Can I trust my neighbours?:
Proving ownership of IPv6 addresses

Authors
Colin Putman, MSc (Royal Holloway, 2018)
Chris Mitchell, ISG, Royal Holloway

Abstract

Throughout the history of the Internet, there has been a question of how to verify that the senders of messages are who they claim to be. Operating under a false IP address opens the way to a wide variety of attacks and can be very difficult to detect. Nowadays, widespread filtering makes this impossible across network boundaries, but there is seldom any such protection within a single network, making public networks especially a difficult space to enforce.
This is likely to become a much larger problem after the migration to IPv6, as the maximum size of each network is increased dramatically over what IPv4 can support. In this article, we examine the mechanisms that have been invented to allow IPv6 users to prove their rightful ownership of an address, preventing others from using it falsely, as well as showing some of the ways in which these measures are incomplete.[a]

[a] This article is published online by Computer Weekly as part of the 2019 Royal Holloway information security thesis series `https://www.computerweekly.com/ehandbook/Proving-ownership-of-IPv6-addresses`. It is based on an MSc dissertation written as part of the MSc in Information Security at the ISG, Royal Holloway, University of London. The full thesis is published on the ISG's website at `https://www.royalholloway.ac.uk/research-and-teaching/departments-and-schools/information-security/research/explore-our-research/isg-technical-reports/`.

The migration of the world's systems to using Internet Protocol version 6 (IPv6) is one of the longest-standing initiatives in communication technologies today. This replacement for the previous version of the Internet Protocol, IPv4, was first proposed in 1998, and was ratified as an Internet Standard as recently as 2017, but still has yet to become fully ingrained into our everyday practices. The key motivating factor in IPv6's development was the shortage of IPv4 addresses, but creating a new suite of protocol standards has also given us a chance to build better security into the very backbone of the Internet.

IPv6 improves security chiefly by making it mandatory to support the use of IPsec, a well-established set of security protocols which can also be used with IPv4. IPsec is a tried-and-true security measure in IPv4, and reinforcing its use in IPv6 is a sensible approach to take. However, there is one protocol within IPv6 for which this doesn't quite work.

The Neighbor Discovery Protocol

When a computer joins a network, there are three things it has to do before it can communicate on the Internet:

1. It needs find out what other computers are on the network, and how best to send messages to each one; in particular, it needs to be able to recognise when an IP address refers to one of its neighbours on the local network, and to be able to translate those IP addresses into the local network's addressing system.
2. It also needs to know where at least one router on the network is, so that it knows where to send messages destined for the wider Internet.
3. It needs to acquire an IP address of its own, so that other computers will be able to send messages to it.

In IPv4, these functions are all handled by different protocols, but in IPv6 they have been consolidated into one protocol, called the **Neighbor Discovery Protocol (NDP)**.

The problem with securing this process comes from that third bullet point. To use IPsec, a few things are needed to set up the process of exchanging encryption keys which will be used to secure the rest of the communications. One of these things is the computer's IP address - but when a computer starts using NDP, it doesn't have an IP address yet. This leads to a chicken-and-egg scenario, where a computer needs to run NDP to get an IP address, but it needs an IP address to run NDP securely, so for at least a crucial first few messages, IPsec can offer no protection.

Unfortunately, the protocol doesn't acknowledge this vulnerability, and without IPsec, NDP treats the network as if every device on it were honest and trustworthy. This is a dangerous way to operate even now, as many networks lack any form of access control, and any network can in theory be infiltrated by a malicious party. Once you're connected to the same network as someone else, it becomes very easy to forge messages and pretend to be that other person by listing their address as the "source address" of your messages - an act known as "spoofing".

Chicken-and-egg scenario

. . . need to run NDP to get an IP address . . . but need an IP address to run NDP securely . . .

This makes a wide range of attacks possible - for example, just by forging NDP messages in this way, an attacker can convince the rest of the network to send messages addressed to a victim to the attacker instead. Since there's effectively no limit on the number of victims the attacker can target, a single malicious user can exert near total control over the flow of information in the network, letting them eavesdrop on and modify unencrypted data, selectively discard messages, or even shut down communication on the network altogether.

Introducing Secure Neighbor Discovery

So how can we defend against these attacks? What we need is a way for devices to prove ownership of an IP address, and to attach that proof to every message they send. This can be done using signatures, since having the private key needed to generate a signature effectively "proves" that you are the owner of that key pair, but to extend this proof of ownership to the IP address it has to be tied in some way to the corresponding public key.

One way we could do this is by storing the public key on the DHCP server which allocates and keeps track of the IP addresses in the first place, but here we run into another feature of IPv6: the ability to support computers choosing their own IP addresses when they join a network. This feature means that many networks will not have a DHCP server, and will be far less centralised as a result - after all, relying on a single device like a DHCP server can be a liability if an attacker disables or takes control of it! But if there is no central server that can be queried on which public keys belong to which IP addresses, we will need another way to connect them together.

This is where **Secure Neighbor Discovery (SEND)** comes onto the scene. SEND is a proposed standard that extends NDP, adding several features and requirements to improve its security. So far, only a few companies have made use of SEND, in quite a limited scope, but it is the only standard of its kind to have seen such use, and is poised to become the de facto standard for protecting NDP messages.

Among other things, SEND includes a mechanism to overcome the problem of proving ownership of an IP address in a decentralised network. Its approach is very simple: since each computer has the freedom to create its own IP address, SEND binds the computer's public key to the address by turning the public key into the address itself.

It works like this: when you want to join a network, you need to decide on an IP address to use. Specifically, you need to choose what the second half (the last 64 bits) will be, since the first half of the address identifies which network you're in. Instead of just choosing at random, when using SEND, you first generate a public key and corresponding private key. You then feed the public key, along with some

other information needed in the protocol, into a hash function, which converts all of that information into a seemingly random string of bits. The first 62 bits in this string become the second half of your IP address (two bits of the address are reserved for other purposes, so they are set to specific values regardless of the hash output). The result is a **cryptographically generated address (CGA)**.

Once this is done, you can claim the address, but you must also include the public key and other input information with each message you send. Other computers on the network can feed this information into the hash function themselves - the hash function is public knowledge and doesn't use any secret information, so anyone can use it, and the same input always results in the same output. This allows other computers to verify that your public key was used to create your address.

SEND and CGA

SEND turns a public key into a cryptographically generated address (CGA) via a hash function.

You must also digitally sign each message, proving that you know not only the public key but also the corresponding private key - no one else should have access to this private key, so this effectively proves that you are the creator and rightful owner of the address. But how secure is this proof? To answer that question, we have to understand what it means to breach that security.

Attacking CGAs

If we want to undermine a proof of ownership, what we need is a way for someone who is not the owner to generate that proof. Doing this would allow an attacker to spoof the owner's address. The proof in this case is the signature - everything else is public information used to bind the key and the address together. The only secret is the private key, so naturally if an attacker can acquire this they can forge the proof - but this would require either breaking into the owner's computer or breaking the public key scheme itself.

So is there a way to forge a proof of ownership without the private key? It turns out that there is. The weakness lies not with the signature but with the hash. While the hash function does ensure that the same input always leads to the same output, it does not ensure that any given output can only come from one input, since we're compressing thousands of bits of public key down to just 62 bits of address.

This means an attacker can find another set of parameters, including their own public key, which binds that key to the target's address and allows the attacker to use their own private key to sign messages. Other computers on the network don't store every single neighbour's public key, so they will be unaware that this claim on the address is forged, and will accept it as valid.

This is why it is important that the hash function cannot be reversed. When a secure hash function is used, the only way to find another input leading to the same output is through trial and error: continually generating new inputs and hashing them until a match is found. Our attacker will have to match 62 bits, each of which we can assume is equally likely to be a 0 or a 1, so they will have to generate on average 2^{62} different, valid hash inputs before they find a match which can be used to spoof the target IP address.

Strengthening CGAs

Unfortunately, 2^{62} is not very much by modern standards. It may be enough to protect an address for a matter of hours, particularly if the attackers do not have a great deal of resources to employ, but if a computer intends to use the same address for a long period of time it will still be vulnerable to this brute-force trial-and-error attack. Moreover, as time goes on, this attack becomes faster and cheaper, so there will come a time when this system offers almost no protection at all.

This is why SEND adds another step to address generation - one which requires a lot of computation, for the express purpose of slowing the whole process down. Everyone is required to do this, but an attacker trying to spoof an address needs to get that specific address (or one of a set of target

addresses), and will have to generate millions of billions of addresses to get it, making this slow-down much bigger.

The way it works is as follows: any computer choosing an address must generate a second hash output (which is imaginatively dubbed *hash2*) using most of the same input information as the first one, including the same public key. We then impose a special condition on *hash2*: if the output doesn't start with several zero bits in a row, it fails the condition, with the result that the address generated with the same inputs is not considered valid. The only way to pass this condition is, once again, through trial and error.

But how many zeroes do we need? This is quite a complicated question. Clearly more zeroes would mean more security, since it would take longer to find a hash input that passes the test, but if we require too many zeroes then everyone will have to go through hours of trial-and-error hashing just to join a network. That would be unacceptable.

This means we have to aim for a window in which the *hash2* test is too slow for most attackers to get through, but not too hard for an honest computer to perform. But it's not quite that simple - this window depends on how powerful the computers involved are, which is very variable among all the different devices that will be using IPv6. It's also a moving goalpost, as computers are continually growing in power, and what is a good *hash2* condition now is unlikely to be secure enough in ten years.

Since it is so difficult to find a single good answer to the question of how many zeroes, and any answer would have a limited lifetime, SEND instead specifies eight different security levels, from 0 to 7, where level 0 ignores *hash2* entirely and each level above that adds sixteen zeroes to the condition *hash2* must satisfy. To put that in perspective, that means that even creating an address at security level 4 takes longer than spoofing an address at level 0. Only a few of these security levels will be useful at any one time, but higher levels will become more practical in future years.

The tricky part to this is telling your neighbours which security level your address uses. It's not enough just to tell them - that way, an attacker could just forge the address without bothering with *hash2* and tell everyone the address uses security level 0. Likewise, we can't just add the security level to the address inputs, because it would still be possible to get that address using different inputs, including a different (lower) security level.

The only answer is to make the security level part of the address directly, overwriting the first three bits of the address. That way there's no denying that particular address uses that security level. That leaves only 59 bits for the actual hash output portion of the address, which lowers the base level of security it provides, and narrows the gap between "too hard to make" and "too easy to forge", but that's a very small effect compared to allowing us to configure how much security *hash2* gives us.

So what's the problem?

On the face of it, it may seem like SEND has solved the problem, and it has certainly taken great strides, but it has also attracted several criticisms which make it clear that the job is not done. If there's one thing to learn from the migration from IPv4 to IPv6, it's that changing these protocols is an immense undertaking, so it is vital to get them right before they become firmly established.

The key issue limiting the lifetime of SEND's anti-spoofing measures is their reliance on very specific algorithms both for the hash function and for creating the digital signatures. If we one day discover a practical attack against either of these algorithms, SEND's security will be completely broken. This is a difficult problem to address, though, since computers which use different algorithms would be unable to verify each other's security, unless there were some added method for choosing between several options.

In its current form, SEND strongly recommends using the RSA public key scheme. RSA is a venerable public key scheme but also quite a cumbersome one. In particular, it uses very large public keys, which will have to be added to every NDP message. Since frequent and regular NDP messages are needed to keep an IPv6 network functioning, keeping these messages small is an important concern. RSA is also beginning to fall out of favour as we prepare for the appearance of quantum computers, which are

expected to weaken the algorithm.

SEND also requires using SHA-1 as the hash algorithm to create addresses, as well as for *hash2*. SHA-1 is already known to be weak in some respects, but not yet in a way that affects the properties SEND needs from it. Still, this does highlight how the security of a hash algorithm might suddenly disappear when a weakness is found. Fortunately, here we do have the option of repurposing the security levels, so that some of the higher numbers refer to using a different hash function instead of more zeroes. It is entirely possible, after all, that SHA-1 would become obsolete before those higher security levels became useful anyway.

Another problem arises from the fact that SEND currently does not include the first half of the IPv6 address - the part which refers to the network, and can't be decided by the joining computer - in the inputs for the *hash2* test, even though it is included when creating the address itself. This was a deliberate decision to help mobile devices to make quick handovers from one network to another, by allowing them to perform the lengthy *hash2* calculations before they were needed. However, it has been argued that this will do more harm than good, because it similarly allows attackers to perform their *hash2* calculations without needing to know which network they would be attacking. This raises the possibility of an enormous database of valid *hash2* inputs being created, which could then be used to completely nullify the security provided by the *hash2* condition in any network in the world.

These and other small but important weaknesses show that SEND is not yet ready for wide-scale deployment. If the protocol is to stand the test of time, it will need ways of mitigating these problems. Many solutions have already been proposed, but most are narrow in scope. It will take a concerted effort to decide which of these are most effective, and how to combine them into a cohesive whole, without introducing further problems in the process.

There are a lot of things to be considered, and a consensus will need to be reached, but this is a small task compared to what was needed to reach this point. If done well, it could result in a protocol that is truly designed for the future, allowing NDP to finally reach the standard IPv6 hopes to attain as an Internet Protocol with security built right into its core.

Biographies

Colin Putman was educated at home and earned a first-class BSc in mathematics and computer science through the Open University at the age of 20, followed by an MSc in Information Security with distinction from Royal Holloway in 2018. Colin was awarded the British Computing Society's David Lindsay prize for the MSc project A Secure Framework for Protecting IPv6 Neighbor Discovery Protocol, and has since remained at Royal Holloway, enrolling in the Centre for Doctoral Training to work towards a PhD in Information Security.

Chris Mitchell received his BSc and PhD degrees in Mathematics from Westfield College, University of London in 1975 and 1979 respectively. He was appointed as Professor of Computer Science at Royal Holloway in 1990, having previously worked at Racal Comsec, Salisbury, UK (1979-85) and Hewlett-Packard Laboratories, Bristol, UK (1985-90). After joining Royal Holloway he co-founded the Information Security Group in 1990, and helped launch the MSc in Information Security in 1992. His research interests lie within information security, focusing on applications of cryptography. He is co-editor-in-chief of Designs, Codes and Cryptography, and section editor for Section D of The Computer Journal.

Series editor: S.- L. Ng