CS-CGA: Compact and more secure CGA

Article · December 2011

DOI: 10.1109/ICON.2011.6168492

3 authors, including:

Ahmad Alsadeh
Birzeit University
12 PUBLICATIONS **57** CITATIONS

SEE PROFILE

Published as: Ahmad AlSa'deh, Feng Cheng, Christoph Meinel, CS-CGA: Compact and more Secure CGA, in Proceedings of the 17th IEEE International Conference on Networks (ICON'11), IEEE Press, Singapore, December 14-16, 2011.

CS-CGA: Compact and more Secure CGA

Ahmad AlSa'deh, Feng Cheng, Christoph Meinel
Hasso-Plattner-Institut, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
{Ahmad.Alsadeh, Feng.Cheng, Christoph.Meinel}@hpi.uni-potsdam.de

***Abstract*— Cryptographically Generated Address (CGA) is one of the most novel security features introduced in IPv6 suite. CGA is designed to prevent addresses theft without relying on trust authority or additional security infrastructures. However, CGA is relatively computationally intensive, and bandwidth consuming. Besides, it has some security limitations. This paper defines a Compact and more Secure CGA (CS-CGA) version. We adopt Elliptic Curve Cryptograph (ECC) keys in CGA instead of standardized RSA keys in order to minimize the size of CGA parameters and reduce CGA generation time. To enhance the security of CGA against the global time-memory trade-off attack, the subnet prefix is included in Hash2 calculations of CGA generation algorithm. For the signature and the key calculations, SHA-256 is used instead of SHA-1, which is known to have security flaws.**

Keywords- Neighbor Discovery Protocol (NDP); Cryptographically Generated Addresses (CGAs); SEcure Neighbor Discovery Protocol (SEND); IPv6 security; IPv6 addressing

I. INTRODUCTION

Cryptographically Generated Address (CGA) [1] is a technique that offers authentication to IPv6 addresses without the need to a third party or additional security infrastructure. CGAs are IPv6 addresses, where the interface identifiers (IIDs) are generated by one-way hashing of the node's public key and other auxiliary parameters. Thus, the IPv6 address of a node is bound to its public key. This binding can be verified by re-computing the hash value and comparing it with the interface identifier of the sender IPv6 address.

CGA is an essential part of SEcure Neighbor Discovery (SEND) [2]. SEND is designed to protect Neighbor Discovery (ND) [3], and StateLess Address Auto-Configuration (SLAAC) [4]. CGA is also proposed as one of the techniques to overcome the security threats in Binding Update in MIPv6 [5], because using other approaches, such as IPSec depends on existing security infrastructure, which is not easy to deploy.

Although CGA is a promising security technique for IPv6, it has some limitations and disadvantages. The main disadvantage of using CGA is the computational cost. Besides, CGA is not a complete security solution; it still has some threats. As shown by Bos et al. [6], CGAs are exposed to global time-memory trade-off attacks. These limitations may prevent the use of CGAs, and leave IPv6 networks vulnerable to many attacks, such as spoofing and Denial of Service (DoS) attacks [9]. Therefore, it is important to mitigate these limitations by enhancing CGAs security and improve its performance before the wide deployment of IPv6.

To enhance CGAs security against the global time-memory trade-off, a more secure version, called CGA++ [6] is proposed. Unfortunately, CGA++ comes with additional cost and it requires longer time than the generation of CGAs for the same security parameters.

In [7], Cheneaua et al. propose an improved method for generating CGAs by using Elliptic Curve Cryptograph (ECC) keys instead of standardized RSA keys. ECC has a shorter key for the same security level as RSA, which leads to smaller packet sizes. Accordingly, the usage of ECC in CGAs can be more suitable especially in resource-limited devices.

In this paper we present a modified CGA implementation that incorporates ECC and CGA++. While CGA++ enhances CGAs security, ECC reduces the size of CGA parameters considerably and accelerates the CGA++ address generation step. We call this modified CGA version as Compact and more Secure CGAs (CS-CGAs). CS-CGAs can be preferred over the standard CGAs in certain scenarios, because it is more secure and provide a privacy protection without scarifying of the performance.

The rest of the paper is organized as follows. Section 2 reviews CGAs, its advantages, limitations, and the possible approaches for improvements. Section 3 describes our adoption of ECC to CGA++ and its integration into SEND. The performance of CGA and CS-CGA implementations are studied in section 4. We conclude the work in section 5.

II. CRYPTOGRAPHICALLY GENERATED ADDRESS (CGA)

This section gives an overview about CGA parameters and algorithm and then discusses the strengths and the advantages of CGAs. The third part of this section introduces the limitations of CGAs and the possible approaches to enhance its security, improve its performance, and provide better privacy protection.

A. CGAs parameter and algorithm

In IPv6, CGAs are known as self-certifying addresses. Any node can generate its own CGA locally and sign messages using its private key. The public key is attached to the signed messages. Thus, the address and the public key are needed for verifying the signature. Therefore, the receiver does not need to have further communication with the sender for completing the authentication verification process.

For using CGAs, the sender and the receiver share the CGA parameter data structure [1]. The "Modifier" field is a 128-bit integer, which is randomly generated to strengthen the robustness of hashed values and to enhance the privacy of the generated addresses. The "Subnet Prefix" field is a 64-bit subnet prefix of the IPv6 address. The "Collision Count" is an 8-bit collision counter used for Duplicate Address Detection (DAD) to ensure the uniqueness of the generated address, initially set to zero. The "Public Key" and the "Extension" fields have variable length.

CGAs require another parameter, which is called security *Sec* value. The *Sec* value lies between 0 and 7 and is designed to increase the CGAs security level against brute-force attacks. The *Sec* parameter is defined in hash extension technique [8], which increases the effective hash length beyond the 64-bit. It is encoded in the three leftmost bits of the interface identifier.

Figure 1 is a schematic diagram showing a representation of CGA addresses generation processes. An address owner computes two hash values (Hash1 and Hash2) using the public key and other parameters. The CGA generation algorithm should fulfill two conditions [1]:

1. The leftmost 64-bit of Hash1 equals the interface identifier. The *Sec*, "u" and "g" bits are ignored in the comparison. The 7th bit from the left in the 64 bit IID is the Universal/Local bit or "u" bit. It is set to "1" to mean that the IID is globally unique. The 8th bit from the left is the Individual/Group or "g" bit, which is set to "1" for multicast addresses.
2. The 16× *Sec* leftmost bits of Hash2 are equal to zero. For example, if *Sec* = 1, the 16 leftmost of Hash2 are zeros.

The verification of the address ownership procedure takes two inputs: CGA address and CGA parameters. The receiver validate the binding between a public key and CGA address by re-computing the Hash1 value based on the received CGA parameters, and comparing it with the IID of the sender. The receiver knows that the message was sent by the owner of the source address by verifying the signature using the sender public key.

B. *Strength and advantages of CGAs*

Authentication by using CGAs can avoid many of the attacks in IPv6 networks. CGAs can prevent address theft of other nodes. When a node generates its address in a secure way, it is hard for a malicious node to claim the ownership of the new address. CGAs also can prevent DoS attacks, which are usually realized based on spoofing of others address, such as Duplicate Address Detection DoS Attack [9].

To impersonate another node's address, an attacker needs to find a public/private key pair, whose truncated Hash1 value matches IID of impersonate node. The number of operations required to impersonate a specific node by using brute-force attack cost, on average, $2^{59+16\times Sec}$ hash function evaluations. The attacker needs to generate a valid modifier, whose cost $2^{16\times Sec}$ hash computation and needs to try 2^{59} different interface identifiers. On the other hand, the address owner is expected to try, on average, $2^{16\times Sec}$ iterations to find the right Modifier value

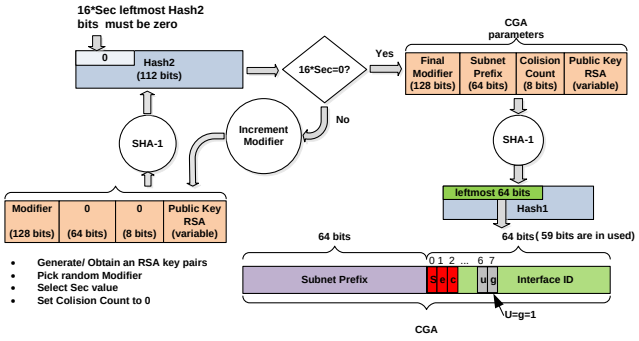

Figure 1. Schematic of CGA address generation

that satisfies Hash2 condition. Therefore, *Sec* value is a scale value parameter that governs the strength of CGAs.

CGAs can obscure the node IID to protect the privacy and achieve anonymity. In a CGA algorithm, the Modifier value should be chosen randomly by using random number generator to avoid unpredictable or/and unlinkable values [1]. So, each time CGA is calculated, a new IID is obtained. Different Modifier values lead to different CGAs even for the same public key. To avoid unique and fixed address, the node is recommended to change its address over time. It is also recommended to generate a new public key for a new network address to avoid the possibility of identifying node by its public key. Even it is still not easy to track based on the known public key, because attacker usually tracks node based on its IP address by using tracking tools, such as *ping* or *traceroute*.

RFC 4941 [10], Privacy Extensions for Stateless Address Autoconfiguration in IPv6, generates a random number for IID and changes it over the time to protect the privacy. However, it does not provide protection against IP address spoofing attacks. Therefore, CGAs are promising for preventing attacks and protecting the privacy. But using CGAs is trade-off between security and privacy protection in one side, and the performance on the other side.

C. *Cost of CGA generation*

Unfortunately, CGAs are computationally heavy and are not a complete security approach. These limitations may prevent its usage in real scenarios. For example, mobile devices have several constraints and requirements that may not be benefit from CGAs security features for different reasons. First, mobile nodes are resource-limited (battery, memory, processor, and bandwidth). Second, mobile nodes need to change their subnet frequently. Consequently it needs to generate a new address, which means it needs a new CGA computation. Third, the time is critical to realize handover operations within a few hundreds of milliseconds to ensure adequate Quality of Service (QoS).

There are several parameters and factors that impact CGAs generation time and security, such as *Sec* value, Modifier, hash function, and public key cryptosystem. These factors and their effects on security, delay, and packet size are studied, and then the possible improvements to a faster and more secure CGA implementation are discussed.

1) Sec value and Modifier

The *Sec* value is the core factor that has great impact on both security level and CGA generation time. The increase in *Sec* value leads to significant delay in the CGA generation algorithm. Theoretically, the cost of generating a CGA increases by 2^{16} for each Sec value.

RFC 3972 makes two recommendations to minimize the delay introduced by using *Sec* value. First, the heavy part of CGA generation algorithm (steps 1-3) [1] can be done in advance, offline, or delegated to more powerful machine. But in case of delegating the computation to powerful machine, the question is how to distribute the calculated parameters to other nodes in a secure way. Moreover, this approach returns back to centralized model, whereas CGAs were originally proposed to avoid relying on a third party. Furthermore, once this powerful machine is compromised, all the other nodes will be compromised as well.

The second approach is to use small *Sec* value. The node should use the *Sec* based on its computational capacity. *Sec* values higher than "1" are not recommended as they require a lot of processing power and too much time. The use of small *Sec*, "0" or "1", is more suitable at this time. Cheneau et al. carried out an experiment on Nokia N800 and they conclude that only Sec = 0 is feasible for mobile devices when using 1024-bit RSA keys [7]. However, the CGA verification does not depend on *Sec* value and it is relatively fast. Only two hash calculations are required.

Another interesting idea to avoid long CGA computational time is to use a stopping time condition for the brute-force search address generation [11]. For *Sec* value greater than "0", there is no guarantee to terminate after certain number of iterations. Large *Sec* value may cause unacceptable delays in CGA generation. Therefore, time threshold can be used as an input in the CGA generation. If the threshold has been exceeded, the CGA generation algorithm terminates. Then, the most secure hash value can be determined by selecting the hash value that has the greatest number of zero bits in Hash2 rounded down to the nearest integer multiple of *Sec*.

2) Public key Cryptosystem

The selection of the public key cryptosystem is vital. It determines the processing delays, computational costs and size of NDP message associated with the cryptosystems. The key pair generation time is substantially prolonged by increasing key size. Therefore, the total CGA generation time is highly influenced by the size of the key. Moreover, due to the SHA-1 intervals, the hashing time increases when the input length exceeds a multiple of 512 bits. Hence, smaller public key reduces the length of Hash1 and Hash2 input. Because, CGA generation algorithm calculates many SHA-1operations to find the valid modifier, it greatly benefits from shorter keys to reduce the size of CGA parameter.

The Elliptic Curve Cryptograph (ECC) public key cryptosystem is proposed to be used in CGA instead of a standardized RSA key [7]. ECC has a shorter key for the same security level as RSA, which leads to smaller packet sizes compared to using of RSA. Also, the signing with the Elliptic Curve Digital Signature Algorithm Cryptograph (ECDSA) is faster than using the RSA equivalent. However, ECDSA needs longer time for the signature verifications. To support alternative public key cryptosystems, two internet drafts are published. The first [12] describes how to use ECC with CGA in SEND. The second [13] describes a mechanism to enable SEND to select between different signature algorithms to use with CGAs.

3) Hash function

Another factor that has impact on CGAs performance in more than one way is the hash function. Standard CGAs use SHA-1 (160 bit hash). But SHA-1 is vulnerable to collision-free attacks [18]. Therefore, RFC 4982 [14] analyzes the implications of attacks against the hash function, and they update the CGAs specifications to support multiple hash algorithms. However, RFC 4982 makes no recommendation for hash function. RFC 6273 [15] analyzes possible threats to the hash algorithms which are used in SEND. The authors conclude that the attacks on the hash functions do not constitute any practical threat to both *RSA Signature Option* and the X.509 certificates, but the attacks against the hash algorithms on CGAs compromise security in SEND. To reduce the time of CGAs generation, Lee and Mun in [16] use MD5 hash instead of SHA-1 in their modified CGA implementation. Since MD5 is simpler and has a shorter processing time. However, MD5 should not be used in CGAs, because it vulnerable to collision free attacks [17].

D. CGA limitations and vulnerabilities

Bos et al. [6] have investigated the attack model of CGA, and they found that CGAs can be vulnerable to global time-memory trade-off and garbage attacks. Accordingly, they propose more secure version of CGAs called CGA++. In this version, the "Subnet Prefix" is included in the calculation of Hash2, and all the "Modifier", "Collision Count" and "Subnet Prefix" values are signed by the private-key corresponding to the public key used. Therefore, CGA++ ensures the ownership of the corresponding private key even without using SEND *RSA signature option*. The time-memory trade-off attack cannot be applied globally, because the node generates new address when it is moved to a new network and signing ND messages provide a protection against garbage attacks [6] when CGAs are deployed alone and not in SEND.

However, CGA++ needs more computations than CGAs for the same *Sec* value. This complexity is due to two reasons. First, the subnet prefix is included in Hash2 calculations. Second, the digital signature is included in Hash1 calculation. In case of using CGA++ within SEND, signing CGA parameters twice is redundant and lead to the high cost. Therefore, signing the message once is sufficient.

Still, there are some other limitations in CGAs. CGAs mechanism can prevent the theft of another node's address, but CGAs cannot provide assurance about the identity of the real node and it is not sufficient to guarantee that the CGA address is used by the appropriate node. Since CGAs are not certified, an attacker can create a new valid address from its own public key and start a communication. For more security requirement, a certificate authority is required to validate the keys. However, the address owner can use the corresponding private key to assert its ownership and to sign SEND messages sent from the

address. An attacker can impersonate other node address from a valid public key, but cannot sign messages. So, the effect of the attack is limited. Moreover, CGAs do not provide any sort of protection against address scanning or any information about the node's privileges on the network. Also, the CGAs are dynamically-generated, so it cannot be used for securing static IPv6 addresses.

III. COMPACT AND MORE SECURE CGA (CS-CGA)

As seen in the previous section, some work has been done to optimize CGAs. Most studies focus on improvement CGAs generation time. The work in [6] focuses on enhancing CGAs security with penalty on the performance. In this paper, we take into account the security, performance and the privacy to obtain an enhanced version of CGAs.

A. CS-CGA Design

Our Compact and More Secure CGA (CS-CGA) implementation comes with some modifications to the standard CGA implementation. First, the subnet prefix is included in Hash2 calculation. Second, ECC is used instead of RSA cryptosystem. Third, CGA parameter is signed and the signature is included in Hash1 calculation. Also, a new flag, *CS-CGA flag*, is defined to allow the use of mixed CGA and CS-CGA. Figure 2 shows the schematic diagram of CS-CGA generation process.

1) Including the subnet prefix in Hash2 calcalation

The prefix is included in Hash2 calculations, as done in CGA++. Including the subnet prefix in Hash2 calculation has the following advantages. First, it enhances the security of CGAs against time-memory trade-off attack. The attacker needs to generate separated database for each subnet. Second, the mobile node privacy is protected, because it has to generate new CS-CGA address for each different subnet.

2) Using ECC Cryptosystem

Because CGAs require to include the public key and other parameters with the message and to affix its signature with every signaling packet that it generates, which means that more than 1K bytes is added to each packet. *CGA Option* carries CGA parameters to enable the receiver to validate the proper binding between the public key and the CGA address. Therefore, minimizing the size CGA parameters in NDP messages is important to reduce the bandwidth consumption.

CS-CGA uses ECC cryptosystem for generating the keys and signing the messages. Using ECC improves CGA generation time and reduces the size of the packet. The CS-CGA uses ECDSA to sign NDP messages. SEND's *RSA Signature Option* carries ECDSA signature when CS-CGA in used.

3) Using SHA-256

Standard CGA uses SHA-1, which is known to have security flaws [18]. ECDSA signature is based on P-256 curve and SHA-256. Therefore, we decided to use SHA-256 in CS-CGA implementation.

4) Set "Reserved field to "1"

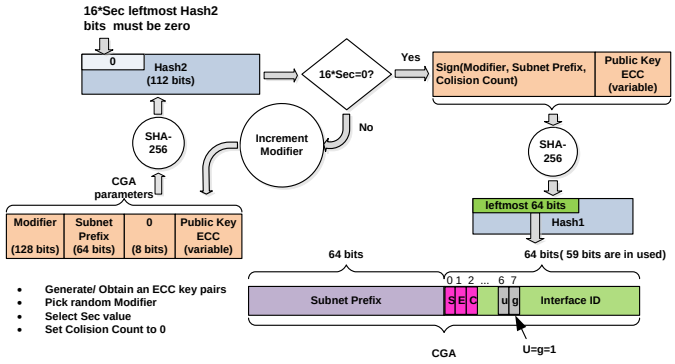

Figure 2. Schematic of CS-CGA address generation

In CGA specifications, the "u" and "g" bits are set to "1" to indicate that node uses CGA address. To distinguish between CGAs and CS-CGAs, the "Reserved" field in *CGA Option* [2] is used. We set the "Reserved" value to "1" to label CS-CGA. Currently, this field is reserved for future use, and using this field makes no change to the standard CGAs.

B. CS-CGA Implementation

As a proof of concept, we chose to modify the existing implementation of CGA and SEND, NDprotector0.5 [19], because it already supports ECC. NDprotector is a user-space Linux platform implementation that has no direct access to kernel space. Figure 3 shows the principle functionality of NDprotector. It is divided into initialization and runtime stages. During the initialization, the CGAs addresses are set up and the firewall rules are defined in *ip6tables*. These rules route all NDP messages to specific *netfilter* queues. NDprotector's runtime component takes the NDP messages out of those queues and secures the outgoing NDP messages, or verifies incoming ones. Afterwards, the NDP messages are either forwarded or dropped, if CGA addresses verification failed.

In NDprotector, the *CGA Option* is added in the address constructor method, `Address.py`, in the initialization phase of NDprotector (see Figure 3). The "*CS-CGA flag*" set to "1" if the calling `Address.py` instance represents an CS-CGA. Figure 4 shows a capture of CS-CGA NS message.

To implement CS-CGA address generation and verification, we added optional parameters to CGA address generation and verification functions in `scapy6send` package. If the "*CS-CGA flag*" is set in the incoming NDP messages, the CGA address verification function is called from `NFQueues.py` with the appropriate parameters. The CGA generation function is called from `Address.py`, depending on whether the `Address.py` instance is representing an CS-CGA address or not.

NDprotector already supports ECC for CGA and no modifications for CS-CGA are necessary. We just made the ECC support more convenient, by adding the possibility to generate an ECC public/private key pair during the instantiation of an CS-CGA `Address.py` object. From a

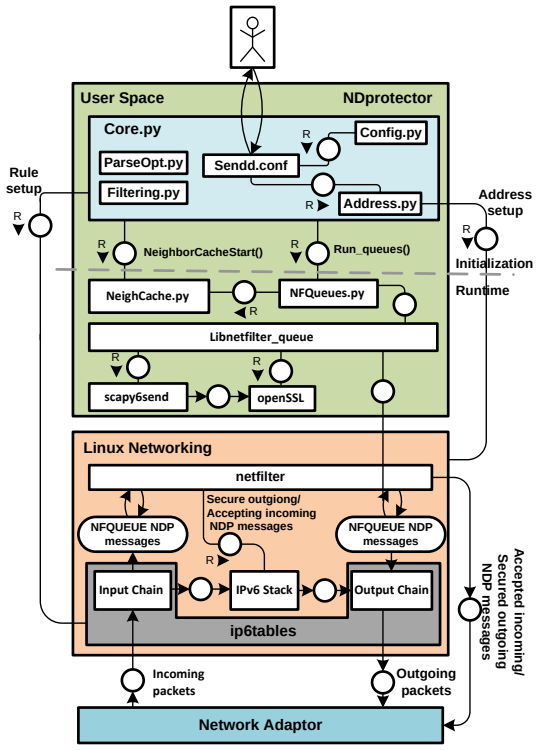

Figure 3. NDprotector architecture

security point of view, it is not recommended to have the keys stored somewhere before the application starts.

IV. Performance Evaluation

This section gives a performance comparison between the standard CGAs and CS-CGAs. The performance analysis is done based on two factors. The first one is the size of NDP messages. Second, the address verification and generation times are compared, when using different cryptosystems algorithms.

The implementation is conducted in a virtual environment, which consists of two Ubuntu 10.10 virtual machines which run the modified NDprotector version. One machine runs the Wireshark to capture the NDP traffic and to determine the size of Neighbor Advertisement (NA) and Neighbor Solicitation (NS) messages.

A. NA and NS Message Size Lengths

NA and NS message sizes highly depend on cryptosystem. We compare the NS and NA message sizes by using different cryptosystems. The comparison is done between CGA with RSA key (3072-bit) and the CS-CGA (P-256 curve), because both have an equivalent security level [7]. Table 1 shows the results. One observation is that the *CGA option* and the *RSA Signature option* constitute the major part of the ICMPv6 messages. Furthermore, ECC greatly reduces the key length. As a result, the use of ECC is advancement in terms of the message sizes for CGAs.

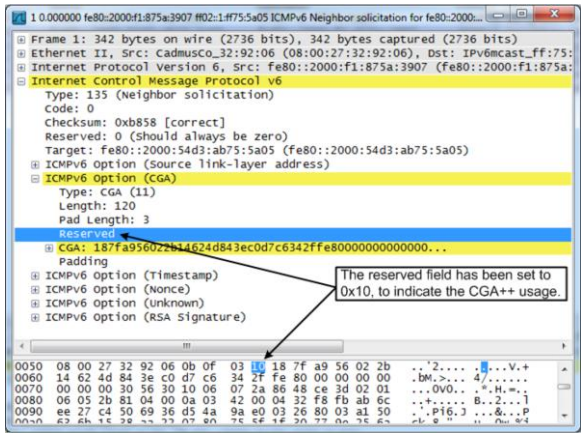

Figure 4. CS-CGA flag

B. CGAs/CS-CGAs Address Generation and Verification Times

We compare the generation and verification time of CGAs with CS-CGAs when using different cryptosystems. The key pair generation time is not included in the comparison, because the node can generate the key pairs in advance to avoid additional delay.

To take the time measurements, a Python script is created. The times are determined with Python's `timeit` module 3, which switches the garbage collection off during the benchmarks. Furthermore, a fine adjustment is done to the ECC public/private key pair class, to avoid generating BER encoding on the fly, during the Hash1 and Hash2 calculation.

CGA/CS-CGA addresses generations times are accelerated significantly through ECC. Table 2 shows CGA/CS-CGA addresses generation and verification times in case of using different cryptosystems. The comparison is made between RSA key with size 3072 bit s and ECC with P-256 curve by using two hash functions, SHA-1 and SHA-256. The generation times for CS-CGA addresses are shorter, because ECC has shorter key size that accelerates the hash calculations. The verification times of CS-CGA addresses are slower than verification times of CGA because CS-CGA include the subnet prefix in the Hash2 calculation, and verifying ECDSA is longer than verifying RSA.

TABLE 1 NS/NA MESSAGES SIZE FOR CGA /CS-CGA

Security level (*Sec* = 1)				
	CGA		CS-CGA	
Cryptosystems	RSA (3072)		ECC (P-256)	
ND message type	NS	NA	NS	NA
ICMPv6 message length (bytes)	928	936	288	280
CGA option length (bytes)	456		120	
RSA signature option length (bytes)	408		96	

TABLE 2 CGA/CS-CGA ADDRESSES GENERATION AND VERIFICATION TIMES

Security level (*Sec* = 1)				
Number of samples (1000 sample)				
Algorithm	cryptosystems	Hash Function	Address generation time (sec)	Address verification time (msec)
CGA	RSA (3072)	SHA-1	2.183	0.695
CS-CGA	ECC (P-256)		1.960	0.723
CGA	RSA (3072)	SHA-256	2.637	0.702
CS-CGA	ECC (P-256)		2.046	0.735

C. CGAs/CS-CGAs security analysis

Including the routing prefix in Hash1 and Hash2 calculation increases the cost of pre-computation attacks by making some brute-force attacks against global scope addresses more expensive. The attacker needs to do an exhaustive search for hash collision or creation of large pre-computed databases of interface identifiers from an attacker's own public key(s) used to find matches for many addresses. The attacker should do this brute-force search for each address prefix separately. However, it is not easy to impersonate a random node in a network because it requires a large storage to carry out this attack. If an attacker wants to impersonate an address of a random node in a network of size 2^{16}, this requires $2^{33-16} = 2^{17}$ gigabytes = 128 terabyte of storage.

V. CONCLUSION AND FUTURE WORK

The proposed, CS-CGA, is an enhanced version of CGA that has more resistance against the time-memory trade-off attack, besides preventing address theft and DoS attacks. Moreover, it assures the anonymity and protects the privacy, because the node needs to generate new CGA once it moves to a new subnet.

Even with using ECC, CGA is still computationally heavy and need to be improved. To avoid unexpectable delay in CGA generation, using a time termination condition can be a practical approach to make CGA feasible. More investigation about the recommend time threshold based on different device specification is needed. For example, based on the CPU speed, the algorithm recommends a proper value for *Sec*, or set time termination condition. Definitely, the termination condition also depends on the application requirement, e.g. MIPv6 needs to finish the address generation within hundreds of milliseconds. A possible way to speed up the CGA generation can be achieved by using Optimistic DAD [20] with CGA. Optimistic DAD approach is a method to minimize address configuration delays that make an address available for use before completing DAD. The DAD in CGA generation algorithm can be one reason for delay, while the probability of collision is too low. Another way to improve the CGA performance could be by optimizing the parameter transmission in each message, for example, to avoid the retransmission of parameters that already known by the receiver. Finally, reducing and eliminating the threats and the attacks against IPv6 network by enhancing the CGA security and make it simple, lightweight, and deployable authentication mechanism is very important. Otherwise, IPv6 network will be left vulnerable to IP spoofing related attacks.

REFERENCES

[1] T. Aura, "Cryptographically Generated Addresses (CGA)," RFC 3972, Internet Engineering Task Force, Mar. 2005, updated by RFCs 4581, 4982.

[2] J. Arkko, J. Kempf, B. Zill, and P. Nikander, "SEecure Neighbor Discovery (SEND)," RFC 3971, Internet Engineering Task Force, Mar. 2005.

[3] T. Narten, E. Nordmark, W. Simpson, and H. Soliman,"Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[4] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[5] T. Aura, M. Roe, "Designing the Mobile IPv6 Security Protocol", Annals of telecommunications, special issue on Network and information systems security, volume 61 number 3-4, March-April 2006.

[6] J. W. Bos, O. Özen, and J.-P. Hubaux, "Analysis and optimization of cryptographically generated addresses," in Proceedings of the 12th International Conference on Information Security, ser. ISC '09. Berlin, Heidelberg: Springer-Verlag, pp. 17–32,2009.

[7] T. Cheneau, A. Boudguiga and M. Laurent, "Significantly Improved Performances of the Cryptographically Generated Addresses Thanks to ECC and GPGPU". Computers & Security 29, pp.419-431. 2010.

[8] T. Aura, "Cryptographically Generated Addresses (CGA)", 6th Information Security Conference (ISC'03), Bristol, UK, October 2003.

[9] P. Nikander, J. Kempf, and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats," RFC 3756 (Informational), Internet Engineering Task Force, May 2004.

[10] T. Narten, R. Draves and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

[11] T. Aura and M. Roe. Strengthening Short Hash Values. http://research.microsoft.com/en-us/um/people/tuomaura/misc/aura-roe-submission.pdf.

[12] T. Cheneau, M. Laurent, S. Shen, and M. Vanderveen, "ECC public key and signature support in Cryptographically Generated Addresses (CGA) and in the Secure Neighbor Discovery (SEND)," November 2009, http: //tools.ietf.org/html/draft-cheneau-csi-ecc-sig-agility-02.

[13] T. Cheneau, M. Laurent, S. Shen and M. Vanderveen, "Signature Algorithm Agility in the Secure Neighbor Discovery (SEND) Protocol. June 16, 2010. http://tools.ietf.org/html/draft-cheneau-csi-send-sig-agility-02.

[14] M. Bagnulo and J. Arkko, "Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGA)", RFC 4982, July 2007.

[15] A. Kukec, S.Krishnan, and S. Jiang, "SEND Hash Threat Analysis", RFC 6273, June 2011.

[16] M. Bagnulo and J. Arkko, "Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGA)", RFC 4982, July 2007.

[17] M. Stevens, Fast Collision Attack on MD5, Cryptology ePrint Archive, Report 2006/104, eprint.iacr.org/ 2006/104.

[18] X. Wang , L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1. CRYPTO 2005: 17-36", 2005.

[19] NDprotector: an implementation of CGA & SEND for GNU/Linux based on Scapy6, 30.06.2010, http://amnesiak.org/NDprotector/

[20] N. Moore, "Optimistic Duplicate Address Detection (DAD) for IPv6", RFC 4429, April 2006. http://tools.ietf.org/html/rfc4429