Asian Journal of Information Technology 5(5) : 549-554, 2006

Strengthening Ecdsa to Be More Suitable to Mobile Networks

Komathy, K. and P. Narayanasamy
Department of Computer Science and Engineering, Anna University, India

Abstract: The primary objective of the paper is to augment the efficiency of signature process in Elliptic Curve Digital Signature Algorithm(ECDSA), which turns out to be a bottleneck in mobile devices due to its heavy consumption of resources. An improved algorithm for strengthening the speed of verification process in ECDSA is proposed and evaluated. Also the recent research studies have demonstrated collision attacks on popularly used hash functions, including the widely deployed MD5 and SHA-1 algorithms. To overcome this threat, this paper attempts to present a few novel solutions to reinforce the ECDSA signature against such attacks. Several simple and efficient message pre-processing techniques have been suggested, which can be combined with SHA-1 so that applications are no longer vulnerable to the known collision attacks. A detailed analysis on the results of implementation of these techniques has also been presented.

Key words: Elliptic curve digital signature algorithm, elliptic curves over prime fields, signature verification, hash algorithm, dsa signature algorithm

INTRODUCTION

Elliptic Curve Cryptography (ECC) has gained increasing popularity in public key cryptography since it was first proposed by Miller[1] and Koblitz[2]. While implementing ECC, there are many factors that need to be considered with respect to mobile networking. The factors include: Security considerations, methods for implementing EC arithmetic, methods for computing EC scalar multiplications, application platform (hardware or software), computing environment (processor speed, memory size, power consumption) and constraints of the communication environment (bandwidth, response time). Compared with other public key cryptography such as RSA[3], ECC utilizes shorter key sizes for the same level of security, which translates into fast computation and less demands on memory and CPU. These advantages make ECC ideal for use in embedded systems and mobile devices where storage, power and computing resources are at a premium.

A major component in ECC is point (or scalar) multiplication. ECDSA algorithm computes kP (k times the EC Point, P) during key generation and signature generation phases and also expends the system resources while computing multiple scalar multiplications such as ($u_1P + u_2Q$) during signature verification. It was reported by Scott Vanstone[4] that the signature verification of ECDSA takes twice the time taken for signature generation considering the same security level. Hence it is a primary concern to devise methodologies for reducing verification time so that ECDSA could be more adaptable to mobile devices. Section-2 of this study analyzes the possible improvements on joint scalar multiplication to meet the demand of mobile devices such as resource conservation.

The recent advances in cryptanalysis on hash functions MD5 and SHA-1 have been incredible and the collision attacks are of particular practical importance as these algorithms are extensively deployed in applications. Wang *et al.*,[5] presented the first attack on the full SHA-1, where they show that finding collisions is at most of complexity 2^{69}. Other recent papers on cryptanalysis of collision-resistant hash functions (CRHF)[6-12] focus mainly on the attacks against MD5 and SHA-1. Section-3 of this paper draws out the feasible strategies towards strengthening the message blocks.

ENHANCING THE SPEED OF ECDSA SIGNATURE VERIFICATION

Many efficient algorithms on point multiplication have been developed such as binary, non-adjacent form (NAF) and several window methods that play tradeoffs[13] between storage space and execution time for scalar multiplication. M¨oller[14] introduced a binary method called interleaving, which aims at reducing the number of ECDBL(EC Point Doubling) operations required for a multi-scalar multiplication. Another popular method called comb algorithm proposed by Lim and Lee[15], uses a binary matrix, which can dramatically speed up the computation of point multiplication keeping the pre-computation space the same as other window methods.

Corresponding Author: Komathy, K., Department of Computer Science and Engineering, Anna University, India

SIMULTANEOUS MULTIPLICATION [SIMUL] ALGORITHM

ECDSA signature verification consumes more time than its generation due to the presence of multiple scalar multiplications. To reduce this, the computation is divided into two stages namely, pre-computation stage and evaluation stage. Most researchers focused on reducing the evaluation time of multiplication[15-17]. Recent studies also look at plausible recommendations on faster execution methods for scalar multiplication[5,18-20]. Only very few works are available in optimizing the pre-computation time, which devours predominant level of resources of the system. An improved algorithm for efficient simultaneous multiplication called SiMul, therefore, is designed and evaluated its strength. Algorithm-1 lists out the steps involved in the design.

STRENGTHENING PRE-COMPUTATION PROCESS

The pre-computation process of SiMul is made stronger by using window non-adjacent form (w-NAF) multiplication. The scalar multiplier k is represented in window-NAF format and the length of w-NAF(k) is at most one longer than the binary representation. Point multiplication kP or kQ is done using w-NAF during pre-computation stage. The running time of w-NAF is approximately [(1ECDBL + (2^{w-2}-1) ECADD) + (m/(w+1) ECADD + mECDBL)], where m is the bit-length of k, w is the window size, ECADD represents EC point addition and ECDBL represents EC point doubling. The amount of storage is brought down to 2^{w+1} number of points with a two-Table implementation.

Algorithm 1: [SiMul] Simultaneous multiple point multiplication

```
Input: Window width w, $k = (k_{m-1},\ldots, k_1, k_0)_2$; $l = (l_{m-1},\ldots, l_1, l_0)_2$; P,Q å E(Fp)
Output: kP + lQ

Pre-computation:
    1. Pre-compute iP + jQ for all i, j $\in [0, 2^{w}-1]$
    2. Write $k = (k_{d-1},\ldots, k_1, k_0)$ and $l = (l_{d-1},\ldots, l_1, l_0)$ where
       each ki and li is a bitstring of length w and d = m/w
Evaluation:
    3.X ← $O$
    4.For $i$ from $d$ - 1 downto 0 do
    4.1X ← $2^w$X
    4.2X ← X + ($k_i$P + $l_i$Q)
        where ($k_i$P + $l_i$Q) are computed using Shamir
        method
    Return(X)
```

Simultaneous inversion: Generally, points are pre-computed and stored in affine co-ordinates to speed up the execution during the evaluation stage. The problem faced in this mode is that every operation in affine co-ordinates requires an inversion. The inverted list of all points is also prepared simultaneously after the pre-computation Table is constructed. Inversion is embedded as a part of the pre-computation process.

Pre-Computed points for simultaneous multiplication: Table 1 shows a sample of pre-computation Table for a window width of 3 used in SiMul. The Table is constructed by omitting certain points from storage during pre-computation process. For example, the diagonal elements (2P + 2Q), (3P + 3Q)…(nP + nQ) are excluded from storage except (P + Q). The scalar multiplication of type (nP + nQ) which is equal to n(P + Q), for n = 2,3… is postponed to evaluation stage. Evaluation of n(P + Q) will consume only one scalar multiplication if necessary, whereas storing (n-2) diagonal points spends out (n-2)ECADD operations. Also, while preparing the pre-computation Table, the following steps are adopted:

Step 1: Points O, P and Q
Step 2: (P + Q), 2P, 2Q, 3P, 3Q,…, multiples of P and Q
Step 3: Rest of the matrix except diagonal elements

Evaluation of siMul algorithm: The evaluation stage of simultaneous multiplication (kP + lQ) uses Shamir's trick, which is explained through Example-1. Here, scalars k and l of m-bit length are divided into d number of portions by window width, w. Each portion of k and l is simultaneously evaluated using Shamir's method and the intermittent result is obtained by multiplying with $2^{w(d-j)}$ for j = 1,2.. d and adding them leads to the final result. It expects a running time of ($2^{2w} - 3$) ECADD + ((d − 1) ($2^{2w} - 1$)/22w ECADD + (d − 1)w * ECDBL) and a storage for 2^{2w} points. Storage increases compared to Comb algorithm as it requires both points P and Q in multiples. Saving in storage is also achieved as discussed in section 2.2.2 by reducing it to (2^{2w}-2^w).

Example-1: To compute 33P + 13Q simultaneously using Shamir method. The parameters for the algorithm are: m = 6; w = 3 ; d = m/w = 2

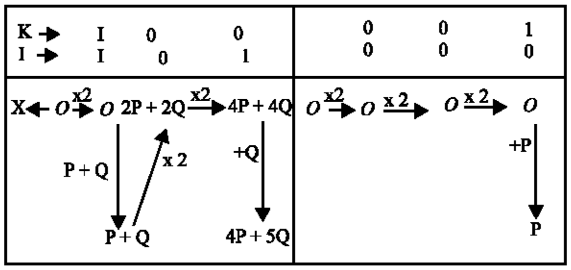

Table 1: Pre-Computation Table for window width w = 3

O	P	2P	3P	. . .	7P
Q	P+Q	2P+Q	3P+Q	. . .	7P+Q
2Q	P+2Q	2P+2Q	3P+2Q	. . .	7P+2Q
3Q	P+3Q	2P+3Q	3P+3Q	. . .	7P+3Q
:	:	:	:	. . .	:
7Q	P+7Q	2P+7Q	3P+7Q	. . .	7P+7Q

Message pre-processing techniques: The message pre-processing techniques for improving the collision resistance assume that the underlying hash function is not to be modified. Let M be a message string to be hashed and H be a standard hash function SHA-1. A message of an arbitrary length is converted into a hash value of 160 bits by the one-way hash function, SHA-1. The following are the formal notations used throughout this paper to represent the parameters of SHA-1 algorithm in ECDSA signature:

M1, M2,…,Mn	512 bit blocks of a message to be hashed
m0, m1, ...,m15	sixteen 32-bit message words for each Mi
w	the expanded message block of m
w0, . . .,w79	eighty 32-bit message words of w
C	compression function
H	hash function
SIG	Signature algorithm using ECDSA

An Overview of collision attacks: Focusing on a single block, the general common strategy behind these collision attacks involves finding a message difference $\Delta(w) = (w\text{-}w`)$ between two expanded messages such that the probability that $C(m)$ equals $C(m`)$ is higher than expected. This is possible when it can be arranged such that during the round computations of the blocks M and M`, the state vectors never deviate significantly and can be corrected with high enough probability. Local collision is a series of a few rounds in which certain small differences in the expanded message words will be absorbed with reasonable probability. Due to the message expansion there will be many different words of M and M`, so these local collisions must be strung together. Disturbance vectors describe how the local collisions are joined. The entire sequence of differences in the state vectors is called a differential path. The overall success probability depends on the simultaneous satisfaction of a set of conditions for each local collision.

The structure of the various attacks consists of analysis of the local collisions, search for a low Hamming weight disturbance vector and a brute force search on input messages. A variety of methods are used to boost the success probability, including specifying concrete conditions for the differential path, message modification so that some conditions always hold and usage of two blocks to construct collisions from near collisions.

There are several strategies, which one might employ to prevent the success of these approaches. The most obvious approach is to prevent the existence of any differential path that leads to (near) collisions and holds with probability greater than $2^{-n/2}$. An additional precaution would be to restrain the power of the message modification techniques, thereby significantly reducing the success probability of the attack. A third possibility is to consider situations in which the Merkle-Damg°ard iterative structure[7,21] cannot be exploited; for example if single message bits were to affect multiple blocks. Easy and a feasible solution would be to randomize the input message before hashing is applied.

Randomization of message block: Message randomization in a single block defines a derived hash function H* which calls H as a subroutine. The proposal is to preprocess the message before it is hashed in a standard way. Let Φ: M ? M* be a preprocessing function mapping strings to strings. For such function, a derived hash function H* may be defined by H*(M) = H (Φ(M)), where we assume that Φ is a relatively simple function and the derived hash function H* is collision resistant with respect to known attacks, even if H is not. The function Φ must be chosen appropriately for a particular H to ensure that H* is secure. For signing a message M, the signer chooses a 512-bit random value r and computes SIG(r,H*(M)). Here, the pair (r,H*(M)) represents a standard encoding of the concatenation of the values r and H*(M). Randomization is applied to the hash input before the hash function is called. The random value r is XORed with each 512-bit block of M. If M is not an exact multiple of 512-bit blocks then the shorter last block is XORed with an appropriately truncated r. H*(M) is now defined as H*(M) = H(M1 ⊕ r, . . . , Mn ⊕ r) where (Mi ⊕ r) will result in giving the same length as the Mi. The random value r is prepended with the hash value such as (r || H(M1 ⊕ r, . . . , Mn ⊕ r)). The signature on message M now consists of the pair (r, SIG(r,H*(M))).

Another variant of this approach is to XOR a different random r with each block instead of the same value r. However, this method poses high overhead on communication since all the random values are to be transferred to the recipient apart from high computation cost on generating random numbers of length equal to the length of the message. Randomization could also be achieved by having the first block of the message XORed with r, the next block with byte-wise circular rotated (left or right) random value and so on. For resource constrained devices such as mobile devices, lower-bit random value could be chosen such as 128 bits and r could be built by concatenating four times to create a 512-bit r.

Message interleaving: In this approach, the basic idea is to duplicate each message word so that each bit appears twice after the preprocessing. Assuming the ith block of message M is broken up into some number of 32-bit words: Mi = (m0, m1, . . .mk), then the preprocessed message would be Φ(Mi) = (m0, m0, m1, m1,...mk, mk), where each word appears twice. Another variant of this approach would be to interleave the 512-bit random r between any two blocks of the original message, thus providing an IV randomization feature for each application of the compression function. The obvious drawback of interleaving method is the added computation i.e. doubling the cost of the original hash function.

Pre-processing with SHA-1 function: For the existing implementation of SHA-1, the ECDSA signature generation is generally carried out for a given input message by three functions as described below:

```
Sign_Init(context)
// context pertaining to the algorithm and the provider
Sign_Update(context, msg_input)
// message input of arbitrary length
Sign_Final(hash, context)
// hash is being used for signing at the final stage
```

The proposed implementation for the improved ECDSA signature with the same sequence of functional calls is as follows:

```
Sign_Init(context)
// same as  original ECDSA
Sign_Update(context, randomized_input)
// modified structure of the call
{
newMsgInput = MsgPreProcess(process_id, msg_input)
//uses the named pre-process
Sign_Update(context, newMsgInput)
// for randomization
}
Sign_Final(hash, context)
// same as ECDSA
```

The pre-processing step is done as a private function call that is invisible to upper layer protocols using ECDSA signature. The computational overhead on preprocessing on message input is negligibly small.

EXPERIMENTAL ANALYSIS OF THE PROPOSED SOLUTIONS

The proposed algorithm for Simultaneous Multiplication, SiMul is tested under Pentium-IV, 3.2Ghz, Windows XP system on JSDK2 platform and its performance is compared with comb algorithm The implementation considers only the NIST prime curves namely, P-192, P-224, P-256, P-384 and P-526 varying the window width w from 0 to 13. It is found that the execution cost for these curves increases exponentially as the window size increases. The results are based on the hash function, SHA-1 with a160 bit hash value.

Figure1 shows the performance of signature verification comparing comb and SiMul algorithms in different co-ordinate systems. Though the process is repeated for window sizes ranging from 8 to 13, this study

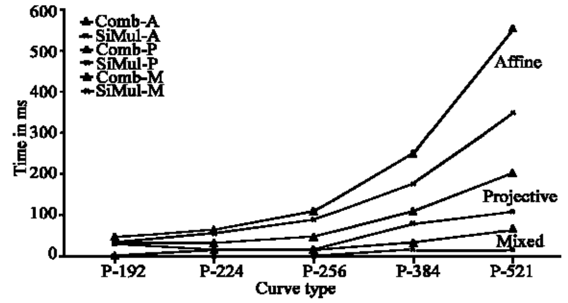

Fig. 1: Comb and SiMuL algorithms on signature verification for window size as10

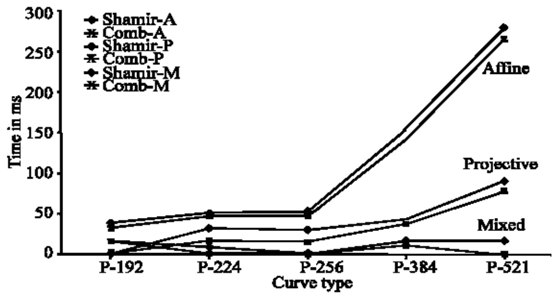

Fig. 2: Comb and SiMuL algorithms on signature generation for window size as10

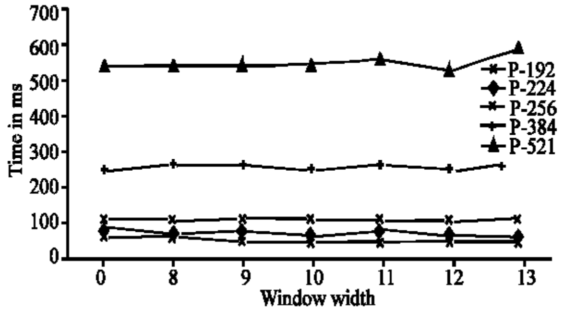

Fig. 3: SiMul for ECDSA verification in Affine for different prime curves

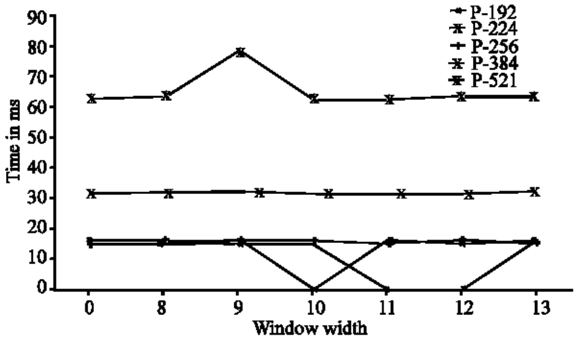

Fig. 4: SiMul for ECDSA verification in mixed for different prime curves

Table 2: Comparing ECDSA and RSA with message pre-processing under the same security level

ECDSA			RSA		
Key Size	Sign time ms	Verify time ms	Key Size	Sign time ms	Verify time ms
192	31	47	1024	15	0
224	15	31	2048	78	0
256	16	47	3072	235	0
384	47	125	7680	3187	31
521	125	328			

presents only the output for the window size 10 as a sample. In all cases, SiMul achieves a much faster execution time than the comb, especially when it is used for higher prime curves such as P-384 and P-521. In mixed mode, time reaches zero for the lower end curves. Generally, ECDSA signature verification uses twice the time taken for signature generation. This implementation demonstrates that the time difference between signature verification and signature generation could be controlled and thereby achieving better efficiency and speed.

Signature generation in comb exhibits relatively the same speed as the counterpart on different co-ordinate systems. The reason being that the signature generation involves only a single scalar multiplication, kP. Figure 2 shows this attitude of comb following SiMul very closely. For mixed mode, the signature time almost reaches

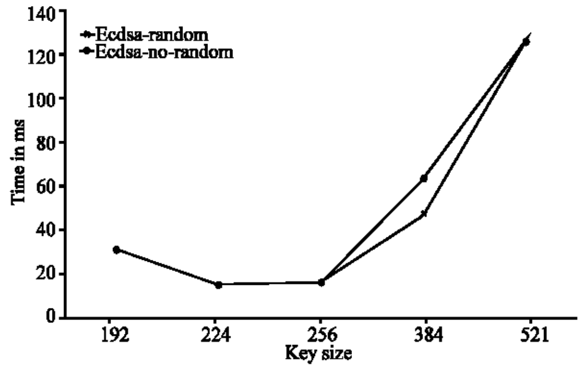

Fig. 5: SiMul ECDSA signature generation with and without randomization

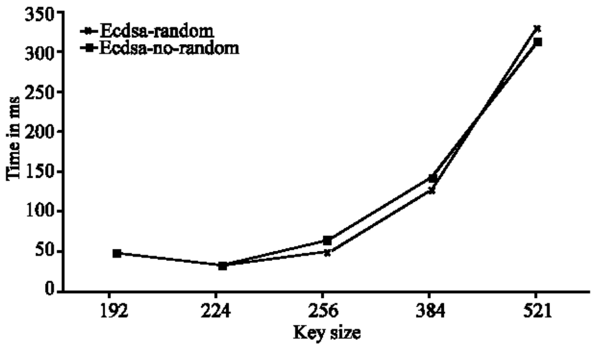

Fig. 6: SiMul ECDSA signature verification with and without randomization

zero neglecting the inconsistencies of Java compiler. Also it is found that key generation in mixed mode tends to be a heavy-duty process than the other two co-ordinate systems and it is due to the preparation of a large pre-computation Table and a simultaneous inverted list. Each prime curve stabilizes itself during signature verification irrespective of the window size under SiMul and this is illustrated in Fig. 3. Affine is found to be much costlier than mixed co-ordinate system as per Fig. 4.

SiMul algorithm is also studied for improvement in performance after message block randomization. The outcomes of the experiment on two popular signature schemes namely ECDSA and RSA are compared in Table 2 keeping the same security level. Signature verification of RSA seems much faster than that of ECDSA whereas the signing is vice versa. Figure 5 and Fig. 6 illustrate that the ECDSA signing and verification with randomized message input have not cost much additional computation. By optimizing the code further, the degradation in the performance could be compensated.

CONCLUSION

This study has attempted to reduce the processing mandatory solution for resource constrained mobile devices. The study imparted few effective methods on SiMul, a proposed algorithm to reduce the time for validating the signature during execution by augmenting the pre-computation process. SiMul, has shown significant progress in the verification process by reducing the computation overheads but at a penalty of large storage compared to comb algorithm. The study has revealed that SiMul in mixed mode gives a better performance than in the other two co-ordinate systems especially for high-end prime curves. Randomizing and interleaving on each message block have been suggested on SHA-1 algorithm for securing ECDSA signature against block collision attack. These methods exploit simple methods of enhancing message mixing to increase security. The drawback is the additional computational cost. The proposed solutions can be viewed as a general purpose, safer, collision resistant way of using SHA-1. Due to their simplicity, the proposal can be appealing for practitioners who wish to increase security in a short term, without changing the underlying hash function at all. Further strengthening of ECDSA with respect to the storage of pre-computed points will be considered as an immediate future work. The objective would be to reach an optimized level between evaluation time and pre-computation time.

REFERENCES

1. Miller, V.S., 1986. Use of Elliptic Curves in Cryptography, Advances in Cryptology-CRYPTO'85, LNCS 218, Springer-Verlag, pp: 417-426.
2. Koblitz, N., 1987. Elliptic Curve Cryptosystems, Mathematics of Computation, pp: 203-209.
3. Rivest, R.L., A. Shamir and L.M. Adleman, 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of ACM, pp: 120:126.
4. Scott Vanstone, 2005. Deployments of Elliptic Curve Cryptography, 9th workshop on Elliptic Curve Cryptography, ECC-2005, at URL http://www. cacr. math.uwaterloo.ca/conferences/2005/ecc2005/vanstone.pdf
5. Kuang, B., Y. Zhu and Y. Zhang, 2004. An Improved Algorithm for uP + vQ using JSF, Applied Cryptography and Network Security-ACNS LNCS 3089, Springer, pp: 467-478.
6. Antoine Joux, 2004. Multicollisions in Iterated Hash Functions-Application to Cascaded Constructions, Crypto.
7. Damg°ard, I.,1990. A Design Principle for Hash Functions, in Advances in Cryptology Crypto’89, Springer-Verlag.
8. Eli Biham and Rafi Chen, 2004. Near-Collisions of SHA-0, Crypto.
9. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet and William Jalby, 2005. Collisions of SHA-0 and Reduced SHA-1, EUROCRYPT.
10. Xiaoyun Wang and Hongbo Yu, 2005. How to Break MD5 and Other HashFunctions, EUROCRYPT.
11. Xiaoyun Wang, Hongbo Yu and Yiqun Lisa Yin, 2005. Efficient Collision Search Attacks on SHA-0, Crypto.
12. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu, 2005. Finding Collisions in the Full SHA-1, Crypto.
13. Blake, I.F., G. Seroussi and N.P. Smart, 1999. Elliptic Curves in Cryptography, Cambridge Univ. Press.
14. M¨oller, B., 2001, Algorithms for Multi-exponentiation, Selected Areas in Cryptography-SAC 2001, LNCS 2259, Springer, pp: 165-180.
15. Lim, C. and P. Lee, 1994. More Flexible Exponentiation with Precomputation, Advances in Cryptology- CRYPTO'94, LNCS 839, Springer-Verlag, pp: 95-107.
16. Cohen, H., A. Miyaji and T. Ono, 1998. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates, Advances in Cryptology-ASIACRYPT LNCS 1514, Springer, pp: 51-65.
17. Morain, F. and J. Olivos, 1990. Speeding Up the Computations on an Elliptic Curve using Addition-Subtraction Chains, Theoretical Informatics and Applications, pp: 531-543.
18. Dahmen, E., K. Okeya and T. Takagi, 2005. An Advanced Method for Joint Scalar Multiplications on Memory Constraint Devices, 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks-ESAS 2005, LNCS 3813, Springer, pp: 189-204.
19. Dahmen, E., K. Okeya and T. Takagi, 2005. Efficient Left-to-Right Multi-Exponentiations, Technical Report TI-2/05, at URL http://www.cdc.informatik. tu-darmstadt.de/reportsREADME.TR.html.
20. Solinas, J.A., 2001, Low-weight binary representations for pairs of integers, University of Waterloo, Technical Report CORR 2001-41, at URL http://www.cacr.math.uwaterloo.ca.
21. Merkle, R., 1990. One Way hash Functions and DES, in Advances in cryptology Crypto’89, Springer-Verlag.