Published as: A. Alsa'deh, H. Rafiee and C. Meinel. "IPv6 Stateless Address Autoconfiguration: Balancing Between Security, Privacy and Usability". 5th International Symposium on Foundations & Practice of Security (FPS). October 25 – 26, 2012 Montreal, QC, Canada.

IPv6 Stateless Address Autoconfiguration: Balancing Between Security, Privacy and Usability

Ahmad AlSa'deh, Hosnieh Rafiee, and Christoph Meinel

Hasso-Plattner-Institute at University of Potsdam,
Potsdam, Germany
{ahmad.alsadeh,hosnieh.rafiee,christoph.meinel}@hpi.uni-potsdam.de

Abstract. Included in the IPv6 suite is a method for devices to automatically configure their own addresses in a secure manner. This technique is called Cryptographically Generated Addresses (CGAs). CGA provides the ownership proof necessary for an IPv6 address without relying on any trust authority. However, the CGAs computation is very high, especially for a high security level defined by the security parameter (Sec). Therefore, the high cost of address generation may keep hosts that use a high Sec values from changing their addresses on a frequent basis. This results in hosts still being susceptible to privacy related attacks. This paper proposes modifications to the standard CGA to make it more applicable security approach while protecting user privacy. We make CGA more privacy-conscious by changing addresses over time which protects users from being tracked. We propose to reduce the CGA granularity of the security level from 16 to 8. We believe that an 8 granularity is more feasible for use in most applications and scenarios. These extensions to the standard CGA are implemented and evaluated.

Keywords: IPv6 Security, IPv6 Address Autoconfiguration, Users' Privacy

1 Introduction

IPv6 comes with many enhancements to IPv4. One major enhancement is the stateless autoconfiguration feature, which allows a node to self-determine its own address without the need of a Dynamic Host Configuration Protocol (DHCP) server. The autoconfiguration feature enables nodes to directly connect to the network. The host uses the information advertised by the router and its interface identifier (IID) information to construct its address.

The most well-known way of setting the IID is based on Neighbor Discovery (ND) [1] and Stateless Address Autoconfiguration (SLAAC) [2]. SLAAC embeds a network devices Ethernet Media Access Control (MAC) address into an IPv6 address. Since every MAC address is unique, IPv6 could allow devices to be globally uniquely identified. Unfortunately this uniqueness property can allow for the tracking of an individuals device thus violating the users privacy. In

practice, a lot of devices (e.g., laptops, cell phones, etc.) are associated with individual users.

IPv6 *privacy extensions* [3] help to protect users from being tracked. This technique is used to assign temporary IPv6 address values that change over time. Changing the addresses over time makes it more difficult for eavesdroppers to track nodes as they roam between networks. It is also harder to make a correlation between addresses when different addresses are used for different activities corresponding to the same node. Although the *privacy extensions* protect the users' privacy, they cannot prevent attacks related to IP spoofing.

Besides dealing with the privacy issues, ND and SLAAC are also directed at malicious operators. RFC 3756 [4] describes possible attacks against ND. The SEcure Neighbor Discovery (SEND) [5] was developed to counteract the vulnerabilities in ND and SLAAC.

SEND is mainly dependent on Cryptographically Generated Addresses (CGAs) [6]. CGA is a technique that offers the authentication of IPv6 addresses without the need of a third party or additional security infrastructure. In CGA addresses, the IIDs are generated by one-way hashing of the nodes public key and other auxiliary parameters such as *Modifier, Subnet Prefix* and *Collision Count*. Thus the IPv6 node address is bound to its public key.

The high computational cost is the main disadvantage for using CGA. It is likely that once a host generates an acceptable CGA, it will continue to use this address. This results in hosts using CGAs still being susceptible to privacy related attacks. It is therefore important to find a way to balance between security, privacy and usability issues of CGA before the wide deployment of IPv6 is undertaken.

IPv6 *privacy extensions* provide the privacy protection necessary for nodes in IPv6 networks but it cannot secure the addresses. While the CGA can prevent address theft related attacks, it is computationally heavy and might be vulnerable to privacy related attacks. Therefore, it is important to find an in between approach which offers security and protection of the users' privacy.

In this paper we define a mechanism that eliminates the security and privacy concerns in IPv6 SLAAC. To protect the users' privacy, we present a modified CGA implementation that integrates the *privacy extensions* approach into CGA. In this way, CGA will prevent IPv6 address spoofing related attacks while changing addresses over time will protect the users' privacy. We also propose to reduce the security level CGA granularity from 16 to 8, to better increase the chance of having a better security level (Sec) and avoiding the large step between the successive Sec values. We also modify the CGA implementation to allow for the generation of the public key pair on-the-fly to increase the randomness of CGA addresses and to enhance the users' privacy protection.

The paper is organized as follows. In Section 2, we briefly introduce the IPv6 Neighbor Discovery Protocol (NDP) and discuss its security and the privacy implications. In Section 3, we discuss the related work to solve the NDP privacy and security issues. In Section 4, we provide the details of the modifications we propose for CGA to achieve the security and privacy in feasible way. In section

5, we evaluate the implementation and usage of the modified CGA and discus the compatibility concerns and deployment limitations. In Section 6, we present our conclusions.

2 Neighbor Discovery Protocol (NDP)

ND for IPv6 [1] and IPv6 SLAAC [2] together are referred to as NDP. NDP is one of the main protocols in the IPv6 suite. It is heavily used for several critical functions, such as discovering other existing nodes on the same link, determining others link layer addresses, detecting duplicate addresses, finding routers and maintaining reachability information about paths to active neighbors.

2.1 Stateless Address Autoconfiguration (SLAAC)

In IPv6 SLAAC, the node creates the rightmost 64 bits (IID), which identifies an individual node within a local network. The IID is often configured from the Extended Unique Identifier (EUI-64) that is generated based on the interface hardware identifier - usually the MAC address of the network card. Afterwards, the node combines the subnet prefix with the IID to form a complete 128 bits IPv6 address. The subnet prefix can be the reserved local link prefix used to generate local link addresses or the prefix which is advertised by the router through the Router Advertisement (RA) message. Finally, the Duplicate Address Detection (DAD) algorithm is run by the node to ensure that there is no address conflict on the same link. The IID includes two bits which are reserved by the IPv6 addressing architecture for special purpose. The 7th bit from the left in the 64-bit IID is the Universal/Local bit (u bit). The 8th bit from the left is the Individual/Group (g bit).

2.2 SLAAC Privacy Implications

IPv6 SLAAC leads to serious privacy implications because IPv6 address may reveal sufficient information to identify the hardware and track the individual. Generating an IID based on the MAC address results in a static IID. This IID will remain the same same in all networks the node contacts. This means that the host MAC address is exposed to the Internet because any website a user visits will log his IP. Consequently, an attacker can use data mining techniques to correlate the users activity based on the IID. A more worrying case concerns mobile devices (e.g., cell phones, laptops, PDAs, etc.) because most of these devices are associated with individual users. An attacker can use the IID to track the movement and the usage of a particular device. Once the location and the identity of the user are determined, an attacker can target the user for identity theft or other related crimes [7].

2.3 NDP Vulnerabilities

It is easy to perform attacks against NDP because it has no authentication mechanism. For instance, it is difficult for a node to distinguish between a fake and the authorized routers' advertisements. The newly connected node cannot validate the routers before having an IP address. Thus, a malicious node can send a fake RA to perform Denial-of-Service (DoS) or Man-in-the-Middle (MitM) attacks and effectively receive, drop, or replay the packets. An attacker can also generate DoS on DAD to prevent a node from obtaining a network address. A malicious node may block the legitimate node from getting a new IPv6 address by always responding to every DAD attempt with the spoofed message that "I have this address". The victim would thus find out that every IPv6 address that it tried to use was being used by other nodes. It would therefore be unable to obtain an IP address to access the network. RFC 3756 [4] describes the possible attacks against NDP.

3 Approaches to Mitigate NDP Privacy and Security Implications

3.1 Privacy Extensions for SLAAC in IPv6

IPv6 *privacy extensions* [3] describe a technique for assigning temporary IPv6 addresses that change over time. Changing the addresses over time makes it more difficult for eavesdroppers and other information collectors to correlate IP address with host (user) when different addresses used for different activity correspond to the same host. The random IID is generated via a hash function using a quantity which forces randomization of the IID. A node can use different IIDs with different prefixes to have a set of global addresses that cannot be easily linked to each other. The temporary address would be used for a certain period of time and then would be deprecated.

Although the *privacy extensions* can protect a users' privacy it cannot prevent IP spoofing related attacks. The privacy extensions have no authentication mechanism with which to enable the receiver to verify the identity of the sender. Therefore, an attacker can usurp other users' addresses to carry out a wide variety of attacks. Fortunately another approach which is called CGA [6] can provide the authentication needed to prevent address theft in the IPv6 environment.

3.2 Cryptographically Generated Addresses (CGAs)

CGA Generation Algorithm. In CGA, the IID portion of IPv6 address is created from a cryptographic hash of the address owner's public key and other auxiliary parameters - *Modifier*, *Collision Count* and *Subnet Prefix*. The address owner computes two hash values (Hash1 and Hash2). The combination of the two hash values increases the computational complexity for the attacker to do the brute-force search attack. Since the 64-bit are not enough to provide sufficient security against brute-force attacks in the foreseeable future, the standard CGA

uses the *Hash Extension* (Hash2) to increase the security strength above 64-bit. The computational complexity of Hash2 depends on the Sec value. Sec is an unsigned 3-bit integer having a value between 0 and 7 which indicates the security level of the generated address.

Each CGA is associated with a CGA parameters data structure, which contains the following fields:

- *Modifier* (128 bits): it is initialized to a random value.
- *Subnet Prefix*(64 bits): it is set to the routing prefix value advertised by the router at the local subnet.
- *Collision Count* (8 bits): it is the result of a collision counter used for DAD algorithm to ensure the uniqueness of the generated address.
- *Public Key* (variable length): it is set to the DER-encoded public key of the address owner.
- *Extension Field* has a variable length for future needs.

Fig. 1 shows a schematic of the CGA generation algorithm. CGA generation begins with determining the address owner's public key and selecting the proper Sec value. The Hash2 computation loop then continues until finding the final *Modifier*. The Hash2 value is a hash of the combination of the *Modifier* and the *Public Key* which are concatenated with a zero-value for the *Subnet Prefix* and the *Collision Count*. The address generator tries different values of the *Modifier* until $16\times Sec$-leftmost bits of Hash2 become zero. Once a match is found, the loop for the Hash2 computation terminates. Then the final *Modifier* value is saved and used as an input for the Hash1 computation. The Hash1 value is a hash of the combination of the whole CGA parameters. The IID is then derived from Hash1. The Sec value is encoded into the three leftmost bits of the IID. Finally, the DAD algorithm is run by the client to ensure that the address is unique within the same subnet. If an address collision occurs, increment the *Collision Count* and compute Hash1 again to get the IID. However, after three collisions, CGA algorithm stops and reports an error.

To assert the ownership of the address and to protect the message, the address owner uses the private key that corresponds to the *Public Key* in the CGA parameters to sign messages sent from that address. Finally, the node will send the message, the CGA parameters, and the signature.

CGA verification takes as input an IPv6 address and CGA parameters. If the verification succeeds, the verifier knows that the public key belongs to that address. Then, the verifier uses the public key to authenticate the signed messages from the address owner.

The CGA algorithm increases the computational cost for both the attacker and the address generator (owner). The address generator needs $O(2^{16\times Sec})$ brute-force search to satisfy the Hash2 condition and for finding the final *Modifier*. The attacker needs to do a brute-force attack against an$(16\times Sec + 59)$-bit hash value which costs $O(2^{16\times Sec+59})$. Fulfilling the condition of Hash2 is the computationally expensive part of the CGA generation. Selecting a high Sec value may cause unacceptable delay in address generation. Even there is a probabilistic guarantee that the CGA address generation will stop after a certain

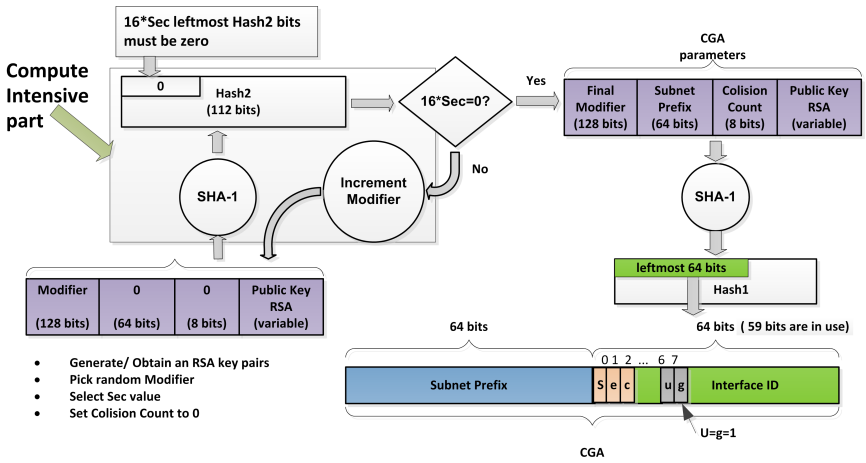

Fig. 1. CGA Generation Algorithm

number of iterations, but it is impossible to tell exactly how long it will take for the CGA generation when Sec is not zero.

CGA Privacy Concerns. With CGA, the *Modifier* is used to enhance the privacy by adding randomness to the address. Changing the *Modifier* over time leads to different IIDs. Therefore, CGA can provide IPv6 addresses with the privacy they need.

However, there are two apparent limitations to this privacy protection. First, hosts that use a high Sec value may choose not to change their addresses frequently. Due to the high computational complexity of generating Hash2, it is likely that once a host generates an acceptable CGA it will continue to use this fixed IID in multiple activities thus reducing its need for frequent regeneration - at least for that subnet. The result is that hosts using CGAs are still susceptible to privacy related attacks. Second, the *Public Key* of address owner is attached with message that is sent to the receiver. This means that the node can still be identified by its public key.

Therefore, the CGA has a privacy implication (especially for high Sec value) and the *privacy extensions* approach is vulnerable to the address spoofing related attacks. In the next section we will integrate the two approaches in a balanced way to attain both the security and the privacy in a usable method.

4 Modifications to Standard CGA

We propose three main modifications to the standard CGA process to attain the users' privacy in a practical manner. First, we modify the CGA to have a lifetime that indicates how long the address is bound to an interface. Second, we reduce the granularity of CGA security levels to get more practical security

levels. Third, we generate the keys, on-the-fly, using CGA code to ensure more security and privacy for the users of CGA addresses.

4.1 Setting a Lifetime for Temporary CGA Addresses

We propose to change the CGA addresses periodically to protect the users privacy. Each CGA address has an associated lifetime that indicates how long the address is bound to an interface. Once the lifetime expires, the CGA address is deprecated. While a CGA address is in a deprecated state, its use is discouraged, but not strictly forbidden. New communication (e.g., the opening of a new TCP connection) should use a new CGA address when possible. A deprecated address should be used only by applications that have been using it and would have difficulty switching to another address without a service disruption. When the lifetime expires and the address is not used by an opened connection, the CGA address is removed from the network interface by the kernel and no longer used.

The lifetime of a temporary CGA address depends on several parameters and actions. For instance, the lifetime should depend on the time needed for a host to generate a new CGA address, the time needed for an attacker to break the CGA address and user desired setting for security and privacy. The following lists the conditions under which a new temporary CGA address should be generated:

- When a host joins new subnet. In this case, the new CGA parameters will be used to generate the new address. A new public key will be used for calculating both the Hash1 and Hash2 values. In the standard CGA it is not necessary to use new CGA when the node moves to new subnet.
- Before the lifetime for the in-use CGA address has expired. To ensure that the CGA address is always available and valid, new CGAs should be regenerated in advance before the predecessor will be deprecated. In practice, a valid lifetime should not be zero. Using the standard *privacy extensions*, the default interval is 24 hours; however, we recommend a minimum lifetime of one hour.
- When the subnet prefix lifetime has expired. A new CGA address will then need to be regenerated. It must include the newly received prefix used in calculating Hash1.
- When the user needs to override the default value of the lifetime in order to generate a new CGA address. The CGA implementation should offer the user the ability to override the current lifetime values and force the CGA algorithm to generate a new address.

Determining the proper lifetime for a CGA address depends on the privacy and security level constraints. For the security level analysis we refer to the security model which has been proposed by Bos et al. [8] for studying the security and efficiency of the CGA. The necessary notation used in the CGA time analysis is defined as follows:

- T_G: The average time needed for a node to generate a CGA.

- T_A: The average time needed for an attacker to impersonate an address.
- T_1: The time needed to compute Hash1.
- T_2: The time needed to compute Hash2.
- b: The number of available bits in the address, which is the truncated output of Hash1 (IID).
- g: The granularity of the security level in CGA.
- s: The number of bits needed to satisfy the Hash2 condition (s=g×Sec), which is the truncated output of Hash2.

The address generator needs on average $(2^s \times T_2)$ in order to fulfill the Hash2 condition, plus T_1 to generate the IID from Hash1. Therefore, the cost of address generation, T_G, is:

$$T_G = (2^{g \times Sec} \times T_2) + T_1 \tag{1}$$

An attacker has two ways to impersonate a node - by satisfying the constraints on Hash1 and then the conditions on Hash2 or vice versa. Beginning with Hash1, the attacker must first perform the attack on Hash1, which takes $(2^b \times T_1)$ hash function evaluations. Once fulfilled, the conditions on Hash2 for the generated *Modifier* should be satisfied, which takes 2^s hash function evaluations. Thus, the total time for impersonation when beginning with Hash1 $(H_1) becomes T_A : H_1 = (2^b \times T_1 + T_2)2^s$.

When the attacker starts from Hash2, the conditions on Hash2 are met at a cost of $(2^s \times T_2)$ hash function evaluations. Next, Hash1 is verified if it matches the target address. Hash1 verification costs 2^b. Therefore, the total cost when beginning with Hash2 $(H_2) becomes T_A : H_2 = (2^s \times T_2 + T_1)2^b$.

The attacker can choose between the two ways to minimize his attack cost. Hence, the time for impersonation an address (T_A) is:

$$T_A = min\left\{(2^{59} \times T_1 + T_2)2^{g \times Sec}, (2^{g \times Sec} \times T_2 + T_1)2^{59}\right\} \tag{2}$$

The resistance of CGA against impersonation is mainly controlled by increasing the number of bits on the Hash2 condition s=g×Sec. For the standard CGA, with g=*16* and Sec value between 0 and 7, the number of operations required for impersonation on a specific node is:

$$T_A = \begin{cases} 2^{59} \times T_1 & \text{if } Sec = 0, \\ (2^{59} \times T_1 + T_2)2^{16 \times Sec} & \text{if } 1 \leq Sec \leq 3, \\ (2^{16 \times Sec} \times T_2 + T_1)2^{59} & \text{if } 4 \leq Sec \leq 7, \end{cases} \tag{3}$$

In next subsection, we propose to reduce the granularity of CGA for practical and usability reasons. When the granularity, g, is ≤ 8 the cost of address generation T_G becomes:

$$T_G = \Big\{(2^{8 \times Sec} \times T_2) + T_1 \quad \text{if } 0 \leq Sec \leq 7 \tag{4}$$

And the number of operations required for the impersonation of a specific node becomes:

$$T_A = \begin{cases} 2^{59} \times T_1 & \text{if } Sec = 0, \\ (2^{59} \times T_1 + T_2)2^{8 \times Sec} & \text{if } 1 \leq Sec \leq 7. \end{cases} \quad (5)$$

We can surmise from equation 5, that it would be easier for the attacker to start by calculating Hash1 then fulfilling the Hash2 condition. Here, the assumption is that the hash function has no known weaknesses.

The lifetime of a CGA address (T_l) should be safe enough so the attacker is not able to impersonate the other nodes' addresses. We recommend that T_A be at least nT_l (where n is an integer) in order to have a safe margin. Clearly, the speed of hash function computation depends on the CPU speed of the computing device. Reading the CPU speed by using the CGA code makes it possible to determine whether or not the selected lifetime is suitable. On the other hand, the T_l time should be greater than the time required for the node to generate a CGA address. It is not feasible to invest the time and resources of the computing device to create an address and then, after a very short period of time, deprecate this address. We recommend that T_l be greater than mT_G (where m is an integer). Therefore, T_l can be described by the following equation:

$$mT_G \leq T_l \leq \frac{T_A}{n} \quad (6)$$

Where m and n are integers.

4.2 Reducing the Granularity of CGA Security Levels

In the CGA generation algorithm, the granularity factor 16 is relatively large. The multiplier 16 was chosen to increase the maximum length of the *Hash Extension* [6] up to 112 bits, but the benefit of this is questionable [9]. Currently, Sec value 0 or 1 can be used in practice. For Sec value 2, the CGA address generation process may take several hours or days. We carried out a test on a set of 5 samples using 2.67 GHz CPU speed which gave us an average CGA generation time of 5923857 Milliseconds (1 hour and 39 minutes). The CGA computation for a Sec value of 3 will take, on average, more than 12 years on a 2.67 GHz CPU.

Smaller granularity is more suitable for CGA computations. Therefore, we proposed to reduce the granularity factor from 16 to 8 for the following reasons:

- The granularity factor 16 is quite large and causes a big jump in CGA computation time for successive Sec values. Having values in between is better than waiting for a very long time to reach the second security level (Sec+1). A smaller granularity factor gives the users the opportunity to have better security level particularly if the user is not willing to wait a long time for the CGA generation. Having a Sec value of 1 with a granularity factor of 8 is better than a Sec value of 0 with a granularity factor of 16.
- Changing the CGA addresses over time in order to protect the users' privacy makes it unnecessary to select a high security level. It does not make sense

for the address owner to select a high Sec value that is expensive in time and CPU cycles if the address will be changed after a short period of time. For instance, it is not reasonable to select a high Sec value which costs the address owner several days if the address will be changed every one hour due to the privacy need. However, the security level should be sufficient to cover a lifetime period. For example, if the lifetime is one day, the security level should be safe enough so that the attacker cannot break the address within several days.
- The privacy concerns are usually much more important for mobile devices. The mobile devices generally have limited resources (battery, memory, and processing power). For a high Sec value, the CGA computation will take too long a time and will consume too much of the computing device's energy. Smaller granularity is more suitable for these devices.
- The multiplication factor of 8 increases the maximum length of the *Hash Extension* up to 56 bits. Therefore, the total hash length will be between 59 and 115 bits, which are adequate for current CPU speeds. Decrementing the granularity to 4 or 2 might lead to weaker hash values which leave small margin of safety. With granularity 4, the total hash length will be between 59 and 87 bits.

4.3 Automatic Key Pair Generation

We propose to generate the key pairs automatically by using the CGA code as proposed in [10]. The default key size is 1024 bits. This can be changed by the use of the CGA parameter setting interface. Setting the keys automatically is better for the following reasons:

- Generating the key pair on-the-fly each time the host needs a new address enhances the CGA security and protects the user's privacy. The automatic generation of the public key increases the randomness of the CGA and consequently enhances its security against a brute-force attack. Moreover, each time the node moves to a new location, it will get a new CGA address and will use a new public key. Therefore, it will not be easy for attackers to track users based on their addresses or even to correlate traffic to their public keys.
- Minimizing the amount of required configurations so that the end user does not need to know the technical details behind the cryptography. There is also no need to use an external program to generate the key pairs. It is not easy for the user to generate key pairs manually each time the host wants to generate a new address. It becomes more tedious when the CGA address changes frequently due to privacy time constrains.
- Keys are not stored in a particular path before starting the CGA computation. The keys are therefore not vulnerable to theft. In our CGA implementation, the key pairs are stored in computer memory (RAM) for quick accessibility, but also the digested keys are stored in an XML file for further usage, such as after rebooting the system while the IP address is still valid or the host is connected to the same subnet.

– The average time to generate a key pair with a RSA 1024-bit key using 1000 samples is 27.8 Milliseconds, while the average time for Standard CGA generation with Sec value 1 and a 1024-bit key size is 439.6 Milliseconds. So, the key generation takes about 6.3% of the total CGA generation time. We took these measurements on a computer with 2.67 GHz.

5 Modified CGA Implementation and its Evaluation

5.1 Modified CGA Implementation

To test the above mentioned modifications, we modified the CGA part of SEND implementation for the Windows operating system (WinSEND)[11]. WinSEND works as a service to provide security for Windows NDP. WinSEND has the SEND functionalities and can generate IP addresses in a secure manner.

Our modified version of CGA automatically offers the default parameters to generate a temporary CGA address. The default value for the minimum lifetime is 24 hours, similar to the proposed preferred lifetime in RFC 4941, and the public key size is 1024-bit. The default value for Sec is 2 and the granularity is 8. The user can override these parameters via CGA setting (See Fig. 2).

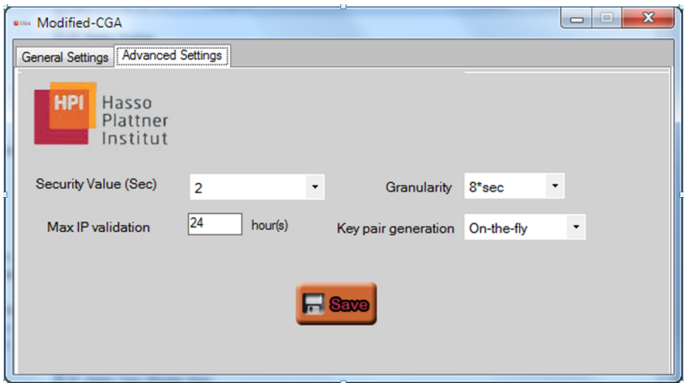

Fig. 2. Modified CGA settings parameters.

To get a rough idea about the time required for generating a CGA address for different Sec values with associated granularity factors, we used the results shown in Table 1. The CGA generation algorithm is run on a 2.67 GHz Quad core CPU computer. The results are taken over 1000 samples with a 1024-bit key size.

5.2 Limitations and Deployment Considerations

Our proposed modification to the standard CGA is compatible with addressing scheme and could be implemented as an extension to RFC 3972. The CGA-

Table 1. CGA Generation Time in Milliseconds (ms) for Different Sec Values with Different Granularity Over 1000 Samples

	Granularity		
Sec	4	8	16
1	117ms	121ms	427ms
2	128ms	425ms	5923857ms
3	135ms	88217ms	*
4	409ms	*	*

enabled nodes need to consider the granularity factor 8 in CGA generation and verification algorithms. This task is not complicated, eventually feasible modifications can be upgraded. The other modification for changing the addresses over time and generating keys on-the-fly do not affect the CGA algorithm and the way of communication. It is more implementation decisions which do not change the CGA algorithm.

There are some implications and deployment considerations for changeable addresses. Most of these limitations are also valid for the privacy extensions approach (RFC 4941) as explained below:

- The changeable address may cause unexpected difficulties with some applications. Some servers reject the connection from clients whose address cannot be mapped into a DNS name that also maps back into the same address.
- Changing the addresses frequently (e.g., every few minutes) has a performance implication and will severely impact user experience.
- Protecting the user's privacy may conflict with the administrative need to effectively maintain and debug the network.
- The implementation needs to keep track of the addresses being used by the upper layer in order to be able to remove the deprecated addresses from the internal data structure when these addresses are no longer used by the upper protocols, but not before.

Tracking users at other layers, such as tracking through DNS, cookies, or browser characteristics is out of the scope of this paper. However, in order to have privacy protection at higher-layers, we believe that the underlying protocols must also have privacy protection mechanisms.

6 Conclusion

It is very important to be sure that the increasing deployment of IPv6 will be done in a secure way without compromising the Internet users' privacy. It is proposed that CGA be used to prove the ownership of an IPv6 address and to prevent spoofing of existing IPv6 addresses, but it might be susceptible to privacy related attacks. On the other hand, the *privacy extensions* protect the users' privacy but are of no value to related address spoofing attacks. In this paper we showed how to integrate the *privacy extensions* into CGA to resolve both

privacy and security issues for IPv6 addresses. We also changed the granularity of the CGA security level and generated the public-key pair on-the-fly to make CGA more practical. We also provided a mechanism for the CGA implementation with which to automatically set the maximum lifetime and to decrease administrative tasks. This approach definitely involves tradeoffs between privacy, security, usability and the cost of address generation but it is a very viable solution.

Our proposal has several benefits over the current CGA scheme, including: (1) the ability to diminish the CGA possible privacy concerns and protect users from being tracked; (2) the ability to configure when new CGA should be created; and (3) the possibility to have finer granularity for CGA security level. We have implemented and tested the CGA modification and found that it generates new CGA address as designed while not impacting Internet activities.

References

1. Narten, T., Nordmark, E., Simpson, W., Soliman, H.: Neighbor Discovery for IP version 6 (IPv6). RFC 4861, Internet Engineering Task Force (September 2007)
2. Thomson, S., Narten, T., Jinmei, T.: IPv6 Stateless Address Autoconfiguration. RFC 4862, Internet Engineering Task Force (September 2007)
3. Narten, T., Draves, R., Krishnan, S.: Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941, Internet Engineering Task Force (September 2007)
4. Nikander, P., Kempf, J., Nordmark, E.: IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756 (Informational), Internet Engineering Task Force (May 2004)
5. Arkko, J., Kempf, Ed., J., Zill, B., Nikander, P.: SEcure Neighbor Discovery (SEND). RFC 3971, Internet Engineering Task Force (March 2005)
6. Aura, T.: Cryptographically Generated Addresses (CGA). RFC 3972, Internet Engineering Task Force (March 2005), updated by RFCs 4581, 4982.
7. Groat, S., Dunlop, M., Marchany, R., Tront, J.,: The privacy implications of stateless IPv6 addressing. In Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW 10, pp. 52:1-52:4, New York, NY, USA, ACM (2010)
8. Bos, J.W., Ozen, O., Hubaux, J-P.: Analysis and Optimization of Cryptographically Generated Addresses. Information Security 2009, pp. 17-32, (LNCS 5735) (2009)
9. Alsa'deh, A., Rafiee,H., Meinel, C.: Stopping Time Condition for Practical IPv6 Cryptographically Generated Addresses. In 2012 International Conference on Information Networking (ICOIN), pp. 257-262, (2012)
10. Rafiee, H., Alsa'deh, A., Meinel, C.: Multicore-based Auto-scaling SEcure Neighbor Discovery for Windows Operating Systems. In 2012 International Conference on Information Networking (ICOIN) pp. 269-274, (2012)
11. Rafiee, H., AlSa'deh, A., and Meinel, Ch.: WinSEND: Windows SEcure Neighbor Dis-covery. 4th International Conference on Security of Information and Networks (SIN 2011), 14-19 November 2011, Sydney, Australia, pp.: 243-246, ACM (November 2011)