Generating Unlinkable IPv6 Addresses

Mwawi Nyirenda Kayuni, Mohammed Shafiul Alam Khan, Wanpeng Li, Chris J. Mitchell(✉), and Po-Wah Yau

Information Security Group, Royal Holloway, University of London, Egham, UK
{Mwawi.NyirendaKayuni.2011,Wanpeng.Li.2013}@live.rhul.ac.uk,
shafiulalam@gmail.com, {C.Mitchell,P.Yau}@rhul.ac.uk

Abstract. A number of approaches to the automatic generation of IPv6 addresses have been proposed with the goal of preserving the privacy of IPv6 hosts. However, existing schemes for address autoconfiguration do not adequately consider the full context in which they might be implemented, in particular the impact of low quality random number generation. This can have a fundamental impact on the privacy property of unlinkability, one of the design goals of a number of IPv6 address autoconfiguration schemes. In this paper, the potential shortcomings of previously proposed approaches to address autoconfiguration are analysed in detail, focussing on what happens when the assumption of strong randomness does not hold. Practical improvements are introduced, designed to address the identified issues by making the random generation requirements more explicit, and by incorporating measures into the schemes designed to ensure adequate randomness is used.

1 Introduction

The move from IPv4 to IPv6 brings with it a range of challenging security and privacy issues. Of course, the vastly larger address space of IPv6 is a huge advantage, allowing the use of globally unique identifiers for all Internet-connected devices. However, this very advantage brings with it possible user privacy problems [1].

That is, if each device has a long-term and globally unique identifier, then use of this identifier enables devices to be tracked. As stated in RFC 4941 [1], if part of the IPv6 address remains fixed then privacy problems arise for mobile devices, since the fixed part of the address can be used to track use of a particular device across networks.

This privacy threat has become increasingly serious with the proliferation of network-enabled personal devices, including phones and tablets. That is, tracking of IP addresses on such devices could enable the movements and activities of a single user to be recorded. This threat will become even more apparent as an increasing variety of devices become IP-enabled, particularly with the emergence of the Internet of Things (IoT).

As a result, a method is needed to enable devices to generate new unique IPv6 addresses on a regular basis with the property that pairs of addresses generated

L. Chen and S. Matsuo (Eds.): SSR 2015, LNCS 9497, pp. 185–199, 2015.
DOI: 10.1007/978-3-319-27152-1_10

by the same device are *unlinkable*. That is, given two IPv6 addresses generated by the method, it should not be possible for a third party to learn anything from the addresses themselves regarding whether or not they belong to the same or distinct devices.

Of course, there is already a substantial body of work addressing this problem, including RFC 4941 [1], discussed further in Sect. 2 below. However, as we discuss in this paper, there are serious practical problems with all the existing approaches. In essence, the existing solutions all depend on the availability of either high quality random bit streams or long-term state (or both) within the device generating its own IPv6 addresses. Meeting these requirements could be very challenging in certain classes of device, particularly those small portable platforms for which the privacy threat may well be greatest. As a result, new solutions are required which can work on a wide variety of platforms while still providing acceptable levels of address privacy.

In this paper, as well as pointing out the scale and scope of the 'randomness' problems with the prior art, we make a detailed proposal for the use of randomness in the existing address generation schemes. The solutions proposed are designed to be readily implemented on current platforms, and should enable significant improvements in the level of privacy offered by the various approaches to dynamic IP address generation.

The remainder of this paper is structured as follows. Section 2 describes previous work on IPv6 autoconfiguration, focussing on proposals for addressing the privacy issue. The limitations of previous approaches are considered in Sect. 3, which leads to Sect. 4 in which new approaches to IPv6 address autoconfiguration are explored. Section 5 summarises the main findings and recommendations, and notes possible directions for future work.

2 Background

2.1 Stateless Address Autoconfiguration (SLAAC)

An IPv6 address is a 128-bit identifier [2] for a specific network interface within a device (referred to as a *host* throughout). That is, a network interface cannot communicate in an IPv6 network unless it has one or more suitably configured IPv6 addresses. Hosts may need to automatically generate (*autoconfigure*) their own IPv6 addresses. This need is addressed by the *IPv6 Stateless Address Autoconfiguration* protocol, or *SLAAC*, specified in RFC 4862 [3]. SLAAC involves a host first generating a *global address* via stateless address autoconfiguration, and then using the *Duplicate Address Detection (DAD)* procedure to verify the local uniqueness of the global address. The mechanism 'allows a host to generate its own [global] addresses using a combination of locally available information and information advertised by routers'.

SLAAC operates in the following general way. A router advertises a 64-bit prefix that identifies the subnet to which the host is attached. The host then generates a 64-bit *interface identifier*, uniquely identifying the host on the subnet. The 128-bit IPv6 address is simply the concatenation of these two values.

The source of the interface identifier, which must be in *modified EUI-64 format* [2], will depend on the underlying link-layer protocol. In many cases, an interface's identifier will be derived directly from that interface's link-layer address [2]. For example, it may be derived from an IEEE 802 48-bit MAC layer address.

This approach is appropriate 'when a site is not particularly concerned with the exact addresses hosts use, so long as they are unique and properly routable'. SLAAC is an alternative to the *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* [4], appropriate when a site requires tighter control over exact address assignments.

2.2 Privacy Extensions to SLAAC

We first observe that, although the first 64 bits of a SLAAC-generated address will change when a host switches subnets, the last 64 bits will stay constant, since they are generated from a fixed interface identifier. This issue has motivated the development of RFC 4941 [1]. As stated in Sect. 2.3 of this RFC, problems arise if the interface identifier contained within the IPv6 address remains fixed and, in such a case, 'the interface identifier can be used to track the movement and usage of a particular machine' (this threat is, of course, particularly relevant to mobile devices). More detailed discussions of the privacy issues arising from the use of SLAAC are provided in Sect. 1 of Gont [5] and in Cooper, Gont and Thaler [6].

The goal of RFC 4941 is to describe methods for a host to automatically generate IPv6 addresses that change over time and which cannot be linked to each other, thereby giving a level of pseudonymity to a host. The focus of RFC 4941 is on the case where the interface identifier used in SLAAC is generated from a fixed IEEE MAC layer address. RFC 4941 seeks to propose new methods for address generation that minimise the changes to SLAAC, and that enable a sequence of apparently random addresses to be generated. RFC 4941 addresses are expected to be used for a 'short period of time (hours to days)' ([1], Sect. 3).

The main change to SLAAC is to replace the fixed interface identifier with a randomised value. Two approaches are described for generating such a randomised identifier.

1. Method 1 (*When stable storage is present*). As the title suggests, this approach assumes that the software responsible for generating the randomised interface identifiers has access to a means of storing changeable data long-term. More specifically, the scheme requires the storage of a 64-bit *history value* H. The scheme also requires the software to have access to a 64-bit random value which is used to initialise the history value. It is further assumed that the 64-bit fixed interface identifier I is available, e.g. as derived from the MAC layer address.

   Whenever a new randomised interface identifier is required, the following steps are performed.

(a) Compute $V = h(H||I)$, where h is the MD5 cryptographic hash function [7], and here, as throughout, || denotes concatenation of bit-strings. Hence V is a 128-bit value.
(b) Set the new history value H to be the rightmost 64 bits of V, and store this value.
(c) Let J be the leftmost 64 bits of V, after setting the 7th bit (counting from the left) to zero to indicate an address of local significance only.
(d) Compare J against a list of reserved interface identifiers and those already assigned to an address on the host (on a different network interface). If J matches a forbidden address then restart the process; otherwise use J as the randomised interface identifier.

The use of MD5 is not mandatory; that is, h could be instantiated as any other suitable cryptographic hash function with an output length of at least 128 bits (longer output lengths can be truncated).

2. Method 2 (*In the absence of stable storage*). In this case it is proposed that the interface identifier can simply be generated 'at random'. No method is specified for random generation, although it is suggested that host-specific configuration information (such as a user identity, security keys, and/or serial numbers) could be concatenated with random data and input to MD5 to generate the interface identifier.

2.3 The Gont Approach

Gont [5] notes that temporary addresses, as proposed in RFC 4941, bring difficulties. From a network management perspective, 'they tend to increase the complexity of event logging, trouble-shooting, enforcement of access controls and quality of service, etc. As a result, some organizations disable the use of temporary addresses even at the expense of reduced privacy [8]. Temporary addresses may also result in increased implementation complexity, which might not be possible or desirable in some implementations (e.g., some embedded devices)'.

As a result, Gont [5] proposes another approach to generating user-privacy-protecting interface identifiers. This scheme generates interface identifiers that are stable within a subnet, but which vary between subnets. That is, when a host migrates from one subnet to another, both the first and second 64-bit components of the IPv6 address change, preventing tracking of hosts as they migrate. As with the RFC 4941 scheme, it is intended that a generated interface identifier cannot be linked to a long-term host identifier (such as the SLAAC interface identifier).

Use of the scheme requires choice of a pseudorandom function f giving a 64-bit output that must be difficult to invert. The choice for f is not mandated, but it is suggested that it could be computed by taking the 64 least significant bits of the output of SHA-1 or SHA-256 [9]. The scheme also requires the address-generating software to have access to a host-unique secret key K (of length at least 128 bits), which is chosen at random at system installation time. It is further assumed that, as in RFC 4941, a fixed network interface identifier I is available, e.g. as derived from the MAC layer address. The scheme then operates as follows.

1. Compute $J = f(P||I||N||D||K)$, where f, I and K are as above, P is the 64-bit SLAAC prefix, e.g. as obtained from a router advertisement message, N is an identifier for the network interface for the generated identifier, and D is a counter used to resolve DAD conflicts (initialised to zero every time this process is run). Hence J is a 64-bit value.
2. Compare J against a list of reserved interface identifiers and those already assigned to an address on the host (on a different network interface). Also perform DAD. If J matches a forbidden address or DAD fails then increment D and restart the process; otherwise use J as the subnet-specific interface identifier.

Including P in the computation ensures that J is subnet-specific; similarly, including N ensures, with high probability, that different network interfaces on the same host have different values of J.

2.4 The Rafiee-Meinel Scheme

In a recent paper, Rafiee and Meinel [10] propose yet another approach to randomised interface identifier generation. They reject the Gont approach (see Sect. 2.3) on the basis that fixing the interface identifier for a given subnet is potentially privacy-compromising, since all accesses to this subnet will be trackable. They also criticise method 1 of RFC 4941 [1] on the basis that stable storage may not be available.

The Rafiee-Meinel scheme can be regarded as a specific instantiation of method 2 of RFC 4941, i.e. it is a specific method of generating randomised interface identifiers that does not make use of stable storage. It assumes that the system generating the identifier has access to the current system time T in the form of a 64-bit integer denoting the number of milliseconds since the beginning of 1970. The scheme operates as follows.

1. Generate a 128-bit random value R.
2. Compute $V = h(R||T||P)$, where T is a timestamp (as above), P is the 64-bit subnet prefix, e.g. as obtained from a router advertisement message, and h is SHA-256 [9]. V is thus a 256-bit value.
3. Let J be the leftmost 64 bits of V.
4. Perform DAD. If DAD fails then increment R and restart the process; otherwise use J as the subnet-specific interface identifier.

2.5 Other Schemes

Before proceeding we also briefly mention two other papers which describe IPv6 address generation schemes which are apparently relevant. Al'Sadeh, Rafiee and Meinel [11] and Rafiee and Meinel [12] describe modified versions of *Cryptographically Generated Addresses (CGA)* [13] designed to address the privacy problem discussed above. CGA is a method of generating 64-bit IPv6 interface

identifiers designed to enable a host that owns an identifier to prove its ownership. To use CGA, a host must generate an asymmetric signature key pair and then calculate the interface identifier as a SHA-1 hash of the public key and certain other parameters. If a third party challenges the host to prove ownership of the identifier, the host can release both the public key and a signature on a third-party-provided challenge created using the signature key.

Clearly CGA-generated interface identifiers are, by definition, random in appearance, and hence appear to address the privacy issue. Thus regular use of CGA would provide 'unlinkable' short-term IPv6 addresses. However, the generation of a key pair is a non-trivial operation, and it would seem that the only reason to adopt such an approach is if the security provided by CGA is required. Of course, improvements in the efficiency of CGA (as claimed in the two papers referred to above) are welcome, but do not change this conclusion. Thus, since the resource requirements of implementing CGA limit its applicability as a general solution, we do not consider CGA, and variants thereof, further here.

2.6 A Summary

If we ignore the CGA variants, three basic approaches have been proposed to generate privacy-protecting interface identifiers:

- RFC 4941 [1] method 1, which enables the generation of a sequence of randomised interface identifiers based on an initial random value;
- RFC 4941 method 2, including a specific instance due to Rafiee and Meinel [10], which enables the generation of a sequence of random identifiers based on 'one off' random values;
- the approach due to Gont [5] which involves the generation of fixed, but unlinkable, subnet-specific interface identifiers.

3 Practical Limitations to Privacy

In practice the schemes we have described all have potential shortcomings arising from poor use of randomness. Before analysing the individual schemes we first consider the use and abuse of random values.

3.1 Use of Randomness

Perhaps the first question that springs to mind when considering the prior art is 'Why not just generate interface identifiers at random?' Indeed, the techniques we have described all, to some extent at least, require the generation of random numbers. This issue is addressed in 3.2.1 of RFC 4941 [1], where it is stated that 'In practice, however, generating truly random numbers can be tricky. Use of a history value [as in method 1] is intended to avoid the particular scenario where two nodes generate the same randomized interface identifier, both detect the situation via DAD, but then proceed to generate identical randomized

interface identifiers via the same (flawed) random number generation algorithm. The above algorithm avoids this problem by having the interface identifier (which will often be globally unique) used in the calculation that generates subsequent randomized interface identifiers'.

That is, the authors of the RFC were very well aware of the difficulties of generating random values, and the possibility that, in practice, a flawed random number generator might be used. Examining the various proposals in more detail, it is clear that in no case are precise instructions provided covering how to generate the necessary random values.

- The specifications of the two methods in RFC 4941 simply contain pointers to RFC 4086 [14] for guidance on how to generate random values. RFC 4086 certainly contains much excellent advice, but does not contain a specific proposal for a random number generator.
- Exactly the same situation holds for Gont [5], who simply refers to RFC 4086 for advice on generating random values.
- Rafiee and Meinel [10] do not address the issue of randomness generation at all.

In the absence of very clear and specific instructions on how random numbers must be generated, or at least a reference to such instructions, there is a great danger that implementers will choose simple, but ineffective, methods for 'random' number generation. Certainly, past experience suggests that implementers cannot be relied upon to make good security decisions, particularly when called upon to generate random values. Examples demonstrating this include the following.

- After conducting a large scale survey of RSA public keys, Lenstra et al. [15] showed that a small but significant proportion offered no security whatever; specifically, 12720 of 4.7 million sampled RSA moduli had a single large prime factor in common. Moreover, 'of 6.4 million distinct RSA moduli, 71052 (1.1 %) occur more than once, some of them thousands of times'. This could only occur because the RSA key generation software used by significant numbers of users makes very poor use of 'randomness'.
- Bond et al. [16] have shown that many EMV (chip and PIN) terminals have a very worrying defect. The EMV protocol requires the terminal to send an 'unpredictable number' to a payment card, which is then used to compute a response to the terminal; the terminal uses this response to authenticate the card, with the unpredictable number being a guarantee of the response's freshness. However, in practice, many terminals (including those from highly reputable manufacturers) generate this unpredictable number in a very predictable way, i.e. very little genuine randomness is involved, meaning that security vulnerabilities result.
- In fact, even when security specifications are apparently precise, implementers cannot be relied upon to implement security correctly. As part of research into the security of IPsec, Degabriele and Paterson [17] looked at six open source IPsec implementations, including those for Linux, FreeBSD and OpenSolaris.

> Their surprising, and very worrying, finding was that not one of them correctly implemented a security-critical part of the protocol. Further evidence of poor use of security specifications has been provided by two separate recent studies [18,19], which have shown that a wide range of serious vulnerabilities can be found in SSL implementations.

This experience suggests that specifications of security protocols need to be absolutely explicit about measures to be taken by implementers. Providing pointers to good advice is not enough.

As a result of the lack of clear specifications of randomness generation in all the schemes we have examined, there is a danger that the unlinkability property of addresses generated by these schemes will be compromised. We examine each of the schemes in greater detail below, following the ordering given in Sect. 2.6 above.

3.2 Privacy Goals

Before analysing the effectiveness of the various schemes, it is important to understand their privacy goals. The two methods proposed in RFC 4941 and the Rafiee-Meinel scheme all aim to provide a degree of privacy protection both within a subnet and between subnets. That is, they provide a means of generating pseudonymous addresses for devices so that no two addresses can be linked either when they are used on the same subnet or when used on different subnets. Of course the degree of privacy obtained from these approaches will depend on a range of other factors, including how long an address is used, but these are outside the scope of the discussion here — that is we focus here purely on the linkability of addresses.

The privacy goal of the other scheme we examine, namely the Gont scheme, is rather different. It proposes use of a fixed address on each subnet, and the only privacy goal is unlinkability of addresses used on different subnets.

In the remainder of this section we consider for each scheme the degree to which its privacy goals are met, and in the next section we consider how the various schemes can be improved to try to more effectively meet their goals.

3.3 RFC 4941 Method 1

The provision of privacy of this scheme clearly relies on the initial assignment of a random value to H. In RFC 4941 it is simply stated that the the initial history value should be hard to guess, and a reference to RFC 4086 is given. All the randomised interface identifiers J are derived as a function of H and I (the fixed interface identifier, e.g. derived from the MAC address).

If the initial value of H has full 64-bit entropy, i.e. it is a 64-bit truly random value, and we assume that h is one-way (and, despite its shortcomings with respect to collisions, MD5 is not known to be not one-way), then the scheme appears secure, assuming that a search of size 2^{63} is infeasible.

However, if H has much less entropy and the method of generation is known to an attacker, then the privacy properties of the scheme are at grave risk. To see why, suppose that the initial value of H has k bits of entropy ($k << 64$) and that an attacker knows how to search through the possible initial values of H in 2^k steps. Now suppose also that such an attacker is temporarily on the same subnet as the target host, and is thus able to observe both the current temporary interface identifier J and also the host's MAC address (and hence can compute I).

If we assume that the host changes addresses once a day, and that the device was initialised less than a year ago, the attacker can perform a simple search through all possible values of H, in each case generating all 365 possible temporary addresses and comparing the generated values with J. Such a search has complexity 365×2^k hash operations (and comparisons). If, for example, we suppose that $k = 32$, this means that an exhaustive search for the initial value of H can be completed in a little over 2^{40} operations. Once the initial value of H is known then all future interface identifiers for this host are simple to compute, i.e. the scheme has been broken.

This analysis makes clear that the address unlinkability property provided by of the scheme is at significant risk if anything other than a very robust method for initialising H is used. Unfortunately, as previous experience shows, this seems to be a very strong and risky assumption.

3.4 RFC 4941 Method 2 and the Rafiee-Meinel Scheme

There is not much one can say about method 2 as described in RFC 4941, except to reiterate the difficulties of generating random values. We instead turn our attention to the Rafiee-Meinel scheme as an example of an attempt to provide a specific implementation of method 2.

This approach requires the host to generate a 128-bit random value R. The correct operation of the scheme depends to a considerable extent on the quality of this value, but no guidance is provided. One is tempted to suspect that in practice this value may be taken from a pseudorandom number function provided by the development environment, which could mean that R has very little entropy. That is, if two devices both attempt to generate a temporary address on the same subnet at the same instant, then they may very well generate the same value J. If replicated across large numbers of devices this could cause significant duplicate address problems, which is precisely why RFC 4941 method 1 was proposed. Whilst this address-collision issue is not privacy-threatening, anything that threatens network connectivity is a major problem, which raises significant doubts about this approach.

3.5 The Gont Scheme

This scheme, like method 1 of RFC 4941, requires the generation of an initial random secret key K, but does not use randomness thereafter. Assuming the robustness of the function f, the security of the scheme rests completely on the

entropy in K. Gont [5] simply states that K shall not be known by the attacker, and points to RFC 4086 [14] for advice on generating random values.

If K has close to 128 bits of entropy, then the scheme appears to be secure. However, if K has much less entropy and the method of generation is known to an attacker (including knowledge of N, the identifier for the network interface used in this particular implementation), then the privacy properties of the scheme are at serious risk. Demonstrating why is rather similar to the attack on RFC 4941 method 1 given above. Suppose the value of K has k bits of entropy ($k << 128$) and that an attacker knows how to search through the possible values of K in 2^k steps. Now suppose also that such an attacker is temporarily on the same subnet as the target host, and is thus able to observe both the current temporary interface identifier J and also the host's MAC address (and hence can compute I).

Then, for each candidate value K^* for K (from a set of size 2^k) the attacker computes $V^* = f(P||I||N||0||K^*)$, which is possible since we assume that the attacker knows P, I and N. The attacker then simply compares V^* against J; if they agree then there is a high probability that $K = K^*$, i.e. the attacker has found K.

Thus the privacy property of this scheme, like RFC 4941 method 1, is at significant risk if anything other than a very robust method for initialising K is used. As discussed above, this appears to be a very risky assumption.

4 Practical Measures to Improve Randomness Generation

Our objective here is to consider ways in which the operational privacy of previous proposals could be improved, even when the host device has very limited capabilities for generating random values. We start by considering the randomness generation problem and the nature of randomness sources that might be available to an implementer. We then consider ways in which the privacy properties of RFC 4941 method 1 and the Gont scheme might be improved. We do not consider the Rafiee-Meinel scheme further here because of the issues with regard to recurring address collisions.

4.1 Generating Randomness

We start by observing that internationally standardised means of generating random bits are given in ISO/IEC 18031 [20]. The models introduced there for *random bit generators (RBGs)* are particularly relevant. The means used in RFC 4941 method 1 to generate the sequence of history values H falls into the class of *Pure Deterministic RBGs (Pure DRBGs)*. The scheme is a *pure* DRBG since entropy is only used once, to generate the initial 'seed value' H, and the method to generate subsequent values of H is purely deterministic. This contrasts with what ISO/IEC 18031 calls a *Hybrid* DRBG, in which a source of entropy is also used as part of the state update function. ISO/IEC 18031 ([20], 7.3) discusses the security advantages of such hybrid DRBGs.

Any DRBG, whether pure or hybrid, relies on a source of randomness to initialise it, and possibly to provide further input during use. We therefore briefly consider possible sources of randomness that are likely to be available to almost any platform. It is important to note that combining a number of sources of randomness, each yielding a modest number of bits of entropy, is just as effective as using a single source of larger quantities of randomness.

- We start by considering the use of timestamps from a system clock, as incorporated into the scheme of Rafiee and Meinel discussed in Sect. 2.4. Such an approach has the great advantage that almost any device will incorporate a system clock, and hence this approach is universally applicable. Moreover, if the clock has a resolution to the millisecond level, then, assuming that the precise time of sampling is not available to an attacker, use of a clock would appear to be able to yield between 10 and 20 bits of entropy.
  However, there are issues with the use of a clock value as a source of entropy. RFC 4086 [14] observes that 'One version of an operating system running on one set of hardware may actually provide, say, microsecond resolution in a clock, while a different configuration of the same system may always provide the same lower bits and only count in the upper bits at much lower resolution. This means that successive reads of the clock may produce identical values even if enough time has passed that the value should change based on the nominal clock resolution'.
  Note that this issue raises further doubts about the operation of the Rafiee-Meinel scheme, i.e. in certain implementations address collisions may be more likely than one might expect. Nonetheless, and despite the words of caution in RFC 4086, a millisecond-accurate clock would appear to be a very valuable and almost ubiquitous source of a modest number of bits of randomness (entropy).
- Memory state information, in particular the number of free (or used) bytes in long-term storage or in RAM, would appear to be a possible source for a few bits of randomness. Again, whilst the number of bits available from each sampling may be modest, this would appear to be a reliable and ubiquitous source of randomness.
- Timings and values of external events make up another source of randomness that is discussed in RFC 4086. One example might be the timings of packet arrivals. In circumstances where an 'entropy-harvesting' process is running continuously in the background, e.g. as part of a hybrid DRBG, such an approach could again be a valuable contributor of modest numbers of bits of entropy.
- Modern mobile devices are equipped with a range of sensors, any of which could be used as a source of randomness. Microphones and cameras will generate large volumes of data likely to be highly unpredictable. A GPS receiver will similarly generate hard to predict data. Even a simple motion sensor, e.g. as used to determine the screen orientation for a portable device, could generate useful material. Of course, some sensors are highly privacy-compromising and hence may not be usable by the address generation software; however others, such as motion sensors, are far less sensitive, and could be readily available.

– Of course, hardware-based non-deterministic sources of randomness, such as those built into implementations of the Trusted Platform Module (TPM) incorporated into large numbers of notebook and desktop PCs (see, for example, Gallery [21]), would be ideal, and should clearly be employed where available. However, not all devices performing address autoconfiguration will have access to such random sources, and the main purpose of this paper is to make provision for devices without a good single source of randomness.

4.2 A Simple Improvement to RFC 4941 Method 1

As we have discussed above, problems potentially arise with RFC 4941 method 1 if the 64-bit 'random value' used to initialise the history value H contains insufficient entropy. Because a pure DRBG is used, if any instance of the history value ever becomes known, then all future outputs can be determined. This is clearly undesirable.

It should be clear that adoption of a hybrid DRBG, incorporating new randomness whenever a new address is generated, would address this problem. Over the long term entropy will 'accumulate', making future address prediction impossible unless almost every address is tracked.

Such an approach is also simple to achieve. Whatever source of entropy is available to generate the initial value of H can be re-used to provide new entropy for each subsequent history value update. We therefore propose the following very simple change to the generation of the value V in step 1, namely to put:

$$V = h(H||I||R)$$

where R is a 'random' value containing new entropy. This should not significantly increase the complexity of using this method, but will ensure a sufficient level of entropy is used to generate each new address, irrespective of the randomness properties of H.

We further propose that R should be mandated to be constructed as the concatenation of:

– a timestamp accurate to the nearest millisecond (guaranteeing 10–15 bits of entropy)[1];
– (optionally, but highly recommended) the number of bytes free in short-term and/or long-term memory;
– (optionally) any other values which contain unpredictable information, notably including the outputs of any device sensors available to the DRBG.

Further items could be added to the list if they are deemed to be likely to be readily available.

[1] One possible issue with using this as a source of randomness in this context is that address updates may occur at fixed times, e.g. at the same time every day. If this is the case then the number of bits of randomness obtained is likely to be significantly reduced.

4.3 Making the Gont Scheme More Robust

A second challenge is to find ways of making the Gont scheme more robust against attacks arising from poor sources of randomness. This is more problematic, since one goal of the scheme is that the same 'randomised' interface identifier will be generated whenever the device is attached to the same subnet. This makes it highly problematic to introduce new randomness during the lifetime of the system.

The only practical solution would therefore appear to be to require the gathering of entropy over a period before generating the key value K. This would involve building an 'entropy-harvesting' hybrid DRBG, with a state of at least 128 bits. The initial state would be set using whatever sources of randomness are available. The system would then be required to be cycled through a number of iterations over a period of hours. On each iteration, additional randomness should be included as part of the state update function. At the completion of such a process, the state of the DRBG should contain a large number of bits of entropy, preventing key-guessing attacks of the type discussed in Sect. 3.5.

A question that naturally arises is in this context is 'How many iterations would be required in practice'? Of course this depends on the number of bits of entropy introduced in each iteration. As a result, one way of deciding on the number of iterations would be to require the implementer to make an estimate for the number of bits of entropy, b say, that are harvested during any one iteration. To try to ensure that the DRBG 128-bit state is 'fully randomised', a minimum of $\lceil 128/b \rceil$ entropy-harvesting iterations will be required.

During the initial period while the key K is being generated, the fixed IPv6 address provided by SLAAC could be used by the device. After all, the main privacy threat arises from use of a single address over a long period of time and across multiple networks. As a result, use of the fixed address for a day or two is unlikely to pose a significant threat.

5 Summary and Conclusions

We have examined three proposed methods for 'randomised' IPv6 address autoconfiguration. Significant shortcomings have been identified in all three of these methods. Two of them do not adequately protect user privacy if only weak sources of randomness are available. The other approach appears likely to give problems with address collisions, at least in some operational environments.

Modifications to two of the three methods have been proposed which are designed to mitigate the threats arising from implementations of systems on devices without hardware RBGs. These modifications have been deliberately designed to involve only minor changes, and should not significantly increase implementation complexity. It would therefore appear reasonable to explore ways of modifying RFC 4941 and the Gont internet draft to incorporate the simple modifications proposed.

Possible future work would include looking at real-life implementations of the schemes we have examined in this paper. It would be particularly interesting to

test the degree of entropy actually being deployed in a range of devices implementing RFC 4941. In some cases, e.g. on smart phones or PCs, implementers may choose to use the random number generation facilities provided by the operating system, in which case the robustness of the solution will very much depend on the quality of the provided random numbers. However, the situation may be very different for small, low-power devices. Finally, it would also appear to be worth building prototype implementations of the proposed modified schemes, to test their randomness properties in practice.

References

1. Narten, T., Draves, R., Krishnan, S.: Privacy extensions for stateless address autoconfiguration in IPv6. RFC 4941, Internet Engineering Task Force (2007)
2. Hinden, R., Deering, S.: IP version 6 addressing architecture. RFC 4291, Internet Engineering Task Force (2006)
3. Thomson, S., Narten, T., Jinmei, T.: IPv6 stateless address autoconfiguration. RFC 4862, Internet Engineering Task Force (2007)
4. Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., Carney, M.: Dynamic host configuration protocol for IPv6 (DHCPv6). RFC 3315, Internet Engineering Task Force (2003)
5. Gont, F.: A method for generating semantically opaque interface identifiers with IPv6 Stateless address autoconfiguration (SLAAC). Internet Engineering Task Force, Internet draft-ietf-6man-stable-privacy-addresses-17 (2014)
6. Cooper, A., Gont, F., Thaler, D.: Privacy considerations for IPv6 address generation mechanisms. Internet Engineering Task Force, Internet draft-ietf-6man-ipv6-address-generation-privacy-01 (2014)
7. Rivest, R.L.: The MD5 message-digest algorithm. RFC 1321, Internet Engineering Task Force (1992)
8. Broersma, R.: IPv6 everywhere: living with a fully IPv6-enabled environment. Presentation at the Australian IPv6 Summit 2010, Melbourne, Australia (2010)
9. International Organization for Standardization Genève, Switzerland: ISO/IEC 10118–3, Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. 3rd edn. (2004)
10. Rafiee, H., Meinel, C.: Privacy and security in IPv6 networks: challenges and possible solutions. In: Elci, A., Gaur, M.S., Orgun, M.A., Makarevich, O.B. (eds.) The 6th International Conference on Security of Information and Networks, SIN 2013, 26–28 November 2013, Aksaray, Turkey, pp. 218–224. ACM (2013)
11. AlSa'deh, A., Rafiee, H., Meinel, C.: IPv6 stateless address autoconfiguration: balancing between security, privacy and usability. In: Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Miri, A., Tawbi, N. (eds.) FPS 2012. LNCS, vol. 7743, pp. 149–161. Springer, Heidelberg (2013)
12. Rafiee, H., Meinel, C.: SSAS: a simple secure addressing scheme for IPv6 autoconfiguration. In: Castella-Roca, J., Domingo-Ferrer, J., Garcia-Alfaro, J., Ghorbani, A.A., Jensen, C.D., Manjon, J.A., Onut, I.V., Stakhanova, N., Torra, V., Zhang, J. (eds.) Eleventh Annual International Conference on Privacy, Security and Trust, PST 2013, 10–12 July 2013, Tarragona, Catalonia, Spain, pp. 275–282. IEEE (2013)
13. Aura, T.: Cryptographically generated addresses (CGA). RFC 3972, Internet Engineering Task Force (2005)

14. Eastlake, D., Schiller, J., Crocker, S.: Randomness requirements for security. RFC 4086, Internet Engineering Task Force (2005)
15. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Ron was wrong, Whit is right. Cryptology ePrint Archive: Report 2012/62 (2012)
16. Bond, M., Choudary, O., Murdoch, S.J., Skorobogatov, S., Anderson, R.: Chip and Skim: cloning EMV cards with the pre-play attack (2012). arXiv:1209.2531 [cs.CY]
17. Degabriele, J.P., Paterson, K.G.: Attacking the IPsec standards in encryption-only configurations. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy (S&P 2007), 20–23 May 2007, Oakland, California, USA, pp. 335–349. IEEE Computer Society Press, Los Alamitos (2007)
18. Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., Freisleben, B.: Why Eve and Mallory love Android: an analysis of Android SSL (in)security. In: Yu, T., Danezis, G., Gligor, V.D., (eds.) ACM Conference on Computer and Communications Security, CCS 2012, 16–18 October 2012, Raleigh, NC, USA, pp. 50–61. ACM (2012)
19. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM Conference on Computer and Communications Security, CCS 2012, 16–18 October 2012, Raleigh, NC, USA, pp. 38–49. ACM (2012)
20. International Organization for Standardization Genève, Switzerland: ISO/IEC 18031:2011, Information technology – Security techniques – Encryption algorithms – Random bit generation. 2nd edn. (2011)
21. Gallery, E.: An overview of trusted computing technology. In: Mitchell, C.J. (ed.) Trusted Computing, pp. 29–114. IEE Press, London (2005)