An Investigation of the Merkle Signature Scheme for Cryptographically Generated Address Signatures in Mobile IPv6

Sana Qadir, Mohammad Umar Siddiqi and Wajdi Fawzi Mohammed Al-Khateeb
(Corresponding author:Sana Qadir)

Department of Electrical and Computer Engineering, International Islamic University Malaysia
P.O. BOX 10, Kuala Lumpur, 50728, Malaysia
(Email: q57sana@yahoo.com)
(Received Feb. 16, 2014; revised and accepted June 22, 2014)

Abstract

Cryptographically Generated Address (CGA) Signatures are a promising feature of Mobile IPv6 (MIPv6) and are slowly being considered suitable for signing the Binding Update (BU) message. However, the use of the Rivest-Shamir-Adleman (RSA) by the CGA signature algorithms is considered to be a very serious limitation. This is because RSA provides only limited computational security and incurs significant performance overhead. This work investigates the use of an alternative signature scheme called Merkle Signature Scheme (MSS) for use in the CGA signature algorithms. The results show that compared to RSA, MSS provides stronger computational security and significantly improves the performance of the key generation and the CGA signature generation operations.

Keywords: Binding update message, cryptographically generated address signature, Merkle signature scheme

1 Introduction

Security was a specific design consideration for IPv6. The use of existing security mechanisms or protocols like IPsec (Internet Protocol SEcurity) were made compulsory and new features like CGAs and CGA signatures were introduced (see RFC 3972).

MIPv6 provides mobility at the network layer (i.e. IPv6 layer) and it contains several improvements over its predecessor MIPv4. One of the most important improvements is that MIPv6 allows for the direct routing of packets between the Mobile Node (MN) and the Correspondent Node (CN) after the Route Optimization (RO) procedure has been completed (see Figure 1). The Binding Update (BU) message is exchanged during the RO procedure as is shown in Figure 2. The lack of a strong authentication mechanism for the BU message is one of the biggest security vulnerabilities in MIPv6 networks.

The existing authentication mechanism is poor and CGA Signature are considered a very promising alternative [15]. The reason why CGA signatures are considered promising is that they are able to provide strong authentication without requiring nodes to share any common information or security infrastructure [7]. Current key management protocols required by IPsec are not sufficiently scalable or manageable in a global MIPv6 setup [7]. This seriously curtails the use of IPsec and as a result CGA signature-based authentication is gaining wider acceptance. For example, the latest enhancement to MIPv6 called Enhanced Route Optimization (ERO) includes the use of CGA signatures [2]. Although CGA signature-based authentication protocols like ERO are considered reasonably secure, the implementation cost of ERO is not insignificant, especially for low-end MIPv6 nodes like the CN and the MN [14]. In some cases, the cost can be so large that it can be exploited by an adversary to launch a denial of service attack (by flooding the CN with signed BU messages that need to be verified) [14, 15].

The main objective of this work is to show that the performance of CGA signatures can be improved by replacing the use of RSA with the Merkle Signature Scheme (MSS). This should help prevent nodes or networks that use CGA signature based authentication from being vulnerable to denial of service attacks.

2 Related Literature

2.1 CGA Signatures

A node with a CGA can use its private key to sign a packet sent from its address and to assert ownership of its address [3]. A CGA signature is generated using the **CGA signature generation algorithm** (see Figure 3 above) and a CGA signature is verified using a **CGA signature verification algorithm** (see Figure 4 above). Both these algorithms use the RSA Signature Scheme with Appendix

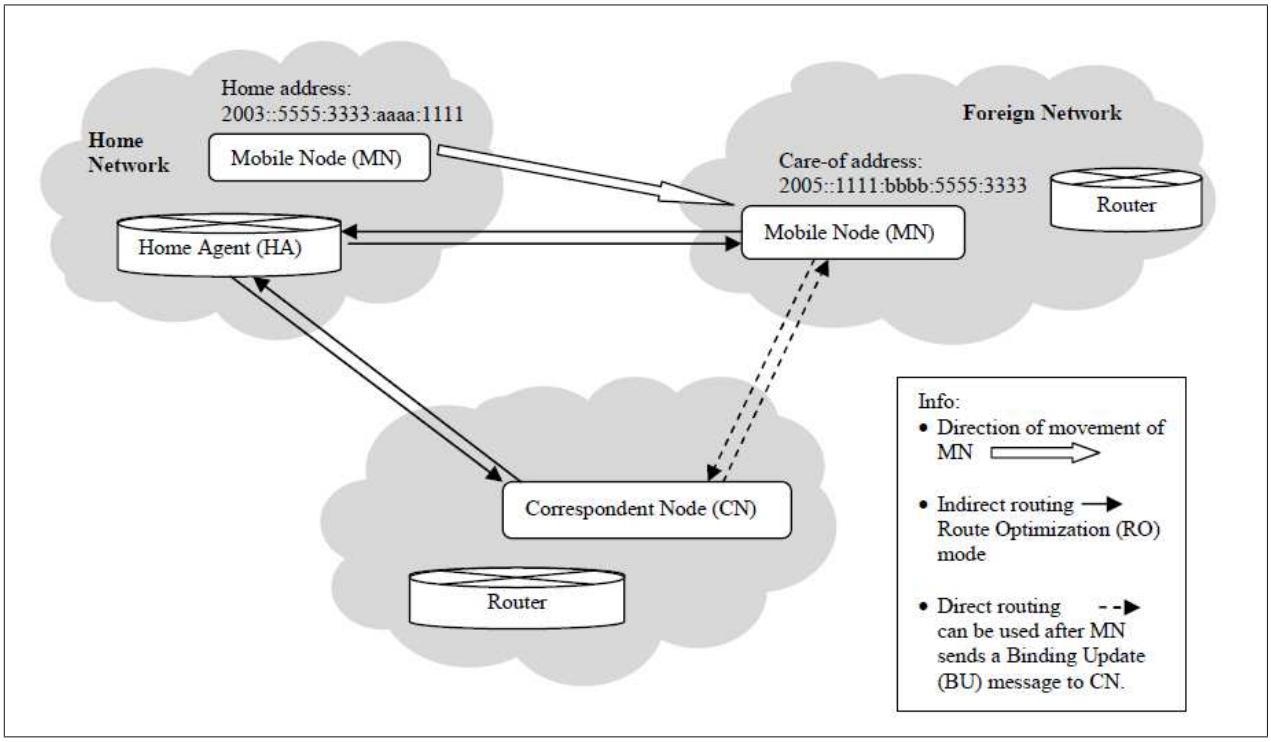

Figure 1: Direct routing in MIPv6

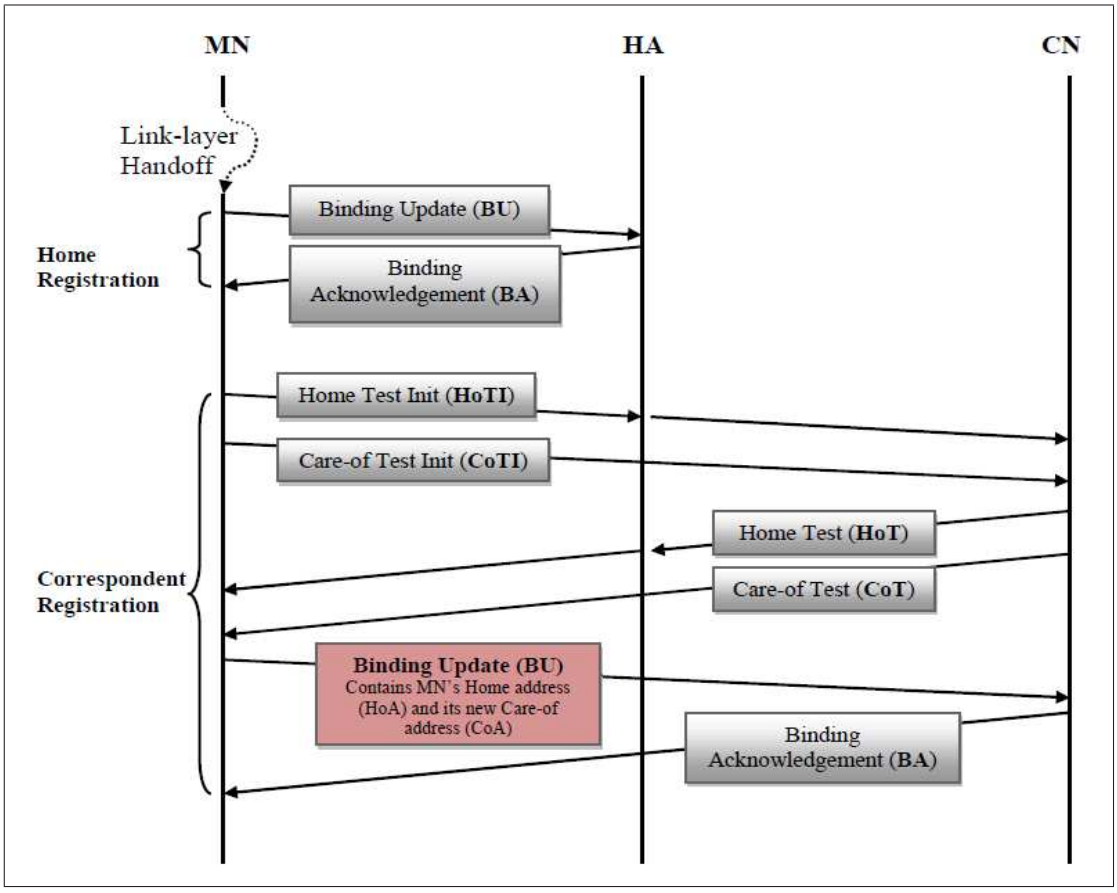

Figure 2: Route optimization procedure [3]

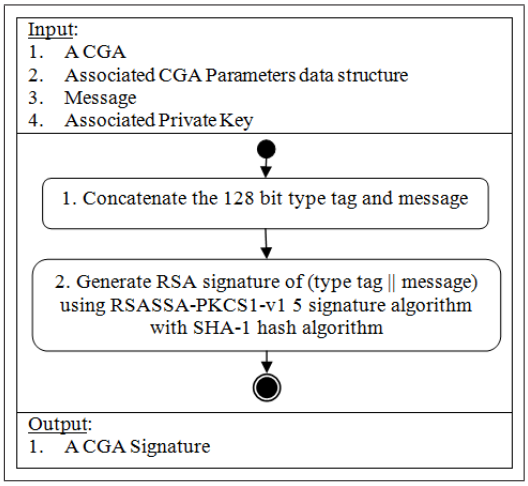

Figure 3: CGA signature generation algorithm [3]

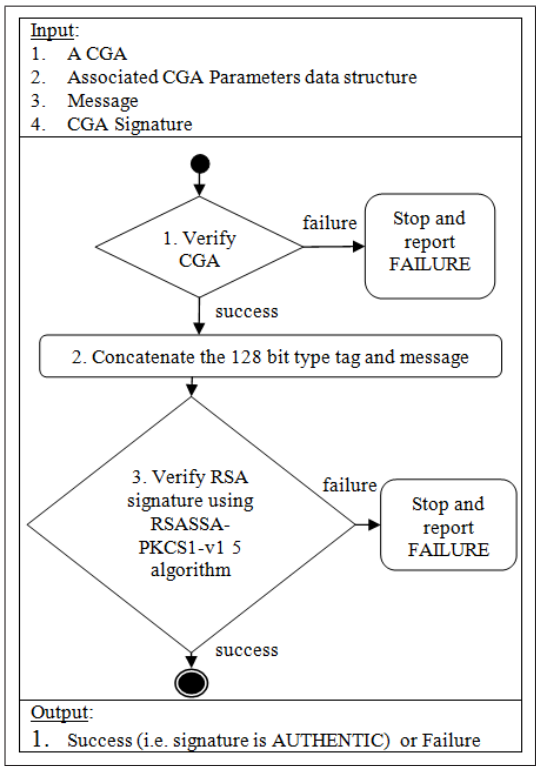

Figure 4: CGA signature verification algorithm [3]

i.e. standardized version 1.5 of Public Key Cryptography Standards #1 (i.e. RSASSA-PKCS1-v1_5). A signature scheme with an appendix means that instead of signing a message directly, a hash function is first used to calculate a digest of the message and then the digest is signed using the private key. This method is popular because the size of the message that can be directly signed must be proportional to the size of the keys and the keys are almost always smaller than the size of the message.

The BU message is sent from a MN to a CN to inform the CN of the MNs new point of attachment [13]. To prevent fake BU messages from being used to launch an attack, a MN must generate a CGA signature and a CN must verify the CGA signature. The use of a CGA signature algorithm to sign a BU message is shown in Figure 5 (on the next page). The sender (i.e. MN) must also share the CGA and the CGA Parameters data structure with the receiver (i.e CN) so that it can verify the CGA signature.

In the SEcure Neighbor Discovery Protocol (SEND), where CGAs were first introduced, the minimum RSA key length required is only 384 bits [3]. RFC 3972 states that if the RSA key is compromised because integer-factoring attacks for the chosen key length have become practical, the key has to be replaced with a longer one [3]. However, existing studies [9, 20] show that:

1) Public key operations are the main contributors to the computational cost of CGA signature generation algorithm;

2) Increasing RSA key lengths significantly degrades the performance of CGA signature algorithms.

Therefore, increasing the RSA key length is neither an acceptable nor a practical solution especially in light of the fact that one of the main reasons for using CGA signatures is to prevent denial-of-service attacks. It is also important that the computational expense on the MN is not large enough to cause significant delays in handover. The most promising solution is to consider replacing the use of RSA with another cryptosystem. Existing studies [8, 9, 19, 20] have investigated alternative signature schemes like Elliptic Curve Digital Signature Algorithm (ECDSA), Digital Signature Algorithm (DSA) and even Modified Feige-Fiat-Shamir (MFFS) scheme. Reference [9] shows that ECC, with its superior performance, small keys and short signatures, is the best alternative to RSA. However, from a performance standpoint, it has one downside: ECDSA signature verification is more costly than RSA signature verification.

2.2 Security of Digital Signatures Schemes

The security of commonly used signature schemes (include ECDSA) depends on the difficulty of solving problems like factoring large composite numbers or computing discrete logarithms. The best cryptanalytic methods of solving these hard problems have sub-exponential run time; meaning that no efficient (i.e. polynomial) algorithm exists to solve these problems. Coupled with the assumption that all adversaries are limited to polynomial time algorithms on classical computers, these schemes are considered secure. However, this assumption is losing credibility in face of the growing acceptance of a near future where quantum computers exist. In 1997, Peter Shor, showed that algorithms for quantum computers (e.g. Shor's algorithm) can solve these hard problems

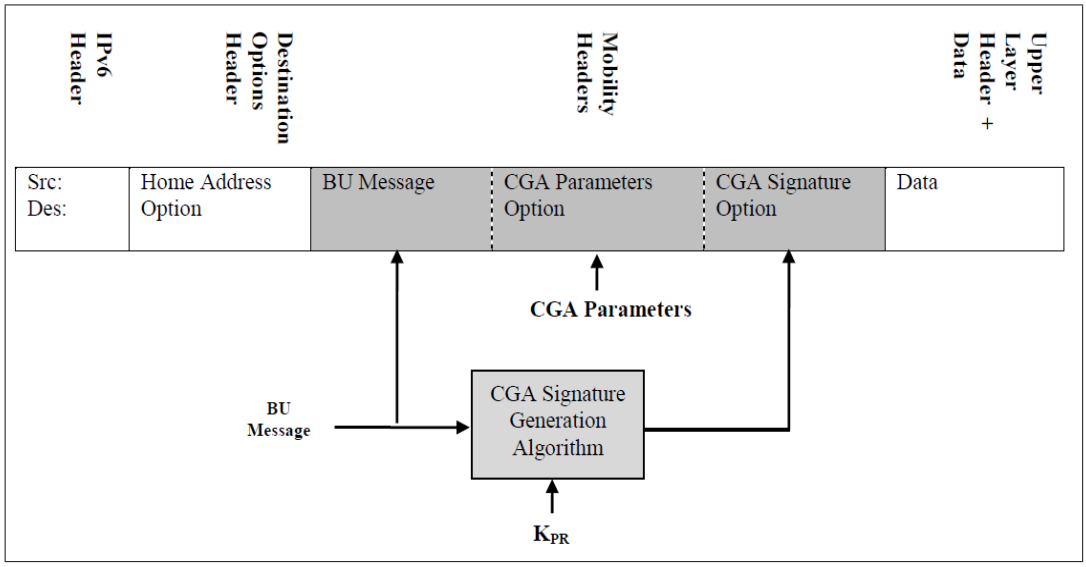

Figure 5: CGA signature of BU message

Table 1: Properties of cryptographic hash functions

Property	Definition	Complexity of Attack Using	
		Classical Computers	Quantum Computers
Pre-image Resistance (One-way)	For any given digest n, it is computationally infeasible to find x such that $g(x) = n$	$O(2^n)$ - Using pigeon hole principle	-
Second-preimage Resistance	For any given block x, it is computationally infeasible to find $y \neq x$ with $g(y) = g(x)$	$O(2^n)$ - Using pigeon hole principle	-
Collision Resistance	It is computationally infeasible to find any pair (x, y) such that $g(x) = g(y)$	$O(2^{n/2})$ - Using birthday paradox	$O(2^{n/3})$

efficiently [4]. As such, it is imperative to consider signature schemes that do not depend on hard mathematical problems.

The signature schemes showing the best potential in a quantum computing scenario are hash-based signature schemes. There are a few other post-quantum cryptosystems that are considered to be extremely difficult to break (e.g. code-based cryptosystems and lattice-based cryptosystems) but hash-based digital signatures schemes have one unparalleled advantage. The security of these schemes depends entirely on the collision resistance property of the hash function used [6]. Refer to Table 1 for a summary of the properties of cryptographic hash functions. If we assume that the hash function used is only vulnerable to generic attacks, then even the fastest quantum algorithm that can be used to attack the collision resistance of a hash function (i.e. Grover's algorithm) has a computational complexity of $O(2^{n/3})$ [6]. So, for example, a hash function with a 256-bit digest will provide computational security of 128-bit security (i.e. $O(2^{128})$) on classical computers and about 85-bit security (i.e. $O(2^{85})$) on quantum computers. Although hash-based digital signature schemes have not formally been proven to resist attacks using quantum computers, their security requirements are minimal and they are independent of hard mathematical problems. In fact, each new hash function can be used to develop a new hash-based signature scheme [6].

2.3 Merkle Signature Scheme

Hash-based signature schemes are based on one-time signature (OTS) schemes and the security of OTS scheme relies on the use of a cryptographically secure hash function. A cryptographically secure hash function is a hash function that is preimage resistant, second preimage resistant and collision resistant (see Table 1).

There are several popular OTS schemes but when selecting a scheme for generating a hash-based signature, it must be kept in mind that the cost of CGA-based au-

thentication depends directly on the performance of the underlying signature scheme. As such, the efficiency and storage requirements of an OTS scheme become the importance selection criteria. In fact, OTS schemes are increasing seen as alternatives to commonly used digital schemes because they are much faster on devices with limited storage and computational resources [5]. This is primarily because they are based on one way functions and one of the most widely accepted class of one-way functions (in cryptography) are hash functions [22].

The most popular OTS scheme, called Lamport-Diffie OTS (LD-OTS) uses large keys and generates huge signatures [4]. In fact, LD-OTS keys and signatures are more than 20 times larger than 1024-bit RSA keys and signatures [4]. The Winternitz OTS (W-OTS) scheme, on the other hand, generates significantly shorter signatures. This scheme provides a parameter, w, through which the signature length can be shortened but at the cost of increased computational cost. The details of W-OTS scheme are given below [6].

Requirements:

1) A one-way function is selected:

$$f : \{0,1\}^n \rightarrow \{0,1\}^n.$$

2) A cryptographic hash function is selected:

$$g : \{0,1\}^* \rightarrow \{0,1\}^n.$$

Key Pair Generation:

1) A Winternitz parameter $w \geq 2$ is selected where w is the number of bits to be signed simultaneously.
2) The t_1, t_2 and t are calculated as follows:

$$\begin{aligned} t_1 &= \lceil \frac{n}{w} \rceil \\ t_2 &= \lceil \frac{\lfloor \log_2 t_1 \rfloor + 1 + w}{w} \rceil \\ t &= t_1 + t_2. \end{aligned}$$

3) The bit strings x_i of the Signature Key X are chosen at random:

$$X = (x_{t-1}, \cdots, x_1, x_0) \in \{0,1\}^{(n,t)}.$$

4) The Verification Key Y is computed:

$$Y = (y_{t-1}, \cdots, y_1, y_0) \in \{0,1\}^{(n,t)},$$

where $y_i = f^{2^w-1}(x_i)$, $0 \leq i \leq t-1$.

W-OTS Signature Generation:

1) Hash digest of the message M is calculated as $g(M) = d$ such that $(d_{n-1}, \cdots, d_0)$.
2) A minimum number of zeros are prepended to d such that the length of d is divisible by w.
3) The extended string d is split into t_1 bit strings $b_{t-1}, \cdots, b_{t-t_1}$ each of length w, i.e. $d = b_{t-1} \,||\, \cdots \,||\, b_{t-t_1}$ where $||$ denotes concatenation.
4) The bit strings b_i are identified with integers in $\{0, 1, \cdots, 2^w - 1\}$ and the checksum c is calculated using:

$$c = \sum_{i=t-t_1}^{t-1} (2^w - b_i).$$

5) A minimum number of zeros are prepended to the binary representation of c such that the length of its binary representation is divisible by w. Then the extended string is split into t_2 blocks: $b_{t_2-1}, \cdots, b_0$ each of length w. Then, $c = b_{t_2-1} \,||\, \cdots \,||\, b_0$.
6) The signature is calculated using:

$$\sigma = (f^{b_{t-1}}(x_{t-1}), \cdots, f^{b_1}(x_1), f^{b_0}(x_0)).$$

W-OTS Signature Verification:

1) For signature $\sigma = (\sigma_{t-1}, \cdots, \sigma_0)$, the bit strings $b_{t-1}, \cdots, b_0$ are calculated as in steps 1 to 5 in W-OTS Signature Generation above.
2) Then, we check if $(f^{2^w-1-b_{t-1}}(\sigma_{n-1}), \cdots, f^{2^w-1-b_0}(\sigma_0)) = (y_{n-1}, \cdots, y_0)$.
3) If the signature is valid, then $\sigma_i = f^{b_i}(x_i)$ and therefore $f^{2^w-1-b_i}(\sigma_i) = f^{2^w-1}(x_i) = y_i$ holds true for $i = t-1, \cdots, 0$.

See Table 2 for an analysis of the computational cost and storage requirements of W-OTS. The flexibility of choosing the value of parameter w and the shorter signatures of W-OTS means that it is considered the best alternative to LD-OTS. Reference [4] states that:

- Signature size of W-OTS varies in inverse proportion to the parameter w.
- Key generation time, signature generation time and signature verification time increase exponentially with the size of parameter w.

An example of the different values for the parameters of W-OTS is given in Table 3. It is evident that, when $w = 32$, the computational cost:

- To calculate the verification key;
- To generate W-OTS signature;
- To verify W-OTS signature.

Total is $t(2^w - 1)$ and therefore infeasible. As such, this work only considers $w = 16$, 8 or 4.

The second point to consider about OTS schemes is that one key pair can be used to sign only one message. This creates an enormous key management problem that renders most OTS schemes more or less

Table 2: Computational cost and storage requirements of W-OTS

	Computational cost	**Storage requirements**
Key generation	$t(2^w - 1)$ evaluations of f	Length of the signature key: $t \times n$ bits Length of verification key: $t \times n$ bits
Signature generation	Worst case: $t(2^w - 1)$ evaluations of f	Length of the signature: $t \times n$ bits
Signature verification	Worst case: $t(2^w - 1)$ evaluations of f	-

Table 3: Different values for parameters of W-OTS

w	n	t_1	t_2	t	$t \times n$	$t(2^w - 1)$
32	256	8	2	10	2560	$4.29e10$
16	256	16	2	18	4608	656350
8	256	32	2	34	8704	2550
4	256	64	3	67	17152	150

impractical. The first step in overcoming this problem was suggested by Merkle and culminated in the creation of the Merkle Signature Scheme (MSS) [17]. MSS requires a cryptographic hash function g and a one-time signature scheme (e.g. W-OTS) [6]. Merkle proposed using a binary hash tree to reduce the validity of an arbitrary but fixed number of one-time verification keys to the validity of one single public key, the root of the hash tree [6]. The details of MSS are outlined below where the height of the hash tree h is 3 [4, 6].

MSS Key Generation (see Figure 6):

1) Select h, such that $h \geq 2$ and $N = 2^h$ (where N is the total number of messages to be signed).
2) Generate 2^h one-time key pairs (X_i, Y_i) where X_i is the OTS signature key and Y_i is the OTS verification key. MSS private key is the sequence of the 2^h OTS signature keys.
3) Calculate the leaves of Merkle tree using $n[0, i] = g(Y_i)$, where $0 \leq i < 2^h$.
4) Calculate the MSS public key (i.e. the root of the Merkle tree). A parent node is the hash value of the concatenation of its left and right children,

$$n[j, i] = g(n[j-1, 2i] \,||\, n[j-1, 2i+1]),$$

where $1 \leq j \leq h, 0 \leq i < 2^{h-j}$.

MSS Signature Generation (see Figure 7):

1) Compute the n-bit digest $d = g(M)$.
2) Generate OTS signature σ_{OTS} of d using the s^{th} OTS signature key X_s.
3) Calculate MSS signature using:

$$\sigma_s = (s, \sigma_{OTS}, Y_s, (a_0, \cdots, a_{h-1})),$$

where:

- s is the index, and
- $(a_0, \ldots, a_{h-1})$ is the authentication path for the verification key Y_s.

MSS Signature Verification:

1) Use the verification key Y_s to verify the one-time signature σ_{OTS} of the digest $d = g(M)$.
2) Validate the authenticity of the one-time verification key Y_s by constructing the path $(p_0, \cdots, p_h)$ from the s^{th} leaf $g(Y_s)$ to the root of the Merkle tree. The verifier uses the index s and the authentication path $(a_0, \cdots, a_{h-1})$ and applies the following construction:

$$p_s = \begin{cases} g(a_{i-1} \,||\, p_{i-1}), & \text{if} \lfloor s/2^{i-1} \rfloor \equiv 1 \bmod 2 \\ g(p_{i-1} \,||\, a_{i-1}), & \text{if} \lfloor s/2^{i-1} \rfloor \equiv 0 \bmod 2 \end{cases}$$

for $i = 1, \cdots, h$ and $p_0 = g(Y_s)$. The authentication of the one-time verification key Y_s is successful if and only if p_h equals the MSS public key.

The Merkle Signature Scheme begins with the signer deciding on the maximum number of messages to be signed with one public key, N. This number of messages must be a power of two and the height of the binary hash tree, h, is calculated as $N = 2^h$. The signer then generates N key pairs (X_i, Y_i) where X_i is the OTS signature key and Y_i is the OTS verification key. The key pairs are used to create a binary hash tree where each leaf is calculated as $n[0, i] = g(Y_i)$

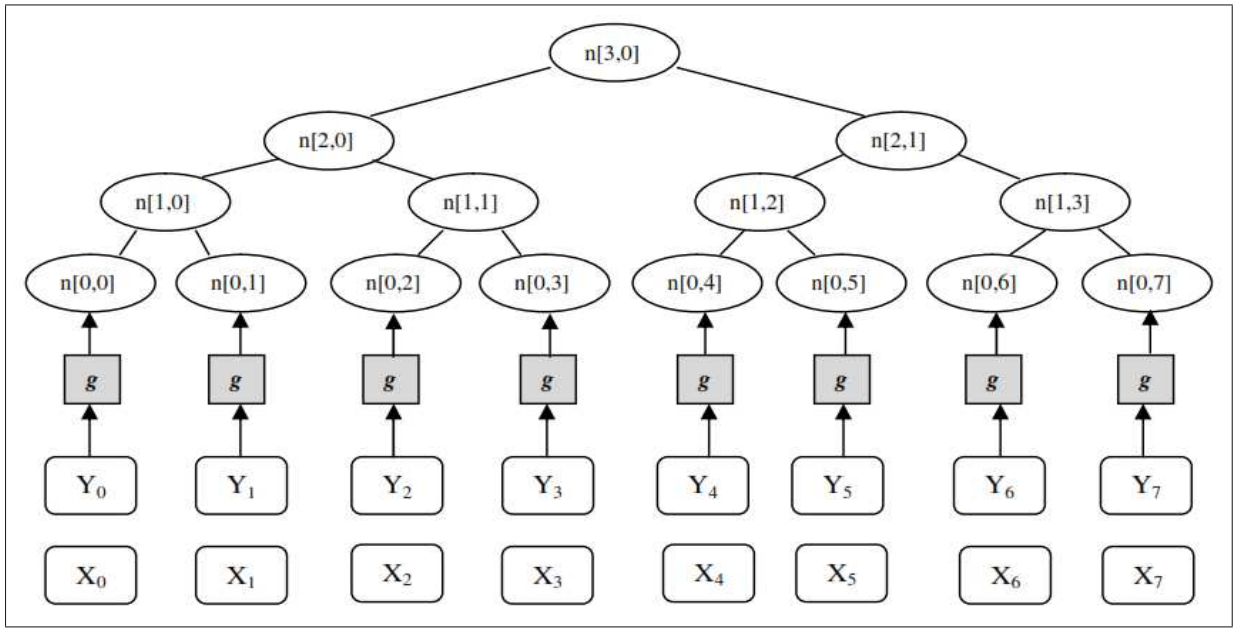

Figure 6: Merkle tree $h = 3$ [4, 6]

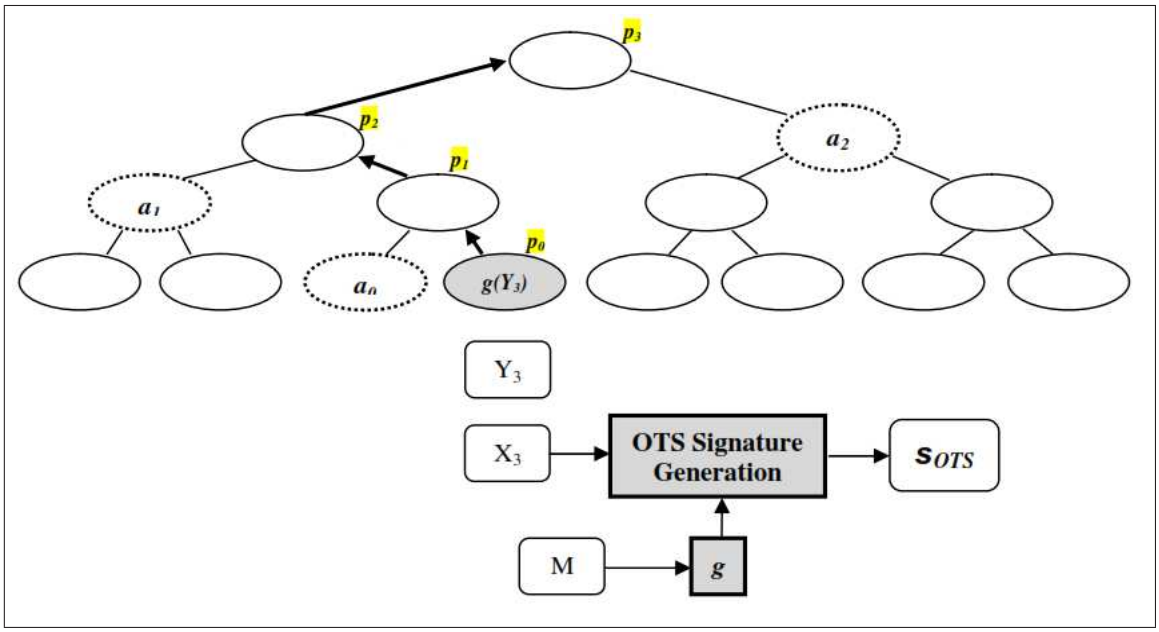

Figure 7: Authentication path for verification key Y_s [4, 6]

and each inner node is the hash value of concatenation of its two children nodes. The root of the hash tree is the MSS public key. The sequence of 2^h OTS signature keys X_i, constitute the MSS private key.

Generating a Merkle signature starts by computing the digest of the message M, $d = g(M)$ and then generating an OTS signature of d using the OTS signature key X_s. Next, the authentication path $(a_o, \cdots, a_{h-1})$ of the OTS verification key Y_s is calculated and sent together with the MSS signature. The verifier begins by verifying the OTS signature using Y_s. If successful, the verifier validates Y_s by calculating the root of the Merkle tree (using Y_s and the authentication path $(a_o, \cdots, a_{h-1})$). If the calculated root of the Merkle tree is equal to the MSS public key sent by the signer, then the signature is considered successfully authenticated.

The strongest advantage of the Merkle signature scheme is its *provable security* - security reductions exist for the Merkle's tree authentication scheme (combined with W-OTS) to the collision resistance of the hash function used [10, 12].

It is also interesting to note that MSS is resistant to differential side channel attacks [4]. Typical techniques for differential side channel attacks require observing the power usage, execution time or electromagnetic signals emitted when different messages are signed using the same private key. This is not possible with MSS because different messages are signed using different private OTS keys.

The only significant drawback of MSS is the relatively expensive generation and storage of the Merkle tree. Table 4 shows an outline of the computation cost and storage requirements of MSS [6].

Generating the entire Merkle tree (in order to calculate the authentication path) for every signature is impracticable especially for big trees. The other alternative of saving all $2^{h+1}1$ nodes results in huge storage requirements. Hence, a good strategy or algorithm is needed to generate a signature in efficient time without having to save too many nodes [4]. This problem is called the *Merkle tree traversal problem*. The traditional algorithm for solving this problem is called the Classical Merkle Tree Traversal [4]. The downside of this algorithm is its storage requirement and existing literature contains proposed improvements to this algorithm as well as some entirely new algorithms to solve the Merkle tree traversal problem (e.g. Logarithmic Merkle Tree Traversal and Fractal / Stratified Tree Traversal) [4, 5, 6]. In the scenario where a CGA-based signature is used to authenticate a BU message (i.e. during route optimization), we do not require a signature scheme capable of signing many messages. In fact, it is better for the security of the network that only about four to eight BU messages can be signed efficiently by the MN before it has to repeat the computationally expensive steps for Key Generation. This is desirable because it will prevent one MN from overwhelming the CN with BU messages that need to be verified - i.e. launching a Denial of Service (DoS) attack.

3 The Design

There are many design choices possible for the underlying hash function g. A hash function that has successfully withstood several years of analysis by the cryptographic community and produces the required digest length can be selected, for example, from the SHA-2 family. This makes MSS a particularly interesting target for implementation research [12]. The hash function can also be chosen in view of the hardware and software resources available. The new hash functions (esp. SHA-3) are investigated in this work. Keccak was selected as winner of the NIST competition for SHA-3 on 2^{nd} October 2012. In spite of the fact that other SHA-3 finalists, like Skein and BLAKE, showed faster performance, Keccak was selected as SHA-3. This was primarily because Keccak is a family of sponge functions and therefore, has a very different design from the SHA-2 family. Also, this study only includes hash functions that produce digests of 256 bits. This is to ensure computational security of at least $O(2^{85})$ on quantum computers and $O(2^{128})$ on classical computers. This is approximately the same as the maximum computational security that is at present achievable in a SEND-enabled MIPv6 network. The reason for this is that generating a CGA on a contemporary CPU using sec value of 1 takes about 402 ms and provides computational security of about $O(2^{75})$ [1, 11]. A CGA that provides higher computational security of $O(2^{91})$ i.e. when sec = 2 will require 1.65 hours to generate. This huge generation time is unacceptable in a MIPv6 environment where address generation must be completed within hundreds of milliseconds [1]. Therefore, all MIPv6 networks at present are limited to using CGAs that have a computational security of about $O(2^{75})$.

4 Implementation

The CGA signature algorithms were coded in C and development was done using the Maemo 5 Software Development Kit (SDK) [16]. The implementation of cryptographic primitives (e.g. RNG and SHA-2) and public-key cryptosystems (i.e. RSA) are from the PolarSSL library [18]. Keccak (i.e. SHA-3), Skein and BLAKE implementations are from the SAPHIR library [21]. The W-OTS scheme was implemented by the authors and then used to implement MSS ($h = 2$). The code was cross-compiled (for ARM architecture) using Scratchbox and run on a Nokia N900. The clock cycles were recorded using the RDTSC instruction. It is important to note that the main processor of a N900 is a 32-bit Cortex A8 running at 600 MHz.

5 Result

Initially, the performance of the different hash functions was recorded for a typical hash operation used in the CGA generation algorithm (i.e. calculation of Hash2) (see Figure 8). It is obvious that SHA-256 provides the best performance in comparison to the other 256-bit hash functions. This performance advantage will guarantee that SHA-256 will continue to be used at least in the near future. BLAKE comes next in terms of performance but it is more than 2.5 times slower than SHA-256. Keccak, although, officially SHA-3, proves to be the slowest hash function included in this study (14 times slower than SHA-256). If, in the future, the SHA-2 hash family needs to be replaced (e.g. because its structure is similar to SHA-1), BLAKE would be the ideal candidate from a performance standpoint.

Figure 9 compares the mean number of clock cycles taken by MSS, RSA-2048 and RSA-4096 for the following operations:

- Key Generation;
- Signature Generation (of BU message);
- Signature Verification (of BU message).

Table 4: Computational and storage requirements of MSS [4]

Operation	**Computational Requirements** (where t and w are parameters of W-OTS; h is the height of the tree and a_i is a node on the authentication path)	**Storage Requirements** (where l is length of digest)
Key Generation	Generate W-OTS key pairs: $2^h \times [t(2^w - 1)$ hash operations + a PRNG operation] Generate MSS public key: $(2^{h+1} - 1)$ hash operations	Entire MSS tree: $(2^{h+1} - 1) \times l$ MSS public key: l
Signature Generation	Generate OTS signature: Worst case: $t(2^w - 1)$ Generate authentication path $a_0, \cdots, a_{h-1}$	$t \times l$ $t \times l + h \times l$
Signature Verification	Verify OTS signature: Worst case: $t(2^w - 1)$ Compute path: h hash operations	

Table 5: Comparison of performance (MSS vs RSA) on a N900 (with a clock rate of 600 MHz)

	RSA-2048		**MSS**		**RSA-4096**	
	Mean no. of clock cycles	Mean time (μs)	Mean no. of clock cycles	Mean time (μs)	Mean no. of clock cycles	Mean time (μs)
Key Generation	12,978,579	21,631	656,134	1,094	138,511,862	230,853
CGA Signature Generation	133,255	222	59,893	100	867,831	1,446
CGA Signature Verification	9,606	16	64,885	108	28,719	48

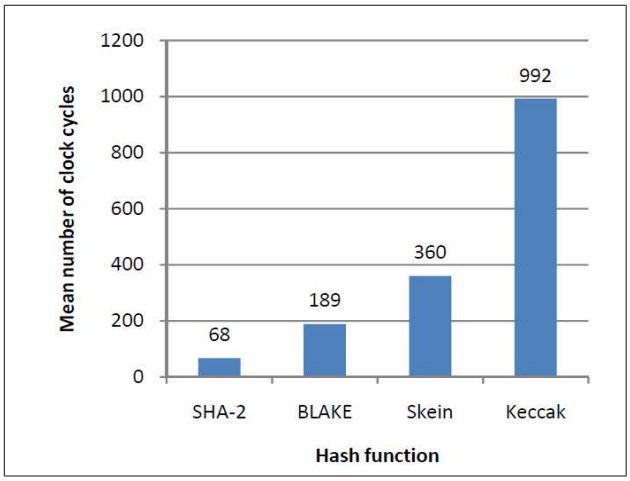

Figure 8: Performance of different hash functions (with 256-bit hash digests)

From the bar charts (Figure 9) and from Table 5, it can be seen that:

- MSS Key Generation time is almost 20 times faster than RSA-2048;
- MSS Signature Generation time is less than half that of RSA-2048;
- MSS Signature Verification time is more than 6 times slower than RSA-2048.

Also, if the performance of MSS is compared to RSA-4096, we find that:

- MSS Key Generation time is 200 times faster,
- MSS Signature Generation time is 14 times faster, but
- MSS Signature Verification time is more than 2 times slower than RSA-4096.

This means that the performance advantage of using MSS improves when compared to increasingly larger RSA key sizes.

A few other points can be deduced from this study:

- When MSS Signatures are used, the computational expense is almost identical for both MN and CN (i.e.

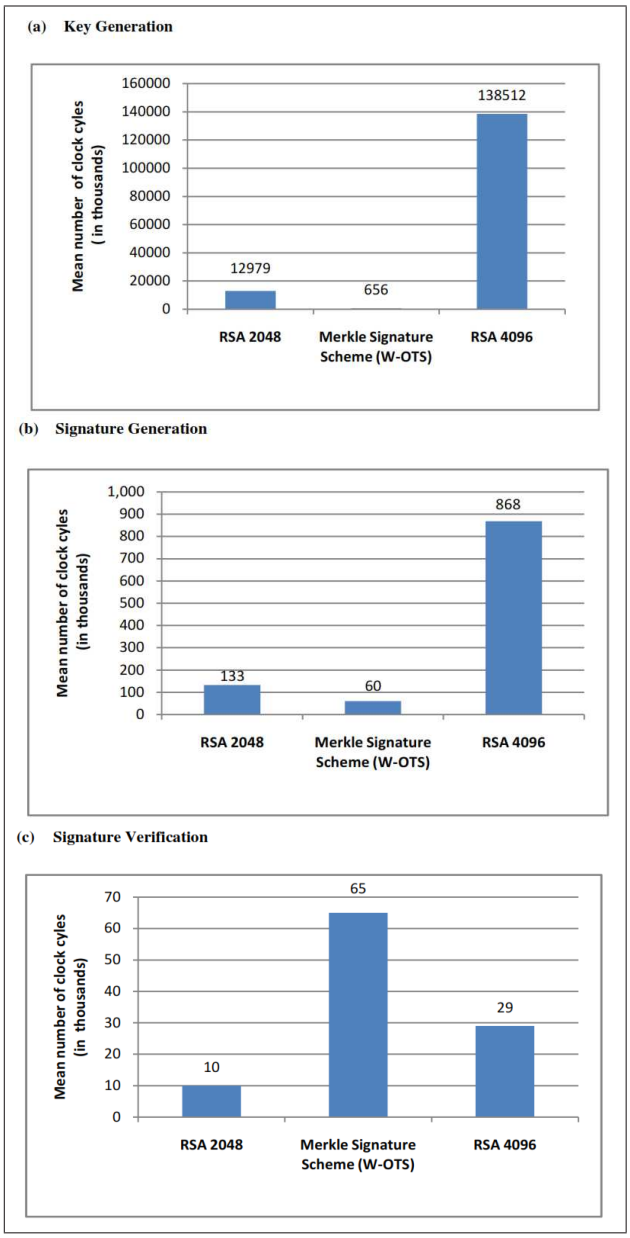

Figure 9: Performance of different hash functions (with 256-bit hash digests)

about 100μs). This is ideal for scenarios where both the MN and CN are low-power mobile nodes.

- MSS public key (32 bytes) is 8 times smaller than RSA-2048 public key (256 bytes).
- MSS signature (2241 bytes) is almost 9 times longer than RSA-2048 (256 bytes) signature.
- The major contributor to computational cost is the generation of the OTS key pairs. It is best therefore to use only four key pairs as in the implementation above ($h = 2$).

6 Conclusion

This work shows that MSS provides an efficient and secure alternative to the use of computationally expensive cryptosystems like RSA for CGA signature. The performance advantage of the MSS Key Generation and Signature Generation Algorithms is very obvious when compared to RSA (esp. for large RSA key sizes). The implementation developed in this study allows for generation of up to four MSS signatures. This means that after four BU messages have been signed, the MN will have to repeat the computationally intensive Key Generation Algorithm. This is an advantage because it will prevent one MN from overwhelming the CN with BU messages that have to be verified (this is one technique of launching a DoS attack). Future work involves bench marking the performance of an actual CGA-based authentication protocol (e.g. ERO) when CGA signatures use MSS with when they use RSA.

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] J. Arkko, C. Vogt, and W. Haddad, "Enhanced route optimization for mobile IPv6," Tech. Rep. RFC 4866, 2007.

[3] T. Aura, "Cryptographically Gnerated Addresses (CGA)," Tech. Rep. RFC 3972, 2005.

[4] G. Becker, "Merkle signature schemes, merkle trees and their cryptanalysis [Ph.D. Thesis]," 2008. (http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf)

[5] D. Berbecaru, "Performance of two one-time signature schemes in space/time constrained environments," in *Proceedings of The 5th IEEE International Symposium of Wireless Pervasive Computing (ISWPC'10)*, pp. 238–243, Modena, Italy, May 2010.

[6] J. Buchmann, E. Dahmen, and M. Szydlo, *Post-quantum cryptography (Eds, Daniel Bernstein, Johannes Buchmann, Erik Dahmen)*, Berlin Heidelberg: Springer-Verlag, 2009.

[7] C. Caicedo, J. Joshi, and S. Tuladhar, "Ipv6 security challenges," *IEEE Computer*, vol. 42, no. 2, pp. 36–42, 2009.
[8] C. Castelluccia, "Cryptographically generated addresses for constrained devices," *Wireless Personal Communications*, vol. 29, pp. 221–232, 2004.
[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] E. Dahmen and C. Kraub, "Short hash-based signatures for wireless sensor networks," *Lecture Notes in Computer Science*, vol. 5888, pp. 463–476, 2009.
[11] E. Durdagi and A. Buldu, "Ipv4/IPv6 security and threat comparison," in *Procedia Social and Behavioral Sciences (WCES'10)*, vol. 2, pp. 5285–5291, 2010.
[12] T. Eisenbarth, "Cryptography and cryptanalysis for embedded systems [Ph.D. Thesis]," 2009. (http://www.emsec.rub.de/media/crypto/attachments/files/2011/02/thesis_eisenbarth.pdf)
[13] M. S. Hwang, C. C. Lee, and S. K. Chong, "An improved address ownership in mobile IPv6," *Computer Communications*, vol. 31, pp. 3250–3252, 2008.
[14] S. Kuang, R. Elz, and S. Kamolphiwong, "Investigating enhanced route optimization for mobile IPv6," in *Proceedings of The 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC'08)*, pp. 1–7, Hsinchu, Taiwan, Aug. 2008.
[15] J. Li, P. Zhang, and S. Sampalli, "Improved security mechanism for mobile IPv6," *International Journal of Network Security*, vol. 6, pp. 291–300, 2008.
[16] Maemo.org, "Intro: The home of the maemo community," 2015. (http://maemo.org/intro/)
[17] R. Merkle, "A certified digital signature," 1978. (http://www.cse.msstate.edu/r̃amkumar/merkle1.pdf)
[18] Polarssi, "mbed TLS," 2015. (https://tls.mbed.org/)
[19] S. Qadir, M. U. Siddiqi, and F. Anwar, "Cryptographically Generated Addresses (CGAs): a survey and an analysis of performance for use in mobile environment," *International Journal of Computer Science and Network Security*, vol. 11, no. 2, pp. 24–31, 2011.
[20] S. Qadir, M. U. Siddiqi, and F. Anwar, "A study of cga- (cryptographically generated address) signature based authentication of binding update messages in low-end mIPv6 node," in *Proceedings of The 4th International Conference on Computer and Communication Engineering (ICCCE'12)*, pp. 510–514, Kuala Lumpur, Malaysia, July 2012.
[21] Saphir, "sphlib 3.0," 2015. (http://www.saphir2.com/sphlib/)
[22] M. Stevens, "Attacks on hash functions and applications [phd thesis]," 2012. (http://www.cwi.nl/system/files/PhD-Thesis-Marc-Stevens-Attacks-on-Hash-Functions-and-Applications.pdf)

Sana Qadir received her MSc in Computer and Information Engineering in 2010. She is cuurently a PhD candidate at the Faculty of Engineering in the 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 F. 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.