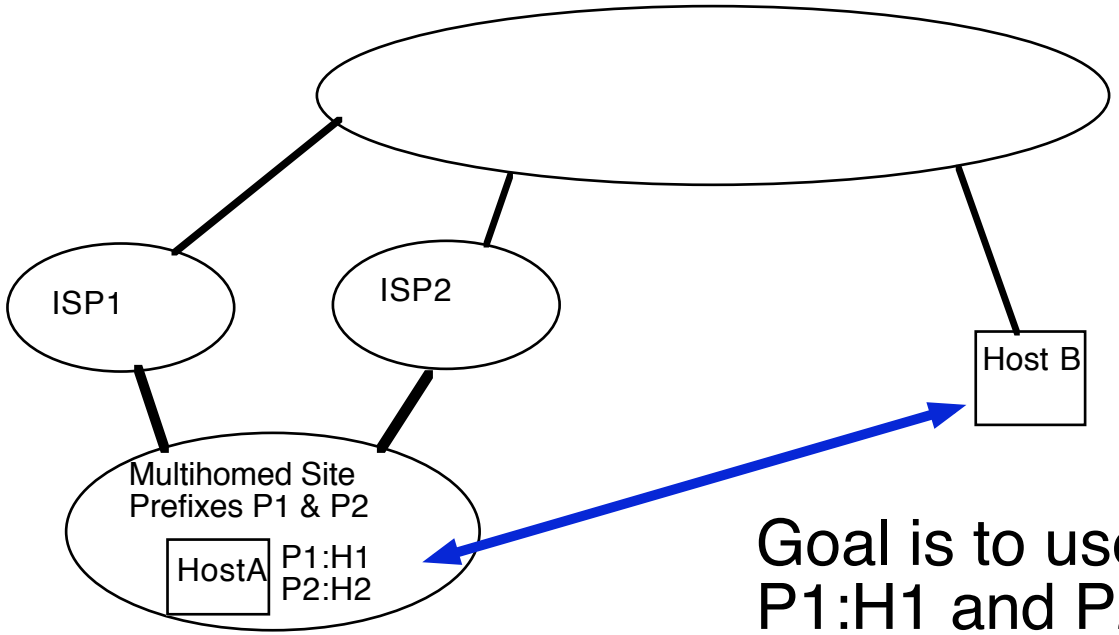
# Crypto Locators
# Hash Based Addresses

shim6 wg meeting - IETF 63

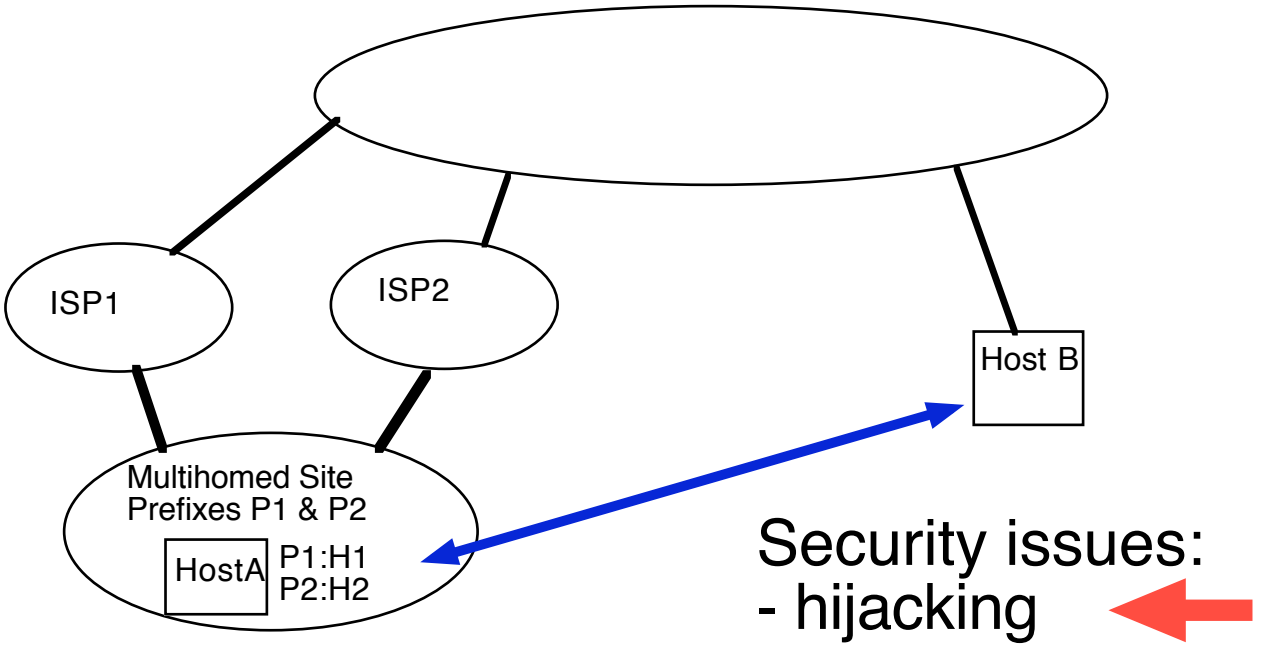
M. Bagnulo

# Multihoming scenario




Goal is to use addresses P1:H1 and P2:H2 in the same communication when needed (e.g.outage)

# Multihoming scenario




Security issues:
- hijacking
- flooding

# Characteristics of HBAs

- Generate sets of securely bound addresses
  - Not vulnerable to time-shifted attacks
  - Not using public key crypto
- The resulting set that is verifiable with hash operation is static

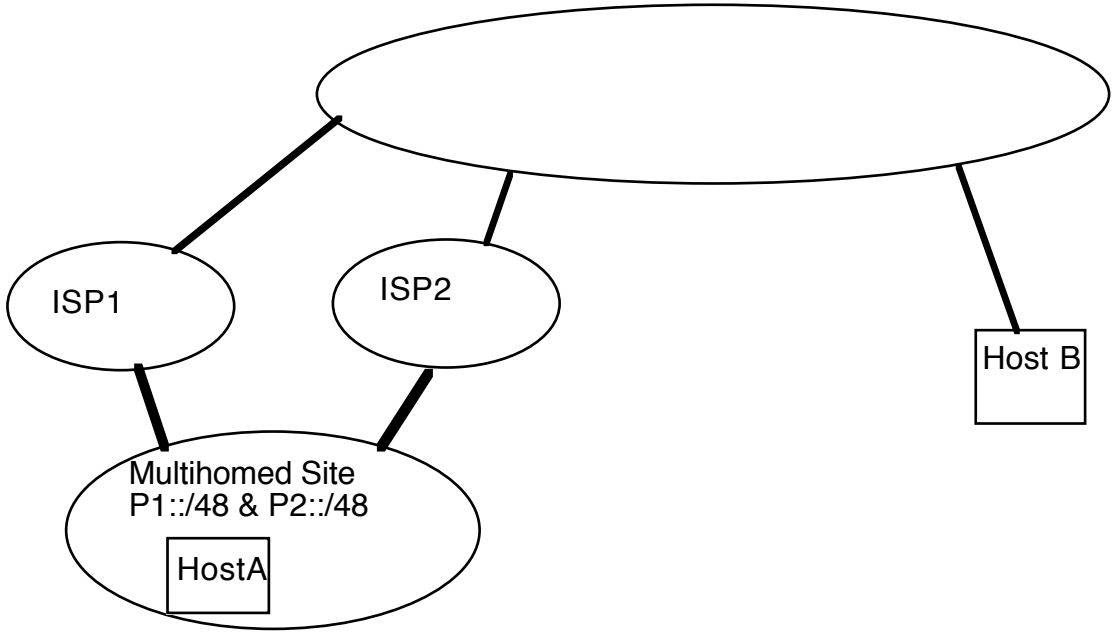
# Main idea of HBAs

- Include the information about multiple prefixes in the addresses themselves
- Available prefixes: P1,P2,...,Pn
- Generate iid as: iid=Hash(P1|P2|...|Pn|rand)
- Resulting HBA is Pref:iid being Pref=Pi for i=1,...,n
- (Privacy is considered later...)

# Application to multihoming

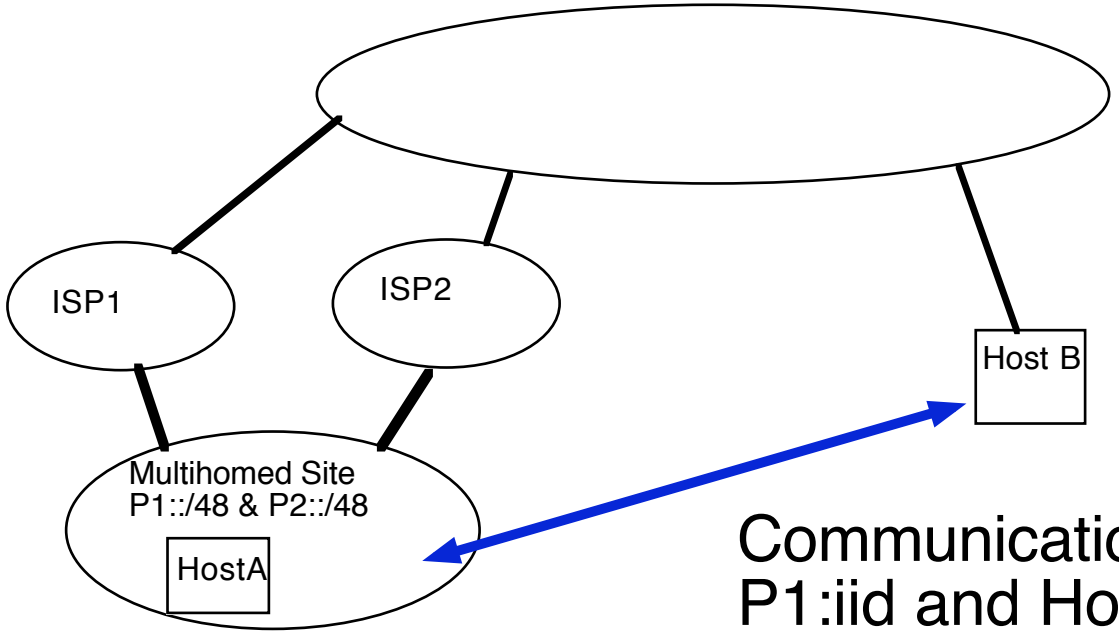
- HostA in multihomed site: P1 & P2
- Generate HBAs for HostA
  - iid=Hash(P1|P2|rand)
  - Addresses for Host1: P1:iid and P2:iid
- Then, HostA communicates with HostB using P1:iid
- HostA informs P1,P2 & rand to HostB
- An outage forces to use P2
- HostB can verify that P2:iid is valid for HostA using HBA

# Application to Multihoming




iid=Hash(P1|P2|rand)
Addresses for HostA
P1:iid
P2:iid

# Application to Multihoming

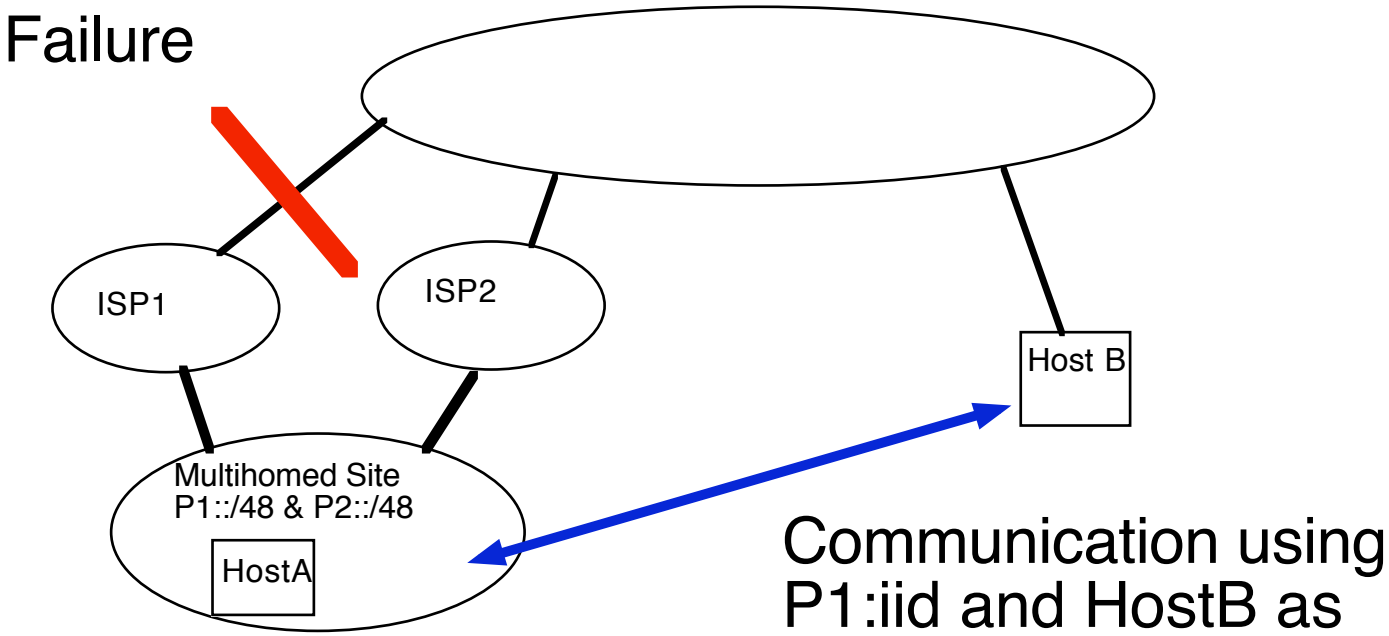



iid=Hash(P1|P2|rand)
Addresses for HostA
P1:iid
P2:iid

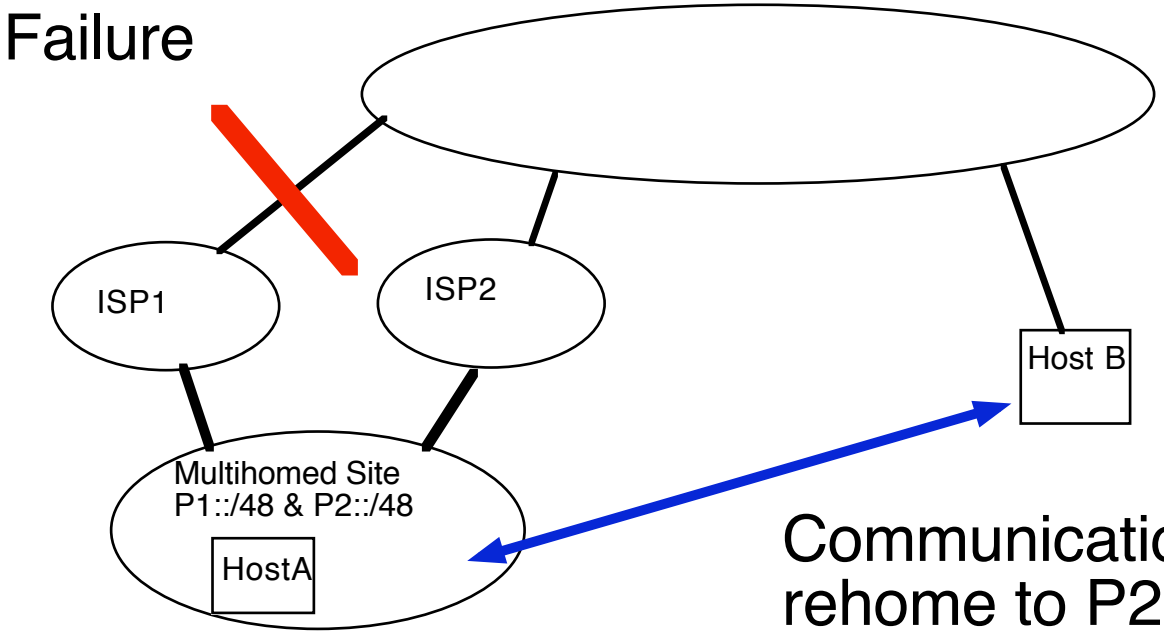
Communication using
P1:iid and HostB as
ULPIDs and locators

(HostA sends rand & set of
prefixes used in HBA)

# Application to Multihoming



# Application to Multihoming




iid=Hash(P1|P2|rand)
Addresses for HostA
P1:iid
P2:iid

Communication must rehome to P2:iid
HostB verifies that P2 is included in the prefixes used to generate iid

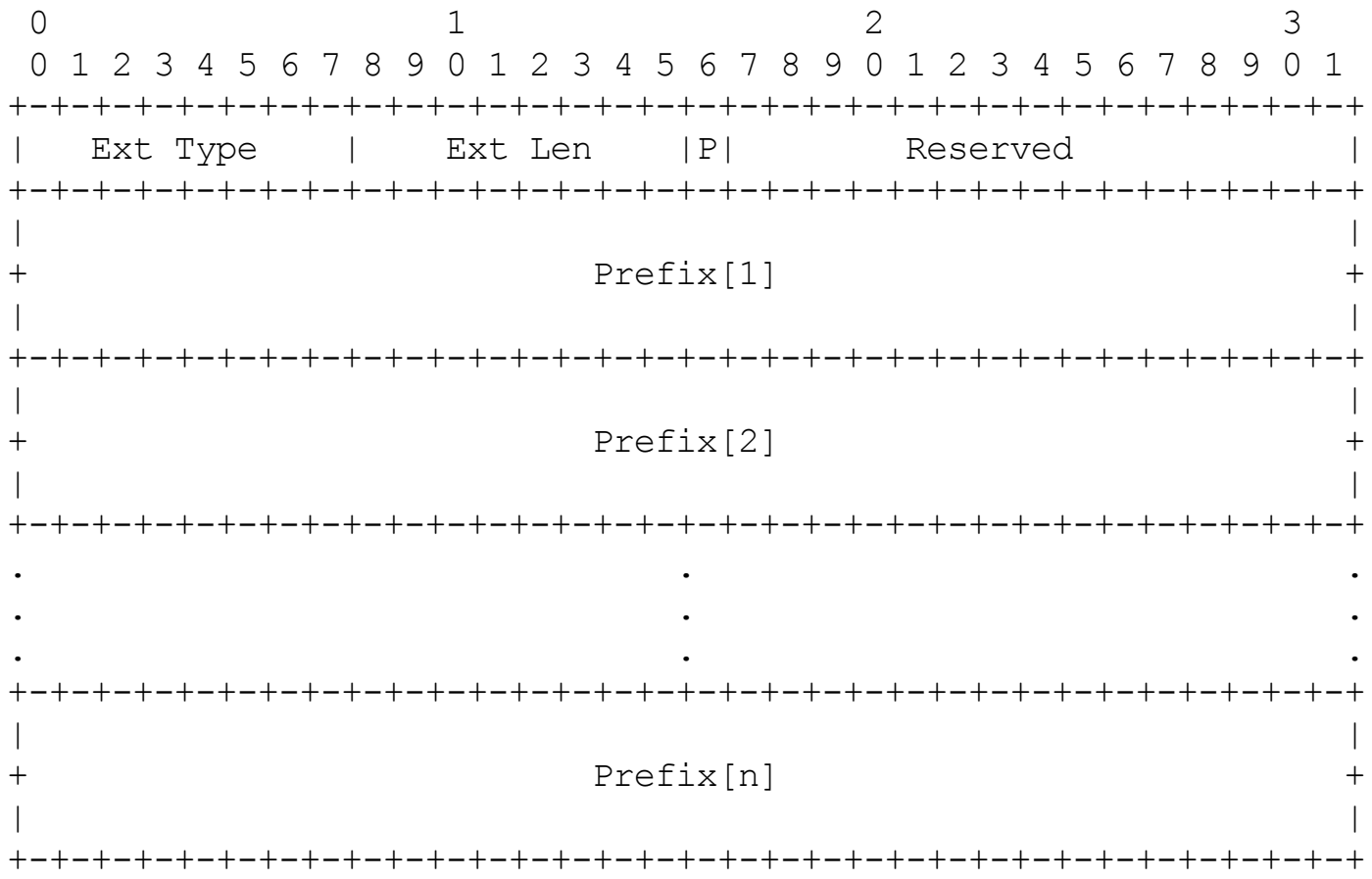
# Compatibility with CGAs

- HBAs and CGAs use the iid bits
- Reasons for compatibility
  - SeND uses CGAs, so compatibility is required to use SeND and HBA-based shim6 simultaneously
  - A CGA/HBA hybrid supports dynamic prefix set, which is useful in some scenarios e.g. mobility, renumbering
- Result: define HBA as a CGA extension

# Resulting address types

- CGA-only addresses:
  - the hash contains public key
  - Support dynamic set address (PK)
- CGA/HBA addresses:
  - the hash contains public key and prefix set
  - Supports a fixed set of prefixes (hash) and dynamic set of addresses (PK)
- HBA-only addresses
  - the hash contains prefix set only
  - Supports a fixed set of prefixes (Hash)

# CGA Multi-Prefix Extension



# HBA-set generation

- Inputs
  - A vector of n 64-bit prefixes
  - A Sec parameter,
  - A public key (in the case of CGA/HBA)
- Outputs
  - An HBA-set
  - their respective CGA Parameters Data Structures

# HBA-set generation

1. Multiprefix extension generation
2. Random Modifier generation
    1. if HBA only, then Ext mod instead of pk
3. Hash2 generation
    1. Modifier|0s|pk*|multpref
4. Verify16*Sec lsbits(Hash2) = 0
5. HBA-set generation. For i=1 to n do:
    1. Generate Hash1[i] (Modifier|Pi|cc|pk*|mp)
    2. Generate HBA[i] (Pi:64lsb Hash1 (u,g,sec))
    3. Generate CGA PDS[i]

# HBA-set verification

- Inputs
  - An HBA
  - A CGA Parameter Data Structure
- Verification process
  - Verify that the 64-bit HBA prefix is included in the Multi-Prefix Extension.
  - Run the CGA verification process (including the Multi-prefix extension in the verification process)

# Security considerations

- Basic attack to HBA:
  - Given an HBA set P1:H1, P2:H2
  - Generate CGA_PDS such as
    - P1 and PX is included
    - the resulting iid==H1
  - so that, the attacker can redirect communications established with P1:H1 to PX:HX
  - The difficulty is to find a suitable modifier i.e. brute force attack
  - The difficulty is O(2^59)
  - Sec>0 then is O(2^(59+16*Sec))

# Privacy considerations

- CGA_PDS[i] contains Pi in the subnet field of basic CGA PDS
- Hence the iid of the HBAs of a given HBA set will be different