# Cryptographically Protected Prefixes for Location Privacy in IPv6

Jonathan Trostle[1], Hosei Matsuoka[2], Muhammad Mukarram Bin Tariq[3], James Kempf[3], Toshiro Kawahara[3], and Ravi Jain[3]

[1] DoCoMo Communications Laboratories USA, Inc.
181 Metro Dr. Suite 300. San Jose, CA 95110
jtrostle@world.std.com
[2] Multimedia Laboratories, NTT DoCoMo, Inc.
3-5, Hikari-no-oka, Yokosuka, Kanagawa, 239-8536, JAPAN
matsuoka@spg.yrp.nttdocomo.co.jp
[3] DoCoMo Communications Laboratories USA, Inc.
181 Metro Dr. Suite 300. San Jose, CA 95110
{tariq,kempf,kawahara,jain}@docomolabs-usa.com

**Abstract.** There is a growing concern with preventing unauthorized agents from discovering the geographical location of Internet users, a kind of security called location privacy. The typical deployments of IPv6 in mobile networks allow a correspondent host and any passive eavesdroppers to infer the user's rough geographical location from the IPv6 address. We present a scheme called *Cryptographically Protected Prefixes (CPP)*, to address this problem at the level of IPv6 addressing and forwarding. CPP randomizes the address space of a defined topological region (privacy domain), thereby making it infeasible to infer location information from an IP address. We present an adversary model and show that CPP is secure within the model. We have implemented CPP as a pre-processing step within the forwarding algorithm in the FreeBSD 4.8 kernel. Our performance testing indicates that CPP pre-processing results in a 40–50 percent overhead for packet forwarding in privacy domain routers. The additional end to end per packet delay is roughly 20 to 60 microseconds. Finally, we present an example showing how CPP can be combined with an encrypting border router to give enhanced protection against compromised routers, maintain limited state on routers, and provide intradomain location privacy in an incremental deployment.

## 1 Introduction

IPv6 addressing, as it is typically deployed, can reveal information about the geographical location of hosts because there is a correlation between the topological location and geographical location of an address. Concern has been increasing about ways to prevent unauthorized agents from using this information to determine the geographical location of fixed users, or to track the geographical location of mobile users as they move [AK,WM]. Protection against such activity is called location privacy. There are regulations in some countries which

mandate that network operators protect their user's location privacy. The need for this protection is likely to grow as Voice over IP (VoIP) and other applications become more prevalent.

Just as our postal addresses are hierarchically arranged with country, state, city, zip code, street address, and the addressee's name information, so that postal office can handle the mail efficiently, the IP addresses are also structured in prefix–suffix format for the reasons of routing efficiency. The prefix part is used for identifying the stub-subnet, and the suffix part identifies the owner of the address in the stub subnet. So in a typical IPv6 deployment, the IP addresses of all hosts that share a subnet will have same prefix; the routers in the Internet maintain routing tables and use different forwarding algorithms, to route the IP datagrams to the destination subnets. Just as it is easy for someone who sees our mail envelops to learn we are located, we are vulnerable to location privacy attacks while communicating over the Internet; an eavesdropper can easily obtain the prefix and use it as an index into the mapping table to determines where the owner of an IP address is located.

In this paper, we discuss a particular technique called Cryptographically Protected Prefixes (CPP), which solves the problem of location privacy in IP addresses by encrypting parts of the prefix such that only appropriate routers in the network can decipher the prefix and obtain the topological information, which they can then use to route the packets to correct destinations. This arrangement eliminates any obvious correlation between the prefix component of the address and its topological belonging.

In particular, CPP provides regional location privacy, such as, within the scope of a regional ISP. In context of IPv6, this means that for a plain IP address of the form $\langle P_0, P, M\rangle$[4], where $P_0$ is the ISP identifier, $P$ is the subnet identifier within that ISP and $M$ is the host identifier, CPP blurs the $P$ component of the address, making it infeasible for an attacker to infer the correct subnet of the host [DH2,DH3]. From the postal address analogy, this means that an attacker can perhaps infer the country or state information (depending on the size of ISP domain) but not any finer granularity information. We refer to the region over which CPP provides location privacy as the *Privacy Domain (PD)*. CPP continues to provide location privacy even if some of the routers in the PD are compromised; without this requirement providing location privacy would be much simpler.

The paper is organized as following. In the next section, we review existing work in this area. Section 3 describes the basic CPP scheme. In Section 4, we discuss security considerations. In Section 5, we present the CPP adversary model, and prove that CPP is secure within the model. We also obtain a probability bound that limits an adversary's ability to obtain location information about a CPP address. Our extension for intradomain PD location privacy is included in this section as well. Section 6 discusses performance results from our implementation. In Section 7, we discuss CPP with respect to network integration and

[4] Through the rest of this paper, we will use the notation $\langle x, y, z\rangle$ to refer to concatenation of bitstrings $x$, $y$, and $z$.

also compare it with some of the other approaches. In Section 8, we draw some conclusions. Appendix A describes CPP forwarding in more detail. In appendix B, we present an extension to CPP for enhanced router security (ERS). This extension gives additional protection against router compromises. We present an example integrating this approach into a Mobile IPv6 framework.

## 2 Related Work

Prior work on location privacy can be categorized into network (IP) layer and application layer solutions. IETF protocols, Mobile IPv6 (MIP) and [JP] and Hierarchical Mobile IPv6 (HMIP) [SC] rely on IP layer agents in the network that relay packets to and from the mobile hosts and provide translation between the globally visible IP address(e.g., Home Address or HoA) of the host and its actual IP address (e.g., care-of-address or CoA), thereby hiding the IP address of the host from its correspondent nodes (CN), and providing location privacy. These schemes, albeit simple, have a number of security and architectural issues. Architecturally, such relays induce traffic concentration points, routing overhead (due to triangular routing), single points of failure that may attract DoS attacks, if they don't crumble under their own weight. From security standpoint, eavesdropping on the links between the network agents and the host, or on the links on either side of the agent can easily reveal the mapping between the regional and global address.

Crowds [RR] is an example of an application specific (web browsing) anonymity service. Other application specific approaches include [Ch1,BF,DD,GT]. Some of these introduce additional network latency making them unsuitable for real time applications. Onion routing (OR) [SG] is an example of a generic privacy approach that secondarily provides location privacy. Anonymity is provided by an overlay network of onion routers that proxy TCP traffic through a series of TCP connections.Although onion routing provides a good privacy/performance tradeoff for many real time applications, the overhead (especially when recovering from a failed onion router) may be problematic for some real-time applications (e.g., Voice over IP). Onion routing was originally designed for communication between two secure sites, and it is optimized for TCP based applications. Onion routing does not protect against host based attacks for example where a malicious software running at the victims machine may obtain the local address and reveal it to attackers through out-of-band means; spyware are classic example of such software. Moreover, since OR is a system of proxies, it cannot coexist with end to end IPsec [KA].

Similar in spirit to OR is Freedom Network [Go1,EH], where a set of Anonymous Internet Proxies (AIP) create a random routes in the private overlay network and associates them with the client's real identity or a pseudo-identity. Again being an application layer solution, Freedom raises some of the same issues as Onion Routing.

Tarzan [FM] is a general purpose anonymity system that operates at the IP layer. In that sense, it closer to our approach than some of the higher layer

techniques. Tarzan uses layers of encryption through a relay network, similar to Onion Routing. Although Tarzan is more application transparent than Onion Routing or Freedom, it is still problematic in some aspects. Since the network address translator (NAT) must perform a source IP address NAT operation, end to end encryption/integrity protection protocols such as TLS/SSL cannot be used (e.g. when application data is encrypted, the source IP address cannot be rewritten in a packet payload, as is required by some protocols such as FTP). Also, the NAT operations are problematic in the sense that they are application specific; thus a NAT module is required for each application that includes source addresses in packet payloads, this is very common for VoIP and other conversational and streaming media applications. As with Onion Routing and Freedom, Tarzan does not protect against host based attacks aimed at obtaining a user's location from its IP address.

## 3 The CPP Scheme

The goal of CPP is to make location privacy a part of basic communication fabric on a public access network, such as that of fixed access or mobile access Internet service providers. We require application transparent location privacy for a full range of IPv6 applications; this includes the applications that may wish to use end-to-end network layer security schemes, such as, IPSec, and/or applications, such as, Session Initiation Protocol (SIP), or Realtime Streaming Protocol (RTSP), that need to communicate the source IP address information to the correspondent node, and thus require knowledge of their IP address. These requirements naturally lead to the conclusion that we need a scheme, whereby, the location privacy is built into the address that is assigned to the host; CPP does precisely that. A brief overview of routing in IP networks is helpful in understanding CPP, this is done in next subsection. In later section we explain how CPP works.

### 3.1 Routing in an IP Network

As explain in the introduction, each host on the IP network is identified by an IP address, structured in prefix-suffix format, where the prefix part identifies the network, and suffix part identifies the host within that network. Internet routing protocols, Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), Routing Information Protocol (RIP), and Border Gateway Protocol (BGP), differ in how routers exchange and maintain routing protocols, but the essence in each case is allow a router to declare which hosts are reachable through it. Since all the hosts under a router share same prefix, the prefix is all that a router needs to advertise to other routers. Each router sends this information to its neighbors, and that neighbors send to their neighbors and so on, so that the routers in the network build a reachability map of the network. For administrative and scalability reasons the routers of an administrative domain do not exchange routing information directly with routers of

other domains, instead only the border routers of different domains talk to each other.

Two aspects in routing information exchange and routing process itself, that are of special interest here are *route aggregation*, and *longest prefix matching*. Route aggregation refers to a situation where a router receives routing information from multiple routers (in form of prefixes), and decides that instead of further advertising the prefixes that it received, it can create and advertise a smaller prefix with out loss of information. For example, if a router receives two 8-bit prefixes, 11010000/4 and 11000000/4 from two routers, it means that it can reach all the hosts, first 4 bits whose address are 1101 or 1100, but it also means that it can reach all the hosts, first three bits of whose prefix are 110, so it can just advertise 11000000/3. Routing aggregation provides smaller and fewer prefixes, resulting in smaller routing tables, which are crucial for routing scalability.

Routing aggregation also creates a problem; in any topology that is not strictly symmetric and hierarchical, or where care is not taken in assigning prefixes, we can end up with prefixes of unequal length in the routing table. Worse yet, we can have prefixes that are prefixes of other prefixes (for details, see [Hu95]). This brings us to second aspect of interest, longest prefix matching. Depending on underlying routing protocol and traffic engineering and policy matters, all routers typically use some variant of longest prefix matching algorithm, which means that while finding matching prefixes in the routing table, the router looks for the entry with the longest match (left to right). This eliminates routing loops, and also avoids sending packets on more aggregated routes, which could be longer, although routing protocols may rely on explicit distance information for shortest path routing.

### 3.2 Overview of CPP approach

CPP IP addresses are designed to hide the location of nodes in an IP access network. The basic idea is that only routers with the proper keys are able to determine how to forward a packet with a CPP destination address to the next hop. At a high level, CPP protects against attacks that attempt to determine a user's geographical location from their CPP IP address. In particular, we would like to protect against the following attacks:

1. prefix correlation attacks - since CPP users in the same subnet have IP addresses with uncorrelated prefixes, an attacker cannot build a prefix table that determines the location of all users from the known location of a few users.
2. host based attacks - with CPP, any local process, even if trusted, is unable to determine the location of a host from its CPP address
3. network attacks - an eavesdropper in some location other than the CPP host's local link is unable to determine the location of the CPP host, (although additional protection is required against more advanced traffic analysis attacks)

4. compromised routers - a router in the CPP access network is only able to determine enough information to forward the packet to its next hop. A malicious router is unable, by itself, to determine the location of the CPP host (except when the next hop is the access router for the CPP host). CPP routers closer to the Internet will be able to obtain limited information about CPP address location, for a large number of hosts (the hosts attached to routers in subtrees below them). CPP access routers will be able to obtain a lot of information about a smaller number of CPP addresses (the hosts directly attached to them).

CPP exploits the route aggregation and longest prefix matching process used in Internet routing. In a moderately structured network topology, the process of routing a packet to a highly aggregated location, such as the border router, to a host in the access network is like progressive decoding, where router find increasingly longer matching prefixes in their routing tables. Exactly how long a prefix a router will *decode* is a function of its position in network topology. With CPP, we want the routers to be able to *see* just enough portion of the IP address necessary to perform longest prefix match and find the next hop. For this, we divide the prefix part of the address into smaller prefix components, each component corresponding to an aggregation level, or router's position in routing topology. The CPP address is constructed such that at each hop in the routing process, the router in question can correctly decipher the component of the prefix that is necessary for routing. Since longest prefix match works left to right, a router in the middle of the routing graph not only needs to decipher the prefix component corresponding to its own level, but also all the prefix component to the left of its own prefix component. The deciphered prefixes are sent by the routers *above* the router in question using an IPv6 hop-by-hop option. In this way each router has just enough information for routing to next hop. Any entity that does not have all the router keys cannot find the plaintext prefix or the correct topological location of the address. With our address notation $\langle P_0, P, M\rangle$, the above means that for a plaintext IP address of form $\langle P_0, P_1, P_2, \ldots P_k, M\rangle$, $P_i$ is the $i^{th}$ prefix component and $P = \langle P_1, P_2 ... P_k\rangle$, the CPP address is of the form, $\langle P_0, X_1, X_2, \ldots X_k, M\rangle$, where $X_i$ is encrypted form of $P_i$, decipherable only by the routers at aggregation level $i$. The CPP addresses are generated by an *address server* that is assumed to have knowledge of the network topology, and that shares symmetric keys with the routers. Following section describes the address structure in more details.

### 3.3 CPP Address Structure

For constructing an IP address, we group the routers in the PD based on their aggregation level (herein after *level*) with respect to the highest aggregation point in the routing graph (such as the border router). All the routers in (or at) the same level share a secret symmetric key with each other and the CPP address server. This key is used by the CPP forwarding algorithm to decrypt the piece of the IP address needed for routing at the level.

For an address of form $\langle P_0, P, M\rangle$, let $P = P_1, P_2, \ldots, P_k$, where $P = \langle P_1, P_2 ... P_k\rangle$ are prefix components corresponding to $k$ aggregation levels, then the CPP IP address is of the form described in Figure 1. The CPP address replaces the plain text prefix components $\langle P_1, P_2 ... P_k\rangle$ with encrypted components $\langle X_1, X_2 ... X_k\rangle$, such that:

$X_j = P_j \oplus H(L_j, M), 1 \leq j \leq k.$

where $H$ is an encryption or hash function, with key $L_j$ and input $M$.

$X_j$ is truncated to same number of bits as $P_j$

The keys $L_1, \ldots, L_k$ are symmetric keys such that each key $L_j$ is shared by the routers at the same level $j$. A router at level $d$ in the routing graph can obtain the prefix component $P_d$ by computing

$P_d = H(L_d, M) \oplus X_d$

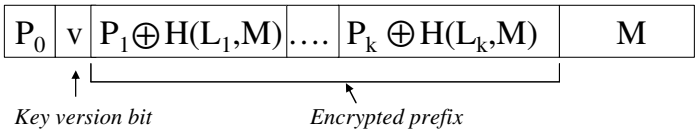


**Fig. 1.** CPP IPv6 Address Showing Prefix Encryption

As the packet is routed from a border router through the routing graph, a router at level $d \leq k$ obtains the prefix components $\langle P_1, \ldots, P_{d-1}\rangle$ from the previous hop router, deciphers $P_d$, concatenates $P_d$ with $\langle P_1, \ldots, P_{d-1}\rangle$ to obtain $\langle P_1, \ldots, P_d\rangle$, uses it for longest prefix match in the routing table, and communicates the newly formed $\langle P_1, \ldots, P_d\rangle$ to the next hop router (presumeably at level $(d+1)$), in an IPv6 hop-by-hop option appended to the forwarded packet. The router at level $(d+1)$ extracts plain text prefix components from the hop-by-hop option, deciphers $P_{d+1}$, looks up the routing table, and forwards the packet to next aggregation level in same way. The process continue, until all the prefixes are exhausted and packet reaches a router that is directly connected to the host that owns the CPP IP address $\langle P_0, X_1, X_2, \ldots, X_k, M\rangle$. We note that each packet is completely encrypted as it traverses the link (tunnel encrypted).

A CPP address uses one bit, $v$, to toggle the key version, for smooth updating of keys. The router uses this bit to determine the key version to be used for decrypting the subnet identifier. Rekeying will be described later in the paper. If an ISP wishes to support both CPP and non-CPP address within the same address space, an additional bit may is used to distinguish between CPP addresses and non-CPP addresses.

An outgoing packet is forwarded up the default route until it hits a border router. The border router forwards locally destined packets using the algorithm described above.

### 3.4 CPP Forwarding Examples

The following example illustrates the CPP forwarding algorithm. Figure 2 depicts the example CPP access network (but not all links are shown). $R_1$ is a border router. A link is denoted by $R_i - R_j$ where $R_i$ is the upstream router and $R_j$ is the downstream router (so the link labeled with the prefix $P_1$ is denoted by $R_1 - R_2$. The user host, host H, is attached to the access router $R_5$. The optimal path for packets from a border router to host H is $R_1 - R_2, R_2 - R_3, R_3 - R_5$.

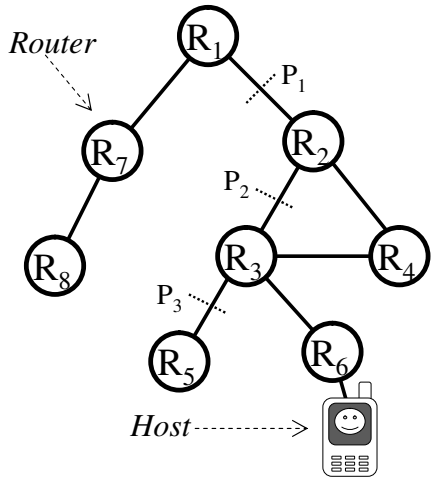


**Fig. 2.** CPP Forwarding Example

There are also a set of level keys: $R_1$ has level key $L_1$, $R_7$ and $R_2$ share level key $L_2, R_8, R_3$, and $R_4$ share the level key $L_3$, and $R_5$ and $R_6$ share the level key $L_4$. The CPP IP address for host H is $\langle P_0, X_1, X_2, X_3, M \rangle$.

The forwarding algorithm for an inbound packet in the case where link $R_2 - R_3$ is down is now described: Since $R_1$ is a border router, it decrypts the first prefix component, $P_1$ using key $L_1$: $P_1 = H(L_1, M) \oplus X_1$

$R_1$ then computes the next hop for the packet by using $P_0 : P_1$ as input to the longest prefix matching forwarding algorithm. The packet is forwarded to $R_2$, and it includes a hop-by-hop option with the prefix $P_1$.

$R_2$ uses key $L_2$ to obtain $P_2 = H(L_2, M) \oplus X_2$. $R_2$ also obtains $P_1$ from the hop-by-hop option. $R_2$ now uses the longest prefix matching algorithm, with $\langle P_0, P_1, P_2 \rangle$ as input, to forward the packet to $R_4$. $R_2$ includes the prefix $\langle P_1, P_2 \rangle$ in a hop by hop option. $R_4$ receives the packet and uses key $L_3$ to obtain $H(L_3, M)$ and $P_3 = H(L_3, M) \oplus X_3$.

$R_4$ then uses the longest prefix matching algorithm to match $\langle P_0, P_1, P_2, P_3 \rangle$ and forwards the packet to $R3$. The prefix $\langle P_1, P_2, P_3 \rangle$ is included in a hop-by-hop option. $R_3$ then uses the longest prefix matching algorithm on $\langle P_0, P_1, P_2, P_3 \rangle$ to forward the packet to $R_5$. The hop-by-hop option also includes the next level which must decrypt the prefix; in this last case, $R_3$ will not decrypt any parts of the prefix since the prefix is completely decrypted. The following pseudocode summarizes the CPP forwarding algorithm:

```
if (I am a border router)
  use level key L to obtain initial prefix component.
  forward packet using this prefix in longest prefix
    match algorithm;
  include prefix in hop-by-hop option
else
  if (no hop-by-hop option is present)
    forward packet up the default route;
  else
    concatenate hop-by-hop prefix with prefix obtained by
      decrypting with key L and input into longest prefix
      match forwarding algorithm,
    include concatenated prefix in hop-by-hop option.
    forward packet to next hop.
  endif
endif
```

The Appendix contains a formal description of the forwarding algorithm.

### 3.5 Rekeying

CPP routers periodically obtain new keys for forwarding CPP addresses. CPP addresses include a key version bit to allow CPP routers to select the correct key. For example, the key period could be 8 hours. During the last half of a key period, CPP routers would obtain keys for the next period. Addresses would expire within 8 hours of being issued. Therefore, a CPP router would never encounter an address that was issued more than one period before the current key period, thus ensuring that a single key version bit is sufficient.

### 3.6 Creating the CPP IP Address

There are a variety of ways that CPP addresses can be mapped onto the 128 bit IPv6 address. The most common address structure expected for general IPv6 addresses [IPv6-Alloc] is that $P_0$ is 48 bits, $P$ is 16 bits, and $M$ is 64 bits, two bits of which are reserved as the 'universal', $u$, and 'group', $g$, bits to designate whether the host identifier is derived from an EUI-64 address [DH2]. However, $P_0$ can be smaller if a particular ISP requires more address space, with 32 bits being a recommended minimum. In addition, the IPv6 addressing architecture

supports addresses with no fixed boundary between P and M. These addresses begin with the special bit pattern 000.

From figure 1, the $X_i's$ require about 16 bits in the address, and $P_0$ requires 48 bits, if IPv6 address blocks are assigned according to [DH3]. The bit $v$ is used for key versioning (see Section 3.5). An additional bit may be required to distinguish CPP addresses from non-CPP addresses (for CPP capable routers) within the domain. So there are 62 bits available for $M$ in the address.

### 3.7 Aggregation and Incremental Deployment

The encryption scheme for the subnet prefix $P$ requires that route advertisements distributed by the ISP's interior gateway protocol be aggregated into distinct levels.

CPP can be deployed incrementally. For example, an overlay network can be used. The CPP routers use IPsec tunneling to tunnel packets to other CPP routers. The non-overlay CPP routers do not have to be CPP capable (one possible exception is the border routers which we discuss below). Therefore, CPP can be used for location privacy, even if the majority of routers do not support CPP. Additionally, the CPP overlay network can be set up to maximize the number of aggregated routes (all CPP overlay routers can aggregate routes). The aggregation can be done automatically, without administrator intervention.

In overlay CPP, border routers must either be CPP routers, or they must be set up to forward CPP traffic to CPP routers and non-CPP traffic to other routers. For example, this forwarding can be accomplished through the use of two distinct global routing prefixes, one for CPP traffic, and the other for non-CPP traffic.

Although CPP routers use tunneling, tunneling is not required between first hop routers and hosts. Many of the other location privacy solutions, such as HMIP also require tunneling, but there tunneling is all the way to the user host which is more problematic due to limited wireless network bandwidths.

## 4 Security Considerations

### 4.1 Eavesdropping

CPP does not necessarily prevent an eavesdropper on the same network link from obtaining the host's location and IP address. If local link encryption combined with some host modifications are employed, then some protection against local eavesdroppers can be provided. If protection against local eavesdroppers is desired, than hosts and routers in the location privacy domain must not use link local addresses, do not perform local link address resolution [NN], and do not perform duplicate address detection on the local link [TN] as most IPv6 hosts typically do. Receipt of a packet with a link local address or any of the messages associated with local link address resolution, which are partially multicast, can provide a hint to an eavesdropper that a particular address is on the local

link, and thus within the geographic area covered by the local link. In order to prevent the router on a link from performing local link address resolution, CPP requires hosts to include their link layer address in unicast Router Solicitation (RS) messages to routers. This message informs the router of the mapping between the hosts' link layer address and CPP IPv6 address. In some cases, routers with co-located Domain Host Control Protocol (DHCP) servers can keep track of the mapping through requests to renew the lease on a DHCP address, and the release of an address by a mobile host that has moved. First hop routers are therefore the only nodes on the link that have any knowledge of a host's presence on the link.

CPP does not by itself protect against eavesdroppers on multiple links. A multiple link eavesdropper can obtain information from each link, and ultimately obtain the location of a CPP address. For example, the attacker can follow the packet from a border router to the access router that the victim host is attached to. CPP uses IPsec tunnels between CPP routers to prevent such an attack. (Alternatively, link encryption can be used to protect all the access network links). However, more sophisticated attackers can still obtain information (e.g. through timing, etc.), especially in a network that is lightly loaded. More powerful countermeasures can also be employed to defeat these attacks (e.g., one of the mechanisms from [SG,FM] at the cost of increased performance impact. For these high privacy environments, CPP should be deployed in combination with other techniques to prevent traffic analysis.

### 4.2 Guessing Attacks and the CPP Extended Address

We now describe a guessing attack against the CPP address. An attacker on a subnet in the PD desires to obtain the location of a victim host, somewhere else in the PD. Suppose the CPP address of the host is $\langle P_0, X_{v1}, X_{v2}, ..., X_{v3}, M_v \rangle$, the suffix of the victim host's CPP address is $M_v$, and $X_{vi}$ being $i^{th}$ encrypted prefix in victims's CPP address. The attacker makes guesses $G_i$ at each of the values $H(L_i, M_v)$ and then XORs these with the prefix components in his own IP address. The attacker then sends a ping packet (or uses some other protocol for which a response is expected), using a guessed destination address of the form: $\langle P_0, G_1, G_2, \ldots G_k, M_v \rangle$

If a response is received, attacker can assume that the guessed destination is active, and the packet was successfully sent to the victim. This is only possible if $G_i = X_{vi}, \forall i = 1 \ldots k$, and hence the attacker has guessed the $H(L_i, M_v)$ values correctly, and can track the victim host at it moves from one subnet to another. Reason behind this attack is that value of encryption function $H$ remains constant for a given host, regardless of its location. The following address structure, the CPP extended address, prevents this situation. The encrypted prefix is $X = X_1, \ldots, X_k$ such that:

$X_j = P_j \oplus H(L_j, \langle M, P_1, \ldots, P_{j-1}, X_{j+1}, \ldots, X_k \rangle)$ where: $j = 1 \ldots k$

A more powerful attacker can launch a similar attack by assuming the suffix $M_v$ of the victim for itself, and then generating guessed CPP addresses and

checking their reachability to itself. Such an attacker would also need the knowledge of plain text prefix for its own location. We will elaborate further on this attack in next section. The CPP basic address should be sufficient for protecting against these attacks in very large access networks, or access network hierarchies, where the prefix size is 32 bits instead of 16 bits.

### 4.3 ICMP

Internet Control Message Protocol (ICMP) messages are often used by routers to signal certain error conditions to a sending host. If the router uses a non-CPP address in the IPv6 header containing an ICMP header, and the ICMP message is in response to a packet that contained a CPP destination address, then an attacker can use this information to help localize the CPP address.

Therefore, a router must always use a one-time random source IP address as the source address in an ICMP error message packet, if the original packet that caused the response contained a CPP destination address. An exception is when IPsec is being used and the peer is authorized to view the real IP address. Per the IETF specifications, nodes do not reply to ICMP error messages.

An instructive example is traceroute. Traceroute is a standard IP tool used for determining the path taken by packets. Traceroute can compromise location privacy unless router sending of ICMP messages is modified. An attacker can use the Time-to-live expiration ICMP messages to build the geographical map of IP addresses or their correlation with a subset of routers.

An exception to the above is to send ICMP error packets with real source addresses when the sender has authenticated using IPsec [KA], and the sender is authorized to request topological information. For example, only a user with administrative privileges should be able to successfully run traceroute and obtain real IP address route information.

## 5 Security of CPP Extended Address

In this section, we present an approach for using CPP extended addresses in a way that prevents guessing attacks. We present an adversary model and prove security against guessing attacks in the model. We also present an extension that provides location privacy across BGP domains. Although correspondents can obtain the global prefix identifier $P_0$ from an address, it is possible to use the same $P_0$ across multiple, geographically distributed BGP domains.

### 5.1 CPP Adversary Model

The adversary can move to different subnets in the CPP access network, register a CPP address of its choosing (but this address must be unique on a given subnet), and receive packets at the address. The adversary is a multi-location entity (may consist of multiple hosts/users in different locations). There is no limit to the number of packets that may be sent and received by the adversary.

There is a restriction on the number of subnets that the adversary can move to in a given key period, (the motivation here is that moving takes some amount of time, and the key period may be only 6-7 hours long).

The adversary can send packets to any destination address. It will receive any packets sent to it by the other adversary components, providing the adversary address is unique on the subnet that it resides. In particular, if the adversary chooses a source address on a subnet where the same source address has already been used, then the adversary cannot receive packets destined to that address. A CPP router neighbor cache will not allow the same CPP address to be reused with multiple different link layer address mappings. Furthermore, the neighbor cache entry will become stale after some period of inactivity, preventing that address from being reused again, even with the same link layer address. Finally, the adversary can move to a new subnet at any time. The following model now formalizes this construction.

**Definition:** The CPP adversary model is the tuple $[G, F, Adv, s, T_1, T_2, \ldots, T_k]$, where $G$ is a CPP routing graph with aggregated prefixes, $F$ is the function described below, $Adv$ is an adversary that can query $F$, $s$ is the number of different locations that $Adv$ can receive packets in, and $T_i$ is the number of prefix components for the $i^{th}$ prefix component. $Adv$ selects an address $A = \langle Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M\rangle$ and a plaintext prefix or known location $Q = \langle Q_1, \ldots, Q_k\rangle$. $F(A, Q)$ returns 1 if the following is satisfied

$G_j = H(L_j, \langle M, Q_1, \ldots, Q_{j-1}, (Q_{j+1} \oplus G_{j+1}), \ldots, (Q_k \oplus G_k)\rangle), \quad \forall j = 1 \ldots k$

$Q_0$ is an empty string. Similarly $Q_i, G_i$ are empty strings if $i > k$. and $A$ has not already been used on the subnet $Q$.

Otherwise $F(A, Q)$ returns no answer. $Adv$ is allowed to select up to $s$ different locations $Q$ in its series of queries. The adversary is adaptive in the sense that the next query is allowed to depend on the results of previous queries.

The CPP adversary model assumes that additional countermeasures prevent traffic analysis attacks on the links of the access network, and that access network routers are not compromised. We will also rule out hash function specific attacks.

We now give a bound for the security of the CPP extended address against guessing attacks. The CPP extended address does not completely prevent guessing attacks; rather, it makes these attacks unlikely to succeed. If $|P|$ is the length of the bit string $P$ then

**Theorem 1.** *Given the CPP adversary model* $[G, F, Adv, s, T_1, T_2, \ldots, T_k]$*, let* $B = \langle P_0, X_1, \ldots, X_k, M\rangle$ *be a CPP address with location* $P_1, \ldots, P_k$*. Let*

$$p_i = 1 - [(2^{|P_i|} - 1)/2^{|P_i|}]2^{-(|P_{i+1}|+\ldots+|P_k|)}$$

*Then the probability that Adv obtains the prefix components* $P_1 \ldots P_j$ *of address B is bounded by*

$q = (1/2)\alpha([x(s-2)p^{s-2} - x(0)]$ *where:*

$p = max\{p_1, p_2, \ldots p_j\}$,

$\alpha = \prod_1^j (1 - p_j)$,

*s is the number of subnets searched, and*
$x(s) = x_2 s^2 + x_1 s + x_0$, *with* $x_2 = 1/(p-1)$, $x_1 = (3-2px_2)/(p-1)$
*and* $x_0 = [2 - px_2 - px_1]/(p-1)$

**Proof:** We can assume that a received packet D has the same host identifier $M$ as B, else the adversary cannot learn any information about B's address (we are ruling out hash function specific attacks as described above, and $M$ is part of the hash computation). Suppose Adv is located on the subnet with plaintext subnet prefix $Q_1, \ldots, Q_k$. Then Adv's address has the form $Q_1 \oplus G_1, \ldots, Q_k \oplus G_k$, for some $Gi$'s. D is forwarded to this subnet, so the following equations (1) are satisfied:

$$
\begin{aligned}
G_1 &= H(L_1, \langle M, (Q_2 \oplus G_2), \ldots, (Q_k \oplus G_k)\rangle) \\
G_2 &= H(L_2, \langle M, Q_1, (Q_3 \oplus G_3), \ldots, (Q_k \oplus G_k)\rangle) \\
G_3 &= H(L_3, \langle M, Q_1, Q_2, (Q_4 \oplus G_4), \ldots, (Q_k \oplus G_k)\rangle) \\
&\ldots \\
G_{k-1} &= H(L_{k-1}, \langle M, Q_1, \ldots, Q_{k-2}, (Q_k \oplus G_k)\rangle) \\
G_k &= H(L_k, \langle M, Q_1, \ldots, Q_{k-1}\rangle)
\end{aligned}
$$

$G_1, \ldots, G_k$ are completely determined by the above equations. The attacker uses the following equations to determine its guesses $G_2, \ldots, G_k$: $Q_2 \oplus G_2 = X_2, \ldots, Q_k \oplus G_k = X_k$; in order to obtain the value $H(L_1, \langle M, X_2, X_3, \ldots, X_k\rangle)$. The attacker can guess $G_1$ such that the first equation above is satisified, but all of the equations must be satisfied in order to receive the reply D. If $P_1 = Q_1$, then $P_2 = Q_2, \ldots, P_k = Q_k$ in order for D to be received. Therefore, either A is on the same subnet as B, or $P_1$ is not equal to $Q_1$ (see the definition of the CPP extended address in Section 4). In the first case, A will not receive a response packet, since we require that A not be able to create an entry in the neighbor cache with its own link layer address and the CPP address B. (The CPP router will also consider a neighbor cache entry to be stale if no packets have utilized the entry in the recent past; therefore, the attacker will not be able to observe packets sent to address B unless it is on the same link as the victim at the same time, or close to the same time. We ignore this case, since there are other ways of detecting the victim's physical location at this point, including physical recognition of the victim.)

In the latter case ($P_1 \neq Q_1$), the above equations are satisfied with probability $1-p_1$. But A learns only $H(L_1, \langle M, X_2, X_3, \ldots, X_k\rangle)$, and $P_1$, from the above equations in this case. The attacker must now repeat the search using $Q_1 = P_1$, in order to learn $H(L_2, \langle M, P_1, X_3, X_4, \ldots, X_k\rangle)$. The attacker can guess $G_1$ and $G_2$; the probability that the remaining equations can be satisfied is bounded by $1 - p_2$. The same argument holds for $1 - p_3$. Therefore the success probability is

$$\alpha \sum_{i_1+i_2+i_3 \leq s-3} p_1^{i_1} p_2^{i_2} p_3^{i_3} \leq$$

$$\alpha \sum_{i_1+i_2+i_3 \leq s-3} p^{i_1+i_2+i_3} =$$

$$\alpha \sum_{i=0}^{s-3} [(i+2)(i+1)/2]p^i$$

where $\alpha = (1-p_1)(1-p_2)(1-p_3)$,
the last quantity is $q = \alpha(1/2)[x(s-2)p^{s-2} - x(0)]$.

For guessing attacks where the attacker moves to multiple subnets, we assume that some time is required to physically traverse between the multiple subnets. For example, if we assume a traversal time of four minutes, then less than 100 subnets can be covered during a key period of 6.67 hours. We compute some probabilities in the next subsection.

### 5.2 CPP Prefix Component Segmentation

Although the above proof shows the difficulty of obtaining the first three components of a CPP address, in some cases it is desirable to prevent an adversary from obtaining any prefix components of an address. The first subnet prefix component $P_1$ can be segmented into multiple single bit prefix components, $W_1, W_2, \ldots, W_m$, where $W_1$ consists of the first bit of $P_1$, $W_2$ consists of the second bit in $P_1$, and so on. The first CPP router (CPP border router) will use $m$ separate encryption keys and decrypt all of $W_1, W_2, \ldots, W_m$ before forwarding the packet. This additional segmentation is independent of, and has no affect on, the forwarding tables and prefix aggregation. We expect $m$ to be between 3 and 7, typically.

With $P_1$ segmentation, the above probability bounds show that it is highly unlikely for an adversary to obtain more than two bits of information about the prefix component $P_1$. We will now show that it is very unlikely for an adversary to obtain more than a single bit of information about $P_1$, when $P_1$ segmentation is used. From the argument in the theorem proof above, the probability that the adversary obtains the first two bits ($W_1$ and $W_2$) is bounded by

$$\alpha \sum_{i_1+i_2 \leq s-2} p_1^{i_1} p_2^{i_2} \leq \alpha \textstyle\sum_{i=0}^{s-2} (i+1)p^i = \alpha[\frac{(1-p^s) - (s)p^{s-1}(1-p)}{(1-p)^2}]$$

where $\alpha = (1-p_1)(1-p_2)$, $p = \max\{p_1, p_2\}$, and $p_1$ and $p_2$ are defined above. In this case, $1 - p_1 = 1/2^{16}$, and $1 - p_2 = 1/2^{15}$.

We obtain a bound less than $1/434187$, when $s = 100$. In other words, the probability that the adversary obtains more than one bit of information about $P_1$ is less than $1/434187$.

To this point we have not considered any hash function specific attacks; more precisely, we are assuming that the hash function outputs are uniform. The uncertainty in the outputs obscures the prefix values in the CPP address. We now relax this assumption. In particular, we consider attacks where the adversary attempts to determine partial information about prefixes. We relabel

$W_1, \ldots, W_m, P_2, \ldots, P_k$ as $P_1, \ldots, P_k$. Instead of trying to obtain $P_1$, the adversary notes the values $G_2$ in (1) when a response packet is returned. These are potential covering values for the prefix component $P_2$. The information obtained by the adversary is the sum over the number of responses received, of the probability that i responses are received times $[\log(n) - E(Y|V)]$ Here $E(Y|V)$ denotes the conditional entropy of the random variable $Y$ given the observed response values $V$ where $Y =$ the hash function output for the particular prefix component under attack.

We have computed bounds for these values for a typical example. Our results indicate very small leakage for the higher prefix components ($P_2$, $P_3$, etc.), with some increase in leakage towards the bottom of the tree. For example, the leakage for $P_2$ is less than 1/1100 bits.

### 5.3 CPP Intradomain Location Privacy Extension

CPP is designed for protecting location privacy within a CPP domain. This section introduces an extension to protect the location over multiple CPP domains.

Our approach is to use the same global identifier $P_0$ across multiple CPP domains, regardless of their geographic location. Each domain has a different AS number and distributes routing information with BGP messages to indicate that packets destined for $P_0$ should be delivered to its domain. The outside domains receive the BGP messages which have the same set of destinations $P_0$ from different ASes. According to the specification of BGP-4 (RFC1771), when an AS receives BGP messages for the same destinations from different ASes, it uses the message with the highest routing metrics. Therefore, the packet destined for $P_0$ would be delivered to the nearest CPP domain. The border gateway of each domain can decrypt $P_1$, which indicates the individual CPP domain, and if the decrypted $P_1$ is not for its own domain, it has to forward the packet to the other border gateway of the domain identified by the decrypted $P_1$. In order to forward packets to the other CPP domains, each CPP domain needs to have another individual global prefix identifier that is routable to the individual CPP domain. Each CPP domain advertises both $P_0$ and its own individual global prefix identifier. The border gateway encapsulates the packet (possibly using an IPsec tunnel) and forwards it to the other border gateway through the addition of tunneling header in which the destination address contains the domain-dedicated prefix of that domain. (With the segmentation of the first prefix component $P_1$ into three segments as described above, the border gateway would decrypt all three pieces before forwarding the packet to the destination CPP domain).

Figure 4 shows an example topology with the CPP intradomain extension. When hosts CH and MN communicate, CH packets are forwarded to domain B since domain B is the closest domain, advertising $P_0$, to CH. CH is not aware of which domain MN actually resides in, amongst the multiple domains that advertise $P_0$.

The intradomain extension that we have presented so far is vulnerable to timing/triangulation attacks. In particular, a correspondent node can determine

how many hops away a user host is. Multiple cooperating malicious correspondent nodes can exchange information and make a reasonable guess about the location of a user host. In order to prevent these attacks, a CPP border gateway can queue packets for flows for an amount of time equal to the round trip time to a remote CPP domain. All of the packets for a particular flow would be queued, prior to forwarding, for an amount of time corresponding to a round trip time. The particular remote CPP domain is selected randomly. In this way, the location of a CPP user, with respect to these attacks, is not visible to correspondent nodes, either within remote CPP domains or within the same CPP domain. The performance impact of this countermeasure should be minimal, since no extra traffic is introduced into the network, and the extra delay time is fairly consistent for a selected flow.

## 6 Implementation and Performance Results

### 6.1 CPP Preprocessor Performance

CPP includes a preprocessing step prior to the conventional forwarding algorithm. We implemented the modifications to the router downward path forwarding algorithm for CPP in the FreeBSD 4.8 kernel to measure the performance of packet forwarding for CPP addresses. The process of computing $Q$, the cleartext prefix, was added prior to the process of forwarding table lookup. To compute the true prefixes, each router obtains the prior prefix components and the bit position from which to begin decryption, from the hop-by-hop option. Fig. 6.1 shows the format of the hop-by-hop option header. The prefix component for the current route aggregation level is obtained by decrypting the current level's component of X. The first 16 bits of the hop by hop option, Type and Length, are

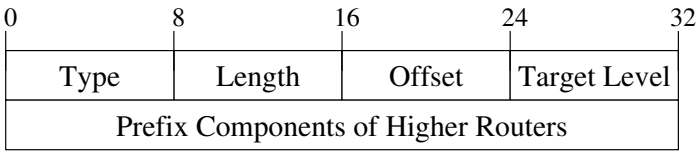


**Fig. 3.** CPP Hop by Hop Option Format

compliant with the IPv6 specification [DH1]. The Offset field (8 bits) identifies the bit position to begin decryption of this level's prefix component. The Prefix Components of Higher Routers field (32 bits) identifies the prefix components to be concatenated with the target prefix component after decryption. The rest of the parameters are preconfigured.

Two different hashing algorithms were tried for the $H()$: SHA-1 and AES (Advanced Encryption Standard) [AES]. The input variables to $Hash()$ are the

router's secret $R's$ key and the suffix $M$, with 64 bits zero-padding satisfying the input size (128bits) in the case of AES since AES is a block cipher algorithm. After the table lookup, the hop-by-hop option field is modified by adding the target prefix component to the prefix components of higher routers field and overwriting the offset field.

The hop-by-hop option of the CPP scheme is processed in the ip6_input() function. The prefix decryption and routing table lookup are also processed in this function. If the packet hasn't reached its final destination, the ip6_input() function forwards the packet to the ip6_forward() function and then the packet goes into the output process. To measure the packet forwarding time, the timestamp at the time of dequeuing the packet from the input queue is attached to the forwarding packet and the timestamp is checked when the packet is enqueued for forwarding. The elapsed time is measured as the packet forwarding time.

We measured the packet forwarding time of unmodified FreeBSD forwarding, and also measured the CPP packet forwarding time for SHA-1 and AES encrypted $Q$. The performance measurements were made on a PC with single Intel Pentium III processor running at 1.0GHz (IBM ThinkPad T23); the FreeBSD 4.8 system contained our implemented kernel. The time in this system can be measured to an accuracy of 1 microsecond. The results are shown in Table 1. With additional work, it is likely that the CPP forwarding times can be improved.

**Table 1.** Forwarding Time Performance for Unmodified and CPP Forwarding

| **Type of Router** | **unmodified** | **SHA-1** | **AES** |
|---|---|---|---|
| **Time to route one packet** | 6 $\mu$sec. | 11 $\mu$sec. | 9 $\mu$sec. |

CPP performance impact will be offset when packets are forwarded based on an entry in the router cache; in this case, no cryptographic processing is required. Also, when MPLS is used, CPP cryptographic processing is typically only required for the first packet with a given CPP address during the address period.

## 7 Discussion

CPP routers perform cryptographic processing on each inbound CPP packet and intra-domain packet, at a cost of one hash operation per packet. There is no performance impact for outgoing packets destined outside the privacy domain (if ERS is not used), since the default routes simply forward outbound packets to the border routers. Given the above performance results, CPP does not appear

to impact performance excessively, especially for non-border routers. Lower level routers are more likely to be limited by the bandwidth of links than by CPU capability, so CPP can be viewed as leveraging unused CPU capacity. Border routers are likely to more heavily loaded. Hardware assist can be of help in reducing the overhead.

CPP configuration is simplest if either all advertised routes are fully aggregated, or overlay CPP is used. In general, route aggregation is considered to be beneficial for good network engineering, because it reduces the size of the routing tables. In CPP, protection against compromised routers is partly facilitated through route aggregation. CPP continues to allow ISPs to institute simple first order security practices. Access routers can still perform ingress filtering if they additionally act as DHCP servers for handing out addresses. If the routers are handing out the addresses, they can check the source addresses on outgoing packets to determine whether they match an assigned address. Instead of filtering on the /64 subnet prefixes, the routers need to filter based on the full /128 address.

ISPs commonly delegate blocks of address space to enterprise customers and allow routing through the ISP to the Internet. This practice is still possible with CPP, except network administrators for delegated address blocks must make sure that the global routing prefix indicates that the address is not a CPP address. The delegated network achieves no location privacy protection, but the addresses can be forwarded using the standard algorithm. Alternatively, if the enterprise network administrator wants location privacy, the ISP can delegate a separate plaintext prefix P0 to the enterprise, allowing the enterprise to set up its own location privacy domain.

The encoding of addresses in CPP causes addresses in the same subnet to be uncorrelated. Network operators often use correlations between addresses to diagnose problems. CPP requires a set of tools that would allow authorized network administrators to access the plaintext address of packets, for such diagnostic purposes. By using IPsec as described above (ICMP discussion), we should be able to use existing tools without modification.

### 7.1 Comparison with Other Approaches

CPP gives up some location privacy when compared with [SG,FreedomNet,FM], since with these approaches, the user's location can be anywhere on the Internet. With CPP, the user is located on some access network associated with the global prefix identifier $P_0$. The latter can still be a large set of geographically distributed locations (especially for a large ISP). Compromise of first hop routers is a problem for CPP, but the ERS extension helps here. Compromise of first hop routers or relays is also problematic for [SG,FreedomNet,FM] in varying degrees (but [FreedomNet,FM] also do encryption on the user host resulting in more protection).

In exchange for giving up some privacy, CPP gains some advantages over the above schemes. CPP network devices maintain no connection specific state; therefore, it is much easier to recover from failures of network elements. Fast

recovery is essential for real time applications such as VoIP. CPP can leverage existing IP based protocols to facilitate recovery.

CPP is well suited for lightweight wireless clients. In [FM], each outgoing (incoming) packet must be encrypted (decrypted) several times (corresponding to the multiple layers). CPP requires either no cryptographic operations on the client or a single layer of encryption for the ERS extension described above. CPP has better resistance to host attacks aimed at disclosing location privacy from an IP address then [SG,FreedomNet,FM], since the CPP address can be used as the host's IP address. Due to the way CPP handles addresses, a larger set of applications can be transparently supported. [SG,FreedomNet,FM] have limits with respect to the number of applications due to encryption and NAT issues.

CPP offers less protection against traffic analysis than [SG], and much less than [FM]. But Tarzan's approach of making user hosts into relays is not currently practical for lightweight wireless clients due to the significant latency of the wireless link. Integration of CPP with the appropriate traffic analysis countermeasures is an area for future research. CPP can be potentially combined with one of these other techniques. If combined with Onion Routing, CPP helps to solve the application layer leakage of information in IP addresses, as well as giving protection against host attacks aimed at location privacy.

## 8 Conclusions

We have presented an approach to location privacy in IPv6 networks, Cryptographically Protected Prefixes (CPP), which directly disrupts the ability of an attacker to map between a mobile host's IPv6 subnet identifier and the geographical location of the access router. CPP does this by encrypting the subnet identifier in the IP address. The encryption is done in a way that only the routers within the routing domain can decrypt. Other approaches to location privacy work by tunneling across parts of the network, including the wireless network, or by masking the IP address. These methods are subject to attacks at various points in the network where the address appears in plaintext, or they result in heavy routing inefficiencies, making them inappropriate for real-time traffic. CPP involves modifications to the basic IP forwarding algorithm, and our implementation results indicate that a modest performance penalty is incurred.

From a security standpoint, CPP achieves a high level of security, when compared to other approaches that facilitate real time traffic, such as Mobile IP and HMIP. The cost is a small performance penalty in forwarding, and the addition of the CPP server. However, as with all cases of security, the main issue is addressing the threat probability. While the current deployment of IPv6 networks is low (and practically non-existent for Mobile IPv6 networks), the situation could change quite rapidly should IPv6 start being deployed for essential services.

## References

[AK] Ackerman, L., Kempf, J., Miki, T.: Wireless Location Privacy: Current State of U.S. Law and Policy. Proceedings of the Workshop on Privacy, Washington DC.

(2003)
[BR] Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. Proceedings of the First Annual Conference on Computer and Communications Security, ACM Press (1993) 62–73
[BF] Berthold O., Federrath H., Kospell S.: Web Mixes: A System for Anonymous and Unobservable Internet Access. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. LCNS vol. **2009** (2001)
[FreedomNet] Boucher, P., Shostack, A., Goldberg, I.: Freedom System 2.0 Architecture.
http://www.freedom.net/products/whitepapers/Freedom_System_Architecture.pdf. (2000)
[Ch1] Chaum D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM **24** (1981)
[DD] Danezis G., Dingledine R., and Mathewson N.: Mixminion: Design of a Type III Anonymous Remailer Protocol. IEEE Symposium on Security and Privacy (2003) 2–15
[DH1] Deering, S., Hinden, R.: Internet Protocol Version 6 (IPv6) Specification. RFC 2460 (1998)
[DH2] Hinden, R., and Deering, S.: Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513 (2003)
[DH3] Hinden, R., Deering, S., and Nordmark, E.: IPv6 Global Unicast Address Format. Internet draft, *work in progress*
[DHCPv6] Droms, R. (editor): Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Internet Draft, *work in progress*
[Dj1] Dijkstra. E.: A Note on Two Problems in Connection with Graphs. Numerische Mathematic, **1** (1969) 269–271
[EH] Escudero, A., Hedenfalk, M., Heselius, P.: Flying Freedom: Location Privacy in Mobile Interworking. Proceedings of INET (2001)
[FM] Freedman, M., Morris R.: Tarzan: A Peer-to-Peer Anonymizing Network Layer. CCS (2002)
[Go1] Goldberg, I.: A Pseudonymous Communications Infrastructure for the Internet. PhD dissertation, University of California, Berkeley (2000)
[GT] Gulcu C., Tsudik G.: Mixing E-mail with Babel. Network and Distributed Systems Security Conference (1996) 2–16
[IPv6-Alloc] IAB, IESG.: IAB/IESG Recommendations on IPv6 Address Allocations to Sites. RFC 3177 (2001)
[JP] Johnson, D., Perkins, C., and Arkko, J.: Mobility Support in IPv6. Internet draft, *work in progress*
[KA] Kent, S., and Atkinson, R.: Security Architecture for the Internet Protocol. RFC 2401 (1998)
[Mo] Moy, J.T.: OSPF: Anatomy of an Internet Routing Protocol. Addison Wesley (1998) 345.
[NN] Narten, T., Nordmark, E., Simpson, W.: Neighbor Discovery for IP Version 6 (IPv6). RFC 2461 (1998)
[AES] National Institute of Standard and Technology.: Specification for the Advanced Encryption Standard (AES). FIPS **197** (2001)
[PK] Perlman, R. and Kaufman C.: Key Exchange in IPsec: Analysis of IKE. IEEE Internet Computing, Vol. **4**, No. **6** (2000).
[RR] Reiter, M. Rubin, A.: Crowds: Anonymity for Web Transactions. ACM Transactions on Information and System Security. Vol. **1**, No. **1** (1998) 66–92

[RoR] Rosen, E., and Rekhter Y.: BGP/MPLS VPNs. RFC 2547 (1999)
[SC] Soliman, H., Castelluccia, C., El-Malki, K., and Bellier, L.: Hierarchical Mobile IPv6 mobility management (HMIPv6). Internet draft, *work in progress*
[SK] Song, R., and Korba, L.: Review of Network-based Approaches for Privacy. $14^{th}$ Annual Canadian Technology Security Symposium. (2002)
[St] Stinson, D.R.: Some Observations on the Theory of Cryptographic Hash Functions. IACR preprint (2001)
[SG] Syverson, P.F., Goldschlag, D.M., and Reed, M.G.: Anonymous Connections and Onion Routing. IEEE Symposium on Security and Privacy. IEEE CS Press. (1997) 44–54
[TR] Tao P., Rudys A., Ladd A., Wallach D.: Wireless LAN Location-Sensing for Security Applications. ACM Workshop on Wireless Security (WiSe 2003)
[TN] Thomson, S., and Narten, T.: IPv6 Stateless Address Autoconfiguration RFC 2462 (1998)
[WM] Warrior, J., McHenry, E., and McGee, K.: They Know Where You Are. IEEE Spectrum, Vol. **50**, No. **7** (2003) 20–25
[YH] Yabusaki, M., Hirata, S., Ihara, T., and Kawakami, H.: $IP^2$ Mobility Management. NTT DoCoMo Technical Journal, Vol. **4**, No. **4** (2003) 16–22
[Hu95] Huitema, C.: Routing in the Internet. Prentice Hall PTR. (2003) 205–208

# A CPP Forwarding Algorithm

This section presents a more formal description of the CPP forwarding algorithm along with a proof of resilience to failures in the network graph.

## A.1 Inbound Forwarding Algorithm

Each R is configured with the level key ($L_i$) for the aggregation level on which it is located. Access routers do not have level keys (and therefore are not level routers). A router having a level key is called a level router. Each level router decrypts the prefix component corresponding to its level and concatenates it with the decrypted component from higher layer routers (if any). Routers use this concatenated prefix to perform longest match and determine next hop. They also pass this concatenated prefix to next hop routers in the hop-by-hop option.

**Network Topology and Level Labeling** We choose one of the border routers as the root of the graph, and exclude the other border routers from the graph, to obtain a new graph $G'$. We use the Shortest Path First (SPF) algorithm to compute a spanning tree $T$ from $G'$, with routes as vertices and links as edges. Let d be the length in hops of the longest path in T from the root to one of the leaves. The root has level d.

We assign levels to T in the following way. We label each of the edges in the tree $T$ with the routing protocol prefixes that will flow along the corresponding link in the network. Given the set of prefixes that flow into a vertex, the vertex can send out the same set of prefixes, plus a prefix that belongs to it. Alternatively, the vertex v may be able to truncate at least one of the outgoing prefixes, thereby performing aggregation. A prefix can be truncated if both of the following conditions apply:

1. The truncated prefix is an initial substring of at least two of the original prefixes in the incoming set,
2. A prefix in an IP address that matches the truncated prefix (some initial substring of the IP address prefix matches the truncated string exactly) does not belong to a router (another vertex) outside the subtree rooted at the vertex $v$. In this case, the two or more matched prefixes in the original set are replaced with the truncated prefix in the set of outgoing advertised prefixes. This truncation is referred to as aggregation. A vertex will be assigned a level key in the tree T if and only if it performs aggregation.

We perform depth first search and label the other vertices of the tree with levels as follows: if a vertex in the tree performs aggregation, and the first aggregating vertex above it on the unique path to the root has level $i$, then it receives level $i-1$. Furthermore, we require that all vertices at level i must perform aggregation and they must aggregate by removing the minimum number of bits that any vertex can remove at that level. This assures that all aggregated prefixes advertised for a particular level have the same number of bits.

Finally, we add the other border routers back into the graph as vertices with level d. Depending on how these other border routers connect to the graph, they may need additional level keys (in addition to the level d keys) and configuration, depending on the relation of the other levels to them. In the best case, they will not require any additional keys or configuration beyond that needed for the original root border router.

### A.2 Handling Failures

When routing links fail, there is the possibility of changes in the ways prefixes are aggregated. The requirement for full aggregation prohibits lower level routers from advertising aggregated routes of the same or a higher level. All aggregated routes must be for routers below the level at which aggregation is being performed. The motivation for this requirement is to allow routes from higher levels to be visible in the forwarding tables of other routers, in case a decryption of a particular level prefix component is needed. The proof follows. For the proof, we impose the requirement that non-level routers do not perform aggregation, even when routing links fail. (In a typical deployment, all the routers would perform aggregation and thus would have level keys).

**Lemma A.2.1:** Suppose a packet arrives in the privacy domain via border router. Suppose also that if the packet passes through a level $i+1$ router S and the next level router R that should handle the packet is down (that is, it is at level i and it is not in the same connected component as S), then R has a sibling router in the same connected component as S, where sibling is defined as another router with level i that is also a descendant of S. Then the packet will be completely decrypted during the forwarding process, since all level keys will be applied and the entire prefix will become available to the routers.

**Proof:** The access router case follows from the border router case. Suppose the packet arrives at a border router. Using induction, suppose prefix components $Q_1, \ldots, Q_i$ have been decrypted. We show that the packet will be forwarded to a level $d-i$ router. Let R$(d-i+1)$ be the level $d-i+1$ router that decrypted Qi. Choose R, in the original spanning tree that is the router which owns prefix Q1, ..., Qi. Then R has level $d-i$.

There are two cases:

Case 1: R is up, and in the same connected component as Ri. If there is a path from $R_i$ to R, where aggregation does not occur, then choose the minimal cost such path, call it A. Then $Q_1, \ldots, Q_i$ is an exact match in the forwarding algorithm for each router along the path. So the packet is forwarded to R as desired.

Otherwise, pick a path B with minimal aggregation and cost, from $R_i$ to R. In the path from Ri to R, let $R_{A1}, \ldots, R_{Ak}$ be the aggregating routers. Then the prefix advertised by $R_{Ai}$ will be successfully matched in the forwarding algorithm for $R_{A(i-1)}$, where $R_{A0} = R_i$. This last statement follows from the fact that we chose a path with minimal aggregation. So the packet gets forwarded to R as desired.

Case 2: R is in a different connected component. By the hypothesis, Ri is in the same connected component with a router $R_A$ that has the same level as R. So RA advertises $Q_1 \ldots Q_{i-1}, Q_r$. Either $Q_1 \ldots Q_{i-1}, Q_r$ is in $R_i$'s forwarding table, or an aggregated prefix $Q_1, \ldots, Q_j$ is visible. In the first case, if no aggregated prefix is visible, then $Q_1, \ldots, Q_{i-1}, Q_r$ is the match. This matching would then continue in the chain of routers between $Ri$ and $R_A$, resulting in the packet being forwarded to $R_A$. $R_A$ then decrypts the subnet identifier as desired.

Once again the best match will be followed until either a decryption occurs (and the proof is complete), or a longer prefix component sequence is encountered. Finally, we must encounter $Q_1, \ldots, Q_{i-1}, Q_{r1}$ for some prefix component $Q_{r1}$. We follow this best match until we arrive at the router that owns this prefix that is a level router by the statement of the forwarding algorithm above. So the next component gets decrypted and the proof is complete.

We now suppose that an aggregated prefix is visible to $Ri$. This best match will be followed until either a decryption occurs (and the proof is complete), or a longer prefix component sequence is encountered: $Q_1 \ldots, Q_k$ where $k > j$. Theorem A.2.1: Suppose a packet arrives in the privacy domain via either a border router or access router, and suppose the conditions of Lemma 4.1 hold. Then the packet will be delivered to the destination access router.

**Proof:** This follows from Lemma A.2.1 and the properties of the existing longest prefix match routing algorithm once the prefix is completely decrypted.

## B Enhanced Router Security Extension (ERS)

We present an extension that uses CPP routers combined with an encrypting border router (BR) that is similar to a HMIPv6 Mobility Anchor Point (MAP). This extension provides the following advantages over HMIPv6:

1. provides additional protection for location privacy in the event of any single router (or host) compromise
2. provides intradomain location privay
3. no storage of per host location state on routers (HMIPv6 MAP's store a mapping between the user's local care of address (LCoA) and regional CoA (RCoA). Protecting against MAP failures in HMIPv6 requires expensive hardware and/or expensive database servers.)

Additionally, only the border routers and the next hop (2nd level) routers are modified. The majority of routers are not modified. The modified 2nd level routers are CPP routers. The basic idea is that a user host (which we denote as MN - the host can be stationary though) will obtain a CPP address A when it enters the network (or is rebooted). The CPP address A is analogous to a HMIPVv6 LCoA. At the same time, the user host obtains a CPP RCoA B which can be derived from the A. The MN and BR will use an encrypted IP tunnel for communication. The BR is unable to determine the location of A, and the inner routers are unable to view the address B (the inner address) which is the address visible to CH (correspondent host). Since the addresses can be derived cryptographically from each other, only relatively static cryptographic keys must stored, instead of highly dynamic address bindings.

Let $A = \langle P_0, X_1, X_2, X_3, M \rangle$, where $|X_1| = 1$, $|X_2| = 1$. Then $B = \langle P_0, Y_1, Y_2, Y_3, Y_4 \rangle$, where the $Y_i$ are defined as follows:

$$
\begin{aligned}
Y_4 &= H(L_4, K, X_1, X_2, X_3) \oplus M \\
Y_3 &= H(L_3, K, X_1, X_2, Y_4) \oplus X_3 \\
Y_2 &= H(L_2, K, X_1, Y_3, Y_4) \oplus X_2 \\
Y_1 &= H(L_1, K, Y_2, Y_3, Y_4) \oplus X_1
\end{aligned}
$$

Here $K$ is a secret value, with the same number of bits as $M$, known only to the BR routers. We assume Mobile IPv6 [DH1]. The CPP address server assigns these addresses. Also, the user host has a home address HoA (which should be relatively static). HoA is used by the transport and application layers of the host. It is location independent. At this point, MN selects one of the BR border routers, BR A, and engages in a cryptographic protocol, such as the IPsec IKE protocol. The result includes dynamic shared session keys between MN and BR A. BR A also stores these keys, in encrypted form, in persistent storage where they will be available to other BR routers. (There are a small number of these BR routers). The BR also stores the MN's home address in persistent storage. The HoA is used as a key identifier, i.e., it is used to identify the correct

session key for decryption of packets sent by the MN. Again, the advantage is that dynamic addresses do not have to be stored on the BR routers. The key identifier can be communicated in packets from MN to a BR in an IPv6 header field (e.g., a hop-by-hop option). Packets sent from MN to the BR have

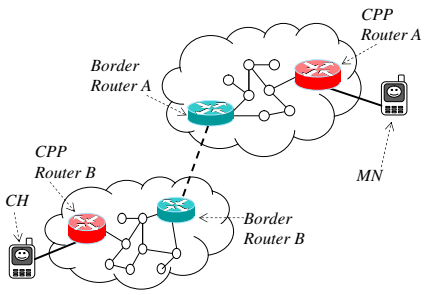


**Fig. 4.** CPP Forwarding with ERS

outer source IP = RCoA, outer destination IP = BR address, inner source IP = RCoA, inner destination IP = CH address. The inner packet is encrypted. The BR will decapsulate and decrypt the packet, before forwarding it to CH. Similarly, inbound packets require that the BR obtain the correct session key for encryption. Packets sent from CH to MN include a type 2 routing header [**?**] with MN's home address (the key identifier). Packets sent from the MN's home agent (HA) to MN can be dealt with in a couple of possible ways:

1. MN gives LCoA to HA instead of RCoA, HA is trusted, so this shouldn't result in much loss of security. Packets without the type 2 routing header are simply forwarded by BR to a CPP router.
2. HA includes the type 2 routing header, or home address option, in its packets as well.

As an example, we consider the sequence of steps when CH sends a packet to MN and vice-versa - see Figure 4. CH sends the packet which is forwarded by Internet routers using the global prefix identifier $P_0$. The packet is forwarded to a BR border router. Using the home address as a key identifier, the BR obtains the session key; it then encrypts and encapsulates the packet. The outer destination address is obtained by the CPP mapping shown above. The outer source address is the BR's address. The packet is then forwarded to any adjacent CPP router. The CPP router A now obtains the IP address of the first hop router for MN (FHR-MN). It then obtains the address of a CPP router B (CPP B) in the same BGP domain as FHR-MN. The common higher level authority ($P_0$) makes it reasonable to assume a trust relationship between these two routers; the packet is encrypted and tunneled to CPP B. This extra level of encryption hides the destination domain from the BR routers. CPP B decrypts and de-encapsulates the packet. It now encrypts and tunnels the packet to FHR-MN.

FHR-MN forwards the packet to MN. MN decrypts and de-encapsulates the packet, obtaining the inner packet with destination address RCoA. The reverse direction is similar.

We have omitted discussion of traffic analysis countermeasures, but these are required for a high level of security. Each link uses encryption so the packet appears differently on every link.

The approach is similar to an enhanced version of HMIPv6; however, it does not provide the reduction in signalling between MN and HA that HMIPv6 provides. This issue can be dealt with using an additional level of hierarchy, or possibly using dynamic home agents. With the former, using expensive hardware is more cost effective since the cost can be leveraged across more users. Alternatively, a more conventional HMIPv6 approach can be used in place of the CPP mapping between the LCoA and RCoA. In this case, we still gain the first two advantages (additional protection against router compromises and intradomain location privacy), but we must now store dynamic IP address information. We also gain the reduction in Mobile IPv6 signalling with this latter approach.