Cryptographically Protected Prefixes for Location Privacy in IPv6

Jonathan Trostle[1], Hosei Matsuoka[2], 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
[2] Multimedia Laboratories, NTT DoCoMo, Inc.
3-5, Hikari-no-oka, Yokosuka, Kanagawa, 239-8536, JAPAN
[3] DoCoMo Communications Laboratories USA, Inc.
181 Metro Dr. Suite 300. San Jose, CA 95110

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. Typical deployments of IPv6 make it possible to deduce the approximate geographical location of a device from its IPv6 address. We present a scheme called *Cryptographically Protected Prefixes (CPP)*[1], 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. CPP can be deployed incrementally. We present an adversary model and show that CPP is secure within the model, assuming the existence of pseudorandom functions. 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. We also give an attack against the address encryption scheme in [Raghavan et al. 2009]. We show that the CPP forwarding algorithm is resilient in the event of network failures.

Keywords: Network layer location privacy, address encryption, IPv6, CPP

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. For example, IP addresses typically narrow the location of the owning user to a city-size area, and in some cases can identify an exact building [19]. More recent work allows an active adversary to locate a user, on average, to within 690 meters [27]. There is increased concern 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 [1]. Location privacy protection technology can be used to prevent such tracking of mobile users. There are regulations in some countries which mandate that network operators protect their users' location privacy [1]. The need for this protection is likely to grow as Voice over IP (VoIP) and other applications become more prevalent.

The location privacy problem is particularly evident with wireless networks. A common deployment pattern for wireless networks is to assign a specific subnet prefix to a collection of wireless cells in a fixed geographical area. For IPv6, this corresponds to having a fixed prefix, typically 64 bits in length, with the form P_0, Q where P_0 is the ISP identifier, and Q is the subnet identifier within that ISP [14, 15]. [2] Any user can determine the location of a host by simply inspecting the IP address.

In this paper, we present a new technique, called Cryptographically Protected Prefixes (CPP), which solves this problem by eliminating any obvious correlation between the topological and geographic location. CPP encrypts parts of the IP address prefix in order to provide location privacy. In particular, CPP encrypts

[1] A preliminary version of this work was presented at the 2004 Privacy Enhancing Technologies Workshop: Springer LNCS 3424, pp. 142-166. This paper is an expanded version of the prelimary paper.

[2] Through the rest of this paper, we will use the notation x, y to refer to the concatenation of bitstrings x and y.

the subnet prefix using a randomized host identifier in a hierarchical manner. The hierarchy is based on segmenting the subnet prefix via routing table aggregated prefixes. Aggregation enables network operators to reduce the size of routing tables, so it is the preferred approach to prefix assignment. Thus the ISP's access network routers are limited to the plaintext address information that they require for packet forwarding (a subset of the full prefix), and no more. Of course, correspondent hosts and routers which are external to the ISP obtain no information about the plaintext address. Our security proof shows security for a multi-location adversary, where the adversary attempts to discern information about the plaintext of a target address based on where packets with chosen CPP addresses are forwarded.

CPP can be easily and incrementally deployed; there is no loss of privacy or interoperability with other Autonomous Systems (AS's) whether or not they deploy CPP. The CPP address is the new IP address of the host.

Given a dynamic IP address of the form P_0, Q, M where P_0 is the global prefix identifier, Q is the subnet identifier, and M is a pseudorandom host identifier. Within an AS, routing is based on the subnet prefix P_0, Q. Network administrators typically lay out the prefixes within the AS so that aggregation is used. For example, if a router has a route to prefixes $s, 0$ and $s, 1$, then using aggregation, it would advertise the prefix s instead of advertising $s, 0$ and $s, 1$. Thus we can segment the subnet identifier as a concatenation of prefix components corresponding to route aggregations:

$$Q = P_1, P_2, \ldots, P_k$$

Let

$$X_j = P_j \oplus h(L_j, M, P_1, P_2, \ldots, P_{j-1}, X_{j+1}, \ldots, X_k), 1 \leq j \leq k.$$

where $h(L_j, ...)$ is a keyed pseudorandom function (prf). (Section 3 discusses motivation for this construction).

The CPP (IPv6) address corresponding to this plaintext address is

$$P_0, X_1, \ldots, X_k, M.$$

A packet with a CPP destination address is forwarded to a border router in the destination AS (via the unencrypted global prefix identifier). The routers within the AS (access network routers) decrypt segments of the subnet prefix which allows them to use the longest prefix matching forwarding algorithm; in this manner the packet is forwarded to the destination host.

The CPP (key) server manages the symmetric keys; it creates the CPP addresses and distributes the addresses to the address servers (e.g., DHCPv6 could be used). These servers would typically be co-located with first hop routers for performance. The CPP address is a dynamic address; each host will obtain a new CPP address for every key period. The CPP key server is a trusted third party (TTP); if an adversary compromises it, then the adversary can obtain the plaintexts of the CPP addresses for the current key period.

CPP is resilient to router failures since the same (symmetric) key is assigned to multiple access routers. Thus if one router or link fails, another route receives a higher preference and the packets will be forwarded via the alternate route. We present an example showing forwarding when a link has failed in Section 3.4. We assign keys across routers of the same depth in the access network routing graph such that routers at different depths have different keys (see Section 3.4 and Figure 3).

1.1 Additional Applications

There are additional applications for CPP. In [12], the authors use the location information from IP addresses to covertly track 100,000 mobile users via dynamic DNS. They construct user profiles that include the user's identity, daily commute pattern, and travel itinerary. CPP can be used to greatly reduce the location information in this scenario. CPP also prevents the leakage of organization information present in the IP address.

Another application is Location Based Services (LBS). Typical solutions involve using an anonymity server or possibly Private Information Retrieval (PIR) to prevent users from being tracked as they download POI's (Points of Interest) to their smartphones. A complete solution also needs to prevent location information that leaks via their IP address as well.

Consider the problem of confidential real time communication between two network entities where the location information for each party should remain private. If existing anonymity solutions [5, 6] are used, then either application level encryption (e.g., TLS/SSL) cannot be used, or many applications will leak the IP address information to the peer (since encryption will prevent address rewriting). Additionally, source addresses must be rewritten in application payloads for many applications. CPP allows a host to have a replacement address which is encrypted.

1.2 Organization

Section 2 reviews IPv6 routing and terminology that we use in this paper. Section 3 overviews the CPP scheme. Our extension for intradomain location privacy is included in this section as well. 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 probability bounds that limit an adversary's ability to obtain location information about a CPP address. Section 6 discusses performance results from our implementation. Section 7 covers related work, including another address encryption scheme [21]. We present an attack against this scheme which is similar to the attack described in Section 3. In Section 8, we summarize. Appendix A gives bounds on the top down attack given the special case where no additional random bits are prepended to the subnet identifier. Appendix B gives an alternate proof, using Gosper's algorithm, of Theorem 3. Appendix C gives additional details for CPP routing graph levels. Appendix D gives examples of aggregated routing graphs and routing tables when prefix components are not the same length at a given level.

2 IPv6 Addresses and Routing

We briefly review IPv6 routing within an access network. IPv6 routing is based on longest prefix matching. For each interface, an IPv6 node (a router or a host) has a routing table which consists of a list of routes. Each route includes the address and prefix length (for matching with the destination address of a packet), next hop (which is the node to forward packets to for this route), preference, and other data. Generally, the longest prefix match is used to determine the correct route for forwarding a packet. In other words, the longest match from the prefixes in the routing table with the destination address in the packet is used to select the correct route. The default route (usually a route with prefix length 0) is used if no other route matches the destination address in the packet. The default route is typically used within an access network in order to forward a packet destined for outside the access network up towards a border router. Dynamic routing protocols (e.g., RIP, OSPF) are used to help populate routing tables and serve to route packets around failure points.

CPP does not impact routing protocols. CPP does require a modified forwarding algorithm which we describe in Section 3.5. CPP leverages route aggregation (or summarization). This technique is used to minimize the size of routing tables, and enables a router to advertise a single route with a shorter prefix in place of multiple routes with longer prefixes. For example, instead of advertising the two prefixes $s, 1$ and $s, 0$ where s is a binary bit sting, a router would aggregate the two and advertise s.

IPv6 addresses have 128 bits; they're divided into a network identifier prefix, P, and a host identifier suffix, M, so the address has the form P, M [14]. A host's first hop router uses M to map between the IPv6 address of a received packet and the link layer address, so it can deliver the packet to a host. P is used to route packets within the ISP's routing domain and on the Internet.

P can be further divided into two parts, P_0, Q, with P_0 being a global routing prefix and Q being a subnet identifier [15]. P_0 is used on the public Internet to identify packets that must be routed to the ISP. Q is a subnet identifier and is used by routers within the ISP's routing domain to determine the first hop router to which a packet is delivered (see Fig. 1). A full IPv6 address can therefore be represented as P_0, Q, M. For example, P_0 could be 32 or 48 bits; the subnet identifier could be 16 or 32 bits, and the host identifier could occupy the remaining bits.

IPv6 Neighbor Discovery [18] allows hosts on a subnet to discover routers on the subnet, the prefixes associated with the subnet, and to map between IP addresses and link layer addresses (the latter using neighbor solicitation messages). Also, redirect messages allow a host to learn about a better first hop router.

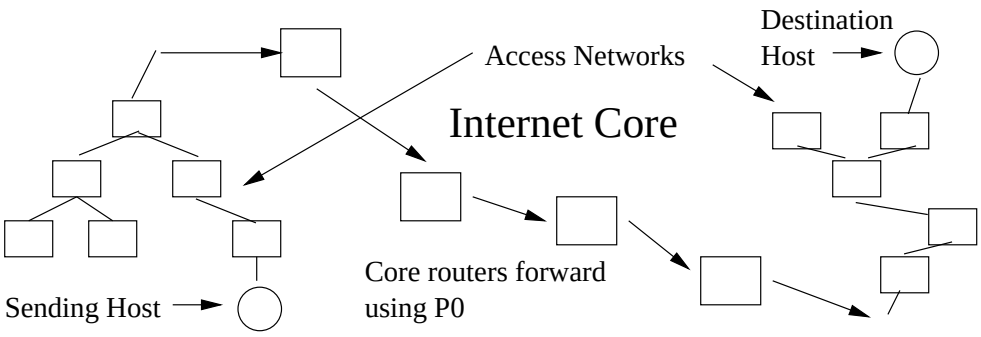


Fig. 1. Router Forwarding of an IPv6 Address

2.1 Terminology

We briefly review terminology used throughout the paper. An access network or site, is attached to a higher level ISP network (or it could be a tier one ISP network). Within the access network are border routers which interface between the internal network and external networks. We refer to the routers within the access network as access network routers. Routers with attached hosts are called first hop or access routers. Typically an incoming packet would first be handled by a border router, then forwarded via a sequence of access network routers to the destination host's first hop router, then to the host itself. Packet forwarding in an access network is based on the subnet prefix and host identifier (with the latter being used at a host's first hop router).

An IP address prefix is an initial substring of an IP address. A routing graph is an undirected graph with routers as nodes and edges denoting the links between the routers. Each node in the routing graph has a routing table which consists of pairs (routes), where each pair contains an IP address prefix and an adjacent node in the routing graph (the next hop). CPP makes some changes to the routing table; we define the CPP routing table in Section 3.4. The directed routing graph is a routing graph with directed edges (representing route advertisements) where the directed edges are labelled with IP address prefixes. Each prefix corresponds to one or more routes that exist in the sending node's route table.

Given the set of advertised prefixes $A(n)$ from a node n. Two or more such prefixes can be replaced (aggregated) with a single shorter prefix P if both (a) P is an initial substring (prefix) of all the replaced prefixes, and (b) every prefix in the graph that is of the form P, x for some bit string x is a member of $A(n)$. (In other words, if a router advertises P, it must have routes to all addresses with prefix P.) The new set of aggregated prefixes at node n is called $Agg(A(n))$, where all aggregations that are possible have been performed. The aggregated routing graph is the directed routing graph with the set $Agg(A(n))$ of prefixes on all edges from n for all nodes n.

3 The CPP Scheme

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. The location of a CPP address is the IPv6 subnet that packets destined for the address will be forwarded to. CPP requires limited modifications to access network routers. Routers outside the CPP network (identified by a global prefix identifier P_0), or privacy domain, are not modified. (But this privacy domain may span multiple BGP domains). Routing protocols are not modified either. CPP uses a new forwarding algorithm (but not all access network routers need be CPP capable).

An IPv6 address is of the form P_0, Q, M where P_0 is the global routing prefix, Q is the subnet identifier, and M is the host identifier. When forwarding packets, each CPP capable router decrypts a piece of the

destination IP address subnet prefix, or prefix component, corresponding to its depth in the routing graph. The router concatenates the prefix components corresponding to the higher routers that were included in a hop by hop option with the prefix component that it decrypts. This concatenated prefix is then used as an input into the longest prefix matching forwarding algorithm. Section 3.4 and Fig. 3 illustrate the CPP forwarding algorithm. Section 3.5 gives pseudocode for the CPP forwarding algorithm. CPP does not hide or obscure the global routing prefix, but we do discuss an extension for intradomain location privacy in Section 3.9.

The segmentation of prefixes into prefix components is accomplished through route aggregation. Route aggregation occurs when a router advertises a single prefix in place of multiple prefixes, where the new prefix is a proper prefix substring of all of the replaced prefixes.

Since each router is only able to decrypt a piece of the prefix, the higher level routers only obtain the information they need to forward the packet. They are unable to determine the subnet where the user host is located (this approach has a security advantage over decrypting the entire prefix at the border router).

A CPP access network requires a CPP key/address server in order to provide symmetric keys to the routers for CPP forwarding. The CPP server also creates and provides CPP addresses to DHCP servers which can then assign addresses to user hosts. These DHCP servers may be co-located with access routers for performance reasons. Section 3.7 will explain how the CPP server creates a CPP address, using cryptographic operations.

3.1 CPP Deployment

An important point is that CPP can be deployed by an Autonomous System (AS) and that the hosts within the AS obtain the privacy benefits of CPP even if no other AS's deploy CPP. Also, there is no loss of interoperability with other AS's whether or not they deploy CPP. The CPP address appears to correspondent hosts as the IP address of the communication peer. Only access routers within the deploying AS need to be modified. When a packet is sent from a correspondent host (Sending Host in Figure 1) the routers within its own access network see that the destination address belongs to a foreign AS and they forward it up the default route towards the core routers. The core routers then forward the packet based on the unencrypted global prefix identifier (P_0). Finally the packet arrives at the destination access network. The routers within this access network use the CPP forwarding algorithm to forward the packet to the destination host.

Routers do not modify the CPP address in order to forward packets which have a CPP destination address; rather, they decrypt part of the address in order to forward the packet. The actual destination address field is not modified however.

Core routers do not need to be modified; CPP is transparent to these routers since they forward packets based on the global prefix identifier P_0 and CPP does not encrypt this field.

CPP can be deployed to only a single router within the deploying AS. In this case, the full privacy benefits of CPP within the AS will not be obtained, but privacy is realized for communications with entities outside the AS.

In the remainder of this section, we will present a threat analysis together with the attacks CPP defends against, the CPP address structure, a forwarding example, CPP forwarding algorithm pseudocode, rekeying, and describe how CPP addresses are created. In the last subsection, we will 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.

3.2 Location Privacy Threats and CPP Defense

Here we discuss location privacy threats, CPP trust assumptions, and list the threats that CPP defends against. Location privacy threats include active and passive network eavesdroppers, correspondent hosts, and compromised hosts and routers on various subnets. In CPP, the key server is trusted. First hop routers are able to discern which CPP addresses are located on their own attached subnets. CPP protects a host's network layer location privacy from its communication peers and it partially protects against the access network routers; such a router is able to map the CPP addresses's location to the subgraph with itself as the

root and no further. We prove security against a multi-location active adversary (the adversary can control hosts on up to s subnets in the access network) in Section 5; our proof assumes that the adversary does not compromise any of the CPP access network routers. The adversary is allowed to receive packets destined for the subnets where the adversary is present and to send packets with arbitrary CPP addresses. Thoughout the paper, we will assume the adversary has no side information for a target CPP address's location based on the global prefix identifier (e.g., using user identity plus the global prefix ID to deduce information about the CPP host's location is outside our model).

We will assume that additional countermeasures (e.g., link encryption) prevent traffic analysis attacks on the interior links of the access network. We also assume any ICMP packets generated by routers in the access network have random source addresses; this restriction prevents the ICMP attack in Section 4.3. More generally, our security analysis assumes that communication peers for the nodes in the CPP access network are selected independently from the CPP nodes subnet location (except we don't restrict adversary controlled nodes). Thus there is no leakage of location information in network protocol traffic patterns. In practice, this assumption may not hold for all protocols (e.g., ICMP), and there are two mitigation approaches for potential leakage in this case: (1) In some cases, a service is trusted not to leak information (e.g., a trusted per subnet service on the CPP access network which does not initiate further communications on the CPP nodes behalf). (2) A CPP node can utilize multiple CPP addresses in which case the untrusted service would see a CPP source address which does not get reused. If the untrusted service initiates further communications, then the service node would also need to utilize multiple CPP addresses. For example, consider a per subnet DNS recursive server deployment, where resolvers utilize the DNS server on the same subnet. The requests from the recursive server to the authoritative server would potentially leak location information for the subsequent request from the CPP node given a compromised host on the same subnet as the CPP node. This leakage is prevented if the recursive DNS server host uses distinct CPP source addresses for each request.

3.3 CPP Address Structure

The set of CPP access network routers is partitioned according to their depth in the aggregated routing graph; these subsets are called levels. (Depth and levels are formally defined in Appendix C, and the definition of aggregation has already been given. Intuitively, the depth of the router is the number of routers between it and the border router, including itself, in the routing graph. The level of the router is d - depth where d is the length of the longest path from the border router to a leaf router in the graph.) All the routers in (or at) the same level share a secret symmetric key with each other and the CPP key server. (Actually, only routers that are close to each other need to share a key at the cost of one extra key version bit, where two routers with the same level are close if there exists a short path between them in the routing graph).

This key is used by the CPP forwarding algorithm to decrypt the piece of the IP address needed for routing at the level.

Suppose the IP address subnet identifier is the concatenation of prefix components corresponding to route aggregations:

$$Q = P_1, P_2, \ldots, P_k$$

The (extended) CPP IP address has the format described in Figure 2. Let

$$X_j = P_j \oplus h(L_j, M, P_1, P_2, \ldots, P_{j-1}, X_{j+1}, \ldots, X_k), 1 \leq j \leq k.$$

where $h(L_j, ...)$ is a keyed pseudorandom function (prf) e.g., using AES and truncating to the length of P_j.[3]

[3] A simpler construction such as $X_j = P_j \oplus h(L_j, M), 1 \leq j \leq k$, is vulnerable to a guessing attack: given an adversary on a subnet $Q_1, \ldots, Q_k$ in the same CPP access network as the victim user. A confederate of the adversary sends packets with destination addresses $P_0, Q_1 \oplus G_1, Q_2 \oplus G_2, \ldots, Q_k \oplus G_k, M_v$ We note that the victim's location is $P_0, P_1, ..., P_k$, and the victim CPP address is $P_0, P_1 \oplus h(L_1, M_v), \ldots, P_k \oplus h(L_k, M_v), M_v$. If a response is received by the attacker, then the $h(L_i, M_v)$ values have been guessed. Since the subnet identifier is often only 16 bits long, the expected number of queries for the attack is 32000 - which is not a problem for an attacker with even modest resources.

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. Suppose the current router is at level $d + 1 - j$ in the routing graph. It obtains the prefix component P_j by computing

$$P_j = h(L_j, M, P_1, \ldots, P_{j-1}, X_{j+1}, \ldots, X_k) \oplus X_j.$$

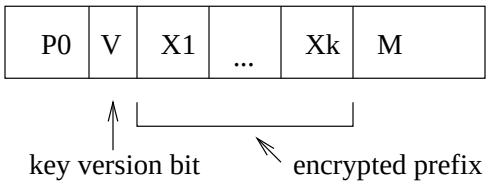


Fig. 2. CPP IPv6 Address Showing Prefix Encryption

The current router obtains the prefix components $P_1, \ldots, P_{j-1}$ from the previous hop router; $P_1 \ldots, P_{j-1}$ are forwarded in an IPv6 hop by hop option. The current router now forwards the packet using the longest prefix matching algorithm, with the prefix $Q = P_1, \ldots, P_{j-1}$, P_j as the input. It also includes the prefix Q in an IPv6 hop by hop option. The next hop will use this prefix Q concatenated with the prefix P_{j+1}, which it will obtain in the same way that the current router obtains P_j; the next hop uses Q, P_{j+1} as an input into the longest prefix matching algorithm. In this way, the packet is forwarded to the destination first-hop router. Each packet can be completely encrypted as it traverses the link (tunnel encrypted or link encrypted) if additional protection against eavesdroppers is desired.

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. An additional bit 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. CPP is compatible with multiple border routers; these routers share the same symmetric key. A packet arriving at any of the border routers would go through initial prefix component decryption and forwarding just as in the case of a single border router.

3.4 CPP Forwarding Examples

We define the *CPP routing table* as follows: It consists of a list of entries where each entry is either:

1. a triple consisting of an IP address prefix, a next hop node, and the level of that next hop node
2. a pair consisting of an IP address prefix and the number of additional bits to decrypt

For each entry of the second type, additional entries must follow so that the final entries are of the first type. (In other words, the final result should be the next hop to forward the packet to.) The number of bits to initially decrypt is also included. If the prefix components are all the same length at each level, then only the first type of entries will be in the routing table. We refer the reader to Appendix D for an example with multiple prefix lengths.

The following example illustrates the CPP forwarding algorithm. Figure 3 depicts the example CPP access network (but not all links are shown). R1 is a border router. A link is denoted by Ri - Rj where Ri is the upstream router and Rj is the downstream router. The user host, host H, is attached to the first-hop router R5. The optimal path for packets from a border router to host H is R1 - R2, R2 - R3, R3 - R5.

There are also a set of level keys: R1 has level key L_1, R7 and R2 share level key L_2, R8, R3, and R4 share level key L_3, and R5 and R6 share level key L_4. The CPP IP address for host H is:

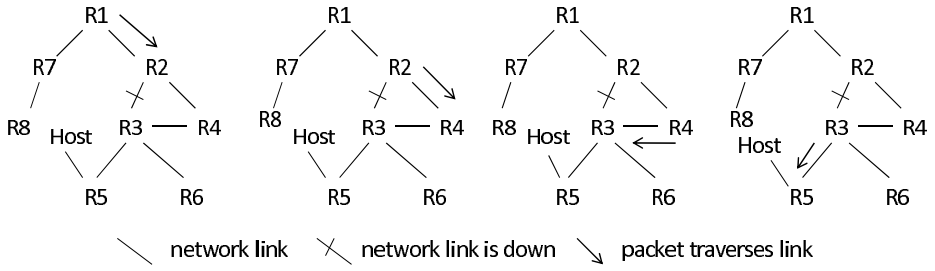


Fig. 3. CPP Forwarding Example Showing Packet Traversal Sequence with Routers Ri

$$P_0, X_1, X_2, X_3, M$$

where $X_j = P_j \oplus h(L_j, M, P_1, \ldots, P_{j-1}, X_{j+1}, \ldots, X_k)$ and P_0, P_1, P_2, P_3 is the prefix for the subnet attached to router R5.

The forwarding algorithm for an inbound packet in the case where link R2 - R3 is down is now described: Since R1 is a border router, it decrypts the first prefix component, P_1 using key L_1:[4]

$$P_1 = X_1 \oplus h(L_1, M, X_2, X_3).$$

R1 then computes the next hop for the packet by inputting P_0, P_1 into the longest prefix matching forwarding algorithm. The packet is forwarded to R2, and it includes a hop by hop option with the prefix P_1 (see the leftmost graph in Fig. 3).

R2 uses key L_2 to obtain $P_2 = X_2 \oplus h(L_2, M, P_1, X_3)$. R2 also obtains P_1 from the hop by hop option. R2 now uses the longest prefix matching algorithm, with P_0, P_1, P_2 as input, to forward the packet to R4 (see the 2nd graph from the left in Fig. 3). R2 includes the prefix P_1, P_2 in a hop by hop option. R4 receives the packet and uses key L_3 to obtain

$$P_3 = X_3 \oplus h(L_3, M, P_1, P_2).$$

R4 then uses the longest prefix matching algorithm to match P_0, P_1, P_2, P_3 and forwards the packet to R3 (see the 2nd graph from the right in Fig. 3). The prefix P_1, P_2, P_3 is included in a hop by hop option. R3

[4] Although our example assumes all of the prefix components at a given level have the same length, and our security analysis is simplified but doesn't require this assumption, CPP doesn't require such a network layout (but being under the control of a single ISP would facilitate it).

If the prefix components from route aggregations on a given level aren't all of the same length, then we can segment into smaller equal length components corresponding to the minimal length from the aggregations, the next longest length, etc. This scheme facilitates giving minimal information to the routers, but requires more decryptions for some prefixes. A router may have to perform multiple decryptions prior to forwarding the packet. The lengths of the components are included in the routing table, so there is never any ambiguity regarding the number of bits to decrypt. Appendix D (Fig. 12) gives a small routing graph example with the routing table. Security is optimized when the route aggregations produce same length components at every level.

Fig. 13 gives another example. Suppose the link between routers R1 and R2 goes down. Then the 1st entry drops out of R1's routing table. Suppose R1 decrypts the 1st bit of an inbound packet prefix, and the decrypted bit is 0. Although there is a partial match with $P_0 00$ in the routing table, the next level for this entry is too low. Thus the best match is the R5 entry. The packet is forwarded to R5 and $P_0 00$ or $P_0 01$ is the resulting decrypted prefix. R5 forwards the packet back to R1 which forwards the packet to R4 without further decryption. If the prefix is $P_0 01$, the packet is ultimately forwarded to R3.

then uses the longest prefix matching algorithm on P_0, P_1, P_2, P_3 to forward the packet to R5. The hop by hop option also includes the next level which must decrypt the prefix; in this last case, R3 will not decrypt any parts of the prefix since the prefix is completely decrypted.

3.5 CPP Forwarding Algorithm Pseudocode

The Figure 4 pseudocode summarizes the CPP forwarding algorithm. The key difference with existing longest prefix match forwarding is that the next hop level must be greater than or equal to the level of the key used to decrypt the next segment of the CPP address. This requirement ensures that the CPP address will be decrypted and the packet forwarded to the correct first hop router. The greater than allows the packet to be forwarded to the correct part of the graph prior to decryption of the next prefix component (and ensures resiliency given network failures).

```
if we are a border router AND packet arrives on external network interface then
  decrypt initial prefix component; forward packet based on longest prefix match algorithm where next hop level
  ≥ d-1; include prefix and level=d in hop by hop option;
else
  if no hop by hop option is present then
    forward packet up default route;
  else
    h = hop by hop prefix; l = hop by hop level; if(l = 1) forward packet using longest prefix match.
    if our level equals l-1 then
      pc = decryption of next prefix component; prefix = h,pc; hop by hop level = l-1;
    else
      prefix = h;
    end if
    j = our level - 1; input prefix into longest prefix match forwarding algorithm to get next hop; if next hop level
    ≥ j then goto SEND else use next longest match on prefix and check if next hop level ≥ j, if so goto SEND;
    repeat until empty prefix is used (empty prefix match allows any next hop with level ≥ j to match). If no
    match, discard packet.
    SEND: include prefix and hop by hop level in hop by hop option; forward packet;
  end if
end if
```

Fig. 4. CPP Forwarding Algorithm Pseudocode

3.6 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 key period (e.g., 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 one version bit for the key is sufficient. If long-lived network failures are not highly unlikely, an extra bit can be allocated for key versioning.

3.7 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 [16] is that P_0 is 48 bits, Q is 16 bits, and M is 64 bits.

From Figure 2, 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 [15]. The bit v is used for key versioning (see Section 3.6). 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. Additional bits will be available for the subnet identifier, if P_0 has a shorter length.

CPP addresses can be constructed by a key/address server which provides CPP keys to the routers as well. The key/address server can create CPP addresses and provide them to DHCP servers co-located with first-hop routers. The DHCP server would obtain a CPP address for a host by contacting the key server; the key server sends a CPP address created from a pseudorandom M value and prefix corresponding to the requesting subnet. Since CPP encryption is probabilistic, host addresses change during every key period.

3.8 Aggregation and Incremental Deployment

The encryption scheme for the subnet identifier Q requires that route advertisements distributed by the ISP's interior gateway protocol be aggregated into distinct levels. A route is fully aggregated when it summarizes all the routers under it using a reduced size prefix. For example, consider the privacy domain shown in Figure 3. Here, the Ri are routers. A fully aggregated routing table means that no routes for R3 through R6 and R8 occur in R1's routing table, since these routes are aggregated by R2 and R7. All routers in the domain contain a default route to R1, since it is the border gateway router to the Internet.

CPP can be deployed incrementally (not all routers within the AS need be CPP capable). 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. Another option is to deploy CPP to only the border routers in a NAT-like manner. In this case, location privacy is obtained from the communication peers external to the ISP.

3.9 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.[5]

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 highest preference CPP domain (which could be the one with the shortest AS path). 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

[5] An adversary may have side information about a user's location which can be used to defeat CPP privacy. For example, if the user's location, given their address global identifier $\bar{P_0}$ is always fixed (e.g., the user is always in her home), then $\bar{P_0}$ is sufficient to disclose the location. In our security proof in Section 5, we will assume that the adversary does not have any side information about P_0. In general, CPP provides less benefit for non-mobile users.

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 a tunneling header in which the destination address contains the domain-dedicated prefix of that domain.

Figure 5 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.

[15] allows the global identifier P_0 to be hierarchically structured. In this case, CPP can be applied to the lower components of the global identifier, similar to the subnet identifier components.

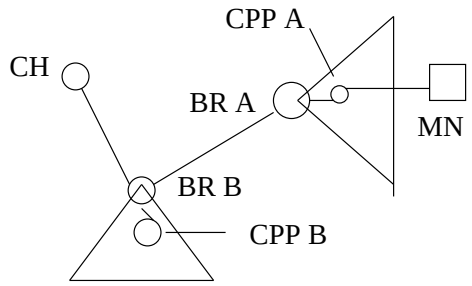


Fig. 5. CPP Intradomain Extension; CPP A and CPP B are CPP domains, BR A and BR B are border routers, MN has a CPP address

4 Security Considerations

4.1 Local Eavesdroppers

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 protection against local eavesdroppers can be provided. If protection against local eavesdroppers is desired, hosts and routers in the location privacy domain must not use link local addresses, do not perform local link address resolution [18], and do not perform duplicate address detection on the local link [26] 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 messages to routers. This message informs the router of the mapping between the hosts' link layer address and CPP IPv6 address. First hop routers are therefore the only nodes on the link that have any knowledge of a host's presence on the link.

4.2 Neighbor Solicitations

If an attacker has the resources to simultaneously send neighbor solicitations on all subnets, then the attacker can locate a target address. For example, when the attacker is on the same subnet as the target address, the attacker will start receiving traffic for the address after sending the solicitation. We prevent this last attack from occurring, by requiring that CPP access routers set the neighbor cache lifetime for a CPP address to

link layer address binding equal to the lifetime of the CPP address. With this countermeasure, an attacker cannot create a new mapping in the router neighbor cache with the CPP address of the victim and the attacker's link layer address. Therefore, the attacker cannot directly mount a subnet search attack.

4.3 ICMP

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. More generally, to best ensure security, routers within a CPP network should only send packets with CPP addresses as the source addresses.

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. Traceroute determines the topological location of a client by sending out packets with gradually increasing time to live (TTL) values until the correspondent on the other end of the connection is reached. TTLs that are smaller than required to reach the correspondent are decremented to zero enroot. If the TTL value is decremented to zero, the router doing the decrementing sends an ICMP Time Exceeded message back to the source. The source address on this packet is the router where the TTL = 0 was detected. An attacker could use such information to build up a location map of IP addresses.

An exception to the above rule is to send ICMP error packets with real source addresses when the sender has authenticated using IPsec 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. The general case for protocols that leak location information based on their selection of communication peers is discussed in Section 3.2.

5 Security of CPP Address

In this section, we prove that a CPP address does not reveal more than negligible information about user location to an attacker who can both send packets with various destination addresses and receive packets in up to s subnets (or locations) (see Theorem 2). Our proof is organized as a sequence of games [23, 2]. Game 0 models the active adversary's actions (sending packets with arbitrary destination addresses and receiving packets in up to s locations) aimed at learning about the plaintext of a chosen CPP address. Game 1 replaces the prf with a random function f. Game 2 aborts if the low probability attack where the adversary determines the covering value for P_i for some i, $3 \leq i \leq k$ succeeds, by first guessing the initial prefix components. Game 3 shows that the remaining attack, learning about the distribution $f_i(M, Q_1, \ldots, Q_{i-1}, X_{i+1}, \ldots, X_k)$ over $Q_1, \ldots, Q_{i-1}$ for $i \geq 2$, has limited probability of success. We give an example and compute a concrete bound for security. We show that the privacy of the plaintext address can still be protected when the adversary has side information at the cost of additional bits in the address, using an indistinguishability definition (see Section 5.6).

Throughout this section, $h(L, ...)$ is a keyed pseudo-random function. For our prototype, we instantiate h using the AES algorithm.

For IPv6 addresses, the global identifier P_0 may have 48 bits, and the subnet identifier could have 16 bits. In order to increase security, we will prepend two or more extra pseudo-random bits to the subnet identifier. We also use one bit for key version. Given the above parameters, then M would have 61 bits. If P_0 uses 32 bits, then our analysis still holds, but the bounds will be smaller (i.e., better).

5.1 Definitions

We write $w \leftarrow W$ to denote selecting an element w from the set W using the uniform distribution. We write $x \leftarrow f()$ to denote assigning the output of the function f, or algorithm f, to x. We let $|S|$ = length of the bit string S.

A function f is negligible if $f(n) < 1/P(n)$ for sufficiently large n, given any polynomial P. We sometimes use $negl$ to denote a negligible function. Throughout the paper, the adversary is an algorithm which we denote as $\mathcal{A}$. We sometimes use the term guess to denote the output of $\mathcal{A}$.

We follow [8] as explained in [23] for the definition of a pseudo-random function: Let l_1 and l_2 be positive integers, and let $\mathcal{F} = \{h_L\}_{L \in K}$ be a family of keyed functions where each function h_L maps $\{0,1\}^{l_1}$ into $\{0,1\}^{l_2}$. Let H_{l_1,l_2} denote the set of functions from $\{0,1\}^{l_1}$ to $\{0,1\}^{l_2}$.

Given an adversary $\mathcal{A}$ which has oracle access to a function in H_{l_1,l_2}. The adversary will output a bit and attempt to distinguish between a function uniformly randomly selected from $\mathcal{F}$ and a function uniformly randomly selected from H_{l_1,l_2}. We define the PRF-advantage of $\mathcal{A}$ to be

$$|Pr[L \leftarrow K : \mathcal{A}^{h_L}() = 1] - Pr[f \leftarrow H_{l_1,l_2} : \mathcal{A}^{f}() = 1]|$$

We define $\mathcal{F}$ to be *pseudo-random* if any adversary's PRF-advantage is negligible.

Intuitively, $\mathcal{F}$ is pseudo-random if it is hard to distinguish a random function selected from $\mathcal{F}$ from a random function selected from H_{l_1,l_2}.

5.2 CPP Adversary Model, Game 0, and Game 1

We now give the intuition behind the CPP adversary model, which is the basis for our initial game that models CPP security. The adversary can move to different subnets in the CPP access network, register a CPP address of its choosing (but this address must meet neighbor cache restrictions - see next paragraph for the definition of this property), and potentially 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. The total number of compromised locations during any key period (a compromised location is a location containing at least one compromised host controlled by the adversary) is less than or equal to s.

The adversary can send packets to any destination address. It will receive any packets sent to it by the other adversary components or non-compromised hosts, providing the neighbor cache restrictions are met, and provided equations (1) below are satisfied. (These equations must be satisfied to receive a packet at address $A = P_0, Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M$ and location $Q = Q_1, \ldots, Q_k$). 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, $\mathcal{A}$, s), where G is an aggregated routing graph, F is the function described below, $\mathcal{A}$ is an adversary that can query F, and s is the number of different locations that $\mathcal{A}$ can receive packets in. $\mathcal{A}$ selects an address $A = Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M$ and a location $Q = Q_1, \ldots, Q_k$. $F(A, Q)$ returns 1 if the following equations are satisfied (which models the case where $\mathcal{A}$ has address A at location Q)

$$G_1 = h(L_1, M, (Q_2 \oplus G_2), \ldots, (Q_k \oplus G_k))$$
$$G_2 = h(L_2, M, Q_1, (Q_3 \oplus G_3), \ldots, (Q_k \oplus G_k))$$
$$G_3 = h(L_3, M, Q_1, Q_2, (Q_4 \oplus G_4), \ldots, (Q_k \oplus G_k))$$

$$\begin{aligned} &\dots \\ G_{k-1} &= h(L_{k-1}, M, Q_1, \dots, Q_{k-2}, (Q_k \oplus G_k)) \\ G_k &= h(L_k, M, Q_1, \dots, Q_{k-1}) \end{aligned} \tag{1}$$

and A has not already been used at location Q. Otherwise $F(A, Q)$ returns no answer. The adversary can only receive packets if $F(A, Q) = 1$.[6] $\mathcal{A}$ is allowed to select up to s different (A, Q) pairs 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.[7]

Unless otherwise stated, we assume the subnet identifier is 16 bits with two additional initial pseudo-random prefix components, P_1 and P_2, prepended to it. Typically, P_1 and P_2 would have at least 1 or 2 bits each. As we will see below, P_1 and P_2 help protect against attacks. Thus the total length of the subnet identifier $P_1, \dots, P_k$ is 18-20 bits, or longer, depending if additional security is desired. Our arguments will be stated generally and can be applied to other subnet identifier lengths.

Game 0 is depicted in Figure 6. In game 0, the adversary selects a target address B to attack. The

Game 0. This game represents the adversary's possible actions given the CPP adversary model described above. Given the CPP adversary model instance: (G, F, $\mathcal{A}$, s). Given location $J_i = Q_1^i, \dots Q_k^i$. We often abuse notation by omitting the superscripts.
$r \leftarrow R$ for some set of coins R. $L \leftarrow K$ for the set of keys K.
$B \leftarrow \mathcal{A}(r, G)$ where $B = P_0, X_1, \dots, X_k, M_v$ not used before by $\mathcal{A}$ and has location $P_1, \dots, P_k$.
for $i \leftarrow 1...s$ do $(J_i, A_i) \leftarrow \mathcal{A}(r, G)$; $\{\emptyset, 1\} \leftarrow F(A_i, J_i)$; end for loop
$(B, \bar{P}_3, \dots, \bar{P}_k) \leftarrow \mathcal{A}(r, G, J_1, A_1, F(A_1, J_1), \dots, J_s, A_s, F(A_s, J_s))$,
output $(B, \bar{P}_3, \dots, \bar{P}_k)$

Fig. 6. Game 0: the CPP Adversary Model

adversary's goal is to obtain location information for B. In particular, the adversary wins if it guesses the location of address B (see Figure 6 and Equation 2 for the formal definition). The adversary may receive packets (depending if the address A_i it selects can receive packets at its current location J_i) allowing it to observe packets with various CPP addresses.[8] Receiving packets is dependent on the value of $F(A_i, J_i)$; the adversary learns this value for each (A_i, J_i) pair that it selects. Finally, after processing within the s locations, it outputs a guess, $\bar{P}_3, \dots, \bar{P}_k$, regarding the plaintext address or decryption of B. B must not have been assigned to the adversary previously, at the location $P_3, \dots, P_k$. Let S_0 be the event that $\bar{P}_3 = P_3, \dots, \bar{P}_k = P_k$. Then we define $\mathcal{A}$-*advantage* to be

$$\mathcal{A}\text{-}advantage = \left| Pr[S_0] - \frac{1}{2^{|P_3|+\dots+|P_k|}} \right| \tag{2}$$

In our main theorem (Theorem 2), we will show that $\mathcal{A}$-*advantage* is negligibly small.

[6] Intuitively, we can think of the (A,Q) pairs as ciphertext plaintext guesses and the function F as an oracle which allows the adversary to obtain confirmation whether any of the ciphertexts decrypts to its corresponding plaintext in the same pair.

[7] Note that our model captures the adversary's ability to send packets with arbitrary destination addresses and receive packets in up to s locations.

[8] We do not need to explicitly model sending packets in Game 0. All packets sent by the adversary are forwarded to a border router without regard to their source address. Then further forwarding is included in the model since $\mathcal{A}$ is a multi-located entity (any packet could have been forwarded from outside the access network). Similarly, receiving packets only serves to allow the adversary to learn the value of $F(A_i, J_i)$, which is included in Game 0.

Remark: In game 0, we assume the adversary has no side information about the target CPP address $B's$ location (or plaintext). We formalize this requirement as follows: at the point at which the adversary selects the target address B in game 0, the adversary's probability of outputting the plaintext corresponding to B is $1/2^{P_3+\ldots+P_k}$. In game $\bar{0}$ below, we will remove this assumption and allow the adversary to have side information; we will give a security definition below that reflects this point. The current definition will be adequate for some environments and has the advantage of consuming fewer bits of the address.

We now describe Game 1. Game 1 is depicted in Figure 7. Our analysis uses the fact that h_L is pseudo-

Game 1. Game 1 is identical to Game 0 except we replace the function h_L, where $L = L_1, \ldots, L_k$, in the definition of the function F with a random function.

Fig. 7. Game 1: Replace prf h_L with random function

random. We define S_1 to be the event that $\bar{P}_3 = P_3, \ldots, \bar{P}_k = P_k$ in Game 1. Then by the argument in [23], we have

$$|Pr[S_0] - Pr[S_1]| = \epsilon_{prf}$$

where ϵ_{prf} is the PRF-advantage, which is negligible since h is pseudo-random.

5.3 Game 2

The adversary can attempt a top down attack against B whereby it guesses the values for the intial prefix components $(P_1, \ldots, P_i)$ and if successful, can then obtain P_{i+1} if the following overdetermined equations are satisfied: $G_{i+2} \oplus Q_{i+2} = X_{i+2}$, $\ldots$, $G_k \oplus Q_k = X_k$, and $G_j = f_j(M_v, Q_1, \ldots, Q_{j-1}, X_{j+1}, \ldots, X_k)$, $i+2 \leq j \leq k$. We describe Game 2 and then show that the success probability of such an attack is negligible.

Game 2 is depicted in Figure 8.

Game 2. Game 2 is identical to Game 1, except if a certain packet is received based on a successful guess, it will cause a low probability failure event. The probability is bounded in the theorem below. Given the CPP adversary model instance: (G, F, $\mathcal{A}$, s).

$r \leftarrow R$ for some set of coins R. $f = (f_1, \ldots, f_k) \leftarrow H_{l_1,l_2}$.
$B \leftarrow \mathcal{A}(r, G)$ where $B = P_0, X_1, \ldots, X_k, M_v$ not used before by $\mathcal{A}$ and has location $P_3, \ldots, P_k$.
for $i \leftarrow 1...s$ do $(J_i, A_i) \leftarrow \mathcal{A}(r, G)$; $\{\emptyset, 1\} \leftarrow F(A_i, J_i)$; end for loop
event W_i occurs, $3 \leq i \leq k$, if $\mathcal{A}$ selects address
$A = G_1 \oplus Q_1, \ldots, G_{i+1} \oplus Q_{i+1}, X_{i+2}, \ldots, X_k$
where $G_j \oplus Q_j = P_j \oplus f_j(M_v, G_1 \oplus Q_1, \ldots, G_{j-1} \oplus Q_{j-1}, X_{j+1}, \ldots, X_k)$, $1 \leq j \leq i$,
and $G_{i+1} \oplus Q_{i+1} \neq X_{i+1}$ with location $Q_3, \ldots, Q_k$ such that $F(A, Q) = 1$.
event F_1 occurs if any of the W_i events occur
if F_1 occurs, abort Game 2.
$(B, \bar{P}_3, \ldots, \bar{P}_k) \leftarrow \mathcal{A}(r, G, J_1, A_1, F(A_1, J_1), \ldots, J_s, A_s, F(A_s, J_s))$,
output $(B, \bar{P}_3, \ldots, \bar{P}_k)$

Fig. 8. Game 2: Abort if Packet is Received That Reveals a Prefix Component Based on Successful Guess

We define S_2 to be the event that the components in the adversary's guess are equal to the plaintext prefix components of the address B, or event F_1 occurs. By Lemma 1 in [23], since Game 1 and Game 2 are identical until the failure event, we have

$$|Pr[S_1] - Pr[S_2]| \leq Pr[F_1]$$

The event F_1 is defined in Fig. 8.

Theorem 1. *Given a fixed i, $3 \leq i \leq k$. The probability of event W_i is bounded by $(2^{b_2}-1)/2^{b_2+b_1+b_3}$ where Let $|P_1|+|P_2|+\ldots+|P_i| = b_1$, $|P_{i+1}| = b_2$, and $|P_{i+2}|+\ldots+|P_k| = b_3$. Thus the probability of adversary success is negligible as $|P_1|+|P_2|$ increases.*

Proof. The probability that the adversary can initially output $G_j \oplus Q_j = P_j \oplus f_j(M_v, G_1 \oplus Q_1, \ldots, G_{j-1} \oplus Q_{j-1}, X_{j+1}, \ldots, X_k)$, $1 \leq j \leq i$, is bounded by 2^{-b_1}. Also, $G_{i+1} \oplus Q_{i+1} \neq X_{i+1}$ with probability $(2^{b_2}-1)/2^{b_2}$. To receive the packet for the corresponding address and location, the following overdetermined equations must be satisfied: $G_{i+2} \oplus Q_{i+2} = X_{i+2}$, $\ldots$, $G_k \oplus Q_k = X_k$, and $G_j = f_j(M_v, Q_1, \ldots, Q_{j-1}, X_{j+1}, \ldots, X_k)$, $i+2 \leq j \leq k$. Since f_j is random, this occurs with probability 2^{-b_3}.

5.4 Game 3

Given the relatively small sized prefix components, an adversary may gain significant information from sampling the probability distributions associated with the various prefix components. More precisely, the adversary observes

$$G_i = f_i(M_v, Q_1, Q_2, \ldots, Q_{i-1}, X_{i+1}, \ldots, X_k)$$

for various values of $Q_1, \ldots, Q_{i-1}$, where M_v and $X_{i+1}, \ldots, X_k$ are from the victim's CPP address. These are potential covering values for the prefix component P_i. The adversary obtains one of the values in the set of all possible covering values for P_i, each time a response packet is returned. The following security arguments will prove that CPP successfully defends against these attacks as well as all others that game 0 allows.

We will prove some lemmas that will be useful in the following security arguments. We will then prove bounds, with high probability, on the number of balls in any bin, given m balls tossed into n bins, where balls are equally likely to land in any of the bins. If a bin receives more balls than the bound, then this event will be a failure condition for game 3. The G_i values in the above equation can be represented by bins, and the $Q_1, \ldots, Q_k$ tuples represent balls. In general, we will model additional bins based on the limited probability of receiving response packets.

Both Stirling's formula and the Chernoff Bound below will enable us to prove some probability bounds:

Stirling's Formula: $k! \geq \sqrt{2\pi k}(k/e)^k$ and thus $k! \geq (k/e)^k$.

Lemma 1. *Suppose $me < nl$ and we toss m balls into n bins where each bin is equally likely to receive each ball. Let $w = me/nl$. Then*

$$Pr[\textit{bin } j \textit{ receives } l \textit{ or more balls}] \leq w^l \left(\frac{1}{1-w}\right)$$

Proof.

$$\begin{aligned} Pr[\geq l \textit{ balls in bin } j \ &= \sum_{i=l}^{m} \left(\frac{1}{n}\right)^i \frac{m!}{i!(m-i)!} \left(\frac{n-1}{n}\right)^{m-i} \leq \sum_{i=l}^{m} \left(\frac{1}{n}\right)^i \frac{m!}{i!(m-i)!} \\ &\leq \sum_{i=l}^{m} \left(\frac{me}{in}\right)^i \leq \sum_{i=l}^{m} \left(\frac{me}{ln}\right)^i = \sum_{i=l}^{m} (w)^i \leq \sum_{i=l}^{\infty} (w)^i = w^l \left(\frac{1}{1-w}\right) \end{aligned}$$

where the 3rd inequality follows from Stirling's formula above.

Chernoff Bound: Given n independent random variables $X_1, \ldots, X_n \in \{0,1\}$, and let $\mu = \sum_{i=1}^{n} Pr[X_i = 1]$. Then for any $\delta \leq 2e-1$,

$$Pr\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq e^{-1/4\delta^2\mu}$$

Lemma 2. *Suppose $m > n(\ln(n))$, and we toss m balls into n bins where each bin is equally likely to receive each ball. Let $l = m/n + \sqrt{(wm/n)\ln(n)}$, where $\sqrt{(wn/m)\ln(n)} \leq 2e-1$. Then*

$$Pr[l \text{ or more balls in any bin}] \leq 1/n^{(w/4)-1}.$$

Proof. We let the random variable X_i denote whether or not ball i falls in bin j for a fixed j. Thus $Pr[X_i = 1] = 1/n$. Thus $\mu = m/n$. Let $\delta = \sqrt{(wn/m)\ln(n)}$. By the Chernoff bound above, we have

$$\begin{aligned} Pr[\geq l \text{ balls in bin } j] &= Pr[\sum X_i \geq l] = Pr[\sum X_i \geq (1+\delta)\mu] \\ &\leq e^{-1/4\delta^2\mu} = e^{-1/4(wn/m)\ln(n)(m/n)} = e^{-1/4w\ln(n)} = n^{-w/4}. \end{aligned}$$

Thus, by the union bound, we have

$$Pr[l \text{ or more balls in any bin}] \leq 1/n^{(w/4)-1}.$$

Lemma 3. *Let $b_2 = |P_i|$, and let $b_1 = |P_1| + \ldots + |P_{i-1}|$, where $3 \leq i \leq k$. Suppose m balls are thrown into n bins where $m = s$, and $n \geq 2^{b_2}$. Suppose also that all bins have $l-1$ or fewer balls. Then the probability that the adversary successfully outputs the prefix component P_i from the address B is less than or equal to*

$$\frac{(2^{b_2}-1)(l-1)}{2^{b_1+b_2}} + \frac{1}{2^{b_2}}.$$

Proof. (Recall from above the $Q_1, \ldots, Q_k$ tuples are the balls and $G_i = f_i(M_v, Q_1, \ldots, Q_{i-1}, X_{i+1}, \ldots, X_k)$ are the bins.) Since f_i is random, the adversary's optimal strategy is to output a guess for P_i based on the bin with the most balls, since $P_i = X_i \oplus f_i(M_v, P_1, \ldots, P_{i-1}, X_{i+1}, \ldots, X_k)$. The probability of a correct guess is less than or equal to the probability of a correct guess when $P_i \neq Q_i$ for all Q_i associated with the maximal bin values (so that any value $Q_1, \ldots, Q_{i-1}$ can be equal to $P_1, \ldots, P_{i-1}$). Finally, we will ignore all values outside the maximal bin in order to further increase the probability, since one of these could also equal $P_1, \ldots, P_{i-1}$ thus implying the adversary's guess would be incorrect. Now given a random function $f : \{1, \ldots, 2^{b_1}\} -> R$ where $f(1) = \ldots = f(r)$, $|R| = n$, $S \subseteq R$, $|S| = 2^{b_2}$ and $r < l$. Then the probability that $f(1) = f(x)$ for some random $x \in \{1, \ldots, 2^{b_1}\}$ where $f(1), f(x) \in S$ is bounded by

$$\frac{(l-1)}{2^{b_1}} + \left(\frac{2^{b_1} - (l-1)}{2^{b_1}}\right)\frac{1}{2^{b_2}} = \frac{(2^{b_2}-1)(l-1)}{2^{b_1+b_2}} + \frac{1}{2^{b_2}}$$

Lemma 4. *Given i where $3 \leq i \leq k-1$. Let $|P_1|+|P_2|+\ldots+|P_{i-1}| = b_1$, $|P_i| = b_2$, and $|P_{i+1}|+\ldots+|P_k| = b_3$. We also set $p = \left(\frac{2^{b_1+b_2}-1}{2^{b_1+b_2}}\right)\frac{1}{2^{b_3}}$. Let $m = s$, and let $n = \lfloor (1/p)2^{b_2} \rfloor$. Given failure probability ϵ_i. If $m > n(\ln(n))$, we select w such that $2^{b_2}n^{-w/4} < \epsilon_i$. In this case $l = m/n + \sqrt{(wm/n)\ln(n)}$. Otherwise, we select l such that $w = me/nl$ satisfies $2^{b_2}w^l\left(\frac{1}{1-w}\right) < \epsilon_i$. Then*

$$Pr[\mathcal{A} \text{ guesses } P_i] \leq \frac{(2^{b_2}-1)(l-1)}{2^{b_1+b_2}} + \frac{1}{2^{b_2}}.$$

with probability greater than $1-\epsilon_i$.

Proof. The intuition here is that the attacker seeks to learn various values of

$$G_i = f_i(M_v, Q_1, \ldots, Q_{i-1}, X_{i+1}, \ldots, X_k)$$

by ranging over tuples $Q_1, \ldots, Q_k$ where $Q_{i+i} \oplus G_{i+1} = X_{i+1}, \ldots, Q_k \oplus G_k = X_k$. The equations

$$G_{i+1} = f_{i+1}(M_v, Q_1, \ldots, Q_i, X_{i+2}, \ldots, X_k)$$

$$\vdots$$
$$G_k = f_k(M_v, Q_1, \ldots, Q_{k-1})$$

are satisfied with probability $1/2^{b_3}$ (these equations must be satisfied in order to receive a response packet). Also, we can't have $Q_1 = P_1, \ldots, Q_i = P_i$, else we won't receive a response packets since then $Q_{i+1} = P_{i+1}, \ldots, Q_k = P_k$ and the attacker address and location is identical to the victim's. The attacker cannot get the victim's address in the victim's location. Therefore

$$p = \left(\frac{2^{b_1+b_2}-1}{2^{b_1+b_2}}\right)\frac{1}{2^{b_3}}$$

is the probability of receiving a response packet at a given address.

Thus we may conceptualize $n = \lfloor (1/p)2^{b_2} \rfloor$ bins that the m balls are tossed into, where each bin is equally likely to receive each ball. Here the bins represent possible values for the G_i covering variables, and the balls represent various $Q_1, \ldots, Q_k$ tuples. The additional bins represent the probability $1-p$ of not receiving a packet. Thus we are only interested in the number of balls in each of the first 2^{b_2} bins. We slightly overestimate the probability since $p\lfloor (1/p)2^{b_2} \rfloor \geq 2^{b_2}$. This overestimation is okay since we are obtaining an upper bound.

Given m balls tossed into the n bins, Lemma 3 combined with Lemmas 1 and 2 gives that

$$Pr[\mathcal{A} \textit{ guesses } P_i] \leq \frac{(2^{b_2}-1)(l-1)}{2^{b_1+b_2}} + \frac{1}{2^{b_2}}$$

with probability greater than $1-\epsilon_i$, where $2^{b_2}n^{-w/4} < \epsilon_i$ and $l = m/n + \sqrt{(wm/n)\ln(n)}$ if $m > n\ln(n)$. Otherwise, we select l such that $w = me/nl$ and $2^{b_2}w^l\left(\frac{1}{1-w}\right) < \epsilon_i$. This completes the proof.

We now consider the lowest prefix component in a CPP address. We note that the $Q_i \oplus G_i = X_i$ equations do not have to be satisfied, so the attacker can select $G_1, \ldots, G_k$ through multiple guesses until a packet is received in every one of the s subnets.

Lemma 5. *Given the lowest prefix component P_k. Let $|P_1| + |P_2| + \ldots + |P_{k-1}| = b_1$, and $|P_k| = b_2$. Let $m = s$, and let $n = 2^{b_2}$.*

Given failure probability ϵ_k. If $m > n(\ln(n))$, we select w such that $n^{-w/4+1} < \epsilon_k$. In this case $l = m/n + \sqrt{(wm/n)\ln(n)}$. Otherwise, we select l such that $w = me/nl$ satisfies $nw^l\left(\frac{1}{1-w}\right) < \epsilon_k$. Then

$$Pr[\mathcal{A} \textit{ guesses } P_k] \leq \frac{(2^{b_2}-1)(l-1)}{2^{b_1+b_2}} + \frac{1}{2^{b_2}}.$$

with probability greater than $1-\epsilon_k$.

Proof. The proof is analogous to the proof of Lemma 4, except that the probability of receiving a response packet is always 1 at a given address, assuming the address B isn't used. Thus $n = 2^{b_2}$ and the proof follows.

Remark: We note that Lemma 4 and Lemma 5 imply asymptotic security: as we increase $|P_1|$ and $|P_2|$ then b_1 increases. Furthermore, l is bounded when $m > n\ln(n)$, since m equals s and thus m is bounded. We see that a similar argument shows that l is also bounded when $m \leq n\ln(n)$. Thus the probability of a successful guess approaches $1/2^{b_2}$. Note that the adversary can use random guessing with a success probability of $1/2^{b_2}$ in order to guess P_i.

Game 3 is depicted in Figure 9. Game 3 is identical to Game 2, except

1. We restrict the adversary's received packets to have destination addresses of the form $A = Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M_v$ where M_v is the host identifier from the target address B. Without this restriction, the packet gives no information about the address B, since $f = (f_1, \ldots, f_k) \leftarrow H_{l_1,l_2}$. Thus this restriction does not affect the security. Note that an adversary strategy based on observation of the target's network traffic patterns cannot yield information on $B's$ location due to our assumption from Section 3.2.

2. We incorporate the failure events corresponding to receiving more than the bounded number of balls in a bin (from Lemma 4 and Lemma 5 above).

Game 3. Given the CPP adversary model instance: (G, F, $\mathcal{A}$, s).

$r \leftarrow R$ for some set of coins R. $f = (f_1, \ldots, f_k) \leftarrow H_{l_1,l_2}$.
$B \leftarrow \mathcal{A}(r, G)$ where $B = P_0, X_1, \ldots, X_k, M_v$ not used before by $\mathcal{A}$ and has location $P_3, \ldots, P_k$.
for $i \leftarrow 1...s$ do $(J_i, A_i) \leftarrow \mathcal{A}(r, G)$ where $A_i = Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M_v$
and $G_j = f_j(M_v, Q_1, \ldots, Q_{j-1}, X_{j+1}, \ldots, X_k)$, for some j, $1 \leq j \leq k$.
$\{\emptyset, 1\} \leftarrow F(A_i, J_i)$ end for loop
event W_i occurs, $3 \leq i \leq k$, if $\mathcal{A}$ selects address
$A = G_1 \oplus Q_1, \ldots, G_{i+1} \oplus Q_{i+1}, X_{i+2}, \ldots, X_k$
where $G_j \oplus Q_j = P_j \oplus f_j(M_v, G_1 \oplus Q_1, \ldots, G_{j-1} \oplus Q_{j-1}, X_{j+1}, \ldots, X_k)$, $1 \leq j \leq i$,
and $G_{i+1} \oplus Q_{i+1} \neq X_{i+1}$ with location $Q_3, \ldots, Q_k$ such that $F(A, Q) = 1$.
event F_1 occurs if any of the W_i events occur
if F_1 occurs, abort Game 3. Define events F_i, $3 \leq i \leq k$:
if the number of observed values for the maximally occurring value (number of balls in a single bin) for $G_i = f_i(M_v, Q_1, \ldots, Q_{i-1}, X_{i+1}, \ldots, X_k)$ exceeds the bound in Lemma 4 or Lemma 5 then event F_i occurs.
if any of the events F_3-F_k occur, then abort Game 3.
$(B, \bar{P}_3, \ldots, \bar{P}_k) \leftarrow \mathcal{A}(r, G, J_1, A_1, F(A_1, J_1), \ldots, J_s, A_s, F(A_s, J_s))$,
output $(B, \bar{P}_3, \ldots, \bar{P}_k)$

Fig. 9. Game 3: Abort if Too Many Balls Land in Any One Bin

We define S_3 to be the event that the components in the adversary's guess are equal to the plaintext prefix components of the address B, or any of the failure events occur. By Lemma 1 in [23], since Game 2 and Game 3 are identical until the failure events (we can ignore the unnecessary packets in Game 2), we have

$$|Pr[S_2] - Pr[S_3]| \leq \sum_{i=3}^{k} Pr[F_i]$$

Theorem 2. *In game 0, $\mathcal{A}$-advantage is negligibly small.*

Proof. Our proof strategy is to first prove that the probability the adversary successfully guesses the prefix given that no failure events occur in game 3 is negligibly close to $1/2^{|P_3|+\ldots+|P_k|}$. Then we can use the triangle inequality combined with the negligible probability of the various failure events and ϵ_{prf} to show that $\mathcal{A}$-*advantage* is negligibly small.

In game 3, the adversary receives various packets with destination addresses of the form $A = Q_1 \oplus G_1, \ldots, Q_k \oplus G_k, M_v$ where $G_j = f_j(M_v, Q_1, \ldots, Q_{j-1}, X_{j+1}, \ldots, X_k)$, for various j, $1 \leq j \leq k$. The adversary is restricted to receiving packets in at most s locations, say the adversary receives in i_1 locations where $j = 1$, in i_2 locations where $j = 2, \ldots$, and in i_k locations where $j = k$. We can upper bound the probability of adversary success by assuming the adversary receives packets in s locations for each value j.

The adversary has no side information for the address B (see the Section 5.2 Remark.) Now

$$Pr[\mathcal{A}\ guesses\ P_1, \ldots, P_i] \leq Pr[\mathcal{A}\ guesses\ P_1] = negl$$

for $i \geq 1$. Thus any attack which determines higher prefix components from lower ones using the extended CPP address equations has a negligible probability of success. Also,

$$Pr[\mathcal{A}\ guesses\ P_3, \ldots, P_k] \leq \prod_{i=3}^{k} Pr[\mathcal{A}\ guesses\ P_i] + negl$$

since determining any of $P_3, \ldots, P_k$ gives negligible information on the other prefix components per the preceding sentence. Thus

$$Pr[\mathcal{A}\ guesses\ P_3, \ldots, P_k] - \prod_{i=3}^{k} 2^{-|P_i|} \leq \prod_{i=3}^{k} Pr[\mathcal{A}\ guesses\ P_i] + negl - \prod_{i=3}^{k} 2^{-|P_i|}$$

Also,

$$Pr[\mathcal{A}\ guesses\ P_i] \leq \frac{(2^{|P_i|}-1)(l_i-1)}{2^{|P_1|+|P_2|+\ldots+|P_i|}} + \frac{1}{2^{|P_i|}}$$

which as we have seen from above, approaches $\frac{1}{2^{|P_i|}}$ as $|P_1| + |P_2|$ increases. Thus the probability $\bar{p}$ that the adversary successfully guesses the prefix given that no failure events occur is negligibly close to $1/2^{|P_3|+\ldots+|P_k|}$.

We have[9]

$$\begin{aligned}
Pr[S_0] = Pr[S_0] - Pr[S_3] + Pr[S_3] &\leq |Pr[S_0] - Pr[S_3]| + |Pr[S_3]| \\
&\leq |Pr[S_0] - Pr[S_3]| + \bar{p} + \sum_{i=3}^{k} Pr[W_i] + \sum_{i=3}^{k} Pr[F_i] \\
&\leq |Pr[S_0] - Pr[S_1]| + |Pr[S_1] - Pr[S_2]| + |Pr[S_2] - Pr[S_3]| + \bar{p} \\
&\quad + \sum_{i=3}^{k} Pr[W_i] + \sum_{i=3}^{k} Pr[F_i] \\
&\leq \epsilon_{prf} + 2\sum_{i=3}^{k} Pr[W_i] + 2\sum_{i=3}^{k} Pr[F_i] + \bar{p} \\
&\leq \epsilon_{prf} + 2\sum_{i=3}^{k} Pr[W_i] + 2\sum_{i=3}^{k} \epsilon_i + \bar{p} \leq negl + \bar{p}
\end{aligned}$$

where *negl* denotes a negligible function. Thus we see that $\mathcal{A}$-*advantage* in game 0 is negligibly small as desired.

5.5 Example

In this section, we present an example where we instantiate the various parameters and bound the probability for an adversary to obtain the plaintext address or location for a given CPP target address.

Example: We set $s = 64$. We also set $|P_1| = 2$, and $|P_2| = 2$. Suppose the network is configured so that $|P_3| = 3$, $|P_4| = 4$, $|P_5| = 4$, and $|P_6| = 5$.

The following lemma will be helpful for computing tighter bounds for the highest prefix component.

Lemma 6. *Given CPP target address $B = P_0, X_1, \ldots, X_k, M_v$ and i where $3 \leq i \leq k-1$. Let $b_1 = |P_1| + |P_2| + \ldots + |P_{i-1}|$, and $b_2 = |P_i|$. We define $\beta = (2^{b_1} - 1)/(2^{b_1+b_2} - 1)$. Then*

$$Pr[\mathcal{A}\ guesses\ P_i | 1\ response\ packet] = \beta\frac{1}{2^{b_2}} + (1-\beta)\left[\frac{1}{2^{b_1}} + (1 - \frac{1}{2^{b_1}})\frac{1}{2^{b_2}}\right]$$

Proof. Note that $\beta = Pr[P_i = Q_i]$, where Q_i belongs to the adversary plaintext address. When $P_i = Q_i$, then we can't have $P_1 = Q_1, \ldots, P_{i-1} = Q_{i-1}$. (Else no response packet can be received.) Thus $\bar{G}_i$ for the victim address is obtained from a different domain point and equals G_i with probability $\frac{1}{2^{b_2}}$. When $P_i \neq Q_i$, then either $P_1 = Q_1, \ldots, P_{i-1} = Q_{i-1}$, (with probability $1/2^{b_1}$) in which case $\bar{G}_i = G_i$, or when $P_1 = Q_1, \ldots, P_{i-1} = Q_{i-1}$ is false, then since it's a different domain point, we have $\bar{G}_i = G_i$ with probability $1/2^{b_2}$.

[9] Here we use $Pr[A \bigcup B] = Pr[(A \bigcap B^c) \bigcup B] \leq Pr(A/B^c) + Pr(B)$ for events A and B.

The lemma also holds when the condition is replaced with 2 response packets. The reason is that the adversary can make its decision after receiving the first response packet, since if the 2nd response packet gives a different value, there is no probability advantage for the 2nd of two distinct values.

We will also make use of the following:

$$Pr[correct\ guess] \leq Pr[0\ responses]Pr[correct\ guess|0\ responses] + \ldots + Pr[i\ responses]Pr[correct\ guess|i\ responses] + Pr[> i\ responses] \quad (3)$$

We now analyze the example above in order to bound the probability of the adversary's determining the location of a CPP address. We analyze each of P_3, P_4, P_5, and P_6 individually. We set $\epsilon_6 = 2^{-50}$, $\epsilon_5 = 2^{-15}$, and $\epsilon_4 = 0$. Also, $\epsilon_3 = 0$.

P_3 : Here we use Lemma 6 and (3) above in order to bound the probability of the adversary's success. The intuition here is that for the high order prefix components such as P_3, the probability of receiving response packets is low. Thus an analysis based on the probability of receiving small numbers of response packets gives good bounds. We set $b_1 = 4$, $b_2 = 3$, and $b_3 = 13$. Let

$$p = \left(\frac{2^{b_1+b_2} - 1}{2^{b_1+b_2}}\right)\frac{1}{2^{b_3}}$$

which is the probability of receiving a response packet. Thus $p = (127/128)(1/2^{13}) \approx .000121116$. We set $q = 1-p \approx 0.999879$. From Lemma 6, we have $\beta = 15/127$, and $\rho = Pr[\mathcal{A}\ correctly\ guesses\ Pi|1\ response\ packet] = 15/127(1/8) + (112/127)(1/16 + (15/16)1/8) \approx 0.173228$. We note that

$$\rho = Pr[\mathcal{A}\ correctly\ guesses\ Pi|2\ response\ packets]$$

as well, since an optimal strategy is for the adversary to make a guess after receiving the first packet. The second response packet does not affect the adversary's choice in this case. The probability of receiving 3 or more responses is

$$\lambda = 1 - q^s - q^{s-1}ps - \binom{s}{2}q^{s-2}p^2$$

We have

$$Pr[successful\ guess] \leq q^s\frac{1}{2^{b_2}} + sq^{s-1}p\rho + \frac{s(s-1)}{2}q^{s-2}p^2\rho + \lambda$$

Given the above values, we compute $\lambda \approx .000000669$, and

$$Pr[successful\ guess] \leq 0.125373 \approx \frac{1}{7.9762}$$

P_4 : Using Lemma 6 we have $b_1 = 7$, $b_2 = 4$, and $b_3 = 9$. Then $m = 64$. We have $p = (2^{11}-1)/2^{11}(1/512) \approx .00195217$. We set $q = 1 - p \approx 0.998048$. From Lemma 6, we have $\beta = 127/2047$, and $\rho = 127/2047(1/16) + (1920/2047)(1/128 + (127/128)(1/16)) \approx 0.0693698$. Given the above values, we compute $\lambda \approx .000272948$, and

$$Pr[successful\ guess] \leq 0.766475 \approx \frac{1}{13.04674}$$

P_5 : We have $b_1 = 11$, $b_2 = 4$, and $b_3 = 5$. Then $m = 64$. Using Lemma 4, we have $n = \lfloor(1/p)16\rfloor = 512$. Since $m < n(\ln(n))$, we have $w = me/nl = 64e/(512)l = e/8l$. We require $(16w^l)/(1-w) < \epsilon_5$. $l = 5$ is the smallest value for l which satisfies the inequality in the preceding sentence. Thus $Pr[successful\ guess] \leq ((16-1)4)/2^{15} + 1/16 \approx .064331 \approx 1/15.544592$.

P_6 : Here $b_1 = 15$ and $b_2 = 5$. $n = 32$. Since $m < n(\ln(n))$, we have $w = me/nl = 2e/l$. We require $(32w^l)/(1-w) < \epsilon_6$. $l = 32$ satisfies the inequality in the preceding sentence. Thus by Lemma 5, $Pr[successful\ guess] \leq ((32-1)31)/2^{20} + 1/32 \approx .0321665 \approx 1/31.08826$

Thus the attack aimed at determining the individual prefix components reduces the uncertainty about the subnet prefix by less than 2 bits. But since $|P_1| + |P_2|$ is small, an adversary can try to guess P_1 and

P_2 and other high prefix components, and then use the CPP decryption algorithm to obtain the rest of the subnet prefix, i.e., the Game 2 attack. We now evaluate this attack.

P_3 : As in Theorem 3, the probability of receiving a sequence of packets with the correct subnet prefix components is

$$\alpha \sum_{i_1+i_2+i_3 \leq s-3} p_1^{i_1} p_2^{i_2} p_3^{i_3} \leq \alpha \sum_{i_1+i_2+i_3 \leq s-3} 1 \leq \alpha \frac{(s-1)s(2s-1)-6}{12}$$

where $\alpha = (1-p_1)(1-p_2)(1-p_3)$. The probability that the initial guess for P_1 and P_2 is correct is $1/2^4$ which must be multiplied with the value from the above expression to get the probability bound:

$$Pr[\mathcal{A} \textit{ outputs } P_3,\ P_4,\ \textit{and } P_5] \leq$$
$$2^{-4}(7/8)2^{-13}(15/16)2^{-9}(15/16)2^{-5}((64)(63)(127)-6)/12 \approx 2^{-16}$$

Note that a single additional packet suffices to give P_6 if $P_1, \ldots, P_5$ are known.

P_4 : In this case, the adversary initial guess for P_1, P_2, and P_3 has success probability 2^{-7}. The probability of receiving a sequence of packets with the correct prefix components is bounded by

$$\alpha \sum_{i_1+i_2 \leq s-2} p_1^{i_1} p_2^{i_2} \leq \alpha \sum_{i=0}^{s-2} (i+1)p^i =$$
$$\alpha[\frac{(1-p^s)-(s)p^{s-1}(1-p)}{(1-p)^2}] \tag{4}$$

where $\alpha = (1-p_1)(1-p_2)$, $p = \max\{p_1, p_2\}$, and p_1 and p_2 are defined above. Applying this formula with $p_1 = 1-(15/16)2^{-9}$, and $p_2 = 1-(15/16)2^{-5}$, and $s = 64$, we obtain $Pr[\mathcal{A} \textit{ outputs } P_1, \ldots, P_6 \leq 1/1280$. This is the optimal attack for the adversary. It's easy to see that the same type of attack aimed at P_5 or P_6 has a lower success probability.

Remark: If we set $|P_1| = |P_2| = 3$, then the above success probability is reduced by a factor of 4.

5.6 Security Given Adversary Knowledge of the Plaintext

In this section, we show that the privacy of the plaintext address can still be protected given the case where the adversary has side information about the subnet prefix. Based on the structure of the CPP address, this type of attack applies to prefix components P_i where $i \geq 3$.

We will replace game 0 with the following game $\bar{0}$, which is depicted in Figure 10. This game represents the adversary's possible actions given the CPP adversary model described above, where we allow the adversary to have knowledge of some parts of the plaintext address. We follow [9] for our indistinguishability definition below.

Game $\bar{0}$. Given the CPP adversary model instance: (G, F, $\mathcal{A}$, s).
$r \leftarrow R$ for some set of coins R. $L \leftarrow K$ for the set of keys K.
$Addr_0, Addr_1 \leftarrow \mathcal{A}(r, G)$ where $Addr_0, Addr_1$ are two distinct plaintext addresses.
$b \leftarrow \{0,1\}$, where b is unknown to the adversary
The CPP encryption of $Addr_b$, B, is given to $\mathcal{A}$.
for $i \leftarrow 1...s$ do $(J_i, A_i) \leftarrow \mathcal{A}(r, G)$; $\{\emptyset, 1\} \leftarrow F(A_i, J_i)$; end for loop
$\acute{b} \leftarrow \mathcal{A}(r, G, J_1, A_1, F(A_1, J_1), \ldots, J_s, A_s, F(A_s, J_s))$,
output 1 if $\acute{b} = b$ else 0.

Fig. 10. Game $\bar{0}$: Indistinguishable Encryptions

We define $\bar{S_0}$ to be the event that game $\bar{0}$ outputs 1. We can prove that this event occurs with probability less than negligibly more than one-half. In this case, we say that we have indistiguishable encryptions within

the CPP adversary model. The argument is similar to the proof above. Although security as defined in Game $\bar{0}$ is stronger than in Game 0, Game $\bar{0}$ requires additional bits in the address to be utilized. Thus the Game 0 definition is useful.

6 Implementation and Performance Results

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 P_i, 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. 11 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 the encrypted subnet identifier, X.

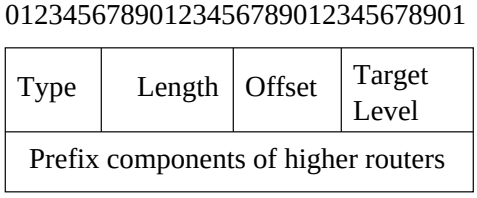


Fig. 11. CPP Hop by Hop Option Format

The Offset field (8 bits) identifies the bit position to begin decryption of this level's prefix component. The Target Level gives the level of the router that last decrypted a prefix component (see the Figure 3 example for a case where not every router on the forwarding path decrypts a prefix component as part of forwarding). 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 algorithms were used for $h()$: SHA-1 and AES. The input variables to $h()$ are the router's secret key L_i, $P_i's$, $X_i's$, and M, with additional zero-padding bits 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 the higher routers field, overwriting the offset field, and updating the target level field. (If the target level field value is the level of the current router, then the current router will not decrypt the next prefix component, rather a succeeding router on the forwarding path will perform the next decryption).

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 subnet identifier. 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. Our results indicate that AES is preferable over SHA-1. Our AES result suggests CPP imposes a 50% performance impact. We believe this impact is acceptable, since (1) with further optimization processing time can be reduced (2) increasing hardware capabilities including commodity multi-processor systems enables the overhead to be ameliorated over multiple packets.

Type of Router	**unmodified**	**SHA-1**	**AES**
Time to route one packet	6 μsec.	11 μsec.	9 μsec.

Table 1. Forwarding Time Performance for Unmodified and CPP Forwarding

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 [22] is used, CPP cryptographic processing is typically only required for the first packet with a given CPP address during the address period.

6.1 Optimized Routing

The above performance analysis focuses on the packet forwarding overhead for packets with CPP destination addresses in the destination privacy domain. CPP incurs another performance overhead that we discuss here. Specifically, packets sent from hosts within the CPP privacy domain to hosts within the same privacy domain (but external to the sender's subnet) are forwarded up to a border router and then forwarded through routers to the destination host. Without CPP, these packets may have a shorter path (fewer routers to traverse); in other words, they don't necessarily need to be sent up to a border router. Thus CPP requires routers within the privacy domain to process more packets then they would if CPP was not deployed.

The performance impact can be partially mitigated by using the router cache. In this case, a router within the privacy domain would cache the next hop for CPP destination addresses within the domain. The first packet of a flow may still need to be forwarded on the longer path through the border router, but subsequent packets of the flow would follow the optimized forwarding path. There is a trade-off with security here; the optimized forwarding path is vulnerable to timing attacks. For example, a sending host could approximate the number of hops between itself and a peer in the same privacy domain based on the round trip times. Thus a CPP privacy domain that desires to maximize location privacy should avoid using the router cache for packets with CPP destination addresses and source addresses within the domain.

7 Related Work

Prior work on location privacy can be separated into IETF protocol work including Mobile IPv6 [17] and HMIPv6 [24] (which are the location privacy equivalent of an anonymizing proxy), anonymity techniques that also provide location privacy, and a technique that modifies the basic underlying IPv6 routing, IP^2 [28]. Onion routing [25], Tor [5], Freedom [7], and Tarzan [6] are examples of overlay approaches specifically developed for providing anonymity that are discussed here as location privacy solutions. The review in [4] contains more information on existing generic network level privacy approaches that are not specifically mobility related.

Mobility management protocols like Mobile IPv6 [17] provide means whereby the geographical location of the mobile node can remain hidden to the correspondent and some eavesdroppers, though that is not its

primary purpose. The basic location management technique of Mobile IP requires correspondent hosts to tunnel traffic to and from a mobile host through a home address in the mobile host's home network. As the mobile host moves, its home address does not change, and so the correspondent host and eavesdroppers on the correspondent host to home agent traffic cannot obtain any information about the mobile host's immediate geographical location.

Both Mobile IP and HMIPv6 force all the traffic to/from a mobile host to traverse via the home agent or a regional CoA server. This includes the traffic within the coverage area of the regional CoA server for HMIP, thus enforcing suboptimal routes within the region and making the home agent or regional CoA server into traffic concentration points. As a result, these approaches are vulnerable to host attacks, against both mobile nodes and the CoA server. In particular, the regional CoA server is a stateful single point of failure, and is vulnerable to denial of service attacks as well. Both HMIPv6 and Mobile IP are vulnerable to intersection attacks, including in the case where IPsec encryption is used. These attacks occur when an attacker matches up outgoing packets on the internal network with outgoing packets between the regional CoA server, or home agent, and the Internet. Multiple associated sets of packets can be intersected which then yields the mapping between CoA and HoA. Application specific work includes [3, 13].

Onion routing [25] is seminal work and is an example of a generic privacy approach that secondarily provides location privacy. Low latency anonymity is provided by an overlay network of onion routers that proxy TCP traffic through a series of TCP connections. Onion routing does not protect against host attacks aimed at obtaining a host's location, and it does not furnish an easily scalable means for removing source IP addresses from application layer data. Since it is a system of proxies, it cannot coexist with end to end IPsec. Therefore, Onion Routing does not provide an easily scalable means of achieving location privacy for all applications.

Tor [5] is the second generation of Onion routing; it incorporates additional features such as location hiding for servers, as well as other improvements. For the purposes of this paper, it is fairly similar to Onion routing, as is the Freedom Network [7] and Tarzan [6]. The main limitations of these approaches compared to CPP is that they require routing inefficiencies that significantly increase latency for applications, and the need to rewrite source addresses at the application layer. They are also vulnerable to intersection attacks when the communication endpoints are located in the same access network.

An approach that requires modifying the border routers and access routers is IP^2 [28]. IP^2 requires that servers in the control plane maintain the mapping between the home address and the CoA in a logically centralized manner. This approach may lead to scalability and performance issues. Additionally, the approach assumes that all routers and operators in the IP^2 network are trusted.

There has also been work on layer two location privacy [11]. Our focus in this paper is on layer three, and in particular, IPv6.

7.1 Address Hiding Protocol (AHP)

In [21], the authors take the same approach as we do, using encrypted addresses to obtain anonymity. Their focus is on IP (version 4), and their scheme corresponds to CPP deployed to only the border routers. Their encryption algorithm includes both destination and source ports as well as the destination IP address as inputs. Although it produces distinct encrypted source addresses for each connection, this is problematic for some applications (e.g., http cookies).

In their published encryption scheme [21], an attack exists that enables a server and a single internal confederate host to decrypt the address of another arbitrary internal host.

The Attack: We now describe the attack against the scheme in [21]. Given the victim source (encrypted) address $F_k(M \oplus H(t)) \oplus H(t)$ where t is the destination IP address of the server concatenated with the destination port of the server, M contains the two lower bytes of the victim plaintext IP address plus the client source port, F_k is the 32 bit symmetric key cipher, and H is the hash function that is truncated to four bytes of output. (The two high order bytes of the AHP encryption replace the host portion of the source IP version 4 address in the outgoing packet, and the two low order bytes of the AHP encryption similarly replace the source port. The network portion of the source address is necessarily the network identifier for

AHP protected ISP). The curious server constructs destination IP addresses of the form

$$F_k(M \oplus H(t)) \oplus H(t) \oplus H(t) \oplus H(\bar{I}, \bar{P}) = F_k(M \oplus H(t)) \oplus H(\bar{I}, \bar{P})$$

and sends the packet using source IP $\bar{I}$ and source port $\bar{P}$. The receiving decrypting router will compute

$$M \oplus H(t) \oplus H(\bar{I}, \bar{P})$$

Suppose

$$H(\bar{I}, \bar{P}) = H(t) \oplus M \oplus M_2$$

when each value is truncated to the first two bytes, and the first two bytes of M_2 are equal to the last two bytes of the confederate host's IP address. Then the packet is forwarded to some port on the confederate host. The expected number of packets that the server must send before the confederate host receives a packet is about 44000 which makes this attack easy to carry out. There is a significant chance of success with less than half as many packets. Note also that the last equation above allows the adversary to compute the first two bytes of M which is the last two bytes of the victim's IP address (the first two bytes are the organization's prefix identifier and are known to the adversary). Also, since $\bar{I}$ concatenated with $\bar{P}$ is 6 bytes, then with very high probability, $H(\bar{I}, \bar{P})$ ranges over the full 4 byte space.

This attack is outside the security model discussed in [21] where the adversary is restricted to being external to the ISP. It is possible that an alternative cipher could be used in their scheme to improve security against this particular attack. Nevertheless, the attack is realistic since among the large set of hosts within the ISP, it is reasonable to assume that at least one host is compromised and under the control of the location privacy adversary. In this sense, the AHP model of the adversary is too restricted since it doesn't protect against this realistic attack.

8 Summary

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 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. Only routers within the protected domain can decrypt the address. CPP involves modifications to the basic IP forwarding algorithm, and our implementation results indicate that a performance penalty in forwarding is incurred, which we have argued is acceptable. We have given a proof of security within our security model against active bounded adversaries.

References

1. L. Ackerman, J. Kempf, and T. Miki. Wireless Location Privacy: Current State of U.S. Law and Policy. In *Proceedings of the Workshop on Privacy, Washington, DC*, pages 24–31. ACM, 2003.
2. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - Eurocrypt 2006*, pages 409–426. Springer Berlin/Heidelberg, 2006.
3. O. Berthold, H. Federrath, and S. Kospell. Web mixes: A system for anonymous and unobservable Internet access. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 409–426. Springer, 2001.
4. Council Canada, Ronggong Song, and Larry Korba. Review of network-based approaches for privacy. In *Proceedings of the 14th Annual Canadian Information Technology Security Symposium*, pages 13–17, 2002.
5. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of 13th USENIX Security Symposium*, pages 303–320, 2004.
6. Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, 2002.
7. Ian Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, University of California, Berkeley, Berkeley, CA, USA, 2000.

8. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33:210–217, 1986.
9. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
10. R.W. Gosper. Decision Procedure for Indefinite Hypergeometric Summation. *Proc. Nat. Acad. Sci. USA*, 75:40–42, 1978.
11. M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers. *ACM Mobile Networks and Applications (MONET)*, 10:315–325, 2005.
12. S. Guha and P. Francis. Identity trail: Covert surveillance using DNS. In *Privacy Enhancing Technologies (PET 2007)*, pages 315–325, 2007.
13. C. Gulcu and G. Tsudik. Mixing e-mail with babel. In *Network and Distributed Systems Security (NDSS 1996)*, pages 2–16, 1996.
14. R. Hinden and S. Deering. Internet Protocol Version 6 (IPv6) Addressing Architecture. *RFC 3513*, April 2003.
15. R. Hinden, S. Deering, and E. Nordmark. IPv6 Global Unicast Address Format. *RFC 3587*, 2003.
16. IAB and IESG. IAB/IESG Recommendations on IPv6 Address Allocations to Sites. *RFC 3177*, 2001.
17. D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. *RFC 3775*, June 2004.
18. T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). *RFC 4861*, 2007.
19. Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *Proceedings of the SIGCOMM 01*, SIGCOMM '01, pages 173–185, New York, NY, USA, 2001. ACM.
20. R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
21. B. Raghavan, T. Kohno, A. Snoeren, and D. Wetherall. Enlisting ISP's to improve online privacy: IP address mixing by default. In *Privacy Enhancing Technologies (PETS 2009, LNCS 5672)*, pages 143–163. Springer-Verlag, 2009.
22. E. Rosen and Y. Rekhter. BGP/MPLS VPNs. *RFC 2547*, 1999.
23. V. Shoup. Sequences of games: a tool for taming complexity in security proofs. *http://www.shoup.net/papers/games.pdf*, January 2006.
24. H. Soliman, C. Castelluccia, K. El-Malki, and L. Bellier. Hierarchical Mobile IPv6 Mobility Management (HMIPv6). *RFC 4140*, 2005.
25. P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54. IEEE CS Press, 1997.
26. S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. *RFC 2462*, 1998.
27. Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards street-level client-independent IP geolocation. In *8th USENIX Symposium on Networked System Design and Implementation NSDI '11*, pages 27–27. USENIX Association, 2011.
28. M. Yabusaki, S. Hirata, T. Ihara, and H. Kawakami. IP^2 Mobility Management. *NTT DoCoMo Technical Journal*, 4(4):16–22, 2003.

APPENDIX

A Security in the Case Where No Random Bits are Prepended to the Plaintext Subnet Identifier

If no random bits are used in the subnet identifier, then the adversary can attempt a top down attack against B whereby it obtains a packet with covering value $f_1(M_v, X_2, \ldots, X_k)$ thus revealing P_1, then a packet with covering value $f_2(M_v, P_1, X_3, \ldots, X_k)$ thus revealing P_2, and finally a packet with $f_3(M_v, P_1, P_2, X_4, \ldots, X_k)$. We will show that the success probability of such an attack is low. The remaining attacks are covered in Section 5, but since we don't include the random bits, the success probability will be higher for these other attacks.

Theorem 3. *Given the CPP adversary model (G, F, $\mathcal{A}$, s) as described in the above definition. Let $B = P_0, X_1, \ldots, X_k, M_v$ be the CPP address with location $P_1, \ldots, P_k$. Let*

$$p_1 = 1 - [(2^{|P_1|} - 1)/2^{|P_1|}]2^{-(|P_2|+\ldots+|P_k|)}$$
$$p_2 = 1 - [(2^{|P_2|} - 1)/2^{|P_2|}]2^{-(|P_3|+\ldots+|P_k|)}$$
$$p_3 = 1 - [(2^{|P_3|} - 1)/2^{|P_3|}]2^{-(|P_4|+\ldots+|P_k|)}.$$

Then the probability, up to ϵ_{prf}, that $\mathcal{A}$ obtains the prefix components P_1, P_2, and P_3 of address B is bounded by

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

where $p = max\{p_1, p_2, p_3\}$, $\alpha = (1-p_1)(1-p_2)(1-p_3)$, s is the number of subnets searched, and $x(s) = x_2 s^2 + x_1 s + x_0$ where

$$x_2 = 1/(p-1), x_1 = (3 - 2px_2)/(p-1),$$

$$x_0 = [2 - px_2 - px_1]/(p-1).$$

Remark: The basic idea behind the proof is that the adversary subnet identifier values ($Q_i \oplus G_i$) are determined by the victim's X_i address values, as well as the adversary's location. Thus it is with small probability that this overdetermined set of equations is satisfied. When the equations aren't satisfied, the adversary does not receive a packet, and therefore it learns no information about the victim's location.

Proof. We can assume that a received packet D has the same host identifier M_v as B, else the adversary cannot learn any information about B's address since we may assume equations (1) uses a random function f as in Game 2. Suppose $\mathcal{A}$ is located on the subnet with subnet identifier $Q_1, \ldots, Q_k$. Then $\mathcal{A}$'s address prefix 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 are satisfied:

$$\begin{aligned}
G_1 &= f_1(M_v, (Q_2 \oplus G_2), \ldots, (Q_k \oplus G_k)) \\
G_2 &= f_2(M_v, Q_1, (Q_3 \oplus G_3), \ldots, (Q_k \oplus G_k)) \\
G_3 &= f_3(M_v, Q_1, Q_2, (Q_4 \oplus G_4), \ldots, (Q_k \oplus G_k)) \\
&\ldots \\
G_{k-1} &= f_{k-1}(M_v, Q_1, \ldots, Q_{k-2}, (Q_k \oplus G_k)) \\
G_k &= f_k(M_v, Q_1, \ldots, Q_{k-1})
\end{aligned}$$

$G_1, \ldots, G_k$ 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 $f_1(M_v, X_2, X_3, \ldots, X_k)$. 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 $\mathcal{A}$ 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 the first case, $\mathcal{A}$ will not receive a response packet, since we require that $\mathcal{A}$ 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 not equal to Q_1), the above equations (1) are satisfied with probability $1 - p_1$ (due to the random function $f = (f_1, \ldots, f_k)$. But $\mathcal{A}$ learns only $f_1(M_v, X_2, X_3, \ldots, X_k)$, 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 $f_2(M_v, P_1, X_3, X_4, \ldots, X_k)$. 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 equal to

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

This last equality can be obtained by induction on s. For the base case, let $s = 4$. Then

$$\begin{aligned} x(2)p^2 - x(0) &= (4x_2 + 2x_1 + x_0)p^2 - x_0 = x_0(p^2-1) + (4x_2 + 2x_1)p^2 \\ &= (2 - px_2 - px_1)(p+1) + (4x_2 + 2x_1)p^2 \\ &= (p^2 - p)x_1 + (3p^2 - p)x_2 + 2p + 2 \\ &= p(3 - 2px_2) + (3p^2 - p)x_2 + 2p + 2 \\ &= (p^2 - p)x_2 + 5p + 2 = 2(3p+1) \\ &= 2(\sum_{i=0}^{1} [(i+2)(i+1)/2]p^i) \end{aligned}$$

The inductive step follows from the equality $s(s-1) + x(s-2) = x(s-1)p$ which can be obtained in a similar manner.

We present an alternative proof of Theorem 3 in Appendix B.

B Alternate Proof of Theorem 3

Here we use Gosper's algorithm [10] to give an alternate proof of Theorem 3. We may apply Gosper's algorithm to compute the closed form sum of a series $\sum t_n$, where $r(n) = t_{n+1}/t_n$ is a rational function of n.

The steps of the algorithm are as follows:

1. Compute $r(n) = \frac{a(n)c(n+1)}{b(n)c(n)}$ where $a(n)$, $b(n)$, and $c(n)$ are polynomials satisfying $\gcd(a(n), b(n+h)) = 1$ for all nonnegative integers h.
2. Compute a nonzero polynomial solution $x(n)$ of $a(n)x(n+1) - b(n-1)x(n) = c(n)$ if one exists.
3. Compute $z_n = \frac{b(n-1)x(n)}{c(n)} t_n$; and $t_n = z_{n+1} - z_n$.

We apply this algorithm to the series $\sum_{i=0}^{s-3} [(i+2)(i+1)/2]p^i$ from Theorem 3.

We fix $b(n) = n + 1$, and let $c(n)$ accumulate the needed factors for equality. We have

$$r(n) = t_{n+1}/t_n = [(n+3)/(n+1)]p.$$

Then $a(n) = pn$, and $c(n) = (n+2)(n+1)n$. We now proceed to step 2.

We must find $x(n)$ such that

$$px(n+1) - x(n) = (n+2)(n+1).$$

We let $x(n) = an^2 + bn + c$. Then we can solve the polynomial equality to obtain $a = \frac{1}{p-1}$, $b = a^2(p-3)$, $c = a(2 - ap - bp)$.

Finally, we compute $z_n = \frac{b(n-1)x(n)}{c(n)} t_n = \frac{x(n)}{2} p^n$:

$$\sum_{n=0}^{s-3} t_n = \sum_{n=0}^{s-3} (z_{n+1} - z_n) = z_{s-2} - z_0 = \frac{x(s-2)p^{s-2} - x(0)}{2}$$

.

C CPP Forwarding and Graph Levels

This section presents additional details for levels in the aggregated routing graph. Each router is configured with the level key (L_i) for the aggregation level on which it is located. A router having a level key is called a level router. Access routers do not have level keys (and therefore are not level routers).

Network Topology and Level Labeling We choose one of the border routers as the root of the aggregated routing graph, and exclude the other border routers from the graph, to obtain a new graph G'. We use the Prim-Jarnik algorithm [20] to get a spanning tree T from the graph G', where the weight of the edges is the length of the minimal prefix component on the edge minus the distance from the root. (Another algorithm could also be used). 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 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 $i-1$.

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.

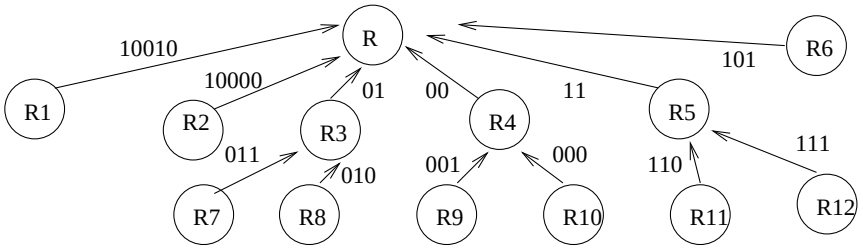
Theorem 4. *Suppose the destination first hop router D for a packet P and all other level routers (at least one router for each level including a border router) are in the same connected component. Then P is forwardable to D.*

Proof Sketch: Suppose the level i decryption occurs at router R. The next level to be decrypted is level $i-1$. So any node with level $\geq i-1$ can be forwarded to. So there is either a one hop path to such a node, or the longest prefix match is being advertised at a higher level (level $\geq i-1$) (routing around a failure point), or an aggregated prefix (shorter match) is advertised at a higher level (level $\geq i$). In the last two cases, there is a forwarding path to the longest prefix match. The only possible problem with the forwarding path is if all adjacent nodes to R have level $i-2$ or less; assume this holds in order to obtain a contradiction. But there is also a forwarding path from a border router (level d) to node R (level i). This last node prior to R on this path has level $i-2$ by our assumption. Then the sequence of levels going back to the border router can only increase one level per hop, at most. Thus there is a level i node on the path, contradicting our supposition that the level i decryption occurs at router R.

At the end of the path, the next decryption occurs, if not earlier. Thus the next level will be decrypted, as long as a level $i-1$ router is in the same connected component with R. By induction, the packet will be completely decrypted and forwarded to the destination first hop router D.

D Multiple Prefix Component Lengths Example

We give the forwarding example with multiple prefix component lengths. Figure 12 depicts the network topology with the aggregated routing graph, and gives the routing table for the node R in the graph. When additional decryptions are needed, the routing table specifies the number of additional bits to decrypt, so there is never any ambiguity regarding the number of bits to decrypt. Figure 13 gives an example with two routing table prefixes where one is a prefix of the other (see explanation in the Section 3.4 footnote).

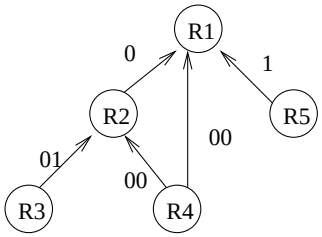


Router R's Routing Table:
Decrypt 1st 2 bits (minimum prefix component length). Match decrypted component with entry in the routing table:
01 forward to R3 (Level 1); 00 forward to R4 (Level 1); 11 forward to R5 (Level 1);
10 decrypt 1 more bit

If 2nd decrypt occurs, match the concatenation of decrypted strings with an entry in routing table: 101 forward to R6 (Level 1); 100 decrypt 2 more bits

If 3rd decrypt occurs, match the concatenation of decrypted strings with an entry in routing table:
10010 forward to R1 (Level 1); 10000 forward to R2 (Level 1)

Fig. 12. Example of Aggregated Routing Graph and Routing Table with Multiple Prefix Component Lengths



R1's Routing Table:
0 forward to R2 (Level 1)
00 forward to R4 (Level 0)
1 forward to R5 (Level 1)

Fig. 13. Example of Aggregated Routing Graph and Routing Table with Redundant Link