A Fault Tolerant Parallel Computing Scheme of Scalar Multiplication for Wireless Sensor Networks

Yanbo SHOU[1] and Hervé Guyennet[1]

University of Franche-Comté, Besançon, France
{yshou, hguyennet}@femto-st.fr

Abstract. In event-driven sensor networks, when a critical event occurs, sensors should transmit messages back to base station in a secure and reliable manner. We choose Elliptic Curve Cryptography to secure the network since it offers faster computation and good security with shorter keys. In order to minimize the running time, we propose to split and distribute the computation of scalar multiplications by involving neighboring nodes in this operation. In order to improve the reliability, we have also proposed a fault tolerance mechanism. It uses half of the available cluster members as backup nodes which take over the work of faulty nodes in case of system failure. Parallel computing does consume more resources, but the results of simulation show that the computation can be significantly accelerated. This method is designed specially for applications where running time is the most important factor.

Keywords: Wireless sensor networks, Elliptic curves, Scalar multiplication, Parallel computing, Fault tolerance.

1 Introduction

The advances of micro-electro-mechanical technology in recent years have enabled the fast development of smart sensor node. A sensor node is small electronic device which is usually composed of processing, sensing, radio communication and power supply units [1, 2]. It is programmable and is able to collect environmental data and communicate with other sensor nodes by using wireless technologies. Such sensor nodes have limited resources and are often battery-operated, they are not designed to handle complicated tasks and that's why a sensor node rarely works alone. Sensor nodes are supposed be deployed massively and be programmed to form automatically a fully functional network, called wireless sensor network.

Ideally, a sensor network consists of a large number of low-cost and low-power sensor nodes, which are interconnected with each other and operate in unattended manner. This kind of networks are often deployed in inaccessible and hostile zones where human interventions are not always possible. Today we can find a wide range of applications using wireless sensor technologies, such as environment monitoring, industrial sensing, medical care and military surveillance [3–7].

However sensor nodes are quite fragile and are vulnerable to various attacks due to the lack of resources and unreliability of wireless communication. [8] gives a detailed presentation of almost all possible attacks in wireless sensor networks. An efficient way to protect sensor networks is to use cryptographic techniques [8, 9]. On one hand, symmetric cryptographic algorithms are usually computational less expensive and easier to implement in hardware and software, but as we use the same key for data encryption and decryption, the key management in sensor networks becomes a challenging issue. On the other hand, asymmetric algorithms need more complicated computation, but as we use two different keys respectively for encryption and decryption, a compromised node can not provide clue to the private keys of other nodes.

In this paper we choose *Elliptic Curve Cryptography* (ECC) to secure the communications in sensor networks. It has become recently one of the most famous asymmetric cryptographic mechanism because of its shorter key length requirement and faster computation compared with other asymmetric mechanisms, such as RSA [10]. ECC has successfully attracted the interest of researchers, especially in the domain of embedded system where most of the devices have strict resource restriction.

The performance of ECC can be significantly improved by using mathematical tools, such as NAF, windowed method, projective coordinates [11]. Besides traditional algorithmic optimization, parallel computing is an other choice for accelerating the computation on elliptic curves. The computation task is split into smaller independent ones which are then distributed to neighboring sensor nodes and carried out simultaneously. However, sensor nodes communicate with each other through wireless connection, which is considered as unreliable and sensitive to radio interference. In addition, sensor nodes might suffer external attacks during computation. In this paper we present our new fault tolerant parallel computing scheme for elliptic curve cryptography in wireless sensor networks, which is actually the following of our research published in [12].

The rest of the paper is organized as follows. Section 2 presents basic concepts of ECC, and section 3 describes our parallel computing scheme. In section 4 and 5 we present possible fault tolerance issues and the related work. In section 6, we propose a fault tolerance mechanism designed specially for our parallel computing scheme, then it is tested by a simulator and the results are illustrated in 7. Section 8 concludes the paper.

2 Basic Concepts of Elliptic Curve

Elliptic curve cryptography (ECC) is an approach to public-key cryptography which is proposed independently by Koblitz [13] and Miller [14] in the 80's. It's suitable for creating lightweight and efficient cryptosystem, especially for embedded systems which have limited resources. Experiments prove that a 160-

bit elliptic curve key can provide the same security robustness as a 1024-bit RSA key [15].

In cryptography we work with the curves which are defined over finite field $\mathbb{F}_q$ where $q = p^m$. p is a prime number, called the characteristic of $\mathbb{F}$. $\mathbb{F}$ is finite prime field when $m = 1$ and $p \neq 2, 3$, which is also the configuration that we use in this paper. Such curves can be represented by the simplified Weierstrass equation (see formula 1).

$$y^2 = x^3 + ax + b \tag{1}$$

whose discriminant $\mathit{\Delta} = -16(4a^3 + 27b^2)$ and $\mathit{\Delta} \neq 0$.

All points on an elliptic curve, including the point at infinity, constitute an abelian group whose law is point addition, denoted by $+$, which combines 2 points on the curve to form a third point which is on the same curve. Suppose that $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are 2 points on an elliptic curve, then $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$ can be calculated by using the formulas 2 and 3.

$$\begin{cases} x_3 = s^2 - x_1 - x_2 \\ y_3 = s(x_1 - x_3) - y_1 \end{cases} \tag{2}$$

where

$$s = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2 \\ (3x_1^2 + a)/2y_1 & \text{if } P_1 = P_2 \end{cases} \tag{3}$$

Based on point addition, we may also perform point multiplication, also called scalar multiplication. For example $Q = kP$, where Q and P are 2 points on an elliptic curve, and k is a big integer. The most basic method to calculate kP is using Double-and-Add algorithm [16].

The security of ECC relies on the difficulty of solving the discrete logarithm problem. For $Q = kP$, given points Q and P, it's extremely difficult to compute the value of k if it's big enough. However it's not always easy to transform the data we want to protect into a point on elliptic curve. Thus in most of the cases, we only use ECC to establish a shared key between both parties, then we can use a symmetric-key cryptographic algorithm to encrypt our data.

A example of key exchange protocol is Elliptic Curve Diffie-Hellman [17]. Suppose that Alice and Bob share the same elliptic curve E whose generator point is G. To establish a shared key, Alice computes and sends her public key $Q_A = s_A G$ to Bob where s_A is her private key, meanwhile Bob sends $Q_B = s_B G$ to Alice where s_B is his private key. Then the shared key $K_{AB} = s_A Q_B = s_B Q_A = s_A s_B G$.

We can notice that the scalar multiplication is the most essential operation on elliptic curve, and it's also the most computationally intensive operation we need to perform. It is obvious that this kind of computation is too complicated for sensor nodes, performance optimization is then absolutely necessary.

3 Parallel Scalar Multiplication in WSN

We suppose that in a cluster based sensor network, cluster members send periodically environmental data to their cluster head which is responsible for data processing. Whenever a critical event is detected, the cluster head has to warn the base station as fast as possible in a secure and reliable manner.

Our parallel scalar multiplication method is based on the idea of [18] which doesn't need shared memory and offers an efficient scalar decomposition. For example we need to compute $Q = kG$ where Q and G are 2 points on an elliptic curve, and k is a large integer of length l which can be represented in binary $k = \sum_{i=0}^{l-1} k_i 2^i$. The cluster head splits k into n blocks B_i of $b = \lfloor \frac{l}{n} \rfloor$ bits according to the number of available cluster members.

$$B_i = \sum_{j=ib}^{(i+1)b-1} k_j 2^j$$

As all sensor nodes share the same elliptic curve which is configured and preloaded before deployment, then it is possible to precompute points $G_i = 2^{ib}G$ where G is the generator point. Thus the computation of kP can be rewritten in the following manner.

$$\begin{aligned} Q = kG &= \sum_{i=0}^{l-1} k_i 2^i \\ &= Q_0 + Q_1 + \ldots + Q_{n-1} \\ &= B_0 G + B_1 2^b G + \ldots + B_{n-1} 2^{b(n-1)} G \end{aligned} \tag{4}$$

We can notice in equation 4 that each Q_i can be considered as an independent task and can be treated separately. The entire process of parallel computing is graphically illustrated in figure 1.

1. Cluster members send data to cluster head periodically.
2. Cluster head detects a critical event. It broadcasts a call for parallel computing and then waits for responses.
3. Available cluster members send positive responses back to cluster head and become *Slave nodes* of parallel computing.
4. Cluster head decomposes the computation according to the number of available slaves. Then it sends the tasks to slaves, and it become the *Master node* of parallel computing.
5. Slaves complete the task and send the results back to their master.
6. Master nodes combines received results to obtain the final result, by which the master can encrypt its message and send it to the node of next hop.

As we can see that the parallel scalar multiplication on elliptic curve is a complicated operation, and every step must be carried out carefully. Moreover, the process needs reliable wireless communication which is not always possible in wireless sensor networks.

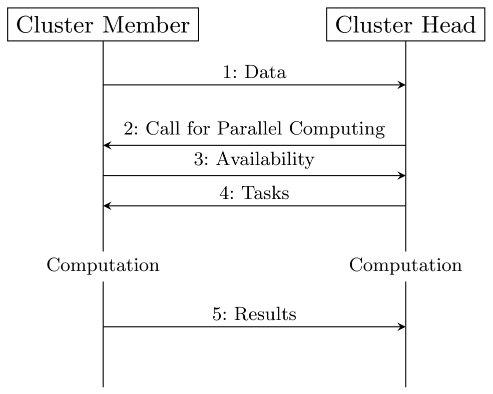

Fig. 1. Parallel Scalar Multiplication Process

4 Fault Tolerance Issues

Three terms have been introduced in [19] in order to explain the fault tolerance issues in wireless sensor networks.

A *fault* is a defect causing an *error*, which represents an incorrect status of system. Such erroneous status may lead the entire network to a *failure* in which the system deviates from its specification and loses its intended functionality.

As we know, sensor nodes are prone to suffer various faults and attacks[1][2]. There are many different types of fault in wireless sensor networks, it is extremely hard to design a fault tolerance technique which solves all kinds of problem. For our parallel computing scheme, we have identified three main possible errors during computation.

– *Missing result* : The master node needs to receive every single result from its slaves to form the final result. Missing results will make the master unable to complete the parallel computing. The missing result may due to various faults such as battery depletion, radio interference and physical attack.
– *Incorrect result* : The results sent back to master node might be wrong because of low battery level, radio interference and other internal errors of sensor nodes. If the master node compute the final result based on incorrect slave results, the entire encryption process will fail.
– *Faulty cluster head* : The cluster head plays the role of master node during parallel computing, if the cluster head becomes faulty in the course of task distribution and results reception, the parallel computing process has to be aborted.

As we can notice that either of these errors may cause the parallel computing scheme to fail. Before elaborating our fault tolerance mechanism, we will see in the following section a few fault tolerance techniques proposed in literature.

5 Related Work

Fault tolerance techniques mainly consist of 2 actions, *fault detection* which aims to detect malfunction in sensor networks, and *fault recovery* which recovers the system from incorrect status [19]. The main technique is to replace the faulty component by a new one, which is logical, since sensor network is constituted of large number of low-cost sensors. When a sensor node becomes faulty, it is worth nothing that we try to locate and replace it manually. The best way is to replace it by an other node which is already deployed in the zone.

A anomaly detection method based on analysis of sensed data is presented in [20]. We suppose sensors $S = \{s_1, s_2, \ldots, s_n\}$ are connected in a hierarchical topology. At every time interval Δ_k, each sensor measures a feature vector x_k^i, and very vector contains attributes $x_k^i = \{v_{kj}^i : j = 1 \ldots d\}$.

Two approaches are presented, in centralized way, every sensor sends its measured vectors $X_i = \{x_k^i : k = 1 \ldots m\}$ to its immediate parent, which merges the received data with its own vectors and then send them to its parent. All vectors are sent back step by step to the gateway node, on which a clustering algorithm [21] is executed. Vectors belonging to a cluster are kept, the others are discarded. The inconvenience of this approach is the big data transmission load which may reduce the lifetime of the network. In distributed way, the clustering algorithm is applied at every step, only necessary statistics are sent to the next step for following analysis. The volume of data to transmit can be considerably reduced.

An other method to identify faulty node is to launch a vote. In [22] sensors are deployed randomly in a zone, and we suppose that every sensor has at least 3 neighbors. Sensors are considered as neighboring sensors if they're in the radio range of each other, and each sensor broadcasts periodically to the neighbors its measured data which is then stored in their memory.

Sensors deployed in the same area are supposed to have similar measured values. A sensor S_i is interested in history data when more then half of its neighbors have significantly different values. It can use $\Delta d_{ij}^{\Delta t_l}$ which represents the difference between measured value of sensor S_i and S_j from time t_{l+1} to t_l. The status of the sensor T_i is likely faulty (LF) if the measurements change over time significantly.

The status of sensor S_i is good(GD) if $\forall S_j \in N(S_i)$ and $T_j = LG$ (likely good), $\sum(1 - c_{ij}) - \sum c_{ij} \geq \lceil |N(S_i)|/2 \rceil$, where $N(S_i)$ represents all neighbors of S_i and c_{ij} is a test result generated by S_i and its neighbors. $c_{ij} = 0$ if S_i and S_j have the same status, otherwise $c_{ij} = 1$.

In [23], a fault tolerant clustering mechanism is proposed for sensor networks. Cluster heads evaluate their status and diffuse status update messages to other cluster heads through inter-cluster communication. Faulty cluster head can be identified by vote of all cluster heads. Once the identification is done, the network pass to the next step, system recovery.

If a cluster head becomes faulty, all its cluster members are then divided into smaller groups and merged with neighboring clusters. A sensor S_j belongs to cluster head G_i if $S_i \in RSet_{G_i} \Leftrightarrow (R_{G_i} > d_{S_j \to G_i}) \wedge (R_{S_{j.max}} > d_{S_j \to G_i})$ where

$RSet_{G_i}$ is the range set of G_i, $d_{S_j \to G_i}$ is the distance between G_i and S_j, R_{G_i} and $R_{S_{j.max}}$ are respectively the ranges of G_i and S_j. Once S_j is assigned to G_i, a final set $FSet_{G_i}$ is constructed based on minimum communication cost between sensors and cluster head.

A cluster head can still construct a backup set, denoted $BSet$, containing sensors which belong to its $RSet$, but not its $FSet$. $S_j \in BSet_{G_i} \Leftrightarrow (S_j \in RSet_{G_i}) \wedge (S_j \notin FSet_{G_i})$. If the cluster head of S_j becomes faulty, it can be recovered if it's present in an other cluster head's $BSet$.

In a landslide monitoring application presented in [24], sensors are formed in clusters which contain a cluster head (CH) and a node leader (NL). The NL aggregates the data collected from other members and sends it to CH, which will then forward the data to base station via multi-hop transmission.

For fault tolerance purpose, the intersection zone of 2 overlapping clusters is considered as a sub cluster, which contains only a NL. When a CH is failed, its NL will send the latest aggregated data to the NL of the sub cluster, then the data will be forwarded to the CH of the other cluster. If NL is failed, its CH will take over its work, and a NL election algorithm will be executed at the same time.

We can notice that in sensor networks, one sensor node is barely able to detect system malfunction or recover the network from faulty status alone. The idea of sensor networks is to make sensor nodes cooperate together to achieve a common goal. For fault tolerance, sensor nodes should also work collectively for fault detection and system recovery. We can also see that the basic method of system recovery is to prepare a backup node which takes over the work of the faulty node.

6 Countermeasures Against Sensor Node Failure

According to [25], fault tolerance techniques can be designed for and applied to (from lowest to highest) hardware, system software, middleware and application layers. Generally speaking, techniques designed for lower layer, such as hardware layer, are more generic, which means they can usually be applied to different applications. However higher layer techniques are more application-specific, a fault tolerance method is designed specially for a particular application, and it can hardly be used in other applications.

Certainly, fault tolerance methods for application layer are not as portable as the other ones designed for lower layers, but they can be carefully tailored to meet the specific needs of applications. For our parallel computing scheme, we have decided to design and deploy our fault tolerance technique on application layer.

Our fault tolerance method is also based on the idea of backup nodes. Suppose that there are n available members in a cluster, and the cluster head CH needs to perform a parallel scalar multiplication. Instead of involve all available cluster members in parallel computing, we use only $\lfloor \frac{n}{2} \rfloor$ members for parallel computing, the rest of cluster members are used as backup nodes. If the probability of a member is faulty is p, which also means the probability that a parallel scalar multiplication fails. The use of backup node can reduce the probability of system failure to $p/2$.

The selection of slaves nodes and of backup nodes is driven by algorithm 1 where L_a is the list of all available cluster members, L_s and L_b are respectively 2 lists of selected slave nodes and backup nodes.

Algorithm 1: Algorithm of backup selection

```
Data: $L_a$
Result: $L_s, L_b$
Sort($L_a$);
forall the $\varphi \in L_a$ do
    $\omega \leftarrow getBackup(\varphi)$;
    $L_a \leftarrow L_a - \varphi$;
    $L_a \leftarrow L_a - \omega$;
    $L_s \leftarrow L_s + \varphi$;
    $L_b \leftarrow L_b + \omega$;
end
```

Before node selection, we sort L_a by the number of its available neighbors in ascending order, since we want the φ having the least neighbors to choose its backup node first. *getBackup*() is a function which returns the most suitable backup node for φ. Every φ is evaluated by multiple parameters, such as RSSI, remaining energy, distance to φ. We always choose a backup node having the most reliable connection.

As we have explained in our last paper [12], to avoid large radio communication overhead, the number of slave nodes participating in parallel computing should be limited at less than 5. Thus the possibility for performing fault tolerant parallel scalar multiplication depends on the number of available cluster members, denoted by n :

- $n \in [0, 1]$: Unable to perform fault tolerant parallel computing.
- $n \in]1, 8[$: Parallel computing using $\lfloor \frac{n}{2} \rfloor$ slaves and $\lfloor \frac{n}{2} \rfloor$ backup nodes.
- $n \in [8, \infty]$: Parallel computing using 4 slaves and 4 backup nodes.

6.1 Result detection

To deal with the missing result error, we ask the backup node to prepare an other copy of result for the slave that it's monitoring. We can see in figure 2 that tasks are sent not only to slave nodes, but also to backup nodes. Both slave node and backup node have the same task. After computation, if a backup node *detects* that the slave node doesn't send result back to master node, it will send its local copy instead of the slave.

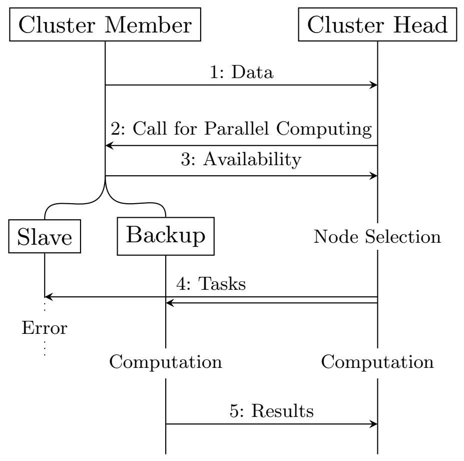

Fig. 2. Parallel scalar multiplication with result detection

Suppose that all cluster members have the same distance to their cluster head. The energy cost of wireless transmission is a function of message size m and distance d, $E_{T_x} = e(m, d)$, and the energy cost of reception is a constant E_{R_x}. We may notice that the parallel scalar multiplication with result detection doesn't consume more energy than the original parallel computing scheme. However, the fault tolerant version takes a little more time, since backup nodes have to wait a small period before sending its result.

6.2 Result verification

For incorrect result error, the slave node does send result back to its master, but an incorrect one. Just like the case of missing result, both slave node and backup node have the same task.

Backup node is supposed to *verify* the result sent by the slave node. When slave node sends its result, as the backup node is in its radio range, it can also receive the result. Backup node compares the result with its local one. If the results are different, the backup node will send immediately a warning message to their master node so that the result is discarded, and the task in question will be repeated locally by the master node.

When no fault occurs, the system consumes almost the same energy and time than the original version. But when some slave nodes become faulty, the energy and time cost is proportional to the number of faulty nodes.

For the third case, when cluster head is faulty, the system has to execute a cluster head election algorithm such as the one described in [26].

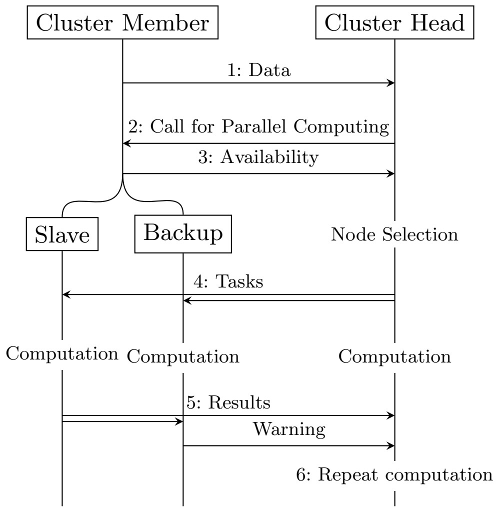

Fig. 3. Parallel scalar multiplication with result verification

7 Simulation and Performance Evaluation

To assess the performance of our method, we have created a simulator in Java which estimates the running time and the energy cost of our fault tolerant parallel computing scheme.

The simulator creates a random sensor cluster in an ideal free space without obstacle, and asks the cluster members to perform parallel scalar multiplications. We suppose that the cluster uses a TDMA based MAC protocol which allows only one sensor to send data at a time. Moreover, all nodes share the same elliptic curve which is defined over a finite prime field and uses $NIST_{192}$ recommended parameters [16]

We also suppose that the sensor nodes communicate with each other using Zigbee protocol whose throughput is studied in [27]. As in sensor networks, most of the energy is consumed during wireless communication, only the energy cost of radio communication is taken into account, and the formula for energy cost estimation is presented in [26] and [28]. In addition, the running time of scalar multiplication is estimated based on the results published in our last paper [12].

The simulator mainly focus on the execution of our parallel computing protocol. Our goal is not to retrieve the precise values of the running time and the energy cost, but to illustrate the impact of our fault tolerance method on the parallel computing scheme. Tests are run is three different cases:

– There are just enough nodes for parallel computing without fault tolerance, but not enough to have a backup node for each slave participating in computation.
– There are enough nodes for parallel computing with the proposed fault tolerance mechanism activated.

– There are enough nodes for fault tolerant parallel computing, but faults occur during the computation.

In our last paper, we have concluded that for efficiency reasons, the maximum number of slaves participating in computation should be limited at 4. We define that n is the number of available cluster members, and the tables 1 and 2 contain the simulation results of the first case where $n \leq 4$.

Nb of cluster members	Running time (ms)	Energy cost (mJ)
0	2308.77	0.06639066
1	1158.88	0.17869498
2	777.75	0.27243610
3	588.81	0.34194090
4	476.75	0.41024986

Table 1. Performance of the parallel computing scheme ($n \leq 4$)

Nb of cluster members	Running time (ms)	Energy cost (mJ)
0	2308.70	0.06639066
1	2308.77	0.06639066
2	1159.01	0.16637882
3	1159.01	0.16637882
4	778.01	0.24765018

Table 2. Performance of the fault tolerant parallel computing scheme ($n \leq 4$)

The results are also graphically illustrated in figures 4. As the node usage is doubled when our fault tolerance method is applied, so on the one hand, only $\lfloor n/2 \rfloor$ nodes can participate in parallel computing and it takes more time to finish the tasks, on the other hand backup nodes don't need to communicate with the cluster head when no fault occurs, the cluster can consumes less energy.

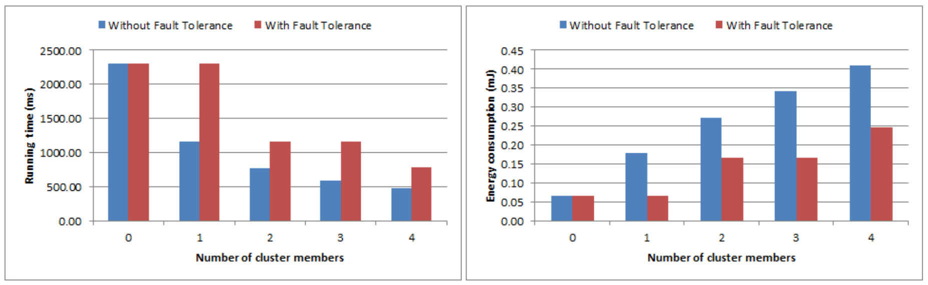

Fig. 4. Running time (ms) and energy cost (mJ) ($n \leq 4$)

In the second case where $n \leq 8$ which means the cluster head always has enough slaves to perform parallel scalar multiplication no matter if our fault tolerance method is used.

Nb of slaves	Running time (ms)	Energy cost (mJ)
0	2308.70	0.06639066
1	1159.01	0.16637882
2	778.01	0.24765018
3	589.20	0.34410666
4	477.27	0.43921370

Table 3. Performance of the fault tolerant parallel computing scheme ($n \leq 8$)

As we can see in table 3 and in figure 5, when our fault tolerance method is used, the cluster needs slightly more time to complete the computation, since more nodes are involved in parallel computing, and backup nodes also need to receive tasks from cluster head during task distribution. The total energy cost increases gradually with increase in number of slaves. The difference of energy cost between the tests with and without fault tolerance is mainly due to the randomness of node deployment.

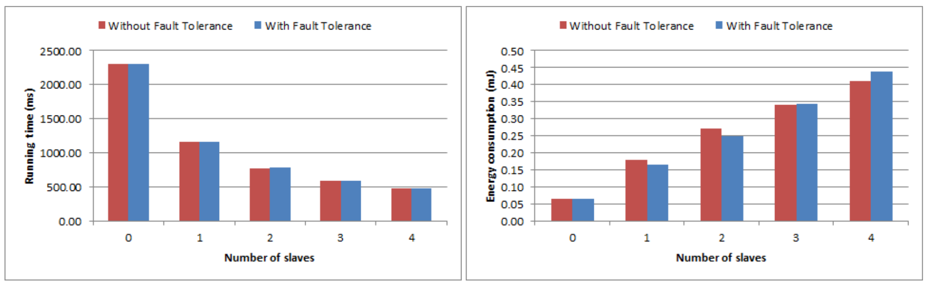

Fig. 5. Running time (ms) and energy consumption (mJ) ($n \leq 8$)

For the 3rd test, we suppose that we may use up to 4 slaves to perform parallel computing, and each slave has a backup node. Some slaves don't function properly during the computation, in this case, backup nodes are supposed to detect the fault and try to recover the computation.

When a backup node detects an erroneous result, it has to send a warning to the cluster head, and the latter will redo the faulty slave's task locally. Thus compared with results without faulty node, we can see in table 4 and figure 6 that the cluster needs more time and energy to finish the computation. The total energy cost increases slightly when more slaves become faulty, since more backup nodes need to warn the cluster head. However as the size of warning message is relatively smaller, the energy cost doesn't increase very fast, its value remains between $0.44mJ$ and $0.45mJ$.

A backup node is also supposed to send result back to cluster head when the slave that it's monitoring doesn't respond at all. However we can notice in table 5 that the running time doesn't increase when more slaves become faulty, since all backup nodes monitor the behavior of slaves and react simultaneously.

Nb of faulty slaves	Running time (ms)	Energy cost (mJ)
0	477.27	0.43921370
1	938.86	0.44161946
2	1400.44	0.44564062
3	1862.03	0.44965998
4	2323.61	0.45207022

Table 4. Performance of the fault tolerance method when erroneous result occurs

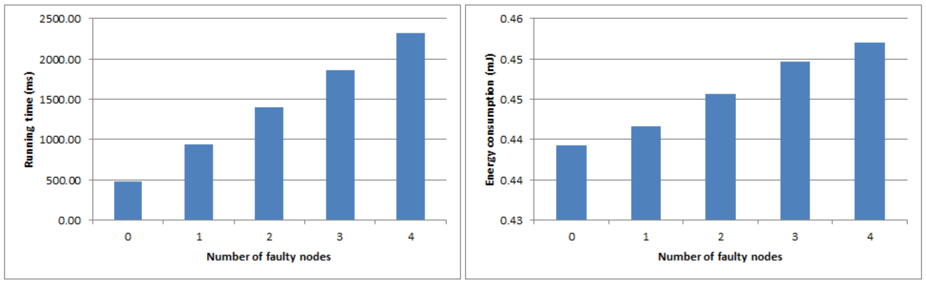

Fig. 6. Performance of the fault tolerance method when erroneous result occurs

Nb of faulty slaves	Running time (ms)	Energy cost (mJ)
0	477.27	0.43921370
1	517.27	0.43859930
2	517.27	0.47736890
3	517.27	0.49638458
4	517.27	0.47632058

Table 5. Performance of the fault tolerance method when missing result occurs

The irregularity of energy cost in figure 7 is due to the random deployment of sensor nodes. In our power model, the energy consumption is proportional to the distance of wireless communication.

8 Conclusion

In this paper we have proposed parallel computing scheme to accelerate the scalar multiplication on elliptic curves, we have also designed a fault tolerance mechanism which can significantly improve the reliability of the system. We have tested our method by using a simulator, and the results show a considerable acceleration of computation. Even in case of anomaly, such as sensor failure and incorrect result, the system is still able to detect the error and recover the system from errors. In addition, when no fault occurs, this fault tolerance mechanism doesn't have serious negative impact in term of running time and energy cost. The only drawback of the parallel computing scheme is the energy consumption since nodes have to communicate with each other for task distribution and result retrieval. Thus it shouldn't be used as the default computation scheme in wireless sensor networks. it can only be used in cases where running time is the most critical factor, like in disaster monitoring and military applications.

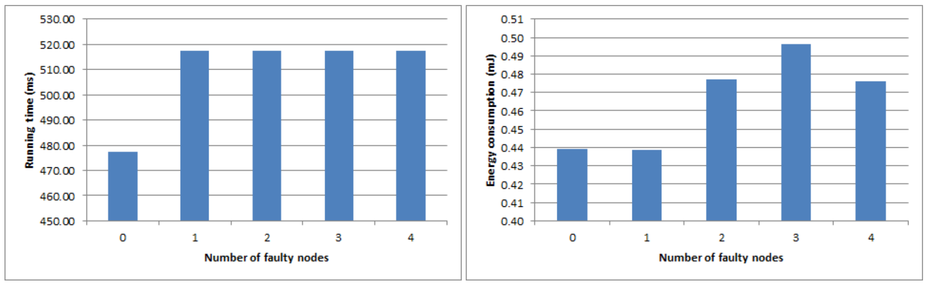

Fig. 7. Performance of the fault tolerance method when missing result occurs

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer networks **38**(4) (2002) 393–422
2. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Computer networks **52**(12) (2008) 2292–2330
3. Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., Welsh, M.: Deploying a wireless sensor network on an active volcano. Internet Computing, IEEE **10**(2) (2006) 18–25
4. Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., Turon, M.: Wireless sensor networks for structural health monitoring. In: Proceedings of the 4th international conference on Embedded networked sensor systems, ACM (2006) 427–428
5. Baker, C.R., Armijo, K., Belka, S., Benhabib, M., Bhargava, V., Burkhart, N., Der Minassians, A., Dervisoglu, G., Gutnik, L., Haick, M.B., et al.: Wireless sensor networks for home health care. In: Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on. Volume 2., IEEE (2007) 832–837
6. Malan, D., Fulford-Jones, T., Welsh, M., Moulton, S.: Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In: International workshop on wearable and implantable body sensor networks. Volume 5. (2004)
7. Gosnell, T., Hall, J., Jam, C., Knapp, D., Koenig, Z., Luke, S., Pohl, B., Schach von Wittenau, A., Wolford, J.: Gamma-ray identification of nuclear weapon materials. Technical report, Lawrence Livermore National Lab., Livermore, CA (US) (1997)
8. Walters, J.P., Liang, Z., Shi, W., Chaudhary, V.: Wireless sensor network security: A survey. Security in distributed, grid, mobile, and pervasive computing **1** (2007) 367
9. Zhou, Y., Fang, Y., Zhang, Y.: Securing wireless sensor networks: a survey. Communications Surveys & Tutorials, IEEE **10**(3) (2008) 6–28
10. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2) (1978) 120–126
11. Gordon, D.M.: A survey of fast exponentiation methods. Journal of algorithms **27**(1) (1998) 129–146
12. Shou, Y., Guyennet, H., Lehsaini, M.: Parallel scalar multiplication on elliptic curves in wireless sensor networks. In: Distributed Computing and Networking. Springer (2013) 300–314

13. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation **48**(177) (1987) 203–209
14. Miller, V.: Use of elliptic curves in cryptography. In: Advances in Cryptology - CRYPTO'85 Proceedings, Springer (1986) 417–426
15. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.: Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: Cryptographic hardware and embedded systems–CHES 2004: 6th international workshop, Cambridge, MA, USA, August 11-13, 2004: proceedings. Volume 6., Springer-Verlag New York Inc (2004) 119
16. Hankerson, D., Vanstone, S., Menezes, A.: Guide to elliptic curve cryptography. Springer-Verlag New York Inc (2004)
17. Diffie, W., Hellman, M.: New directions in cryptography. Information Theory, IEEE Transactions on **22**(6) (1976) 644–654
18. Lim, C., Lee, P.: More flexible exponentiation with precomputation. In: Advances in Cryptology - CRYPTO'94, Springer (1994) 95–107
19. Mishra, S., Jena, L., Pradhan, A.: Fault tolerance in wireless sensor networks. International Journal **2**(10) (2012) 146–153
20. Rajasegarar, S., Leckie, C., Palaniswami, M., Bezdek, J.C.: Distributed anomaly detection in wireless sensor networks. In: Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on, IEEE (2006) 1–5
21. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection. In: Applications of data mining in computer security. Springer (2002) 77–101
22. Chen, J., Kher, S., Somani, A.: Distributed fault detection of wireless sensor networks. In: Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks, ACM (2006) 65–72
23. Gupta, G., Younis, M.: Fault-tolerant clustering of wireless sensor networks. In: Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE. Volume 3., IEEE (2003) 1579–1584
24. Raj, R., Ramesh, M.V., Kumar, S.: Fault tolerant clustering approaches in wireless sensor network for landslide area monitoring. In: Proceedings of the 2008 International Conference on Wireless Networks (ICWN 08). Volume 1. (2008) 107–113
25. Koushanfar, F., Potkonjak, M., Sangiovanni-vincentelli, A.: Fault tolerance in wireless sensor networks. In: Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems. (2004)
26. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, IEEE (2000) 10–pp
27. Burchfield, T.R., Venkatesan, S., Weiner, D.: Maximizing throughput in zigbee wireless networks through analysis, simulations and implementations. In: Proceedings of the International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks Santa Fe, New Mexico, Citeseer (2007) 15–29
28. Heinzelman, W.B., Chandrakasan, A.P., Balakrishnan, H., et al.: An application-specific protocol architecture for wireless microsensor networks. IEEE Transactions on wireless communications **1**(4) (2002) 660–670