
Enterprise

Libfabric Authorization Key Ring Proposal

Ian Ziemba

August 8, 2023

Agenda

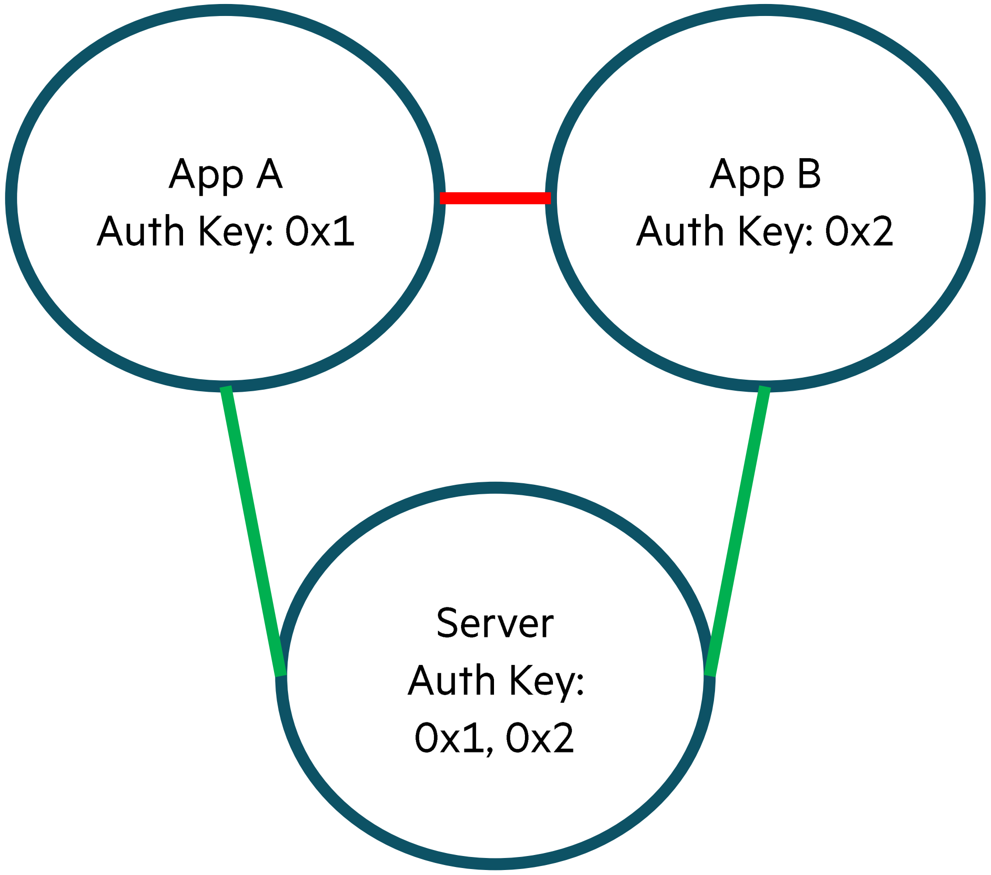
- Authorization Key Overview
- Problem Statement
- Solutions
- New Capabilities to Support Authorization Key Ring
- Libfabric Authorization Key Ring Object
- Server Workflows
- Retrieving Authorization Key from CQ Event
- Using Authorization Key for Local RDMA Operations
- Key Takeaways

Authorization Key Overview

- Authorization keys are used to limit communication between endpoints
  - Only peer endpoints that are programmed to use the same authorization key may communicate
  - fid_domain, fid_mr, and fid_ep may be associated with a **single** authorization key
- In practice, each independent libfabric application is associated with at least one authorization key
  - Using unique authorization keys per application prevents applications from erroneously or maliciously issuing RDMA operations to each other

Problem Statement

- In client/server environments with RDM endpoints, expectation is persistent server instances need to communicate with multiple independent clients in a secure manner
- Issue: While client/server software may implement encryption, authentication, and authorization, none of these isolate RDMA traffic from independent clients
  - Solution: Use unique authorization keys for each independent client



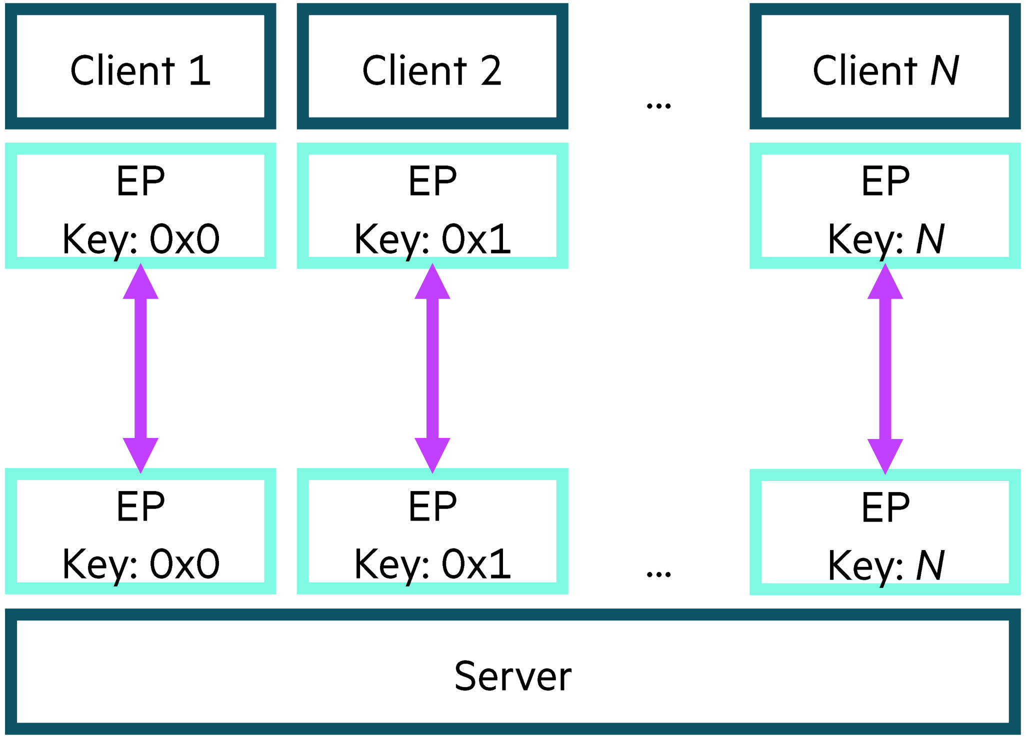
App A and App B can communication with server since same auth keys are used (and vice versa). App A cannot communicate with App B since different auth keys are used (and vice versa).

Problem Statement

- **Issue: Servers need to support multiple client authorization keys in a scalable and performant manner**
  - Requirement 1: Servers must scale to many client authorization keys
    - Need at least 16-bits worth of authorization keys
    - 32-bits worth of authorization keys would be ideal
  - Requirement 2: Servers must map incoming RDMA operation fi_addr_t to authorization key
    - Need handles to represent authorization keys instead of operating on opaque blobs
  - Requirement 3: Servers must respond to clients using the client specific authorization key
    - Need ability to assign an authorization key per RDMA operation

Solution 1: Endpoint per Authorization Key

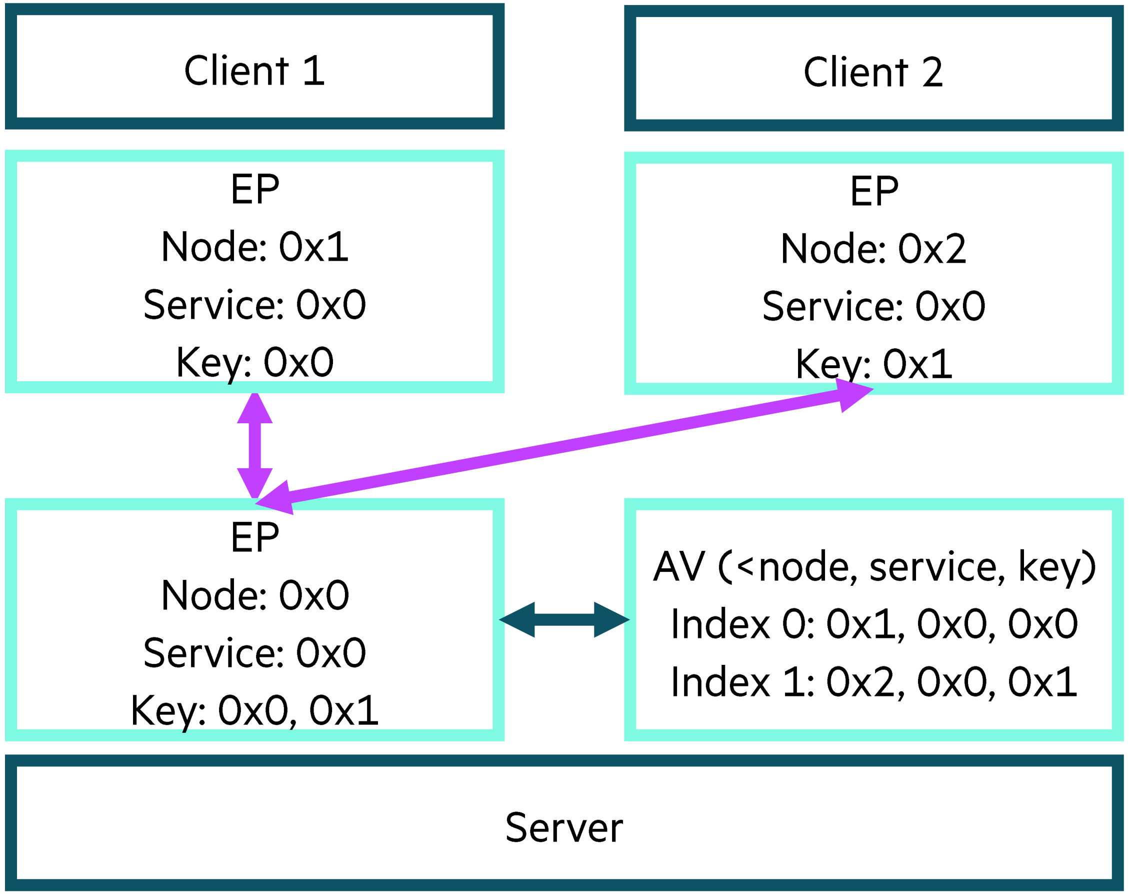
- Servers allocating an endpoint per client authorization key would satisfy all requirements
- Pros
  - Supported by libfabric today
- Cons
  - Servers must manage multiple endpoints
  - **Providers may not be able to support required number of endpoints**



Each client has its own auth key to communicate with the server. This requires the server to have *N* endpoints where *N* equals number of client auth keys.

Solution 2: Add Authorization Key Extension to Address Vector (AV) APIs

- AV API would be extended to support inserting authorization key with endpoint address
  - Example:
    ```
    fi_av_insert_auth_key(
    struct fid_av *av, void *addr,
    size_t count, fi_addr_t *fi_addr,
    void *auth_key, size_t auth_key_len,
    uint64_t flags, void *context)
    ```
- Pros
  - Single endpoint can support multiple authorization keys
- Cons
  - Complicates endpoint's view of valid authorization keys
- Example: After endpoint enablement, the server inserts a new client AV entry with a new authorization key
  - Should the endpoint support this new authorization key?
    - If yes, this requires dynamically updating endpoint state
    - If no, how can servers support new clients with new authorization keys?
  - **Dynamic modification of endpoint state may lead to unexpected runtime failures**



Client 1

Client 2

EP
Node: 0x1
Service: 0x0
Key: 0x0

EP
Node: 0x2
Service: 0x0
Key: 0x1

EP
Node: 0x0
Service: 0x0
Key: 0x0, 0x1

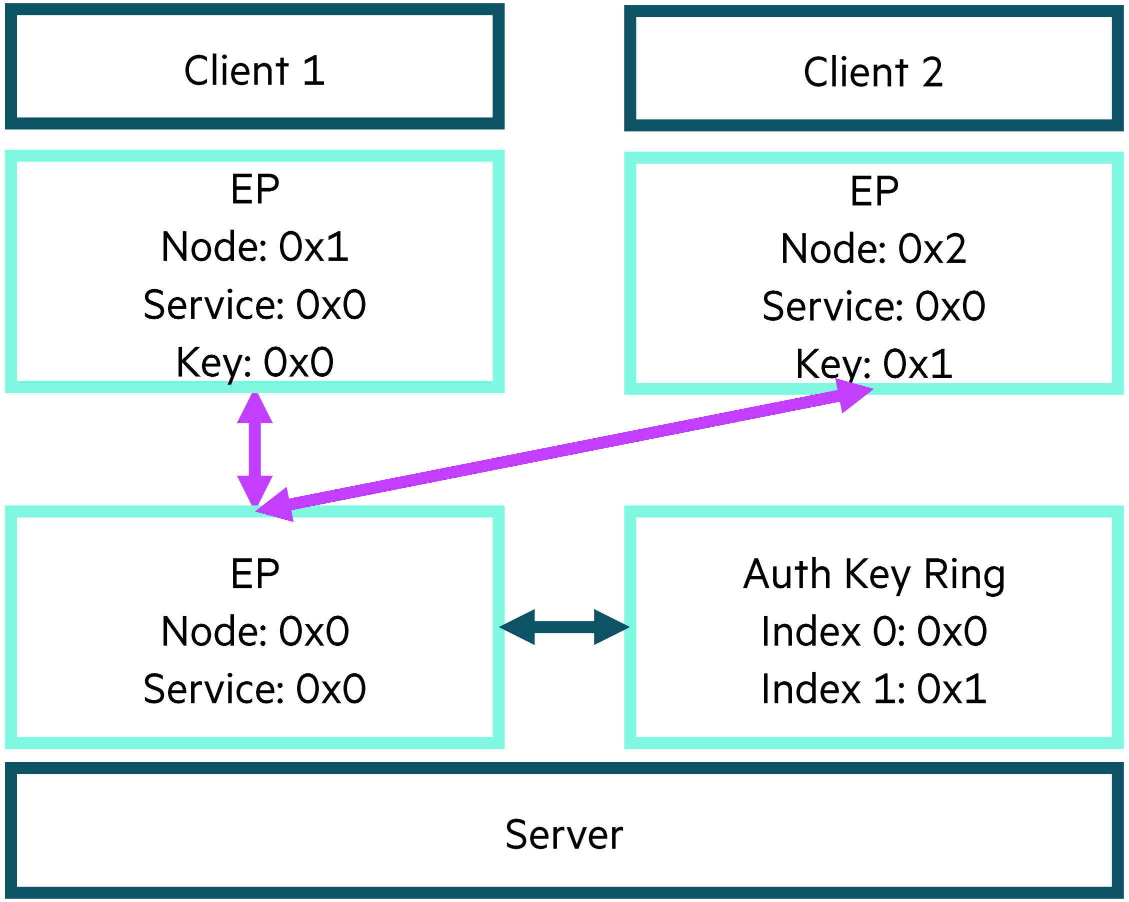
AV (<node, service, key)
Index 0: 0x1, 0x0, 0x0
Index 1: 0x2, 0x0, 0x1

Server

Server AV defines the supported auth keys. Adding new auth keys against already enabled endpoint may lead to runtime failures.

Solution 3: Authorization Key Ring

- Support an endpoint to be bound and enabled against *N* authorization keys
  - Authorization key ring is container of authorization keys which are bound to endpoints
  - Receive based RDMA operation will report the authorization key used
  - Authorization key can be specified per local RDMA operation
- Pros
  - Single endpoint can support multiple authorization keys
  - All needed authorization keys can be bound to endpoint before enablement
    - Avoids runtime failures
- Cons
  - None?



Authorization key ring defines the auth keys the endpoint should be enabled against. AV management is separate from authorization key ring management.

New Capabilities to Support Authorization Key Ring

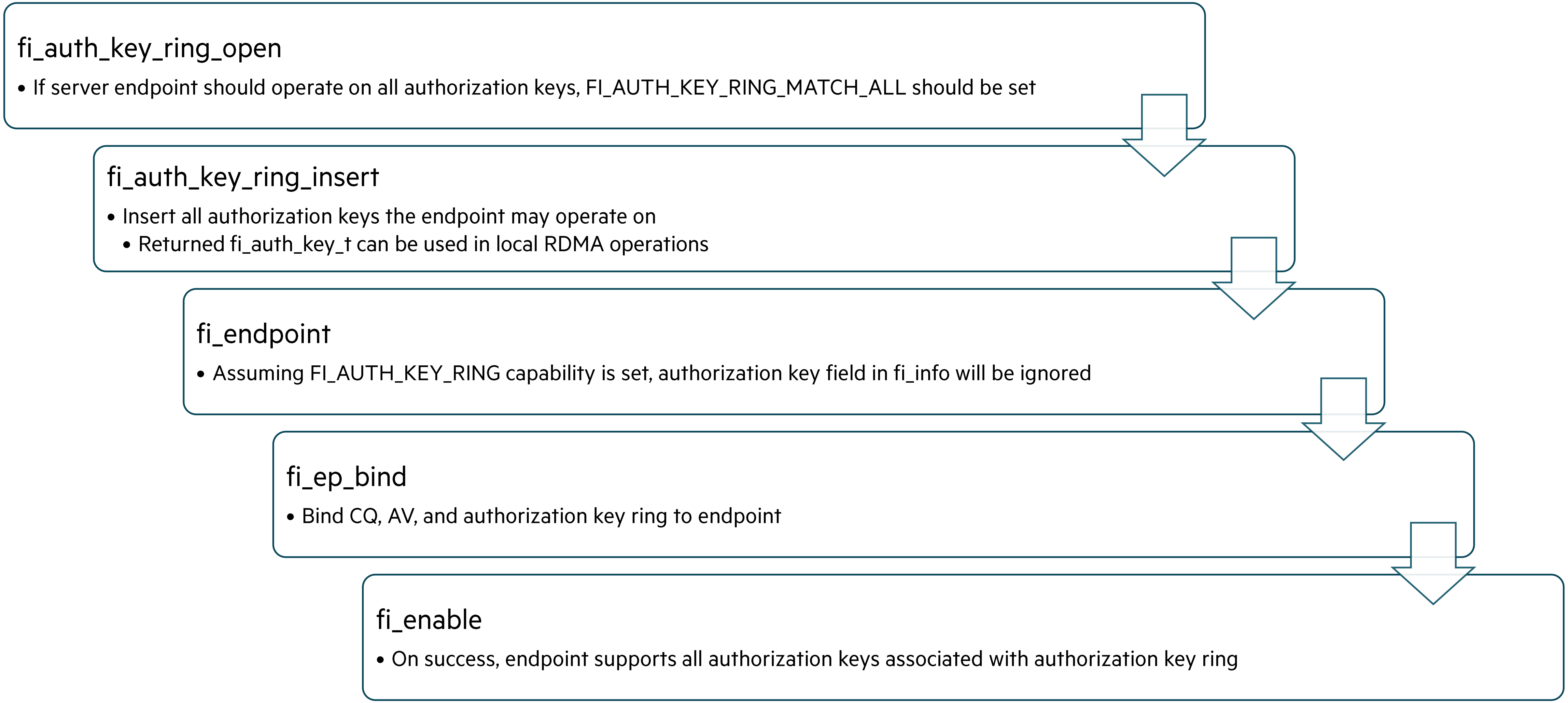
- FI_AUTH_KEY_RING
  - A provider supports binding an authorization key ring to an endpoint
  - If capability is set during endpoint creation, the authorization key in the endpoint fi_info will be ignored
  - Requires providers to support changing authorization key per transmit-based RDMA operation
  - Requires providers to support an endpoint receiving on one or more authorization keys
    - Authorization key ring would define the exact number an endpoint would need to support
- FI_SOURCE_AUTH_KEY
  - Paired with FI_AUTH_KEY_RING
  - Requests that the endpoint return the source authorization key (fi_auth_key_t) data as part of its completion data
- FI_NO_SOURCE_AUTH_KEY
  - Paired with FI_SOURCE_AUTH_KEY
  - Flag used per MR and recv/trecv to signal to provider that corresponding completion events do not need to return source authorization key data
  - Optimization to avoid potential reverse lookup to retrieve fi_auth_key_t
- FI_RECV_AUTH_KEY
  - Paired with FI_AUTH_KEY_RING
  - Provider supports restricting a recv/trecv to a specific fi_auth_key_t

Libfabric Authorization Key Ring Object

- A fid_auth_key_ring is a vector of authorization keys which can be optionally bound to an endpoint
- Two operations supported
  - Insert: Insert a new authorization key
    - On success, a fi_auth_key_t handle will be returned
  - Lookup: Retrieve the authorization key for a given fi_auth_key_t
- FI_AUTH_KEY_RING_MATCH_ALL flag signals to providers that all authorization keys will be used
  - Providers should set up authorization key ring to support all authorization keys
  - No requirement to insert authorization keys into key ring
    - Inserts can still be done and a fi_auth_key_t will be returned
- FI_AUTH_KEY_RING_SYMMETRIC flag signals to providers that authorization keys must map to the same fi_auth_key_t
  - Enables multiple endpoints belonging to different domains to have the same authorization key ring view

```
#define FI_AUTH_KEY_RING_MATCH_ALL (1U << 0)
#define FI_AUTH_KEY_RING_SYMMETRIC (1U << 1)

struct fid_auth_key_ring_attr {
    uint64_t flags;
};

static inline int fid_auth_key_ring_open(
struct fid_domain *domain,
struct fid_auth_key_ring_attr *attr,
struct fid_auth_key_ring **auth_key_ring,
void *context);

static inline int fid_auth_key_ring_insert(
struct fid_auth_key_ring *auth_key_ring,
void *auth_key, size_t auth_key_len,
fi_auth_key_t *fi_auth_key);

static inline int fid_auth_key_ring_lookup(
struct fid_auth_key_ring *auth_key_ring,
fi_auth_key_t fi_auth_key, void *auth_key,
size_t *auth_key_len);
```

Server Workflow: Endpoint Initialization with Authorization Key Ring



fi_auth_key_ring_open

- If server endpoint should operate on all authorization keys, FI_AUTH_KEY_RING_MATCH_ALL should be set

fi_auth_key_ring_insert

- Insert all authorization keys the endpoint may operate on
  - Returned fi_auth_key_t can be used in local RDMA operations

fi_endpoint

- Assuming FI_AUTH_KEY_RING capability is set, authorization key field in fi_info will be ignored

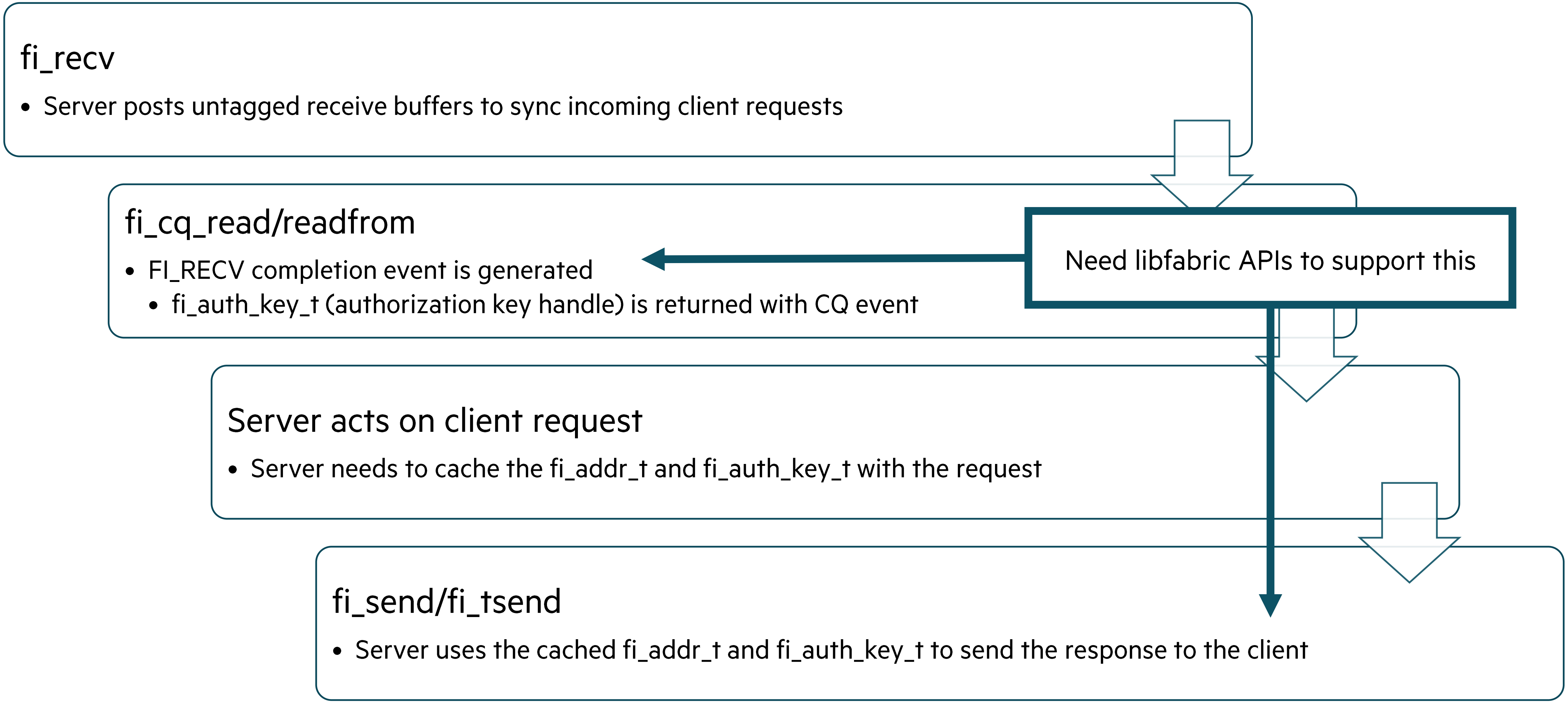
fi_ep_bind

- Bind CQ, AV, and authorization key ring to endpoint

fi_enable

- On success, endpoint supports all authorization keys associated with authorization key ring

Server Workflow: Using Authorization Keys



fi_recv

- Server posts untagged receive buffers to sync incoming client requests

fi_cq_read/readfrom

- FI_RECV completion event is generated
  - fi_auth_key_t (authorization key handle) is returned with CQ event

Need libfabric APIs to support this

Server acts on client request

- Server needs to cache the fi_addr_t and fi_auth_key_t with the request

fi_send/fi_tsend

- Server uses the cached fi_addr_t and fi_auth_key_t to send the response to the client

Retrieving Authorization Key from CQ Event

Option 1: Encode fi_auth_key_t in fi_addr_t

- Based on proposed libfabric-2.0: Tagged message enhancements, fi_auth_key_t will be encoded in the fi_addr_t returned from fi_cq_readfrom
  - A reserved number of bits in the fi_addr_t will encode the fi_auth_key_t
    - Bits [0,31] of the fi_addr_t will be for fi_addr_t
    - Bits [32,47] of the fi_addr_t will encode fi_auth_key_t
- FI_SOURCE_AUTH_KEY flag in CQ event will denote if fi_addr_t encodes a fi_auth_key_t
- If fi_cq_reader() returns FI_EADDRNOTAVAIL, CQ error olen field is treated as fi_auth_key_t
- Getters and setters will be defined to extract fi_auth_key_t and fi_addr_t from encoded fi_addr_t
- Pros
  - Aligns with libfabric-2.0: Tagged message enhancements proposal
- Cons
  - Forces AV to be always be FI_AV_TABLE
  - May limit number of authorization keys to ~65,536

```
/* Used if CQ event returns FI_SOURCE_AUTH_KEY */
static inline fi_auth_key_t
fi_addr_decode_auth_key(fi_addr_t fi_addr);

static inline fi_addr_t
fi_addr_decode_addr(fi_addr_t fi_addr);

/* Used when fi_cq_readerr() returns
 * FI_EADDRNOTAVAIL
 */
static inline fi_auth_key_t
fi_cq_err_get_auth_key(
struct fi_cq_err_entry *err);
```

Retrieving Authorization Key from CQ Event

Option 2: New fi_cq_readfrom APIs

- Define fi_cq_readfrom2 APIs which would return a fi_auth_key_t
  - Precedence already set for defining version 2 of APIs
    - Example: Domain and Endpoint allocation
- Pros
  - ~64-bits worth of authorization keys could be supported
- Cons
  - Not all providers may support new fi_cq_readfrom2/sreadfrom2 calls
    - May be able to stub a core implementation for providers which do not support these calls directly

```
static inline ssize_t fi_cq_readfrom2(
struct fid_cq *cq, void *buf, size_t count,
fi_addr_t *src_addr, fi_auth_key_t *src_key);

static inline ssize_t fi_cq_sreadfrom2(
struct fid_cq *cq, void *buf, size_t count,
fi_addr_t *src_addr, fi_auth_key_t *src_key,
const void *cond, int timeout);
```

Using Authorization Key for Local RDMA Operations

Option 1: Encode fi_auth_key_t in fi_addr_t

- Intended to be paired with *Retrieving Authorization Key from CQ Event: Encode fi_auth_key_t in fi_addr_t*
  - fi_auth_key_t will be encoded in the fi_addr_t
- Endpoint configured with FI_AUTH_KEY_RING
  - A valid fi_auth_key_t must be provided with FI_SEND, FI_RMA, and FI_AMO operations
- Endpoint configured with FI_RECV_AUTH_KEY
  - A valid fi_auth_key_t must be provided with FI_RECV operations
  - FI_AUTH_KEY_UNSPEC (i.e. match any) will be supported
- Pros
  - Aligns with libfabric-2.0: Tagged message enhancements proposal
- Cons
  - Forces AV to be always be FI_AV_TABLE
  - May limit number of authorization keys to ~65,536

```
/* Return fi_addr_t can be passed into local RDMA
 * operations.
 */
static inline fi_addr_t
fi_addr_encode_auth_key(fi_addr_t fi_addr,
fi_auth_key_t fi_auth_key);
```

Using Authorization Key for Local RDMA Operations

Option 2: New FI_OPT_AUTH_KEY_TRANSMIT/RECV Endpoint Operation

- For non-message style local RDMA operations (e.g. fi_send/fi_sendv), the fi_auth_key_t used for these operations comes from an endpoint property
- fi_setopt + FI_OPT_AUTH_KEY_TRANSMIT
  - Set fi_auth_key_t for non-msg style transmit based operations (e.g. FI_SEND, FI_RMA, and FI_AMO)
- fi_setopt + FI_OPT_AUTH_KEY_RECV
  - Set fi_auth_key_t for non-msg style receive based operations (e.g. FI_RECV)
  - FI_AUTH_KEY_UNSPEC will be supported
- Users must set endpoint authorization keys before issuing first non-msg style RDMA operation

```
 /* Endpoint option levels */
enum {
        FI_OPT_RX_SIZE,
        FI_OPT_FI_HMEM_P2P,
        FI_OPT_XPU_TRIGGER,
+       FI_OPT_AUTH_KEY_TRANSMIT,
+       FI_OPT_AUTH_KEY_RECV,
};
```

Using Authorization Key for Local RDMA Operations

Option 2 Continued: Add fi_auth_key_t to all Message Structures

- All message style structs (e.g. struct fi_msg) will be extended with a fi_auth_key_t field
  - Enables users to provide a different authorization key per RDMA operation
- Endpoint configured with FI_AUTH_KEY_RING
  - A valid fi_auth_key_t must be provided with FI_SEND, FI_RMA, and FI_AMO operations
- Endpoint configured with FI_RECV_AUTH_KEY
  - A valid fi_auth_key_t must be provided with FI_RECV operations
  - FI_AUTH_KEY_UNSPEC will be supported
- Pros
  - ~64-bits worth of authorization keys could be supported
- Cons
  - fi_msg_rma and fi_msg_tagged will exceed 64-byte cache line

```
struct fi_msg {
        fi_addr_t                  addr;
        void                       *context;
        uint64_t                   data;
+       fi_auth_key_t             auth_key;
};
```

Key Takeaways

- Current libfabric authorization key definition cannot meet client/server security requirements in RDM endpoint environment
- A new libfabric object, called authorization key ring, is needed to support single RDM endpoint transmitting/receiving RDMA operations with different authorization keys
- Completion queue (CQ) API changes are required to associate an authorization key with a CQ event
- Local RDMA API changes are required to associate an authorization key with a RDMA operation

Thank you

Email: ian.ziemba@hpe.com