Cryptographically Generated Addresses (CGAs): A survey and an analysis of performance for use in mobile environment

Sana Qadir and Mohammad Umar Siddiqi

International Islamic University Malaysia, Kuala Lumpur, Malaysia

Summary

CGAs are cryptographically generated IPv6 addresses and are one of the most novel features introduced in IPv6. They have the promising potential of being the basis of authentication mechanisms for Mobile IPv6 because they do not require hosts to share information or security infrastructure. A mobile environment however has several resource constraints that must be considered before any mechanism can be deemed feasible. This paper undertakes to survey all the work done on CGAs and their performance. The goal is to identify and discuss parameters that have an impact on performance (e.g. the public-key cryptosystem being used). This should help in proposing possible modifications and parameters to ease the adoption of CGAs in a mobile environment.

As a starting point, the CGA generation and verification algorithms are implemented with the help of a cryptographic library designed especially for embedded systems. An evaluation of the performance of this implementation is undertaken and a comparison is made with the results reported in previous studies. Based on this, a recommendation is made for the parameters that should be used by mobile nodes when using CGAs. The long-term modification that has the most potential for improving the performance of CGAs in a resource-limited setup is also identified

Key words:

CGA, Performance, Public Key Cryptosystem

1. Introduction

Two benefits of adopting IPv6 are improved support for security and mobility. To help achieve these goals a novel mechanism called Cryptographically Generated Addresses (CGA) was introduced. CGAs were initially included in SEcure Neighbor Discovery (SEND) protocol to protect against IP address spoofing and stealing attacks. However, the most promising area of application for CGAs is in mobile environments where the use of conventional security mechanisms like IPsec/IKE is not realistic. This is because these mechanisms require hosts to share common information or to depend on an existing security infrastructure whereas the use of CGAs is not bound by this limitation [1]. CGAs also have the advantage over alternatives like Host Identity Protocol (HIP) in that it does not require an additional layer between the network and transport layer [2].

It is thus imperative that an analysis of all work related to CGAs is made and their performance studied so that optimizations can be proposed to ease their adoption in a mobile environment.

The rest of this paper is organized as follows. Section 2 presents the details of CGAs and a review of improvements proposed for CGAs. Section 3 describes the performance testing environment and section 4 provides the results. Section 5 gives the conclusion of the paper.

2. Overview and Related Work

2.1 Overview of CGA

CGA are IPv6 addresses for which the interface identifier is generated by computing a cryptographic one-way hash function using a public key and auxiliary parameters [3]. This ensures that the IPv6 address of a host is bound to the public key it is using. This binding can be verified by re-computing the hash digest and comparing it with the interface identifier of the IPv6 address [4].

The most basic use of a CGA is to prevent an attacker from impersonating an existing IPv6 address [3]. CGAs can also be used for authentication. For example, to prove that the sender of a packet is the actual owner of a CGA, the packet can be signed by the sender's private key. This signature, the public key and the auxiliary parameters can be sent with the packet to the receiver. The receiver can verify the signature of the packet to confirm that the sender of the packet is also the owner of the CGA [5].

The use of CGAs requires the sender and receiver to share the CGA Parameters data structure. Essentially this is the concatenation of [3]:

- a 128-bit randomly generated Modifier,
- a 64-bit Subnet Prefix,
- an 8-bit Collision Count,
- variable length Public Key, and
- variable length Extension Fields (optional)

CGAs also require a security parameter or `sec`. This is an unsigned 3-bit integer that indicates the security level of

Manuscript received February 5, 2011
Manuscript revised February 20, 2011

the CGA against brute force attack. After a CGA is generated, the sec parameter is encoded in the three leftmost bits of the interface identifier (see Fig. 1).

The CGA generation and verification algorithms are defined in the RFC 3972. Fig. 2 and 3 show the steps in brief [3]. A CGA generated using the algorithm shown in Fig. 2 will satisfy the following two conditions [3]:

- Hash1 equals the interface identifier of the address
- 16 * sec leftmost bits of Hash2 are zero

2.2 Areas of Application of CGA

a. To protect against denial-of-service attacks during IPv6 address auto-configuration, duplicate address detection (DAD) and neighbor discovery (ND) [6]. A node can prove ownership of its address by using its private key to sign the DAD and ND messages that it sends.

b. To protect against denial-of-service attacks in Mobile IPv6 [7]. A binding update (BU) message can be authenticated by the correspondent node if the sending mobile node signs the message. Inadequate mechanisms to protect BUs is one of the most important shortcomings of Mobile IPv6

2.3 Cost Analysis of CGA generation and verification algorithm

Advances in technology make it easier to attack the underlying hash function used by a CGA. For example, a 2nd pre-image attack on a 64-bit hash digest requires O(2^{64}) hash function evaluations [2]. To prevent against such attacks, the *hash extension* technique was introduced to achieve the effective extension of the hash digest length [6]. Basically this technique requires that the input to Hash2 be modified (by incrementing the Modifier value) until the 16 * sec leftmost bits of the hash digest are zero [3]. This increases the cost of a brute force attack from O(2^{59}) to O($2^{59 + 16 * sec}$). It also means that the cost of generating a CGA increases from O(2^{59}) to O($2^{59 + 16 * sec}$).

However, once a CGA has been generated, the cost of using and verifying a CGA does not depend on sec. In fact, the verification algorithm requires a constant amount of computation and it is relatively fast (requires at most two computations of SHA-1 functions) [3].

2.4 Related Work

In a mobile environment, minimizing the time taken by the CGA generation and verification algorithm is vital. This is for two reasons. Firstly, handover operations have to be completed within a few milliseconds in order to ensure an adequate quality of service. Secondly, mobile nodes have limited resources (like battery, bandwidth and memory) that have to be efficiently used to prevent unacceptable delays. It is thus important to review all the work related to factors that affect the time taken by CGA algorithms:

i. sec value

In general, a mobile node should use a sec value based on its computational capacity, risk of attacks and the expected lifetime of the address. Currently, values between 0 and 2 are considered adequate [3]. However, any increase in sec value introduces significant delay and this is undesirable in a mobile environment. For example, [8] finds that increasing sec from 0 to 1, causes the average execution time of CGA generation algorithm to jump from15.57µs to just over 0.1 seconds. A sec value of 2, increases the average execution time to 100 seconds. Table 1 presents a summary of the results of CGA generation time reported in several studies.

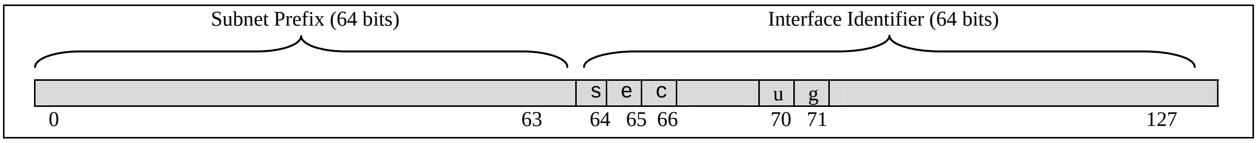


Fig. 1 CGA

Table 1: CGA generation time for different sec values

Source	Specification of setup	sec = 0	sec = 1	sec = 2	sec = 3
[8]	Pentium 4.3GHz, Memory 1GB. Linux (Kernel 2.4)	15.57µs	just over 0.1 seconds	100 seconds	more than 200 hours
[9]	Machine with moderate processing power	n/a	1 minute	16 days	n/a
[2]	A modern PC (AMD64)	n/a	0.2 seconds	3.2 hours	24 years

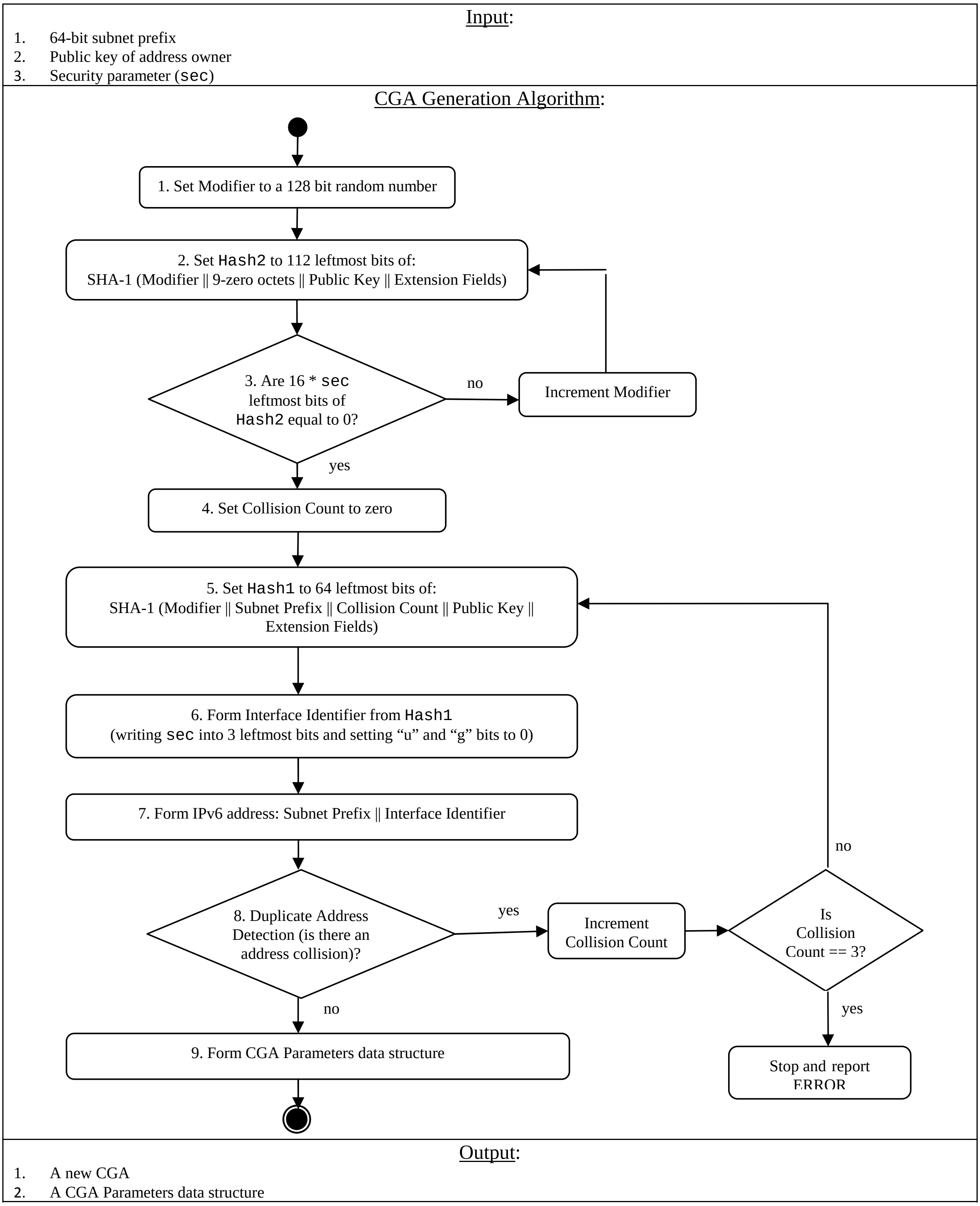
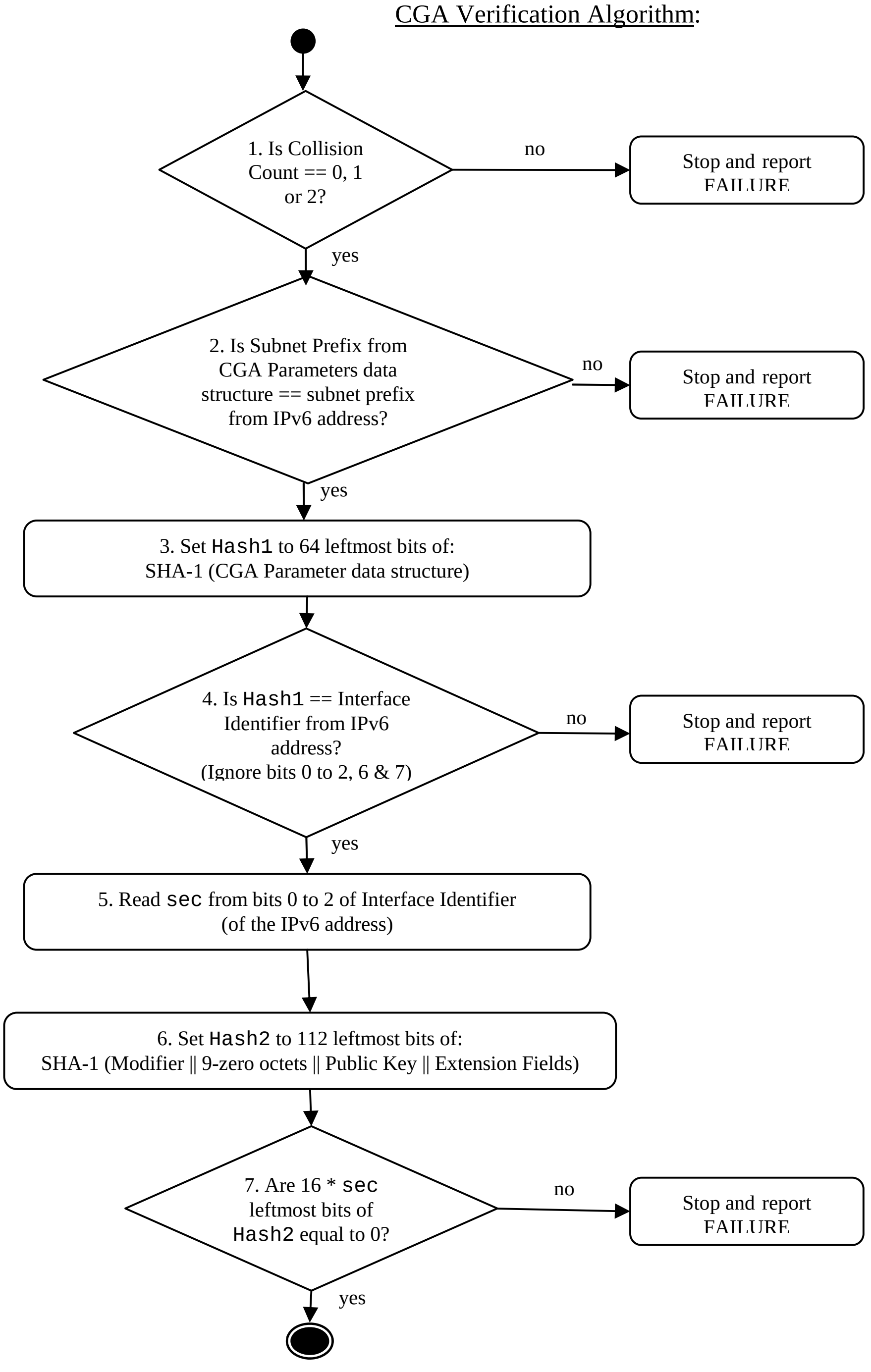

Input:
1. 64-bit subnet prefix
2. Public key of address owner
3. Security parameter (sec)
CGA Generation Algorithm:
1. Set Modifier to a 128 bit random number
2. Set Hash2 to 112 leftmost bits of:
SHA-1 (Modifier || 9-zero octets || Public Key || Extension Fields)
3. Are 16 * sec leftmost bits of Hash2 equal to 0?
no
Increment Modifier
yes
4. Set Collision Count to zero
5. Set Hash1 to 64 leftmost bits of:
SHA-1 (Modifier || Subnet Prefix || Collision Count || Public Key || Extension Fields)
6. Form Interface Identifier from Hash1
(writing sec into 3 leftmost bits and setting “u” and “g” bits to 0)
7. Form IPv6 address: Subnet Prefix || Interface Identifier
no
8. Duplicate Address Detection (is there an address collision)?
yes
Increment Collision Count
Is Collision Count == 3?
no
yes
9. Form CGA Parameters data structure
Stop and report ERROR
Output:
1. A new CGA
2. A CGA Parameters data structure

Fig. 2 CGA Generation Algorithm

Input:

1. An IPv6 address
2. A CGA Parameters data structure

CGA Verification Algorithm:


1. Is Collision Count == 0, 1 or 2?
no
Stop and report FAILURE
yes
2. Is Subnet Prefix from CGA Parameters data structure == subnet prefix from IPv6 address?
no
Stop and report FAILURE
yes
3. Set Hash1 to 64 leftmost bits of: SHA-1 (CGA Parameter data structure)
4. Is Hash1 == Interface Identifier from IPv6 address? (Ignore bits 0 to 2, 6 & 7)
no
Stop and report FAILURE
yes
5. Read sec from bits 0 to 2 of Interface Identifier (of the IPv6 address)
6. Set Hash2 to 112 leftmost bits of: SHA-1 (Modifier || 9-zero octets || Public Key || Extension Fields)
7. Are 16 * sec leftmost bits of Hash2 equal to 0?
no
Stop and report FAILURE
yes

Output:

1. Success or Failure

Fig. 3 CGA Verification Algorithm

Based on the performance test carried out on an actual Nokia N800, by Cheneau and colleagues, only sec value of 0 is feasible for mobile devices at the medium security level provided by 1024-bit RSA keys [10].

To help minimize the delays introduced by sec values greater than 0, RFC 3972 makes two recommendations:

a. Steps 1-3 of the CGA generation algorithm can be done in advance or offline (i.e. on separate, more powerful machine) [3]. The result of these steps can be transmitted to a mobile node which can then carry on the algorithm from step 4. If the subnet prefix changes or an address collision occurs, a mobile node can simply use the earlier results to generate a new address (starting from step 4).

b. Use small `sec` values. Higher `sec` values can be used either in the future (when the computational capacity of mobile nodes is higher) or when the risk of denial-of-service attacks based on brute-force search is too high to ignore [3].

ii. Hash function

The hash function used by the CGA algorithm also has an impact on the level of security of the CGA and on performance. RFC 4982 introduces support for multiple hash Algorithms in CGAs [11]. This is to help protect applications where CGAs are vulnerable to attacks based on the collision-free property of SHA-1. The RFC, however, makes no recommendations of a hash function.

In an effort to reduce execution time, Lee & Mun use MD5 (instead of SHA-1) in their design of a Modified CGA (MCGA) because it is simpler and has a shorter processing time [8]. SHA-1 should not be replaced by MD5, because its 128-bit hash digest is vulnerable to collision attack [12]. The possibility of using a hash function other that SHA-1 was investigated in [10]. The performance metric used shows SHA-1 as the most efficient algorithm compared with more secure alternatives like SHA-256, SHA-512, RIPEMD-160, TIGER and WHIRLPOOL.

iii. Public Key Cryptosystem

The public key cryptosystem used by CGAs has an impact on performance in more than one way. Reference [10] reports Total CGA generation time (including key pair generation time) for different RSA key lengths. If the reported RSA key generation time is subtracted from the Total CGA generation time, it gives the time taken by just the CGA generation algorithm. This data is summarized in Table 2.

The setup used is a Nokia N800 @ 400 MHz when `sec` is 0.

It is clear to see that increasing the RSA key size does not have much of an impact on the CGA generation time (which is about 13.8 ms for all three key sizes). It is the RSA key generation time that increases substantially as key size is increased. Using an alternative public key cryptosystem like ECC will reduce the key generation time as well as the size of the CGA Parameters data structure because of the smaller key length. The latter should help reduce packet size and this is desirable in a low bandwidth environment and for devices with limited battery.

[10] reports that it takes 0.079611 seconds to generate an 163 bit ECC key and the time to generate the CGA is 0.068774 seconds (0.148385 - 0.079611). When compared to the 4.685756 seconds needed to generate an equivalent 1024 bit RSA, it is clear that ECC's key generation is much less expensive than RSA's key generation. To deal with unacceptable delays, RFC 3972 suggests generating the key pair on a separate, more powerful machine instead of on the mobile node [3]. Of course, this assumes the secure transmission of the key pair from the machine to the mobile node.

The need to support alternative public key cryptosystems has led to a number of Internet drafts. [13] paves the way for a CGA to be associated with multiple public key while [14] outlines how to use ECC together with CGA and SEND.

iv. Other Studies

Two other studies on CGA generation and verification algorithms are worth mentioning. In [9] two modifications are made. Firstly, they move the Modifier field from the beginning of the CGA Parameters data structure to after the Extension Fields. This enables them to reduce the number steps used in re-computing the hash digest. The hash digest has to be re-computed whenever 16 * sec leftmost bits of Hash2 are not zero. The second change they make is to optimize the SHA-1 algorithm to perform the block operation only on those blocks that contain the new 128 bit Modifier. The block operation is not performed on the rest of the blocks and this saves computation time. Performance test on an implementation of their proposed optimizations shows an almost 80% reduction in the time taken for computing the hash value.

Table 2: RSA Key Length vs. CGA Generation Time (in seconds) [10]

RSA Key Size (in bits)	**384**	**512**	**1024**
Total CGA generation time	0.651353	1.004133	4.699501
RSA key generation time	0.637553	0.990302	4.685756
CGA Generation Time	0.0138	0.013831	0.013745

This they argue is a significant reduction in the time required to generate a CGA.

The second work proposes a more secure version of CGAs for IPv6 called CGA++ [2]. Their main aim was to increase the resistance of CGAs to time-memory trade-off attacks and garbage attacks. To achieve resistance against global time-memory trade-off attacks, they include Subnet Prefix in the calculation of `Hash2`. This prevents a mobile node from offloading the computation expensive steps1-3 to more powerful machine. This however has the cost of reducing the efficiency of CGA++. To protect against garbage attacks, they sign the Modifier, Subnet Prefix and Collision Count with the private key. This signature is concatenated with the Public Key before being used to calculate `Hash1`. Introducing this signature increases the cost of CGA generation and verification algorithms. Their analysis shows that [2]:

- generation time of CGA++ is significantly higher than the generation time of CGA when `sec` is 0.
- generation time of CGA++ is about 0.005 seconds when `sec` is 0 and about 0.18 seconds when `sec` is 1.

3. Development and Performance Evaluation

As specified in RFC 3972, CGAs used by SEND must be able to support RSA public key length between 384 and 2048 bits [3]. In future, higher security level may be required. To this end, it is important to evaluate the performance of CGA algorithms at these key sizes using more efficient software implementations than those used by the studies quoted above. Such a performance analysis should also provide very important feedback on how CGAs should be used in a mobile environment.

The CGA generation and verification algorithms were coded using C. The implementation of multi-precision integers, random number generation and different hash functions used is the one provided by PolarSSL [15]. Most of the studies quoted above use OpenSSL instead because it is the most popular open source implementation of SSL/TLS. PolarSSL is a more recent light-weight implementation of SSL/TLS that is written specifically for use by embedded systems. PolarSSL allows developers to include only the components of the library that are needed into their application instead of the whole library. It has also been successfully ported to several architectures including ARM and Motorola 68000. Both these characteristics make PolarSSL the preferred choice for this study because execution time of algorithms and memory requirements of an application both depend on the software implementation being used.

The development is done using Maemo 5 Software Development Kit (SDK) running on a Debian system (Kernel 2.6.26-2). Maemo is the operating systems used on the Nokia N series of smartphones including the N900 [16]. The code was cross-compiled using Scratchbox (a cross-compilation toolkit included in Maemo SDK) for an ARM target. The executable produced can run on an actual ARM processor like the OMAP ARM SoCs used in Nokia's N series. In this study, however, the executable is run using QEMU. QEMU is an open source processor emulator that can run a program compiled for one machine (in our case ARM) on a different machine [17]. In our case, this different machine is a desktop with Pentium Dual-Core (each CPU at 2.8 GHz) and Memory of 494.3MB GB.

To measure the execution time of an algorithms or function, the RDTSC instruction is recommended. This instruction returns the value of a 64-bit time stamp counter (TSC) that is incremented on each clock cycle [18]. Using clock cycle count is the most precise and accurate method of recording time on x86 architectures. To this end, code is added so the following measurements can be made:

- CGA generation – the number of clock cycles time taken by the steps shown in Fig. 2 (except for Duplicate Address Detection). Keys are pre-generated.
- CGA Verification – the number of clock cycles taken by the steps shown in Fig. 3
- The number of clock cycles taken for `Hash1` and `Hash2` computation

The results obtained are shown in Table 3, 4 and 5. The mean number of clock cycles is reported as well as the corresponding mean time in microseconds (based on the fact the processor speed of an N800 is about 400 MHz).

4. Analysis of Results

As can be seen from Table 3 and 5, `sec` value of 0 (with 1024-bit RSA) results in a mean CGA generation time of 68.61 µs. This is a reasonable delay for a mobile scenario. Reference [10] reports a Total CGA generation time of 4.699501 seconds for the same `sec` value and RSA key length. As noted before, their result includes the time taken to generate the RSA key pair while this study assumes the mobile node has a pre-generated key pair. If the time taken to generate the RSA key pair is excluded from their results, their CGA generation time reduces to about 13.8 ms [10]. This is still more than the 68.61 µs obtained in this study.

Using `sec` value of 1(instead of 0) increases the mean CGA generation time by about 126 times. Although 8.7 ms is still much less than the unacceptable delay of 1s, it is

Table 3: CGA generation using different sec values

Value of `sec`	**0**	**1**
Mean # of clock cycles	27446	3,467,670
Mean time (in µs)	68.61	8669.17

Table 4: Hash1 computation using SHA-1, SHA-256 and SHA-512

Hash function	**SHA-1**	**SHA-256**	**SHA-512**
Mean # of clock cycles	19	25	49
Mean time (in µs)	0.0475	0.0625	0.1225

Table 5: CGA generation and verification using different RSA Key Length

RSA Key Length		**384-bits**	**512-bits**	**1024-bits**	**2048-bits**
CGA Generation	**Mean # of clock cycles**	27290	27404	27446	27549
	Mean time (in µs)	68.22	68.51	68.61	68.87
CGA Verification	**Mean # of clock cycles**	51	51	74	97
	Mean time (in µs)	0.13	0.13	0.19	0.24

still too much in cases where handover operations have to be completed within 5 to10 ms.

It is also obvious from Table 5 that increasing RSA key length does not have a significant an impact on mean CGA generation time (it remains between 68 to 69 µs). Increasing RSA key length has an obvious impact only on Total CGA generation time (as reported in Table 2). This difference is because of the inclusion of generation time of the RSA key pair.

It should be noted that alternatives to RSA should urgently be investigated for use with CGAs for following reasons:

- A mobile node still has to generate the RSA key pair (even if this delay is not part of the CGA generation and verification algorithm). As shown in [10] this introduces unacceptable delay even for the soon to be replaced 1024-bit keys.
- Generating and verifying CGA signatures requires the computationally expensive RSA signature generation and verification operations

The mean CGA verification time in Table 5 is in line with the point made in RFC 3972, that the verification algorithm is relatively fast i.e. less than 0.25 µs even for 2048-bit RSA keys.

Table 4 shows that although using SHA-256 or SHA-512 increases the mean number of clock cycles taken to compute `Hash1`, the increase is not substantial. This should help promote the possibility that SHA-1 can be replaced with alternatives like SHA-256 and SHA-512 that are more robust and provide a higher level of security without any noteworthy degradation in performance.

5. Conclusion and Recommendation

As it can be seen, some work has been done on optimizing CGAs for use in a mobile environment. These include investigations of the impact of different `sec` values, different hash functions, and different public key cryptosystem. Like [10] this study finds that only `sec` value of 0 is feasible for mobile devices at the medium security level provided by 1024-bit RSA keys. Because this study uses a more efficient software implementation, the results of the performance evaluation are better than those reported in previous studies. The mean time to generate a CGA and the mean time to verify a CGA were found to be acceptable for a mobile scenario (i.e. 68.61 µs and 0.19 µs respectively).

Although this study finds that increasing RSA key length does not lead to any significant increase in mean time taken by CGA generation and verification algorithms, it is still vital to investigate replacing RSA. This is because using RSA for operations related to using CGAs (such as key generation, CGA signature generation and verification) is too expensive for mobile nodes. RSA must be replaced by a public key cryptosystem that provides comparable cryptographic strength but has faster key generation, shorter key length and less expensive signature generation and verification. Only when this is achieved can CGAs-based authentication be computationally feasible for mobile environment.

References

[1] C. Caicedo, J. Joshi and S. Tuladhar (2009), IPv6 Security Challenges, *IEEE Computer*, *42*(2), pp.36-42.

[2] J.W. Bos, O. Ozen, and J-P Hubaux (2009), Analysis and Optimization of Cryptographically Generated Addresses, *Information Security 2009* (LNCS 5735), pp17-32.

[3] T. Aura (2005), *Cryptographically Generated Addresses (CGA)* [Online]. Available: http://tools.ietf.org/pdf/rfc3972.pdf.

[4] H. Oh and K. Chae (2007, February 12-14), "An Efficient Security Management in IPv6 Network via MCGA" paper presented at the 9th International Advanced Conference on Communication Technology (ICACT 2007), Phoenix Park, Republic of Korea.

[5] C. Bauer and M. Ehammer (2008, October 12-14), "Securing Dynamic Home Agent Address Discovery with Cryptographically Generated Addresses and RSA Signatures" paper presented at IEEE International Conference on Wireless & Mobile Computing, Networking & Communication (WIMOB'08), Avignon, France.

[6] T. Aura (2003), Cryptographically Generated Addresses. *Information Security 2003* (LNCS 2851), pp.29-43.

[7] J. Arkko, C. Vogt and W. Haddad (2007), *Enhanced Route Optimization for Mobile IPv6* [Online]. Available: http://tools.ietf.org/pdf/rfc4866.pdf.

[8] H. Lee and Y. Mun (2006), "Design of Modified CGA for Address Auto-configuration and Digital Signature in Hierarchical Mobile Ad-Hoc Network" paper presented at the International Conference on Information Networking ICOIN 2006 (LNCS 3961), pp.217-226.

[9] T. Rajendran and K.V. Sreenaath, (2008, January 6-10), "Hash optimization for cryptographically generated address" paper presented at the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 2008), Bangalore, India.

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

[11] M. Bagnulo and J. Arkko (2007), *Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs)* [Online]. Available: http://tools.ietf.org/pdf/rfc4982.pdf.

[12] B. A. Forouzan (2008), *Cryptography and Network Security*. New York: McGraw-Hill.

[13] T. Cheneau, M. Maknavicius, S. Sean and M. Vanderveen (2009, Feb 21), *Support for Multiple Signature Algorithms in Cryptographically Generated Addresses (CGAs)* [Online]. Available: http://tools.ietf.org/pdf/draft-cheneau-cga-pk-agility-00.pdf.

[14] T. Cheneau, M. Laurent, S. Shen and M. Vanderveen (2010, Jun 16), *ECC public key and signature support in Cryptographically Generated Addresses (CGA) and in the Secure Neighbor Discovery (SEND)* [Online]. Available: http://tools.ietf.org/pdf/draft-cheneau-csi-ecc-sig-agility-02.pdf.

[15] polarSSL, http://polarssl.org/.

[16] maemo.org, http://maemo.org/intro/.

[17] QEMU, http://wiki.qemu.org/Main_Page

[18] P. Kankowski (2006), *Performance measurements with RDTSC* [Online]. Available: http://www.strchr.com/performance_measurement_with_rdtsc.