Analysing and Improving Performance and Security of Cryptographically Generated Address Algorithm for Mobile IPv6 Networks

Sana Qadir, Mohammad Umar Siddiqi, and Wajdi F. M. Al-Khateeb
(Corresponding author: Sana Qadir)

Electrical and Computer Engineering Department, International Islamic University Malaysia
P.O. Box 10, 50728 Kuala Lumpur, Malaysia
(Received Nov. 20, 2014; revised and accepted Feb. 20 & Apr. 21, 2015)

Abstract

A Cryptographically Generated Address (CGA) is a self-certifying address that a node generates when it joins a foreign network. Despite its advantages, generating a CGA is computationally expensive. This study examines the security and performance issues related to the use of the CGA Generation algorithm. It also scrutinizes the hash extension mechanism, different hash functions and how multithreading can be used to improve the performance of the CGA Generation algorithm. Based on the results, this research recommends imposing a minimal computational security of $\mathcal{O}(2^{80})$, the use of the HAVAL hash function and parallelizing the algorithm in order to take maximum advantage of multicore architectures of mobile node.

Keywords: CGA generation algorithm, hash functions, multithreading, parallel computing

1 Introduction

A Cryptographically Generated Address (CGA) is an IPv6 address generated by a node using the CGA Generation algorithm as defined in RFC 3972. The input to this algorithm is the public key of the node and some auxiliary parameters. The output of the algorithm is a CGA.

CGAs were introduced in IPv6 as part of *stateless address auto configuration* (SLAAC). This enables nodes to join a subnet and locally generate an IPv6 address. Although CGAs have several advantages, their main shortcoming is high computational cost. The aim of this paper is to carry out an in-depth analysis of the security and performance of the CGA Generation algorithm.

This is important for several reasons. Firstly, Mobile IPv6 (MIPv6) networks usually consist of low-end nodes that have limited resources (computational, memory, bandwidth, power, etc.) and therefore cannot be expected to perform computationally expensive operations. Secondly, CGAs are increasingly being included in protocols like Enhanced Route Optimization - ERO (where they are used to prove ownership of a MN's home address). Proving ownership of an address is important to protect against attacks such as address stealing, flooding, session hijacking and redirect attacks [13, 30]. One of the factors that dominates the cost of CGA-based authentication protocols is the CGA Generation algorithm [12, 16]. In the case of MIPv6 networks, delays have to be minimised to preserve the quality of real-time and interactive applications. In practice, this means operations like handovers should be completed within a few hundred milliseconds.

2 Related Work

Essentially, a CGA cryptographically binds the public key of a node to its IPv6 address. The details of the CGA Generation algorithm are illustrated in Figure 1. The CGA Parameters data structure that the sending node shares with the receiving node is shown in Figure 2. The receiving node verifies a CGA using the CGA Verification algorithm. This study will only focus on the CGA Generation algorithm and not the CGA Verification algorithm.

CGAs require the sending and the receiving node to share a 3-bit integer called `sec` that indicates the security level of the CGA against brute force attack. `sec` can take values from 0 (lowest security) to 7 (highest security) and is encoded in the three leftmost bits of the generated interface identifier (IID).

The main aim of a CGA is to prevent the stealing and spoofing of existing IPv6 addresses [4]. In other words, an impersonation attack - given a CGA, an adversary is able to find another public key that generates the same CGA. This would require the adversary to break the 2^{nd} pre-image resistance of `hash1` [4]. Because only 59 bits of `hash1` make up the IID, the cost of finding a hash collision is only $\mathcal{O}(2^{59})$. 59 bits are too few to provide

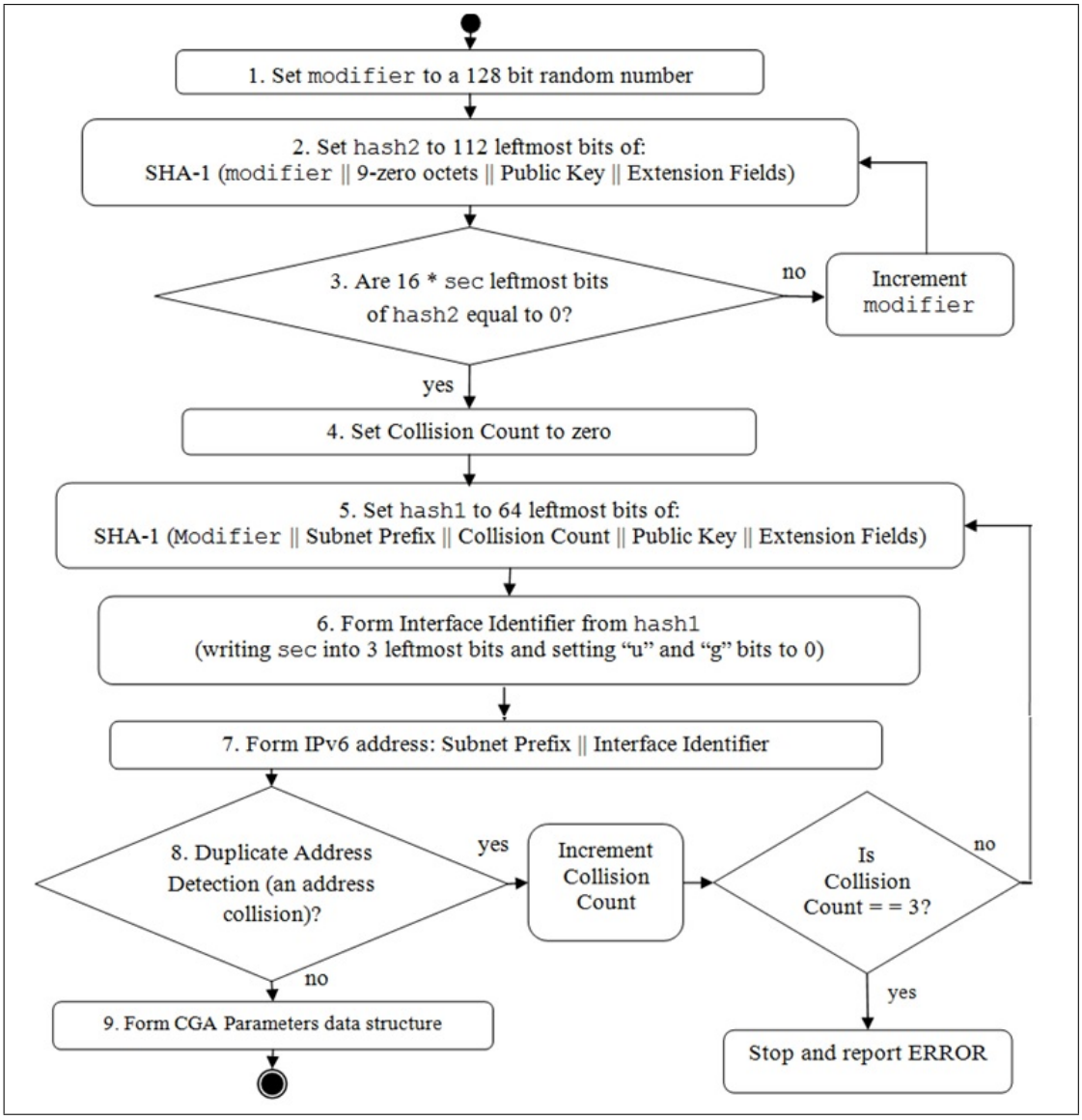

Figure 1: CGA generation algorithm [4]

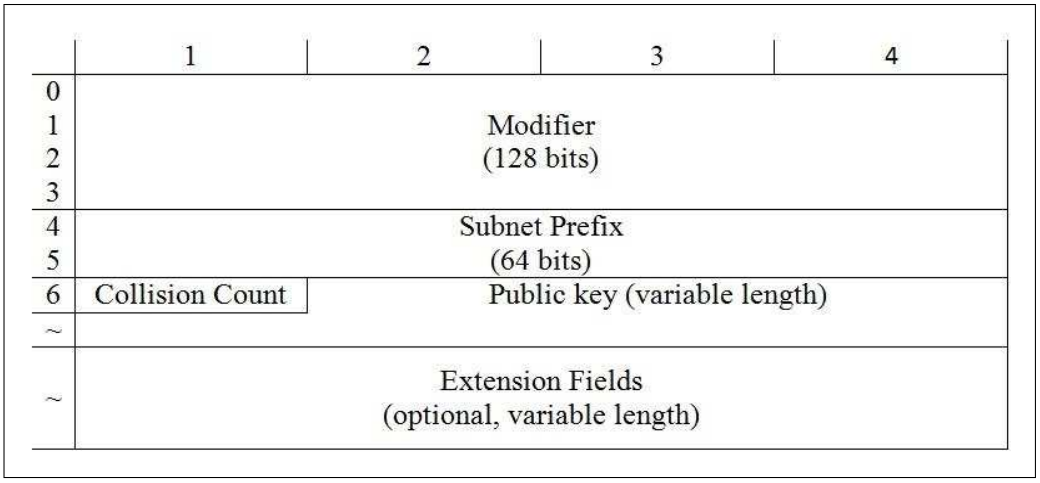

Figure 2: CGA parameters data structure [4]

strong security or any real protection against brute force attacks. CGAs should provide users with the option to increase this cost in the face of exponential growth of computational capacity and memory.

The *hash extension mechanism* was introduced to solve this shortcoming. This mechanism modifies the input to `hash2` until the leftmost $16*\texttt{sec}$ bits of the hash digest are zero [4]. This effectively increases the cost of an impersonation attack to $\mathcal{O}(2^{59+\texttt{sec}*16})$. However, this mechanism has the negative impact of increasing the cost of generating a CGA to $\mathcal{O}(2^{\texttt{sec}*16})$ [4]. In fact, several studies confirm that the largest contributor to the computational cost of the CGA Generation algorithm is the value of `sec`.

2.1 Performance Analysis

Table 1 summarizes the results from studies that have undertaken the performance evaluation of the CGA Generation algorithm. It is obvious to see that the performance of the CGA generation algorithm degrades substantially with increasing `sec` values. It is also important to note that for `sec` values greater than 0, the CGA Generation algorithm is not guaranteed to terminate.

RFC 3972 stipulates that nodes can choose `sec` value based on [4]:

- How long they expect to use the address;
- Their computational capacity;
- The perceived probability of being attacked.

The RFC also stipulates several solutions that can be used to overcome poor performance of the CGA Generation algorithm [4]:

- Using small `sec` values;
- Offloading computationally costly Steps 1-3 to a more powerful machine; or
- Completing the computationally costly Steps 1-3 offline or in advance.

Existing literature contains a number of studies that have investigated the factors that impact the performance of the CGA Generation algorithm. It also contains a number of possible solutions to improve the performance of the CGA Generation algorithm. These studies are summarized in Table 2.

2.1.1 Hash Extension Mechanism

Because it is vital that MIPv6 nodes complete address generation in less than a few hundred milliseconds, one obvious solution to improve the performance of the hash extension mechanism is through some form of time based termination. This was initially proposed in [5] and later refined in [21] as Time-Based CGA (TB-CGA). In TB-CGA, `sec` is not selected by the node. Instead the node decides the time after which CGA Generation algorithm must terminate. The best `hash2` value found during this time (i.e. the `hash2` value where the most $\texttt{sec}*8$ leftmost bits are zeros) is used to generate the CGA. Essentially, `sec` is automatically determined based on the time. A faster CPU will search for more `hash2` values within the same time meaning TB-CGA will automatically adjust `sec` according to the speed of the processor on which it is run [21]. This is a very advantageous design because it automatically adjusts based on the resources of the node.

Despite these advantages, the authors feel that users can be negligent and set an address generation time that is very small. This can result in the generation of an address that is detrimental to the security of the whole network. Also, using $\texttt{sec} * 8$ instead of $\texttt{sec} * 16$ was proposed as a good idea from a performance perspective in [5]. Reference [21] provides empirical evidence to support this claim especially in the case of low-end nodes. This approach also does not change the communication of `sec` to the verifying node nor does it change the CGA Verification algorithm.

2.1.2 Generation of Key Pair

For improving key generation time, the best solution is to use alternative cryptosystems. The best example is provided in [9]. This study reports that CGA Generation time using RSA-1024 (4.70 s) drops 31 times for ECC-163 (0.15 s). However, the choice of which public key cryptosystem is best to use in the CGA Generation algorithm is out of the scope of this paper. One reason for this is that the choice cannot be made solely on the performance of the algorithm used to generate the key pair. The performance of the CGA Signature generation and CGA Signature verification algorithms must also be taken into account as is investigated in [27].

2.1.3 Hash Function

The performance of the hash function is of importance because of the role it plays in the hash extension mechanism where Steps 2 - 3 of the algorithm are repeated in search of a suitable `modifier`. To this end, [14] replaces SHA-1 with MD-5 for use in Mobile Ad-Hoc Networks (MANETS). This is because of the latter's simplicity and superior performance. In [23], the CGA Parameters data structure is restructured and then some operations in SHA-1 and MD-5 are reordered to take advantage of this new structure. They report an 80% improvement in performance [23]. However, it must be noted that both SHA-1 and MD-5 are considered broken. An attack on the collision resistance property of SHA-1 can be carried out in $\mathcal{O}(2^{63})$ instead of $\mathcal{O}(2^{80})$ [19]. The authors are of the opinion that using weak hash functions to improve the performance of the CGA Generation algorithm is not an acceptable approach.

Table 1: Performance of CGA generation algorithm (RSA-1024)

Source	Setup	sec	Sample Size	Performance	Recommendation
[9]	Nokia 800	0	10000	4.7 s	Use sec = 0 for mobile nodes
[1]	Intel Duo 2.67 GHz CPU	0	1000	avg: 93.41 ms	Do not use sec value more than 1
		1	1000	avg: 402 ms	
		2	5	avg: 1 hr 39 min	
[17]	AMD64 with OpenSSL	1	–	0.2 s	Use sec = 1
		2	–	3.2 hr	
[22]	–	2	–	avg: several hours	Users should use sec values of 0 or 1

Table 2: Factors affecting performance of CGA generation algorithm and possible improvements

Source of computational cost	Aim of mechanism (importance to security)	Proposed solutions	Source(s)	Disadvantage
Hash extension mechanism	Increase security level of CGA against brute force attack (only 59 bits of hash digest are used as the interface identifier)	Users use small sec values (0 or 1)	[4, 5]	Computational security $< \mathcal{O}(2^{80})$
		Steps 1-3 can be done on a powerful machine beforehand	[3, 4]	Relies on a centralized model
		Time limit based on application or CPU speed	[21]	
		Time and probability based termination condition	[5]	
		Use cryptographic/graphic accelerator cards. Significant reduction in CGA generation time esp. for higher sec values	[9]	
		Take advantage of parallelism to speedup CGA generation particularly on devices with multiple cores	[2]	
Generation of key pair	The key pair is used in the generation and verification of CGA Signatures	Delegate to a more powerful key server to generate key pair	[3, 4, 29]	Relies on a centralized model
		Use public key cryptosystem with faster key generation time (e.g. ECC)	[9, 10]	
Hash function	Generate hash digest	Replace SHA-1 with an alternative faster hash function (e.g. MD-5)	[9, 14]	MD-5 is broken

2.2 Security Analysis

On a broader note, the performance of CGA Generation algorithm cannot be scrutinized without analyzing the security issues surrounding the use of CGAs (see Table 3). Using CGAs can still leave a network vulnerable to a few types of attacks. The attacks possible against the CGA Generation algorithm are discussed in this section.

2.2.1 Global Time-Memory Trade-Off (TMTO) Attack

This attack is explained in [17]. [17] also proposes an improved CGA algorithm called CGA++ to help prevent this attack. CGA++ protects against replay attack but at the cost of an additional signature generation and signature verification operation. [10] improves CGA++ by proposing the use of faster ECDSA signatures in their Compact and Secure CGA (CS-CGA). They also show that CS-CGA Generation algorithm (with ECC P-256) takes 1.96 s while the original CGA Generation algorithm (with RSA-3072) takes 2.183 s. Using ECC, also has the advantage of generating shorter signatures and smaller CGA Parameters data structures. However, the CS-CGA Verification algorithm (with ECC P-256) is 0.037 ms slower than the original CGA Verification algorithm (with RSA 3072). Despite the benefit of using ECC, the CS-CGA Generation algorithm is still computationally expensive. Moreover, [17] notes that the TMTO attack is prohibitive in the terms of the amount of storage required to launch the attack. Impersonating a random node in a network with 2^{16} nodes would require about 128 TB of storage [17].

2.2.2 Impersonation

The security of a CGA is also affected by the hash function used. Protection against impersonation requires a hash function that it is 2^{nd} pre-image resistant. The hash function must also be very efficient because it is repeatedly used in the computationally intensive Steps 2 - 3. Replacing SHA-1 with a more secure hash function was investigated in [9]. They found that SHA-1 outperforms most other commonly accepted hash functions like SHA-256 and SHA-512 [9]. One more study has also compared the performance of hash functions and found that SHA-256 performs better than BLAKE, Skein and SHA-3 (Keccak) [27]. BLAKE and Skein were included in the study for several reasons. Firstly, BLAKE has a simple design that is easy to implement and lends itself to excellent performance [15]. Skein is flexible, simple and also shows excellent performance on both hardware and software (including a version called Skein-256 that can be implemented on 8-bit smart cards) [26]. Lastly, SHA-3 was chosen because it is based on a sponge construction that is completely different from the Merkle-Damgard construction used in many commonly used hash functions (like SHA-1 and MD-5). The sponge construction is an iterative structure that supports variable length output and in addition to the basic security properties of hash functions, it has been proven to be indifferentiable from the random oracle [11]. There are a few disadvantages to hash functions based on the sponge construction. The most notable is the large state. This basically makes hash functions like Keccak more suitable for large messages and not small ones like in the context of the CGA Generation algorithm. However, we will include Keccak in this study because of its adoption as SHA-3.

We will not go into detail about the DoS attacks against the CGA Verification algorithm. The focus of this paper is the CGA Generation algorithm.

Also, we agree with the use of a timestamp option (in the CGA Parameters data structure) to protect against replay attacks.

We will also not go into further details about the privacy issue surrounding the use of CGAs and the garbage attack (as outline in Table 3).

3 Design of Enhanced CGA Generation Algorithm

3.1 Hash Extension Mechanism

The hash extension mechanism was proposed by [5] as a solution for applications where the hash digest was limited to less than 128 bits. Hash values longer than or equal to 128 bits are considered secure against brute force attacks for any reasonable future while a minimum of 80 bits are acceptable for the immediate future [5]. This is particularly important in scenarios where the adversary has a much more powerful computer while the victims node is a low-end mobile or embedded computer.

We think that the design of the enhanced CGA Generation algorithm should:

- Impose a minimal computational security. Users can be negligent and set an address generation time that is very small. This can be detrimental to the security of the whole network. There should be a minimal security level that a node must provide, i.e. $\mathcal{O}(2^{minimal})$. A reasonable value is 80 bits given the computational capacity of modern nodes. In future, this can be increased for nodes with greater computational capacity.

- Allow the value of `sec` to be guided by the three factors mentioned in RFC 3972:

  1) the duration a node is expected to use an address, i.e. $\mathbf{T}_{expected\_lifetime}$. Nodes frequently move from one subnet to another. It is a waste of resources to generate a CGA with high computational security when the user has no intention of staying in the subnet for any reasonable duration.

Table 3: Limitations of CGAs from a security perspective

Name of attack	**Algorithm / Data**	**Details of attack**	**Mitigation or counter mechanisms**
Denial of Service (DoS) against CGA Verification process	CGA Verification Algorithm	An adversary can reply to each DAD check performed by a node on a tentative CGA telling the node that the address is already in use. Effectively this prevents the node from joining the subnet.	• Sign DAD & NA messages [4]; • Verify each DAD response [1]; • Use DAD extension [22].
	CGA Parameters data structure	Adversary captures/sniffs, replays or changes the sender' CGA parameters so the verification process fails.	Use a *Timestamp Option* when CGA is used in protocols other than SeND [22].
Global Time-Memory Trade-off (TMTO) Attack	CGA Generation Algorithm	The adversary creates a large database of IIDs from its own key pair and then searches for matches for many addresses. • Attack can be assumed to be almost impractical because of massive storage requirements.	• Include subnet prefix in input to `hash2`. This forces adversary to create a separate database for each subnet prefix [3]. • CGA++ (also sign input to `hash1`; expensive and does not solve problem with local-link addresses). This prevents TMTO attack from being applied globally [10, 17, 22].
Garbage Attack	CGA	The adversary uses random data as public-key. • Limited practicality since node does not have corresponding private key.	• Include an authentication mechanism in CGA or use CGA in a protocol that demands authentication [17].
Impersonate an existing CGA	CGA Generation Algorithm	Find another key pair that produces the same CGA. • Break 2^{nd} pre-image resistance of SHA-1(`hash1`). • Cost of attack: $\mathcal{O}(2^{59+\texttt{sec}*16})$.	Replace SHA-1 with SHA-256 (see RFC 4982) [10, 27].
Violation of Privacy	CGA	A node that continues to use a valid CGA (in a subnet) for a long period of time can be tracked.	Set a lifetime for a CGA address [22]: $mT_G \leq T_l \leq T_A/n$ where T_G is time to generate a new CGA, T_l is the lifetime of a CGA, T_A is time to attack a CGA, m and n are integers.
		An adversary can track a node using its public key. • Difficult attack to carry out because nodes are usually tracked using their IP address.	• Generate a new key pair when joining a new network.

2) the perceived probability of an attack , i.e. $\mathbf{P}_{attack}$. This can be set to a high value when a user is joining an untrusted/public network or low when joining a secure/protected network.

3) the computational capacity of a node, i.e. $\mathbf{CPU}_{capacity}$.

Values that can be selected by the user for each of these three factors are shown in Table 4. In this way, the final value of `sec` remains between 0 and 7 and can be securely encoded in the three leftmost bits of the CGA.

- support a maximum computational security of more than 128 bits. This is to ensure that CGAs are applicable well until 2030.
- granularity of 8 (as in TB-CGA) instead of 16 (as in RFC 3972). The option of removing the granularity altogether is very attractive because then the `hash2` value with the most zero leftmost bits found in a given time can be used. However, this strategy is not possible because only three bits are available to securely transmit `sec`.

If all of the above mentioned design changes are adopted, then the overall computational security of the CGA can be calculated as in Equation (1):

$$\begin{aligned} & ComputationalSecurity \\ = \; & O(2^{(\mathbf{T}_{expected\_lifetime}+\mathbf{P}_{attack}+\mathbf{CPU}_{capacity})*8+80)}. \end{aligned} \tag{1}$$

The authors recognize that the above design depends on how accurately a user chooses values for $\mathbf{T}_{expected\_lifetime}$, $\mathbf{P}_{attack}$ and $\mathbf{CPU}_{capacity}$. However, the range of computational security (from $\mathcal{O}(2^{80})$ to $\mathcal{O}(2^{136})$) is optimal.

3.2 Hash Function

Hash functions are usually not considered to be a performance bottleneck specially on desktops. However, on embedded systems (with slower bandwidth), the performance of the hash function can have a more substantial impact (esp. when the hash function is executed in a loop as in Steps 2 - 3).

SHA-1 is used in the original CGA Generation algorithm because of its efficiency. Any hash function that replaces SHA-1 must have superior or comparable performance.

SHA-3 and Skein have been around for a few years, so this study will include them for comparison purposes. This study will also include the new improved version of BLAKE called BLAKE2 which is reported to have comparable performance to MD5 on 64-bit platforms. BLAKE2 comes in two versions. BLAKE2b is optimized for 64-bit architectures and BLAKE2s is optimized for 8-bit or 32-bit architectures [7].

This study will also examine two other hash functions that are not broken and produce hash digests of at least 128 bits. The first hash function is HAVAL. This hash function is based on the Davies-Meyer construction and is not susceptible to attacks that aim to exploit the Merkle-Damgard construction. The downside to this function is that an efficient algorithm, with a complexity of $\mathcal{O}(2^{59})$, has been demonstrated for constructing collisions for the 3-pass version of HAVAL [8]. As such, only the 4-pass and 5-pass versions of HAVAL, for which no weaknesses have been found, are considered secure. Also, HAVAL is reported to be faster than MD5. The last hash function included in this study is MD6 [24]. The current MD6 version is resistant to the buffer overflow error and has been proven to be resistant to differential cryptanalysis. Its design takes full advantage of opportunities for parallelism in multicore architectures. It is also considered to be a relatively simple and efficient hash function [6].

3.3 Timestamp

This is included as an Extension Field in the CGA Parameters data structure to protect against replay attacks. Figure 3 shows the Enhanced CGA Parameters data structure.

3.4 Include Subnet Prefix in Input to `hash2`

This is included to protect against the Global Time-Memory Trade-off (TMTO) attack.

3.5 Parallelism

One method of reducing the cost of the CGA Generation algorithm is to take advantage of the multicore architecture of most recent mobile nodes. Almost all platforms are becoming multicore, as manufacturers have realised that improving performance by increasing raw clock rates is reaching its physical limit and mutlicore chip design is the best approach to adopt. For example, the Qualcomm Snapdragon 808 (arrived at end of 2014) has six cores (a dual core Cortex A57 and four Cortex A53) [28].

Multicore systems have the most impact on performance when the main processing of an algorithm is split into multiple threads. In other words, when the algorithm is parallelized. However, it should be remembered, that the maximum speedup in performance is limited by Amdahl's law. Essentially, this law states that the speedup obtained from multiple processors is limited by the execution time of the sequential part of a program [25].

At first glance, the CGA Generation algorithm looks like a sequential set of instructions. But there are two obvious ways in which the computationally expensive parts of the algorithm can be parallelized. It is important to remember that the way an algorithm is parallelized has an impact on its performance. Two methods are illustrated in Figure 4 and Figure 5. In these examples, the main

Table 4: Values for three factors that determine overall value of `sec`

$T_{expected\_lifetime}$	P_{attack}	$CPU_{capacity}$	value
Small	*Negligible*	*Low*	0
Medium	*Low*	*Average*	1
Large	*Medium*	*Fast*	2
-	*High*	-	3

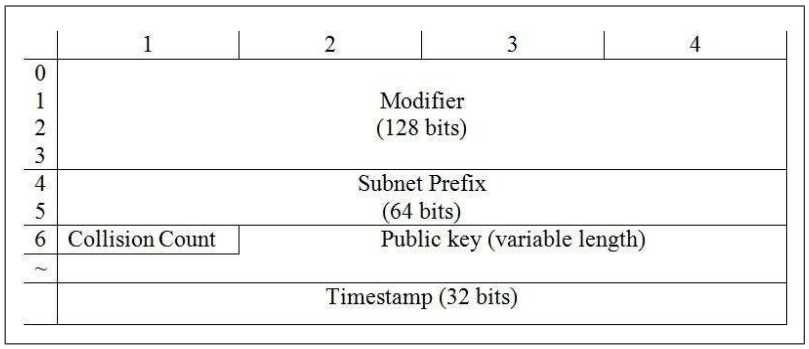

Figure 3: Enhanced CGA parameters data structure [5]

process spawns two additional threads (i.e. $t = 2$). More threads can be spawned if additional cores are available (e.g. four threads $t = 4$ when four cores are available).

Theoretically, assuming:

- T_i is the time taken by Step i that is executed by a thread in parallel;
- T_S is the total time taken by all the sequential steps; and
- c is the number of cores.

Each method can be analyzed in the following ways.

Method 1. *Each thread starts with a different random* `modifier`*:*

$$T_{CGA} \approx \min\left(\sum_1^{m_1}\sum_{i=1}^{3} T_i, ..., \sum_1^{m_t}\sum_{i=1}^{3} T_i\right) + T_S \quad (2)$$

Here, m_1 *is the number of* `modifiers` *searched by thread* 1*,* m_2 *is the number of* `modifiers` *searched by thread* 2 *and so on until* m_t *(i.e. number of* `modifiers` *searched by thread* t*).*

Method 2. t *threads equally share the number of* `modifiers` *to be searched, i.e.* m_{Total}*:*

$$T_{CGA} \approx \frac{m_{Total}}{t}\left(\sum_{i=2}^{3} T_i\right) + T_S \quad (3)$$

This study implements and reports results from both these methods.

4 Implementation of Enhanced CGA Generation Algorithm

4.1 Hash Function and Hash Extension Mechanism

The enhanced CGA Generation algorithm is implemented in C. The Meamo 5 SDK is used and the code cross-compiled for ARM architecture. Also, every effort is made to use the same library or implementation (e.g. of hash function) in order to ensure that performance indicates difference in design rather than difference in implementation [20]. The SAPHIR library (for SHA-2, SHA-3, Skein, HAVAL) and reference C implementations are used (e.g. `blake2_code_20140114.zip` from [7] and `md6_c_code-2009-04-15.zip` from [24]). The clock cycles are recorded for the following operations on an actual mobile architecture (i.e. a Nokia 900):

1) Calculate `hash2`;
2) CGA Generation algorithm.

It should be noted that the Nokia 900 has TI OMAP 3430 chipset with a 600 MHz Cortex-A8 CPU. It also has a PowerVR SGX530 GPU.

4.2 Parallelism

To implement parallelism, POSIX threads (or `Pthreads`) are used. `Pthreads` have a much lower overhead (at least 6 times faster) compared to `fork()`. Apart from basic multithreading, mutexes and condition variables are used to implement Methods 1 and 2 [18].

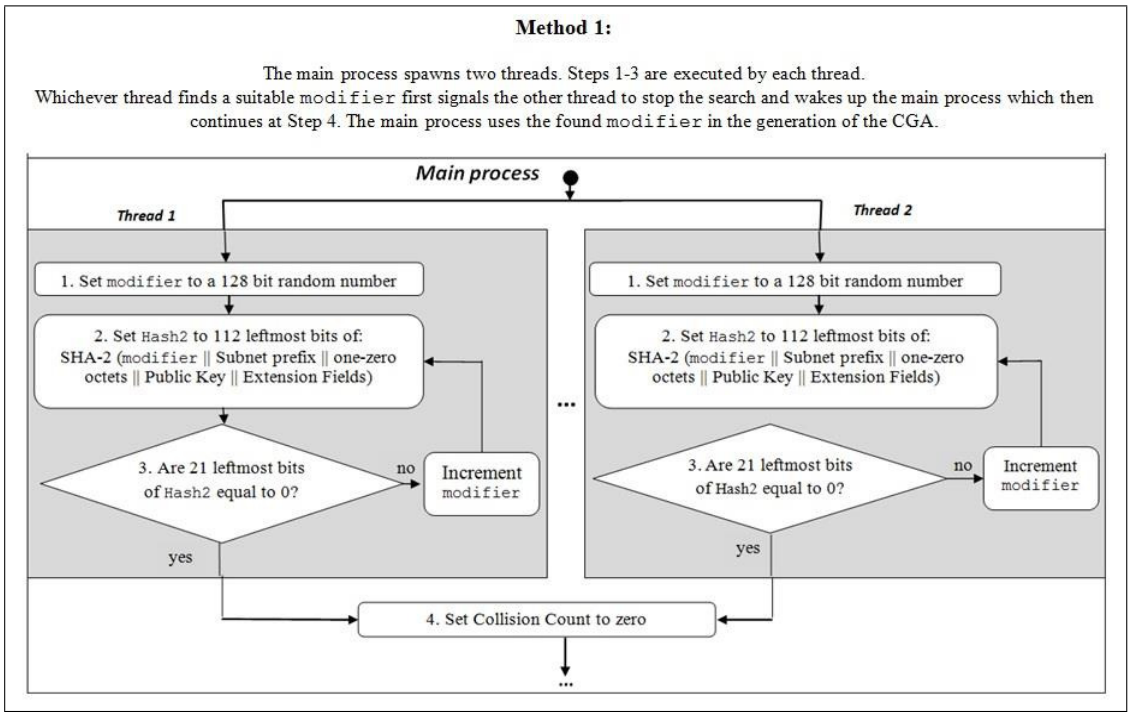

Figure 4: Method 1

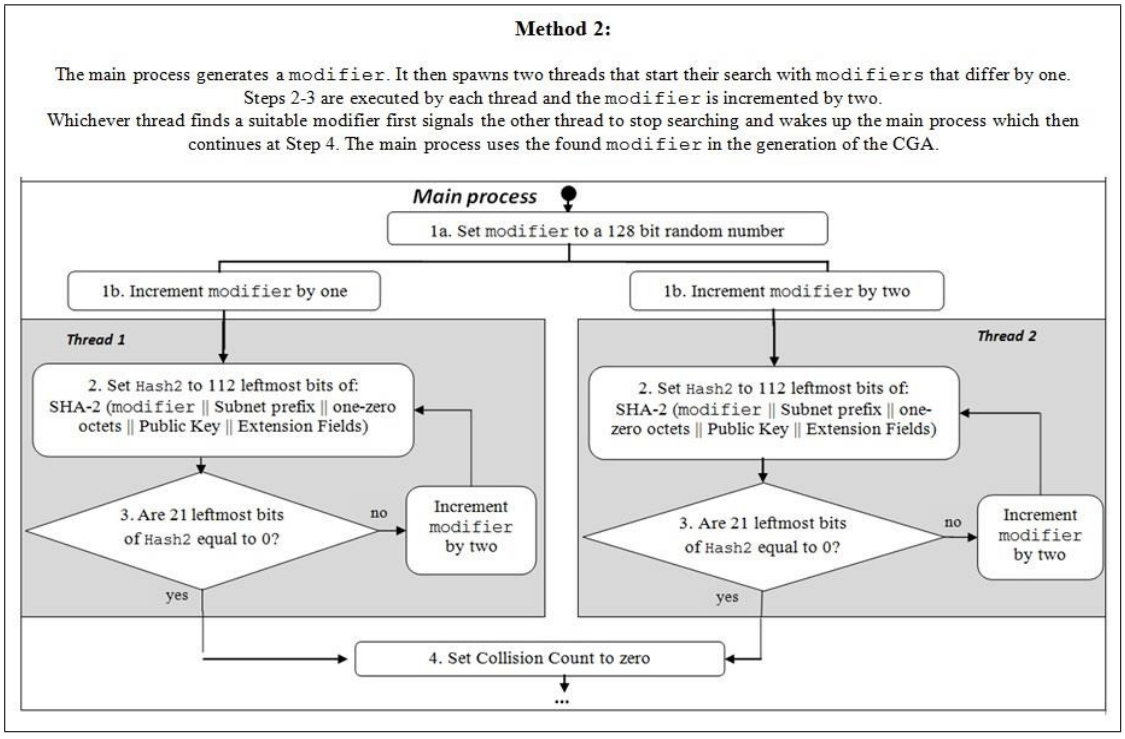

Figure 5: Method 2

The CGA Generation algorithm (implemented using Methods 1 and 2) is run on an Intel Core i7-3537U CPU @ 2.00 GHz (cache size: 4096 KB). This architecture has two cores with each core clocked at 2.0GHz. With hyper-threading, the two cores are capable of handling up to four parallel threads. In other words, the architecture acts as if it has four cores. This provides reasonable estimation since as of 2014 most Android smartphones are quad-cores processors. It is also important to note that only when `pthread_setaffinity_np()` is used to allocate a thread to run on a specific core, the utilization of the core reach 100%. The Gnome/GNU Linux `system monitor` is used to observe CPU utilization.

5 Results

5.1 Different `sec` Values

Table 5 shows the average number of clock cycles (10 runs) taken to generate a CGA for different levels of security. It is clear that the enhanced CGA Generation algorithm with a minimal computational security of $\mathcal{O}(2^{80})$ takes at least 180 ms on a N900. More modern mobile nodes will show better performance.

5.2 Different Hash Functions

Figure 6 shows the average number of clock cycles (30 runs) taken to compute `hash2` using different hash functions. As is obvious from the figure, the 4-pass HAVAL and the 5-pass HAVAL should be considered as excellent substitutes to SHA-3 because of their significantly superior performance. HAVAL-4 also provides the closest performance to SHA-256 out of all the hash functions compared in this work.

It is also important to remember that for hash functions, the level 1 cache size (for instructions) is one of the most important parameters affecting performance [20]. So in order to see improved results, manufactures should increase the level 1 cache size of mobile nodes. The N900 used to obtain the data in Figure 6 has configurable instruction and data caches of 16KiB - 32KiB.

5.3 Parallelism

Figure 7 compares the average number of clock cycles (100 runs) taken by the CGA Generation algorithm at $\mathcal{O}(2^{80})$. There are a few obvious points that can be noted from Figure 7.

- Spawning even one extra thread improves performance by about 20% (regardless of which method is used to parallelize the algorithm).
- In Method 1, the performance improves drastically (39%) when 2 threads (instead of 1 thread) are spawned by the main process. However, this improvement in performance slows down significantly as the number of threads increases to 3 or more.
- Likewise, for Method 2, the performance improves drastically (40%) when 2 threads (instead of 1 thread) are spawned by the main process. This improvement in performance slows down until four threads are spawned. After four threads, the performance actually gets worse.
- The best performance is obtained from Method 2 with four threads. Essentially, this means that the best performance is generally obtained by keeping the number of threads spawned equal to the number of cores (and they are 100% CPU-bound).

6 Conclusion

This paper reports a detailed investigation of the CCA Generation algorithm from a security and performance perspective. It proposes fixing a minimal computational security of $\mathcal{O}(2^{80})$ for the generation of a CGA and finds that this takes 180 ms on a typical mobile node like the N900. Over time (and increasingly powerful machines) this mimimal computational security should be increased. This paper also finds that HAVAL-4 and HAVAL-5 are the best alternatives to SHA-2 and SHA-3 from a performance viewpoint. With regards to taking advantage of multicore architectures, we find that Method 2 (for parallelising the CGA Generation algorithm) provides the maximum speedup when the number of threads spawned by the main thread equals the number of cores.

References

[1] A. AlSa'deh and C. Meinel, "Secure neighbor discovery: review, challenges, perspectives, and recommendations," *IEEE Security and Privacy*, vol. 10, no. 4, pp. 26–34, 2012.

[2] A. AlSa'deh, H. Rafiee and C. Meinel, "Multicore-based auto-scaling secure neighbor discovery for windows operating systems," in *Proceedings of 26th IEEE International Conference on Information Networking (ICOIN'12)*, pp. 257–262, Bali, Indonesia, Feb. 2012.

[3] T. Aura, "Cryptographically generated addresses," *Information Security*, LNCS 2851, pp. 29-43, 2003.

[4] T. Aura, *Cryptographically Gnerated Addresses (CGA)*, Technical Report RFC 3972, Mar. 2005. (http://tools.ietf.org/pdf/rfc3972.pdf)

[5] T. Aura and M. Roe, *Strengthening Short Hash Values*, May 10, 2015. (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.7681)

[6] D. V. Bailey, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, R. Rivest, B. Agre and Y. L. Yin, *The MD6 Hash Function - A Proposal to NIST for SHA-3*, 2009. (`http://groups.csail.mit.edu/cis/md6/docs/2009-04-15-md6-report.pdf`)

Table 5: Different values of `sec`

`sec` security level	Computational security $\mathcal{O}(2^n)$	Average Number of Clock Cycles	Average Time on Nokia 900
0	$\mathcal{O}(2^{59})$	25,842	43μs
1	$\mathcal{O}(2^{67})$	44,201	74μs
2	$\mathcal{O}(2^{75})$	4,724,643	7.9ms
–	$\mathcal{O}(2^{80})$	107,806,357	180ms

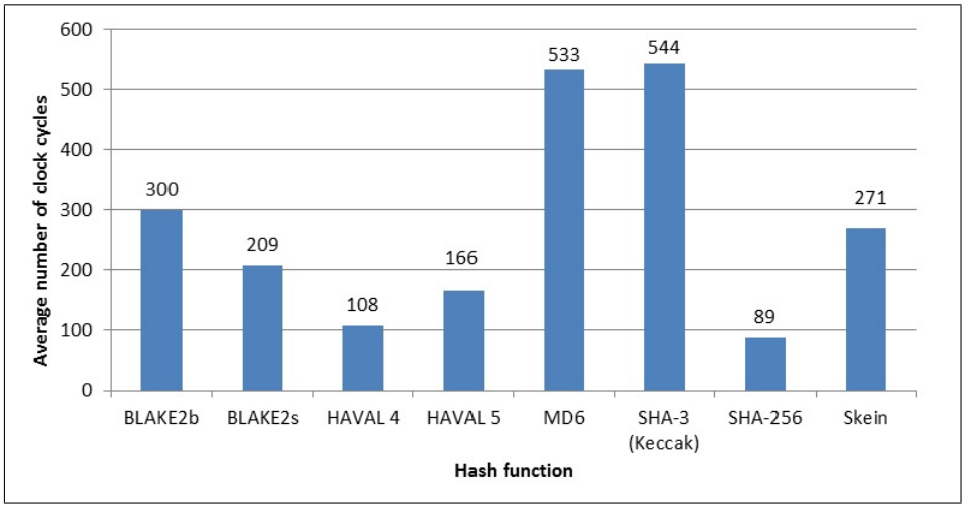

Figure 6: Comparison of different hash functions to calculate `hash2`

[7] Blake, *Blake2: Fast Secure Hashing*, May 10, 2015. (`https://blake2.net`)

[8] C. D. Canniere, J. Lano, H. Yoshida, A. Biryukov and B. Preneel, "Non-randomness of the full 4 and 5-pass haval," in *Security in Communication Networks*, LNCS 3352, pp. 324-336, 2005.

[9] T. Cheneau, A. Boudguiga, and M. Laurent, "Signifiantly improved performances of the cryptographically generated addresses thanks to ECC and GPU," *Computers and Security*, vol. 29, pp. 419–431, 2010.

[10] F. Cheng, A. AlSa'deh and C. Meinel, "CS-CGA: Compact and more secure CGA," in *Proceedings of 17th IEEE International Conference on Networks (ICON'11)*, pp. 299–304, Singapore, 2011.

[11] J. H. Davenport, S. Al-Kuwari, and R. J. Bradford, *Cryptographic Hash Functions: Recent Design Trends and Security Notions*, 2011. (`https://eprint.iacr.org/2011/565.pdf`)

[12] M. Doll, C. Vogt, R. Bless and T. Kuefner, "Early binding updates for mobile IPv6," in *Proceedings of IEEE Wireless Communications and Networking Conference*, vol. 3, pp. 1440–1445, Mar. 2005.

[13] C. C. Lee, M. S. Hwang and S.-K. Chong, "An improved address ownership in mobile IPv6," *Computer Communications*, vol. 31, no. 14, pp. 3250–3252, 2008.

[14] H. K. Lee and Y. Mun, "Design of modified CGA for address auto-configuration and digital signature in hierarchical mobile ad-hoc network," *Information Networking. Advances in Data Communications and Wireless Networks*, LNCS 3961, pp. 217-226, 2006.

[15] W. Meier, R. C.-W. Phan, J. P. Aumasson, L. Henzen, *SHA-3 Proposal BLAKE*, ver. 1.3, Dec. 16, 2010. (`http://131002.net/blake/blake.pdf`)

[16] K. E. S. Murthy, D. Kavitha and S. Z. ul Huq, "Security analysis of binding update protocols in route optimization of MIPv6," in *2010 International Conference on Recent Trends in Information, Telecommunication and Computing (ITC)*, pp. 44-49, Mar. 2010.

[17] O. Ozen, J. W. Bos, and J. P. Hubaux, "Analysis and optimization of cryptographically generated addresses," *Information Security*, LNCS 5735, pp. 17–32, 2009.

[18] A. Park, *Multithreaded Programming (POSIX Pthreads Tutorial)*, May 10, 2015. (`http://randu.org/tutorials/threads/`)

[19] Polarss, *Finding Collisions in the Full SHA-1*, May 10, 2015. (`http://polarssl.org/`)

[20] T. Pornin, *Comparative Performance Review of Most of the SHA-3 Second-round Candidates*, 2010. (`http://csrc.nist.gov/groups/ST/hash/`

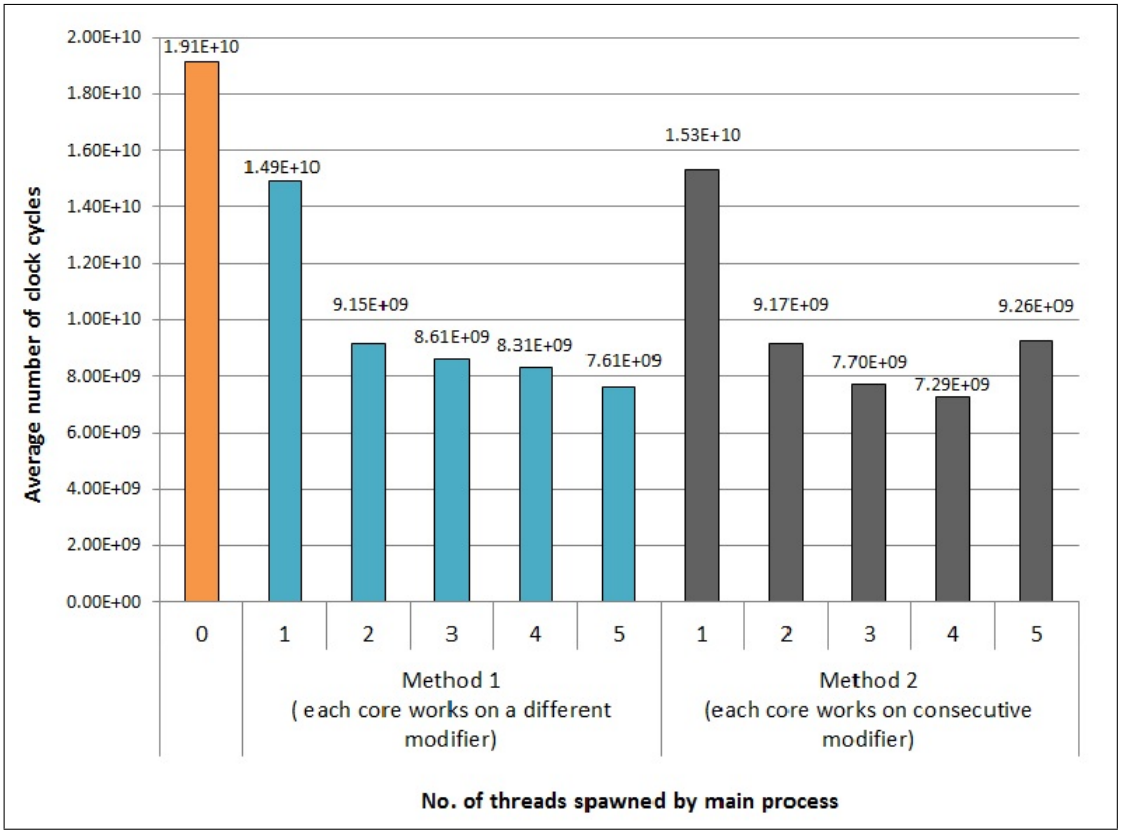

Figure 7: Effect of parallelism on CGA generation algorithm

sha-3/Round2/Aug2010/documents/papers/Pornin-report-sphlib-tp-final.pdf)

[21] H. Rafiee, A. AlSa'deh and C. Meinel, "Stopping time condition for practical IPv6 cryptographically generated addresses," in *Proceedings of International Conference on Information Networking (ICOIN'12)*, pp. 257–262, Bali, Indonesia, Feb. 2012.

[22] AlSa'deh H. Rafiee, A. AlSa'deh and C. Meinel, "Cryptographically generated addresses (CGAS): Possible attacks and proposed mitigation approaches," in *Proceedings of IEEE 12th International Conference on Computer and Information Technology (CIT'12)*, pp. 332–339, Chengdu, Sichuan, China, Oct. 2012.

[23] T. Rajendran and K. V. Sreenaath, "Hash optimization for cryptographically generated address," in *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, pp. 365–369, Bangalore, India, Jan. 2008.

[24] R. L. Rivest, *The MD6 Hash Algorithm*, May 10, 2015. (http://groups.csail.mit.edu/cis/md6)

[25] Scali, *Scali's Openblog: Multi-core and Multi-threading Performance (the Multi-core Myth?)*, June 1, 2012. (http://scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading)

[26] B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, N. Ferguson, S. Lucks and J. Walker, *The Skein Hash Function Family*, ver. 1.3, Oct. 1, 2010. (https://www.schneier.com/skein1.3.pdf)

[27] M. U. Siddiqi, S. Qadir and W. F. M. Al-Khateeb, "An investigation of the merkle signature scheme (MSS) for cryptographically generated address (CGA) signatures in mobile IPv6," *International Journal of Network Security*, vol. 17, no. 3, pp. 311–321, 2015.

[28] R. Triggs, *What to Expect from Smartphone Hardware in late 2014 and into 2015*, 2014. (http://www.androidauthority.com/smartphone-hardware-2015-405022)

[29] G. Xiangyang, Q. Xirong, J. Sheng, S. Guangxue, W. Wendong and G. Xuesong, "A quick cga generation method," in *International Conference of Future Computer and Communication (ICFCC'10)*, pp. 769–773, Wuhan, China, May 2010.

[30] P. Zhang, J. Li and S. Sampalli, "Improved security mechanism for mobile ipv6," *International Journal of Network Security*, vol. 6, no. 3, pp. 291–300, 2008.

Sana Qadir received her MSc in Computer and Information Engineering in 2010. She is cuurently a PhD candidate at the Faculty of Engineering, International Islamic University Malayisia. Her research interests include information security, network security and implementation issues in cryptography.

Mohammad Umar Siddiqi received his B.Sc. and M.Sc. degrees from Aligarh Muslim University (AMU Aligarh) in 1966 and 1971, respectively, and a Ph.D. degree from the Indian Institute of Technology Kanpur (IIT Kanpur) in 1976, all in Electrical Engineering. He has been in the teaching profession throughout, first at AMU Aligarh, then at IIT Kanpur and Multimedia University Malaysia. Currently, he is a Professor in the Faculty of Engineering at International Islamic University Malaysia. His research interests are in coding, cryptography, and information security.

Wajdi Fawzi Mohammed Al-Khateeb received his MSc. Eng. degree in Telecommunications Engineering from the Technical University Berlin in 1968. After graduation he joined the University of Technology, Baghdad and Northern Petroleum Company, Iraq in 1971 as telecommunications engineer where he assumed various professional engineering activities including senior and chief telecommunications engineer until 1993. In 1995, he joined the Department of Electrical and Computer Engineering, International Islamic University Malaysia. Beside his academic activity, he was appointed as leader of consultancy team to plan, design, and supervise the ICT infrastructure project at the Universitys new campuses in Gombak and Kuantan with more than 30 thousand data/voice nodes to support the ICT applications of the University. He was later conferred a PhD in Engineering from IIUM in 2006. Dr. Wajdi is a professional telecommunications and IT engineer with expert knowledge in telecommunications engineering activities gained through 40 years of experience in many telecommunications systems covering: planning, design, consultation, project management and supervision of wide range of communications systems.