Significantly Improved Performances of the Cryptographically Generated Addresses thanks to ECC and GPGPU

Tony Cheneau[a,*], Aymen Boudguiga[a], Maryline Laurent[a]

[a] *Institut TELECOM, TELECOM SudParis, CNRS Samovar UMR 5157, 9 rue Charles Fourier, 91011 Evry, France*

Abstract

Cryptographically Generated Addresses (CGA) are today mainly used with the Secure Neighbor Discovery Protocol (SEND). Despite CGA generalization, current standards only show how to construct CGA with the RSA algorithm and SHA-1 hash function. This limitation may prevent new usages of CGA and SEND in mobile environments where nodes are energy and storage limited.

In this paper, we present the results of a performance and security study of the CGA and SEND. To significantly improve the performances of the CGA, we investigate first replacing RSA with ECC (Elliptic Curve Cryptography) and ECDSA (Elliptic Curve DSA), and second using the General-Purpose computing on Graphical Processing Units (GPGPU). Finally, a performance comparison between different hash algorithms (SHA-256, WHIRLPOOL,...) allows to prepare a better transition for the CGA when SHA-1 will be deprecated.

Key words: CGA, ECC, ECDSA, RSA, Performances, Hash algorithms, GPGPU

Introduction

Neighbor Discovery (ND) is a protocol defined for IPv6 [1] in RFC 4861 [2] and RFC 4862 [3]. ND serves throughout a network link for any IPv6 nodes to determine their neighboring routers, to perform any IPv6 address resolution, and to detect any unreachable neighbors and duplicated addresses. It defines new ICMPv6 (Internet Control Message Protocol - [4]) messages and options. As discussed in [5], the ND protocol suffers from several well known security problems, and is vulnerable to some critical attacks such as the Denial of Service attacks (DoS). To cope with those attacks, the IETF Working Group "SEND" introduced few extensions to the ND protocol, leading to the standardized Secure Neighbor Discovery (SEND) protocol [6] that is essentially based on the Cryptographically Generated Addresses (CGA) [7].

A CGA is an IPv6 address bound to a public key, and it only helps proving that the sender of a SEND message is the real owner of its CGA address. The CGA defines a decentralized mechanism to bind the public key to its owner, and is radically different from the legacy approach which centralizes the binding through an electronic certificate being generated by a centralized unique entity. SEND [6] makes this decentralization possible by the compulsory use of a pair of self-generated RSA keys and an RSA signature option. The CGA also makes use of some electronic certificates but these certificates are only attached to the routers and serve to prove that a router is authorized to announce itself as a router with the declared subnet prefix on the local link. Those CGA works are derived from previous research works relative to SUCV (Statistically Unique and Cryptographically Verifiable) identifier and addresses [8].

The CGA is currently under adaptation to the mobile networks, that is the ND proxying [9] [10] and multihoming [11] issues. That is, first, as required by some mobility protocols [12], a proxy node must be able to announce itself to its neighboring nodes as the owner of the CGA address of a remote mobile node. Second, in a multi-homed scenario, a node can be connected to many Internet Service Providers (ISP) at the same time, thus it is allocated several subnet prefixes, and there is a high interest in making the other neighboring nodes securely identifying the node as a single entity. Further efforts are currently done to adapt CGA to HIP (Host Identity Protocol) [13] and MIPv6 (Mobile IPv6) [12] protocols.

So far, the CGA is strongly associated to the RSA algorithm in the context of SEND. Previously , document [14] proposed to replace the RSA algorithm by a variation of the Feige-Fiat-Shamir (FFS) scheme. This is a great improvement as it reduces the signature generation and verification time. However, it has not offered much about the Public Key and signature sizes. Moreover the FFS scheme as not been standardized yet. Similarly, article [15] offers a more in-depth analysis of the proposal of document [10] and proposes to use Rivest-Shamir-Tauman (RST) ring signature scheme instead of RSA to extend the SEND protocol. While it indeed offers a solution to the ND proxying issues, it does not, however, improve protocol performances at all.

In this paper, we propose another approach, that is, to generate the CGA with the ECC key. Thanks to the underlying mathematical theories of the Elliptic Curve Cryptography (ECC) and the small lengths of the ECC keys, the CGA generation time

*Tony Cheneau: Tel : +33 1 60 76 44 79
Email addresses: tony.cheneau@it-sudparis.eu (Tony Cheneau), aymen.boudguiga@it-sudparis.eu (Aymen Boudguiga), maryline.maknavicius@it-sudparis.eu (Maryline Laurent)

should decrease. In mobile networks, it is of interest decreasing this time as the time is critical, especially during the handover operations that must be realized within few milliseconds. This ECC approach was also motivated by the very promising results that were already obtained by using ECC in the sensor and ad hoc networks that are known for their resource limitation.

The CGA also relies on the SHA-1 hash function that is likely to be broken in a near future ([16] and [17]). To deal with this issue, the alternatives to SHA-1 must be introduced into the CGA. The RFC 4982 standard [18] gives the mechanisms to support multiple hash algorithms in CGA, but it does not recommend any of the current hash algorithms. The selection of the hash function is of high importance. First, it fixes the security level of the CGA, and this criterion should be the main concern. Second, it has a direct impact on the CGA performances, and as such, depending on the resource-constrained device implementing the CGA, a limited set of hash functions is able to apply. This article investigates the performances of some eligible hash functions, and gives a comparative analysis.

The objective of the paper is to analyse the experimental performances of the CGA and also the SEND related operations, first when using RSA vs ECC, and second for several eligible hash algorithms. For more realistic results, the measurements were performed on three devices with different capacities: a desktop, an embedded device (Nokia N800), and a GPGPU graphical processing unit. The results are presented under some synthetic benchmarks.

This article starts by reminding some CGA (section 1) and SEND related (section 2) generalities. Section 3 presents the testing environment and the different timing measurements. We then analyse the performances of the CGA in section 4 and the performances of the SEND protocol in section 5. We further investigate on these performances in an embedded context in section 6. In section 7, we focus on the impact of some new hash functions that are likely to replace SHA-1 in the CGA generation processing in the near future. We further improve the performances of the CGA generation in section 8 using the cheap hardware solution offered by the General-Purpose computing on Graphical Processing Units (GPGPU). Finally, section 9 concludes this article.

Glossary

- *CGA*: Cryptographically Generated Addresses
- *ECC*: Elliptic Curve Cryptography
- *ECDSA*: Elliptic Curve Digital Signature Algorithm
- *GPGPU*: General-Purpose Computing on Graphical Processing Units
- *NA*: Neighbor Advertisement
- *NDP*: Neighbor Discovery Protocol
- *NS*: Neighbor Solicitation
- *RS*: Router Solicitation
- *RA*: Router Advertisement
- *RDTSC*: Read Timestamp Counter
- *SEND*: Secure Neighbor Discovery

1. CGA generation and verification

1.1. CGA generation

The CGA generation algorithm consists in computing two specific hashes. Hashes are computed over part of the *CGA Parameters Data Structure* presented on Figure 1, and according to the fixed SEC value between 0 and 7 as described in [7]. This SEC value serves to determine the robustness of the CGA against some brute-force attacks that might be attempted on the hash algorithm. The current hash algorithm used with the CGA is SHA-1, and the Public Key cryptography algorithm used with SEND is RSA. Both of these algorithms are hard coded in the protocols, and no transition mechanism toward another algorithm is yet defined.

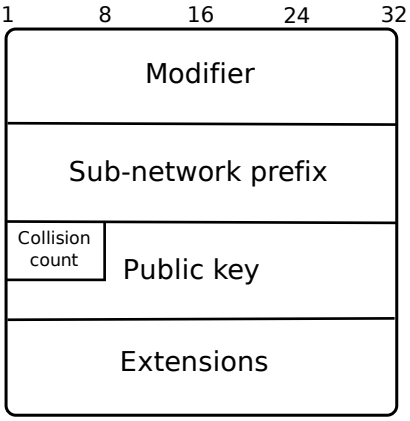

Figure 1: CGA Parameters Data Structure

The *CGA Parameters Data Structure* presented in Figure 1 is the result of the concatenation of a 128 bit random number called *modifier*, the *subnet prefix* associated to the generated address, the collision count, the DER encoded *Public Key* of the node and some (not yet used) optional extensions. The *modifier* is used during the CGA generation to strengthen the robustness of the hashed values and to enhance the address privacy by adding some randomness to the address generation processing (two generation processes with the same *Public Key* will result in two different addresses).

The CGA algorithm relies on the SHA-1 algorithm to compute the two hash values presented in Figure 2:

- *hash1* contains the 64 leftmost bits of the SHA-1 digest computed over the *CGA Parameters Data Structure*. These 64 bits form the CGA's interface identifier [7] after the 3 leftmost bits are set to the 3 bits of the encoded SEC value and the *U* and *G* bits are set to 0. Please refer to [19] for further information on *U* and *G*;

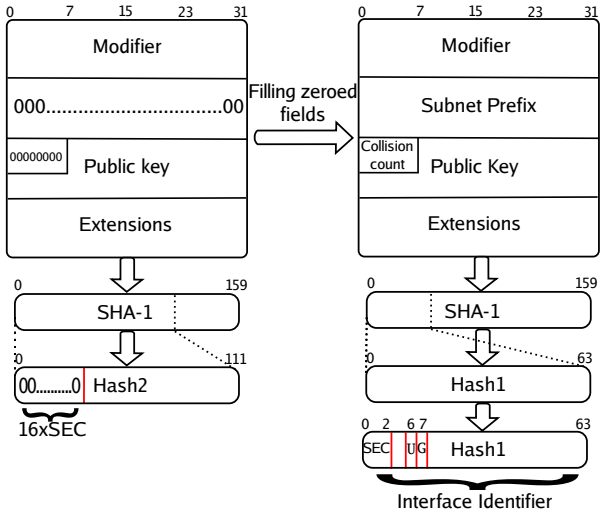

Figure 2: Definition of *hash1* and *hash2*

- *hash2* contains the 112 leftmost bits of the SHA-1 digest computed over the *CGA Parameters Data Structure* with a prefix and a collision count equal to 0, and satisfying the condition that the 16×SEC leftmost bits of the digest are equal to 0. The SEC value increases the computational power/time on the generator and also on the attacker. It is used to increase the cost of a brute force attack against the CGA generation algorithm [7].

The CGA generation algorithm can be split into the two following phases:

1. The computation of *hash2* and thus the *final modifier* generation. This phase is time consuming, and it is generally computed only once during the CGA generation (and possibly before the node joins any network). A *modifier* is contained in the *CGA Parameters Data Structure*, it serves to generate the hashes of the CGA, and it is incremented by 1 each time that the computed hash value (*hash2*) does not satisfy the requirement: 16×SEC leftmost bits equal to 0. The final value of the *modifier* is stored for later use in the *CGA Parameters Data Structure*. We refer to this value in this document as the *final modifier*.
2. The computation of *hash1* and the interface identifier construction. It is computed each time a new CGA for a new subnet prefix is required.

The CGA generation algorithm as depicted on Figure 3 starts with a random *modifier* value, then *hash2* is computed (possibly many times) until verifying that its 16×SEC leftmost bits are equal to 0. Then the collision count is set to 0 and *hash1* is computed. The interface identifier is derived from this *hash1* value. The CGA is formed by concatenating the subnet prefix with the interface identifier. Finally the Duplicate Address Detection processing (DAD) as defined in [3] is launched to verify that there is no address collision. If an address collision occurs, the *collision count* field is incremented by 1 and *hash1* is recomputed to form a new interface identifier. This step is performed maximum three times. If during the third trial, a collision is detected (again), the node gives up attempting to generate an address.

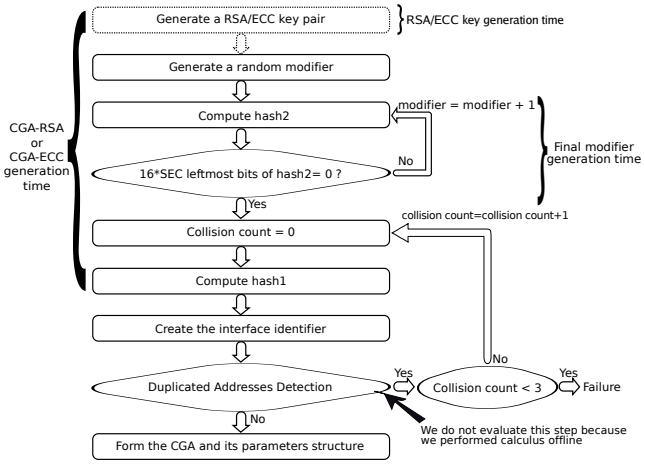

Figure 3: CGA generation algorithm

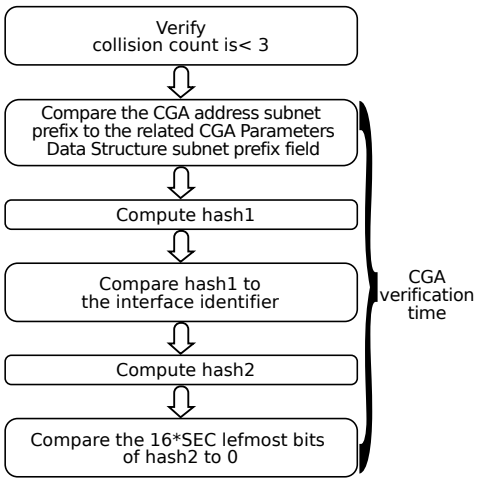

Figure 4: CGA verification algorithm

1.2. CGA verification

When a node receives a packet from one of its neighbors which use a CGA address, it has to execute the CGA verification algorithm. The received packet should contain an ICMPv6 *CGA option* carrying the final *CGA Parameters Data Structure*.

The verification algorithm presented in Figure 4, starts by checking that the *collision count* value is less than 3. The *subnet prefix* from the sender's IPv6 source address (obtained from the packet header) is then compared to the *subnet prefix* obtained in the *CGA Parameters Data Structure*. Next, *hash1* is computed and compared to the interface identifier, omitting bit 0 to bit 2, bit 6 and bit 7 (encoding respectively the SEC value, *U* and *G*). Finally, SEC is extracted from the interface identifier (the 3 leftmost bits) and *hash2* is computed. Its 16×SEC leftmost bits

are checked to be equal to 0. If any of the previous tests fails, the CGA address is considered as unsecured.

2. CGA related operations in SEND

This section first presents the SEND basic protocol and then it proposes a rough solution to integrate the use of ECC and ECDSA into SEND.

2.1. SEND brief description

SEND is defined to secure the neighbor discovery mechanism against the DoS and replay attacks [5], and also rogue routers. It helps detecting rogue routers as it enables to distinguish valid sources of information from invalid ones. Some SEND extensions are appended to the ND messages, that is, the ND messages - Neighbor Solicitation (NS), Router Solicitation (RS), Neighbor Advertisement (NA) and Router Advertisement (RA) - that are usually exchanged between the hosts and the routers (RS, RA) and between the hosts (NS, NA). SEND defines two new ICMPv6 messages and six new ND options [6].

SEND relies on the CGA to prove that the sender is the owner of the declared CGA address. A simple host announcing itself over the link is required to sign the RS, NA and NS messages with its private key, thus proving the receivers that it owns the announced public key and thus the associated CGA. A router playing the key role of relaying the traffic back and forth between the local hosts and the infrastructure network is required to sign the RA messages with its private key and also to strongly prove that it is the owner of the associated public key and it is authorized to announce the declared subnet prefix. This latter mechanism is known as ADD (Authorization Delegation Discovery). It relies on an electronic certificate issued by a trusted third party entity. To permit the host to check the validity of the certificate, the host and the router should agree on a common trust anchor, that is a certification authority that is trusted by both of them and that belongs to one of the certification path of the router's certificate. The ADD mechanism is designed to thwart the threat of rogue routers on the unsecured links, as a trusted third party is certifying the role played by the router, and the subnet prefix it has to announce. The ADD exchanges occur at the very beginning of a session between the local host and the router. A valid certificate guarantees to the host that the router (i.e. its CGA address) is bound to a public key and a subnet prefix. The SEND messages signed with the private key serve to authenticate the router. ADD introduces two new SEND messages - Certification Path Solicitation (CPS) and Certification Path Advertisement (CPA) - that are exchanged between the local hosts and the router.

When a new node joins a network, it starts sending a RS message to get all the information about the possible subnet prefixes in use. Prefixes are required for later generating its CGA. Upon receiving a response RA from a router, the host has to check the validity of the router's signature, and asks the router to send back a certificate that can be checked as valid by itself. The host solicits the router with a CPS message (Figure 5) including a list of its own trust anchors in a *Trust Anchor Option*. Finally the router responds with a CPA message containing the selected trust anchor and a list of the certificates forming a chain from its certificate to the trust anchor's one (in *Certificate Option*). Then the host is able to perform an offline verification of the certificate validity.

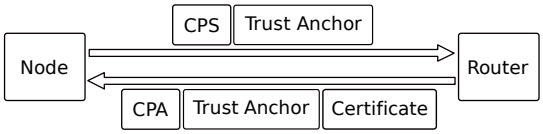

Figure 5: Exchange of messages Certification Path Solicitation/Certification Path Advertisement

As depicted in figure 6, the ND messages are protected thanks to four SEND options. The solicitation message NS or RS sent by a node includes the *CGA Option*, the *RSA Signature Option*, the *Timestamp Option* and the *Nonce Option*. The *CGA Option* contains the *CGA Parameters Data Structure* and enables the receiver to get the public key of the sender node. The *RSA Signature Option* includes the RSA signature over part of the ND message that certifies that the message was generated by the owner of the CGA (bound itself to the public key). The *Timestamp Option* protects the message against the replay attacks. The *Nonce Option* serves to bind a solicited ND message to its advertisement response, and also to detect replayed messages.

An advertisement message NA or RA can be generated by either a host or a router. It includes the same options as in the solicitation messages that serve to authenticate the origin of the message and to detect replays. These messages can be sent when a node changes one of its (physical or CGA) addresses or when a router multicasts new information about its prefixes. In case of an unsolicited message, the message is not bound to a solicitation message and the *Nonce Option* is omitted.

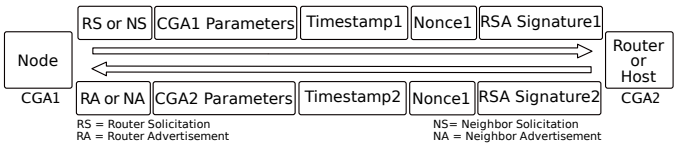

Figure 6: SEND options usage

2.2. ECC / ECDSA integration into SEND

The ECC cryptography [20] is promising for the resource-constrained networks such as the wireless sensor and ad-hoc networks. ECC and its ECDSA signature algorithm offer the following advantages:

- ECDSA provides fast computation of the signature;
- The ECC/ECDSA code size is pretty short, and suits well small capacity devices;

- The ECC keys are shorter than RSA keys for the same security level. This property is very interesting as the public keys are usually transmitted to the receiver to let it verify a signature. Shorter is the key, less bandwidth is consumed and more energy is saved by the devices. Table 1, extracted from document [21], reminds the equivalence between ECC keys and RSA keys length in term of security level.

RSA key length (bits)	ECC key length (bits)
1024	163
2048	224
3072	256
7680	384
15360	512

Table 1: RSA and ECC key length equivalence from a security level point of view

The objective of this section is to describe the rough modifications we brought to CGA and SEND to get a first SEND implementation with ECC and ECDSA, for the performance measurement purpose. We propose the following modifications to the initial SEND specifications:

- Replace the RSA signature algorithm used in SEND [6] with ECDSA;
- Replace the DER encoded RSA key with a DER encoded ECC key in the *CGA Parameters Data Structure* (Figure 1).

We propose to keep the *RSA signature option* format as defined in the RFC [6], even if it contains an ECDSA signature. The verifier is adapting the signature verification to the type of the public key presented in the *CGA Parameters Data Structure*, as it verifies the public key during the CGA verification, and before the signature verification.

These modifications to the SEND protocol are currently revised to define a backward compatible and interoperable technical solution, but this extra work has no impact on the performance measurement results. For further explanations, the interested readers can refer to the IETF drafts [22], [23] and [24].

3. Testing environment

Generally, when we need to recompute a CGA (e.g. in case a node is moving from one network to another), only the subnet changes: this change involves recomputing only *hash1* (and not *hash2*). Also, the *Public key* does not have to be recomputed each time. However, in our testing platform, we are measuring the time needed for the very first CGA generation. As such, each time measurement includes the time needed for generating a new *Public Key*, and so a new CGA. These measurements are also interesting to assess for the CGAs to support the IP Privacy Extension (cf. [25] and [26]) which requires fast CGA generation.

The following time measurements are evaluated in the next sections:

- the *CGA generation time* (or *total CGA generation time*) is the complete duration of the generation of the CGA, from the generation of the Public Key up to the computation of the interface identifier including *hash1* and *hash2* calculus;
- the *CGA verification time* is the time spent verifying the CGA, as described in section 1.1, by testing the CGA address against the *CGA Parameters Data Structure*. It does not include the *RSA Signature Option* verification;
- the *Final modifier generation time* corresponds to the time spent computing a *hash2* value that matches the condition on the first bits;
- the *RSA/ECC key generation time* is the time for generating an RSA or ECC key;
- the *RSA/ECDSA signature generation time* is the time spent for computing the RSA or ECC digital signature. In the SEND protocol, it corresponds to the generation of the digital signature carried into the *RSA Signature Option*;
- the *RSA/ECDSA signature verification time* is the duration of the RSA or ECC Digital Signature verification. In SEND, it corresponds to the time spent to verify the Digital Signature contained in the *RSA Signature Option*.

To evaluate the CGA generation and verification time, we use an assembly instruction RDTSC (Read Timestamp Counter) that returns the internal processor clock value expressed in CPU cycles. By comparing two executions of this function, we can deduce the elapsed time, in CPU cycles, and get an important precision even on small measurements. Then by dividing this elapsed cycles by the CPU clock frequency, we get the elapsed time in seconds.

The major drawback of this method is that the monitored processing can be interrupted by the scheduler whose time will be accounted for. To avoid too many interruptions, a Linux kernel was used in the single mode, so the number of processes and their interactions with our synthetic test program remain very limited.

To determine the appropriate number of measurement samples, our first idea was using the Monte Carlo estimator, but this method was proved to be inappropriate for our application as the evaluated time measurements were random. This randomness is mainly due to the randomness of the key generation and the *final modifier* calculus. That is why, we then decided to perform the testing on a fixed 10000 samples. A later calculus of variance proves that 10000 samples are enough as the variance remains low for all our measurements.

Note that we used an RSA public key exponent equal to 3 during our testing (for a faster signature generation).

Note also that we wanted to use 512-bit ECC keys which are equivalent to 15360-bit RSA keys. However, due to a limitation

in the OpenSSL library which does not implement the 512-bit ECC key, we then decided to round to the higher value 571-bit ECC key. This security level difference is depicted into the tables under the "15360+" RSA key.

4. CGA generation and verification time on a PC

4.1. CGA generation

This section compares the CGA generation time measurements between the RSA key and the ECC key when SEC is equal to 0 and 1. Table 2 gives the mean values computed over the 10000 samples, on a Pentium 4 running at 2593 MHz.

SEC value	0				
RSA key length (bits)	1024	2048	3072	7680	15360+
Equivalent ECC key length (bits)	163	224	256	384	571
CGA-RSA generation time	0.165630	1.047319	3.415941	91.562634	-
CGA-ECC generation time	0.009535	0.010031	0.015732	0.023823	0.129787
SEC value	1				
CGA-RSA generation time	0.302924	1.218577	3.631235	91.957422	-
CGA-ECC generation time	0.103472	0.103276	0.108027	0.135732	0.265401
Final modifier gen. using RSA	0.137296	0.171260	0.215295	0.394790	-
Final modifier gen. using ECC	0.093938	0.093247	0.092297	0.111910	0.135616

Table 2: CGA generation time and *final modifier* generation time using RSA and ECC keys on a Pentium 4 at 2593 MHz (in seconds)

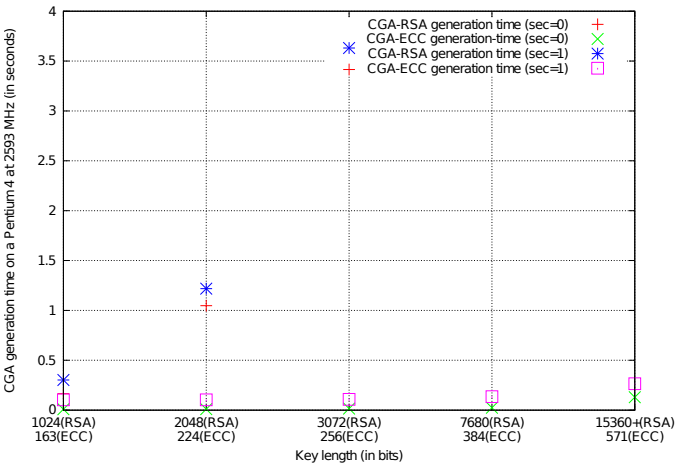

Figure 7: Comparison of CGA generation time between RSA and ECC keys

As depicted on Table 2, the CGA generation time using a 571-bit ECC key with a SEC equal to 1 is shorter than the generation time using a 2048-bit RSA key with a SEC equal to 0. We can deduce that ECC increases the security level while decreasing the CGA generation time. Figure 7 clearly indicates the gap between the generation of RSA and ECC based CGA.

As expected, the generation time for SEC equal to 1 is greater than the generation time for SEC equal to 0. The difference is due to the extra time spent for SEC equal to 1 to generate the *final modifier*, that is, to compute several *hash2* values until the 16×SEC leftmost bits of *hash2* are equal to 0. The generation time for SEC equal to 0 includes only the key generation time and the *hash1* computation time. For SEC equal to 1, there is an average of 2^{16} hash computations before finding the *final modifier*. This value is confirmed by the formula 1 giving the number of trials necessary to get the *final modifier* according to the SEC value. This formula is adapted from the formula of the complexity of a first pre-image attack on a hash function [27].

$$f(SEC) = 2^{16 \times SEC} \quad (1)$$

From the formula 1, we also show that generating a CGA with a SEC value of 2 or higher is not computationally feasible in a timely fashion. A test on an unrepresentative set of 10 samples gave an average of 1.8 hours to generate 1024-bit long RSA key-CGA address for a SEC value of 2. This result is confirmed with Formula 1 which gives scaling factor of 2^{16} between the duration of a *final modifier* calculus with SEC value of 1 and a *final modifier* calculus with SEC value of 2. If 0,137 sec is the average duration of the *final modifier* generation for the 1024-bit long RSA keys, and SEC value of 1 (cf. Table 2), we can deduce a synthetic estimation of the *final modifier* calculus duration of 2.5 hours vs 1.8 hour, which is not acceptable in a real usage scenario.

Table 2 also depicts that the *final modifier* generation time is shorter with the ECC keys than with the RSA keys. This is due to the *hash2* generation time which is shorter with ECC because the SHA-1 hash function is calculated over the *CGA Parameters Data Structure*, whom length heavily depends on the public key length (as explained in the following section).

4.2. Theoretical analysis of the SHA-1 impact on the performances

This section proposes to analyse the impact of the SHA-1 hash function (as required by SEND) on the CGA generation time measurements of table 2. As stated previously, the two digests *hash1* and *hash2* have to be computed when generating a CGA. In order to further understand how significant is the impact of SHA-1 on the CGA generation time, when an ECC key pair and an RSA key pair is used, let us first analyse the internal functioning of SHA-1.

The SHA-1 algorithm is divided into the two following major phases: the preprocessing and the hash computation [28]. The preprocessing phase starts by doing an aligning operation over the input message, then parsing the previously padded message into 512-bit blocks and finally setting the initial hash values. Let us suppose that the input message has been parsed into N blocks of 512 bits. The hash computation applies on each individual block and is composed of 4 rounds, each of them containing 20 steps. This hash computation time is constant for every block, so we can infer that the complexity of this computation is bound to the number of blocks and can be estimated to $O(N)$.

In the CGA context, SHA-1 applies to the *CGA parameter data structure*, and the length of that structure has a direct impact on the computation time of the digests. All the fields of the *CGA parameter data structure* but the DER encoded *Public Key* field are fixed length. As such, the table 3 reports the length of the structure and the number of the corresponding generated

512-bit blocks, when using different key lengths. These blocks have been calculated using the method proposed in [28].

RSA key length (bits)	1024	2048	3072	7680	15360
DER encoded RSA public key length (bytes)	160	292	420	996	1956
CGA parameters data structure Length (bits)	1480	2536	3560	8168	15848
Number of 512 bits blocks	4	6	8	17	32
Equivalent ECC key length (bits)	163	224	256	384	571
Octets encoded ECC public key length (bytes)	66	80	88	120	170
CGA parameters data structure Length (bits)	728	840	904	1160	1560
Number of 512 bits blocks	2	2	2	3	4

Table 3: CGA parameter data structure length (in bits) and corresponding number of blocks

From the Table 3, we deduce that the *hash1* and *hash2* computation times belong to the same range of time duration for a 1024 bits RSA key and a 384 bits or a 571 bits ECC key. We can check that this result is verified in the Table 2. The same remark applies for ECC keys with 163, 224 and 256-bit length which are spilled over 2 blocks.

It is now clear that the generation time of *hash1* and *hash2* is greater with RSA keys than ECC keys because RSA generally generates more blocks (especially when the RSA key length is greater than or equal to 2048 bits). The same remark applies to the CGA verification procedure, as we will see in the following section.

4.3. CGA verification

The CGA verification is the first step performed in the SEND protocol as soon as receiving a ND message coming from a CGA address. It is supposed to be a lightweight check before the (heavier) RSA signature verification. The CGA verification algorithm contains two main computational steps which are *hash1* and *hash2* verification. The other steps are only quick comparisons. The Table 4 depicts these results.

SEC value	0				
RSA key length (bits)	1024	2048	3072	7680	15360+
Equivalent ECC key length (bits)	163	224	256	384	571
CGA-RSA verification time	0.000004	0.000005	0.000006	0.000009	-
CGA-ECC verification time	0.000004	0.000004	0.000005	0.000005	0.000005
SEC value	1				
CGA-RSA verification time	0.000005	0.000006	0.000007	0.000013	-
CGA-ECC verification time	0.000003	0.000003	0.000003	0.000004	0.000005

Table 4: CGA verification time on a Pentium 4 at 2593 MHz(in seconds)

Similarly to the previous section, as ECC provides shorter keys than RSA, ECC offers, as expected, some better timing performances.

5. RSA and ECDSA signature generation and verification times on a PC

This section presents the results relative to the RSA / ECDSA signature generation and verification. For testing purpose, we simulate a SEND scenario where we create a SEND protected NS message, by first generating randomly a message having the same length than an NS message with all the SEND options (but the *RSA signature option*). The length of the messages varies depending on the size of the public key. The resulting messages in use for the signature benchmark are crafted to be of the same size as the ones that would be emitted by an actual SEND [6] implementation.

RSA key length (bits)	1024	2048	3072	7680	15360+
Equivalent ECC key length (bits)	163	224	256	384	571
RSA signature generation time	0.004568	0.022275	0.053676	0.609052	-
ECDSA signature generation time	0.002219	0.002588	0.004439	0.007338	0.042402
RSA signature verification time	0.000069	0.000165	0.000321	0.001422	-
ECDSA signature verification time	0.004398	0.003039	0.005352	0.008775	0.084884

Table 5: RSA and ECDSA signature generation and verification time on a Pentium 4 at 2593 Mhz (in seconds)

Table 5 shows that the RSA signature generation time increases with the key length. This is the expected result as the signature calculus depends on the modulus length. This table also highlights that the ECDSA signature verification procedure always takes more time than its generation. As such, the experimental results confirm the expected behavior of ECDSA [29]. However, neither RSA or ECDSA based signature generation and verification exceed 0.5 seconds. Values exceeding 0.5 seconds would have caused the Duplicate Address Detection (DAD) process to fail as it expects to send and receive 2 Neighbor Discovery messages below 1 second (see [2]) , involving 2 signature generations and 2 signature verifications.

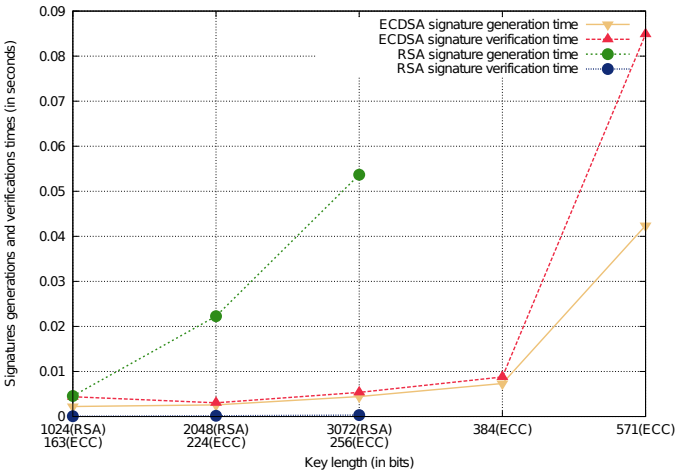

Figure 8: Comparison between the RSA and ECDSA signature generation and verification times on a Pentium 4 at 2593 MhZ

Figure 8 depicts graphically the difference between the signature verification and generation for ECDSA and RSA. For the signature verification, RSA is even faster than ECDSA. Due to this time difference, there could be advantages using one of the RSA or ECDSA signature depending on the SEND model that applies:

- in the one-to-one model, when two nodes are directly communicating together (e.g. to learn each others' link-layer

addresses), with big key sizes, ECDSA is mostly superior to RSA on an average of signature generation/verification operations.

- in the one-to-many model, when a router is multicasting a lot of information on the link (e.g. to refresh the current prefix lifetime), RSA is best suited because the RSA signature verification is faster than the ECDSA's and globally more nodes are likely to verify the signature rather than to generate it.

Unfortunately, we were not able to monitor a real SEND deployment to evaluate which model is more widely used. Heterogeneous networks where routers are using RSA algorithm and where hosts are using ECDSA could provide a good compromise.

6. Performances on a Tablet PC

To have an idea of the performances of the CGA generation times with ECC vs RSA in a resource-constrained environment, we performed the CGA generation tests on a Tablet PC. We used a Nokia N800 with an ARMv6-compatible processor running at 400 MHz.

As far as we know, there is no equivalence of the assembly instruction RDTSC on ARM CPUs, so we decided to measure the elapsed time for the CGA generation based on the function *gettimeofday()* from *time* library. Note that this latter function is imprecise as it measures the global generation time (including the CPU time consumption of other background processes).

SEC value	0			
RSA key length (bits)	384	512	1024	2048
(Total) CGA generation time	0.651353	1.004133	4.699501	35.484486
RSA key generation time	0.637553	0.990302	4.685756	35.470764
RSA signature generation time	0.012324	0.021701	0.114592	0.711035
RSA signature verification time	0.000522	0.000712	0.001837	0.005783
Final modifier generation time	0	0	0	0
SEC value	1			
Final modifier generation time	1.761618	1.754493	2.817053	3.873690

Table 6: CGA/RSA generation time on a Nokia N800 (in seconds)

SEC value	0				
ECC key length (bits)	163	224	256	384	571
Total CGA generation time	0.148385	0.169551	0.308717	0.461143	1.866542
ECC key generation time	0.079611	0.086993	0.135157	0.186858	0.654368
Final modifier generation time	0	0	0	0	0
ECDSA sig. generation time	0.028778	0.037199	0.085092	0.138247	0.604534
ECDSA sig. verification time	0.056505	0.045464	0.102846	0.168529	1.207743
SEC value	1				
Final modifier generation time	1.765540	1.760574	1.760952	2.287539	2.788008

Table 7: CGA/ECC generation time on a Nokia N800 (in seconds)

Although the RSA keys of size greater than or equal to 1024 are recommended in [30], we decided for the sake of the completeness to include shorter keys in our tests. We used the 384 and 512-bit RSA keys to evaluate the key size impact on the generation time.

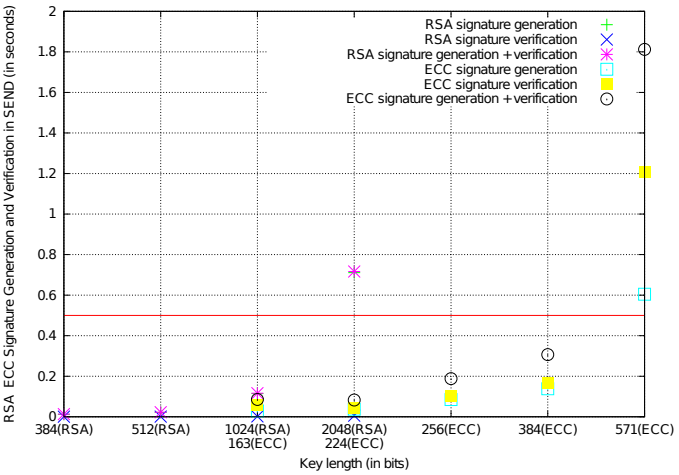

Figure 9: Comparison between CGA/RSA and CGA/ECC signature generation and verification on a Nokia N800

Table 6 illustrates that RSA is too slow to generate a strong enough CGA in a timely fashion. In the case where SEC is equal to 0 (i.e. *hash2* is not computed), we can see that the generation time with 1024-bit RSA keys (which should be at least the default secure size) is greater by 3 seconds than the shorter 384 and 512-bit RSA keys. Given those results, we deduce that it will not be possible to generate on the fly new RSA-based CGA on lightweight mobile nodes. This prevents the use of solutions based on document RFC 4941 [25].

To improve the CGA generation, we recommend to use some pre-generated keys to avoid the key generation delays, so the CGA generation time becomes dependent only on the *hash1* computation time which is about 10^{-4} second. It should also be noted that the pre-computation involves some key storage, that could lead to a security issue when the nodes cannot rely on a secure storage solution.

In the case where SEC is equal to 1, we can note that the *final modifier* computation time is at least 1 second. This time duration is too long in a mobility context. Furthermore, for SEC equal to or greater than 1, we can deduce that the CGA cannot be computed on the fly on the resource-constrained devices. However, since the *final modifier* is computed only once during the CGA generation, it should not affect the handover delays, requiring only *hash1* recomputation to fit to a new subnet prefix.

Table 7 also shows, for SEC equal to 1, that the CGA generation with ECC keys cannot be executed in a timely fashion. The *final modifier* generation time difference we observe between RSA and ECC is mainly due to the key length difference.

In Table 7, we notice that ECC-based CGA shows great improvement for the key generation delays. For larger key sizes, the generation time is smaller than for RSA, but is still too much important for use on a lightweight device. Smaller keys below 256 bits for SEC value of 0 provide a generation time less than 0.15 second and this can help the whole CGA generation pro-

cessing to become seamless to the user.

As stated in the previous section, given the usage scenario (one-to-many or one-to-one), RSA could have some advantages over ECDSA. However, this is a first sight conclusion as the impact of ECDSA shorter keys on the energy consumed during wireless transmission should also be taken into consideration. We believe it could be another point in favor of the use of ECC in CGA. Also note that some RSA and ECC key size can not work on these lightweight devices. As outlined in Figure 9, values over 0.5 second (over the line) indicate the key sizes that do not perform a Duplicate Address Detection process in a timely fashion.

To conclude this section, we recommend that lightweight devices in a mobility context make use of CGA with ECC keys and SEC value equal to 0 when the address cannot be pregenerated beforehand.

7. Influence of a hash function on the CGA generation time

Due to the security flaws ([16] and [17]), SHA-1[28] will soon be phased out by the NIST [31] and it is recommended to replace it with SHA-2 family hash functions. Some on-going works in the "CGA and Send maIntenance" Working Group in the IETF are providing some insights about these security issues and how CGA can be generated with a different hash algorithm [32].

For these reasons, we evaluated the performances of some hash algorithms in the CGA context. Tests are performed on: SHA-256, SHA-512 [28] (current hash algorithms proposed by the NIST), RIPEMD-160 [33], TIGER2 [34] and WHIRLPOOL [35]. SHA-224 and SHA-384 are not evaluated as they are truncated versions of respectively SHA-256 and SHA-512. SHA-512, TIGER2 and WHIRLPOOL are benefitting from our 64-bit processor architecture, which are very popular in recent workstations and servers. Except for TIGER2 and WHIRLPOOL, for which we use available online implementations [36] [35], all the algorithm implementations we used are part of the OpenSSL library [37].

For each hash algorithm, we evaluated the *final modifier* generation time over a basic *CGA Parameters Data Structure* (Figure 1) when generating the RSA-based CGA with a SEC value equal to 1. It helps evaluating the influence of the hash functions on the CPU intensive *hash2* calculus. The comparative results are given in Figure 10. Remind that SEC=1 implies an average of 2^{16} hash calculations (cf. Formula 1).

Figure 10 show that the *final modifier* generation time depends on the key length. This is normal as the hash is computed over the *CGA Parameters Data Structure* that contains the *Public Key*. SHA-256 and SHA-512 meet our expectations as they are known to be slower than SHA-1. SHA-512 benefits from our 64-bit architecture and actually offers better performances than the SHA-256 hash function. WHIRLPOOL [35] as already stated in the literature [38], is known to be slower than SHA-256. However, the counter-performances here are likely to be more related to a lack of optimization in the implementation.

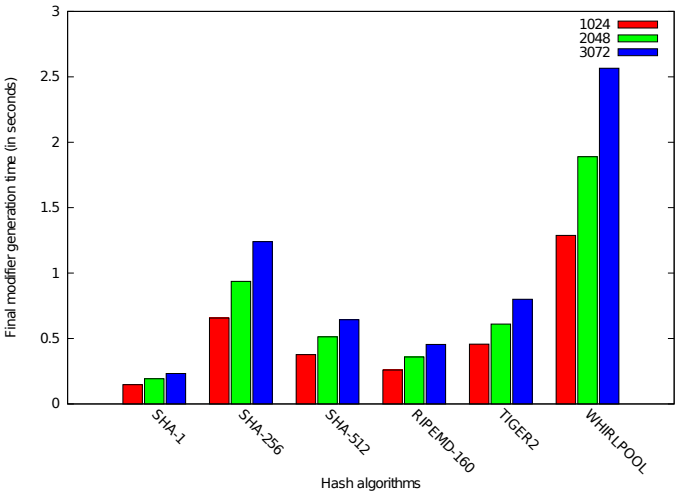

Figure 10: Hash function influence on the *final modifier* generation time on a Pentium 4 running at 2400 MHz (in seconds) with SEC=1

We remark that the most efficient algorithm concerning the *final modifier* generation time is the currently used SHA-1 algorithm. Next, RIPEMD-160 presents also interesting performances. TIGER2 performances are close to RIPEMD-160's ones, but this is mostly due to our 64-bit architecture.

Currently, SHA-256 and the whole SHA-2 family (SHA-384, SHA-512...) are being reviewed among the IETF and should be proposed as the next hash algorithm standard instead of SHA-1 for CGA. This decision is not based on the performances: in our comparative results, we clearly see that SHA-256 and SHA-512 are not the fastest ones. The choice is based on the security level and robustness provided by SHA-2 family that has received a thorough analysis by the NIST and among the cryptographic community. This choice will however slow down the CGA generation processing.

8. Improved performances with the General-Purpose computation on GPU

Nowadays, inexpensive graphic cards shipped with new end-user computers tend to be more and more powerful and are GPGPU enabled. In the contrary, we cannot expect an end-user to acquire some cryptographic accelerator cards. We evaluate in this section the benefits offered by the General-Purpose computations on GPUs (GPGPU), by parallelizing the CGA generation algorithm, and we analyse possible gain using the widely deployed recent graphic card generation.

We ported the lightweight XySSL's [39] SHA-1 implementation into the NVIDIA CUDA [40] platform. The CUDA framework offers a C-like programming language to perform the parallel operations on graphic cards and is relatively easy to use.

The main idea here is to parallelize the *final modifier*'s SHA-1 computations. As explained in section 1.1, the *hash2* computation is the most consuming task of the CGA generation algorithm. Multiple SHA-1 hashes are computed over the *CGA*

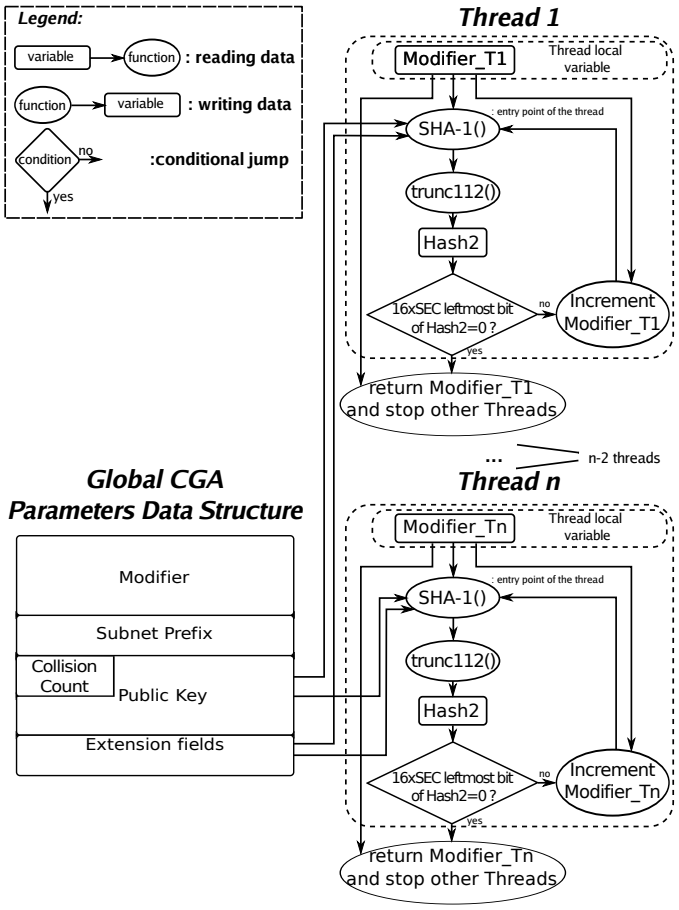

Figure 11: Hash2 parallel computation processing using GPU

Parameters Data Structure which is slightly modified each time the *modifier* value is incremented. Basically, we suggest testing multiple *modifier* values at once. We barely modify the CGA's generation algorithm [7] to achieve parallelization as shown in Figure 11.

The results are obtained using NVIDIA Geforce 8600GT and NVIDIA Geforce 8600GTS graphic cards (CPU type has only little influence here). These two cards have exactly the same number of Multiprocessors (used to compute data) but the GTS version has higher GPU and memory frequencies (GTS has a GPU clocked to 675 MHz and memory running at 1000 Mhz while the GT is limited respectively to 540 Mhz and 700 Mhz). Due to these low end graphic cards' limitations (small shared memory, low number of registers, etc.), our portage of SHA-1's code does not fit completely in the card's "*fast*" memory and some data must be spilled over the computer local (slow) memory, involving performance losses. This drawback is partially mitigated when scheduling multiple threads to hide the memory latency.

We performed, on each card, 500 calculus of *final modifier* with CGA having SEC value equal to 2 and a 1024-bit long RSA key. An average of 1351 s (approx. 22 min) was spent to compute the *final modifiers* on a NVIDIA 8600 GT, vs 852 s (approx. 14 min) on the NVIDIA 8600 GTS. The time difference is due to the GPU and memory clock frequencies. The standard deviation for the NVIDA 8600 GT is 1300 seconds (almost the same as the mean value), which highlight the high randomness of the address generation process. These results should be compared with the approximative estimation of 2.4 hours on a Pentium 4 (see section 4). With the 8600 GTS, the speed increases by a factor of 10. This is a serious improvement, however, 14 min to generate a CGA address with a SEC value of 2 is still too long for an end-user.

We were not able to test out the latest NVIDIA Geforce GT200 series card, but the authors believe that this series can perform at least 7 times faster than the tested NVIDIA. This conjecture is held on the basis that new GT200 series contain at least 7 times more Multiprocessors, while having faster GPUs, more registries and a new memory management mechanism. With GPGPU, CGA with a SEC value of 2 could become more widely used in the next upcoming years as their generation time could drop to few seconds.

There are currently some talks at the IETF for using the DHCPv6 servers to generate CGA on behalf of the computationally limited nodes [41]. We can further improve this proposal by embedding graphic cards, like the ones presented here, into routers. This will permit decreasing the CGA generation time. The very same graphic cards will also be able to serve as cheap cryptographic accelerators for other needs of the router.

As a conclusion of the section, we note that GPGPU could be used in conjunction with the multi-core CPUs to further improve *hash2* calculus. We also note that the calculus duration of a CGA with a SEC value of 3 is supposed to be in average 2^{16} times longer than the one with a SEC value of 2, which means weeks of calculus with the current hardware. Unless another big technological jump improves the performances, we consider the SEC value above 3 to be out of reach of modern computers.

9. Conclusion

CGA have been initially defined with RSA and SHA-1. RSA with key length equal to or greater than 1024 bits is recommended by the NIST for a medium security level. The generation time and the size of such keys make the use of the RSA-based CGA for the constrained devices difficult. Moreover, larger is the key, larger is the size of transmitted packets and worse it is for the wireless operations on battery limited devices.

We proposed and measured the actual performances of an alternative based on the Elliptic Curve Cryptography, providing shorter keys and faster computations. With experimental measurements, we did prove that the usage of ECC was an important breakthrough for CGA, especially for constrained devices like PDA and Tablet PC. We further evaluated the impact of the hash functions on the CGA as the next generation of CGA is highly likely to implement one of the SHA-2 hash algorithm instead of SHA-1.

We then significantly reduced the generation time of higher SEC values by introducing the GPGPU. This demonstrates that

even a higher security could be used in the network without the need for an expensive specialised hardware.

All of this analysis can provide serious leads for standardization bodies such as the IETF where document such as [22], [23] and [24] are proposed.

Future works of the authors will mainly focus on some new usages of the CGA and SEND benefiting from the significantly improved performances presented in this paper.

Acknowledgements

We are thankful to ANR (Agence Nationale de la Recherche) financing the TLCOM MobiSEND project. We also thank anonymous reviewer for their comments and proposed improvements.

References

[1] S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC 2460, Internet Engineering Task Force, updated by RFC 5095 (Dec. 1998).
URL http://www.rfc-editor.org/rfc/rfc2460.txt

[2] T. Narten, E. Nordmark, W. Simpson, H. Soliman, Neighbor Discovery for IP version 6 (IPv6), RFC 4861, Internet Engineering Task Force (Sep. 2007).
URL http://www.rfc-editor.org/rfc/rfc4861.txt

[3] S. Thomson, T. Narten, T. Jinmei, IPv6 Stateless Address Autoconfiguration, RFC 4862, Internet Engineering Task Force (Sep. 2007).
URL http://www.rfc-editor.org/rfc/rfc4862.txt

[4] A. Conta, S. Deering, M. Gupta, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, RFC 4443, Internet Engineering Task Force, updated by RFC 4884 (Mar. 2006).
URL http://www.rfc-editor.org/rfc/rfc4443.txt

[5] P. Nikander, J. Kempf, E. Nordmark, IPv6 Neighbor Discovery (ND) Trust Models and Threats, RFC 3756, Internet Engineering Task Force (May 2004).
URL http://www.rfc-editor.org/rfc/rfc3756.txt

[6] J. Arkko, J. Kempf, B. Zill, P. Nikander, SEcure Neighbor Discovery (SEND), RFC 3971, Internet Engineering Task Force (Mar. 2005).
URL http://www.rfc-editor.org/rfc/rfc3971.txt

[7] T. Aura, Cryptographically Generated Addresses (CGA), RFC 3972, Internet Engineering Task Force, updated by RFCs 4581, 4982 (Mar. 2005).
URL http://www.rfc-editor.org/rfc/rfc3972.txt

[8] NDSS'02, Statistically Unique and Cryptographically Verifiable (SUCV) Identifier and Addresses, The Internet Society, 2002.

[9] S. Krishnan, J. Laganier, M. Bonola, Secure Proxy ND Support for SEND, Internet-Draft draft-ietf-csi-proxy-send-01, Internet Engineering Task Force, work in progress (Jul. 2009).
URL http://www.ietf.org/internet-drafts/draft-ietf-csi-proxy-send-01.txt

[10] J. Kempf, Secure IPv6 Address Proxying using Multi-Key Cryptographically Generated Addresses (MCGAs), Internet-Draft draft-kempf-cgaext-ringsig-ndproxy-00, Internet Engineering Task Force, work in progress (Aug. 2007).
URL http://tools.ietf.org/html/draft-kempf-cgaext-ringsig-ndproxy-00

[11] M. Bagnulo, Hash-Based Addresses (HBA), RFC 5535, Internet Engineering Task Force (Jun. 2009).
URL http://www.rfc-editor.org/rfc/rfc5535.txt

[12] D. Johnson, C. Perkins, J. Arkko, Mobility Support in IPv6, RFC 3775, Internet Engineering Task Force (Jun. 2004).
URL http://www.rfc-editor.org/rfc/rfc3775.txt

[13] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, Host Identity Protocol, RFC 5201, Internet Engineering Task Force (Apr. 2008).
URL http://www.rfc-editor.org/rfc/rfc5201.txt

[14] C. Castelluccia, Cryptographically generated addresses for constrained devices*, Wireless Personal Communications 29 (3) (2004) 221–232. doi:10.1023/B:WIRE.0000047065.81535.84.
URL http://dx.doi.org/10.1023/B:WIRE.0000047065.81535.84

[15] J. Kempf, J. Wood, Z. Ramzan, C. Gentry, Ip address authorization for secure address proxying using multi-key cgas and ring signatures, in: Advances in Information and Computer Security, Springer, 2006, pp. 196–211.

[16] X. Wang, Y. L. Yin, H. Yu, Finding collisions in the full sha-1, in: In Proceedings of Crypto, Springer, 2005, pp. 17–36.

[17] C. McDonald, P. Hawkes, J. Pieprzyk, Differential path for sha-1 with complexity $o(2^{52})$, Cryptology ePrint Archive, Report 2009/259 (2009).
URL http://eprint.iacr.org/

[18] M. Bagnulo, J. Arkko, Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs), RFC 4982, Internet Engineering Task Force (Jul. 2007).
URL http://www.rfc-editor.org/rfc/rfc4982.txt

[19] R. Hinden, S. Deering, IP Version 6 Addressing Architecture, RFC 4291, Internet Engineering Task Force (Feb. 2006).
URL http://www.rfc-editor.org/rfc/rfc4291.txt

[20] D. Hankerson, A. Menezes, S. Vanstone, Guide to elliptic curve cryptography, Springer, 2004.

[21] National Institute of Standards and Technology, NIST special publication 800-57, NIST Special Publication 800 (2007) 57.
URL http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

[22] T. Cheneau, M. Laurent-Maknavicius, S. Shen, M. Vanderveen, ECC public key and signature support in Cryptographically Generated Addresses (CGA) and in the Secure Neighbor Discovery (SEND), Internet-Draft draft-cheneau-csi-ecc-sig-agility-00, Internet Engineering Task Force, work in progress (Oct. 2009).
URL http://www.ietf.org/internet-drafts/draft-cheneau-csi-ecc-sig-agility-00.txt

[23] T. Cheneau, M. Laurent-Maknavicius, S. Shen, M. Vanderveen, Support for Multiple Signature Algorithms in Cryptographically Generated Addresses (CGAs), Internet-Draft draft-cheneau-csi-cga-pk-agility-00, Internet Engineering Task Force, work in progress (Oct. 2009).
URL http://www.ietf.org/internet-drafts/draft-cheneau-csi-cga-pk-agility-00.txt

[24] T. Cheneau, M. Laurent-Maknavicius, S. Shen, M. Vanderveen, Signature Algorithm Agility in the Secure Neighbor Discovery (SEND) Protocol, Internet-Draft draft-cheneau-csi-send-sig-agility-00, Internet Engineering Task Force, work in progress (Oct. 2009).
URL http://www.ietf.org/internet-drafts/draft-cheneau-csi-send-sig-agility-00.txt

[25] T. Narten, R. Draves, S. Krishnan, Privacy Extensions for Stateless Address Autoconfiguration in IPv6, RFC 4941, Internet Engineering Task Force (Sep. 2007).
URL http://www.rfc-editor.org/rfc/rfc4941.txt

[26] G. Montenegro, C. Castelluccia, Crypto-based identifiers (cbids): Concepts and applications, ACM Trans. Inf. Syst. Secur. 7 (1) (2004) 97–127. doi:http://doi.acm.org/10.1145/984334.984338.

[27] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone, R. L. Rivest, Handbook of applied cryptography (1997).

[28] National Institute of Standards and Technology, FIPS 180-3: Secure Hash Standard (SHS), National Institute for Standards and Technology, Gaithersburg, MD, USA, 2008.
URL http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[29] V. Gupta, S. Gupta, S. Chang, D. Stebila, Performance analysis of elliptic curve cryptography for SSL, in: WiSE '02: Proceedings of the 1st ACM workshop on Wireless security, ACM, New York, NY, USA, 2002, pp. 87–94. doi:http://doi.acm.org/10.1145/570681.570691.

[30] National Institute of Standards and Technology, Draft FIPS 186-3: Digital Signature Standard (DSS), National Institute for Standards and Technology, Gaithersburg, MD, USA, 2008.
URL http://csrc.nist.gov/publications/drafts/fips_186-3/Draft_FIPS-186-320_November2008.pdf

[31] NIST Comments on Cryptanalytic Attacks on SHA-1.
URL http://csrc.nist.gov/groups/ST/hash/statement.html

[32] A. Kukec, S. Krishnan, S. Jiang, SeND Hash Threat Analysis, Internet-Draft draft-ietf-csi-hash-threat-03, Internet Engineering Task Force, work

in progress (Mar. 2009).
URL http://www.ietf.org/internet-drafts/draft-ietf-csi-hash-threat-03.txt

[33] H. Dobbertin, A. Bosselaers, B. Preneel, RIPEMD-160: A Strengthened Version of RIPEMD, in: Proceedings of the Third International Workshop on Fast Software Encryption, Springer-Verlag, London, UK, 1996, pp. 71–82.

[34] R. Anderson, E. Biham, Tiger: a fast new hash function, in: Fast Software Encryption, Third International Workshop Proceedings, Springer-Verlag, 1996, pp. 89–97.

[35] The WHIRLPOOL Hash Function website.
URL http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html

[36] Tiger: A Fast New Hash Function website.
URL http://www.cs.technion.ac.il/~biham/Reports/Tiger/

[37] OpenSSL website.
URL http://www.openssl.org

[38] J. Nakajima, M. Matsui, Performance Analysis and Parallel Implementation of Dedicated Hash Functions, in: EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Springer-Verlag, London, UK, 2002, pp. 165–180.

[39] XySSL website.
URL http://xyssl.org

[40] CUDA Zone website.
URL http://www.nvidia.com/object/cuda_home.html

[41] S. Jiang, Z. Xia, Configuring Cryptographically Generated Addresses (CGA) using DHCPv6, Internet-Draft draft-jiang-csi-cga-config-dhcpv6-00, Internet Engineering Task Force, work in progress (May 2009).
URL http://www.ietf.org/internet-drafts/draft-jiang-csi-cga-config-dhcpv6-00.txt