

# Key Establishment in Multi-Core Parallel Systems

Meikang Qiu

Department of Electrical Engineering  
University of New Orleans  
New Orleans, LA 70148, USA  
mqiu@uno.edu

Jing Deng

Department of Computer Science  
University of North Carolina at Greensboro  
Greensboro, NC 27412, USA  
jing.deng@uncg.edu

**Abstract**—The trend toward high processing power at a reasonable cost continues with the emergence of multi-core architectures with large number of cores. In such computing systems, a major technological challenge is to design the internal, on-chip communication network. This not only depends on high performance in latency, bandwidth, and fairness in contention under heavy loads, but also depends on an efficient authentication mechanism and protection of integrity and privacy of applications from hardware and software attacks. In this paper, we present a technique to establish secret keys for the cores employed by the same application to communicate securely. Our approach is based on key pre-distribution, in which each of the cores chooses a number of keys from a large key pool. When a group of cores are employed by an application, they discover their shared keys and establish a group key for secured communication. The key discovery and the process to connect partitioned groups use space-efficient Bloom filter to ensure the security and efficiency of the key establishment process. Our performance evaluation demonstrates the efficiency of the proposed framework.

## I. INTRODUCTIONS

The trend toward high processing power at a reasonable cost continues with the emergence of multi-core architectures with large number of cores. Building modular processors with *multiple cores* is far more cost-effective than building monolithic processors. Their use and scales are expected to increase dramatically in the coming years [1]. For example, 16-core processors will be common within the next four years [2]. Intel plans to deliver processors that have dozens or hundreds of cores during the next decade [3].

Cell Broadband Engine (Cell BE) processor is launched by IBM, Sony, and Toshiba in 2006 [4], [5]. The design goal is to improve performance an order of magnitude over that of desktop systems shipping in 2005. This includes gaining the most performance per area invested, reducing the area per core, and having more cores in a given chip area. In this way, the design would exploit application parallelism while supporting established application models and thereby ensure efficiency. It implements a single-chip multiprocessor with nine processors operating on a shared, coherent system memory (see Fig. 1 in Section II). There are two types of processor elements in this processor: the Power Processor Element (PPE) is optimized for control tasks; the Synergistic Processor Elements (SPEs) provides an execution environment optimized for parallel data processing.

In all multi-core processors, a major technological challenge is to design the internal, on-chip communication network. This

not only depends on high performance in latency, bandwidth, and fairness in contention under heavy loads, but also depends on an efficient authentication mechanism and protection of integrity and privacy of applications from physics attacks as well as software attacks. Usually, the cores are treated as trusted entities. On the other hand, all other components such as memory, cache, bus, and co-processors are untrusted [6]. The unique structure of multi-core processors exposes the data bus connecting the cores to outsiders. In addition, the communication between these cores and increasing-size cache may be under attack. The goal of this paper is to present a technique *to allow the cores to establish a group key for secured communication among themselves at run-time level*. To the best of our knowledge, this is the first paper to address this problem in technical literature.

Securing the multi-core communication system is more different than simply applying conventional security techniques such as public/private key schemes and the secret key schemes. The main reasons of the difficulty include small local store and heat/cooling problem of the large number of cores in one processor package. The potentially frequent switch of cores employed by different applications makes it impractical to use the costly public/private key techniques to secure the multi-core communication. Even the secret key scheme requires a careful design to make use of the local store.

In this paper, we present a technique to establish secret keys for the cores employed by the same application to communicate securely. We employ random key pre-distribution to establish secure communications among cores that are working in the same application. In our scheme, each of the cores chooses a number of keys from a large key pool. When a group of cores are used in an application, they discover their shared keys and establish a group key for secured communication. The key discovery and the process to connect partitioned groups use space-efficient Bloom filter to ensure the security and efficiency of the key establishment process.

Our paper is organized as follows: The architecture of multi-core system is presented in Section II. The details of our scheme are presented in Section III, with our performance evaluation shown in Section IV. We provide concluding remarks of our work in Section V.

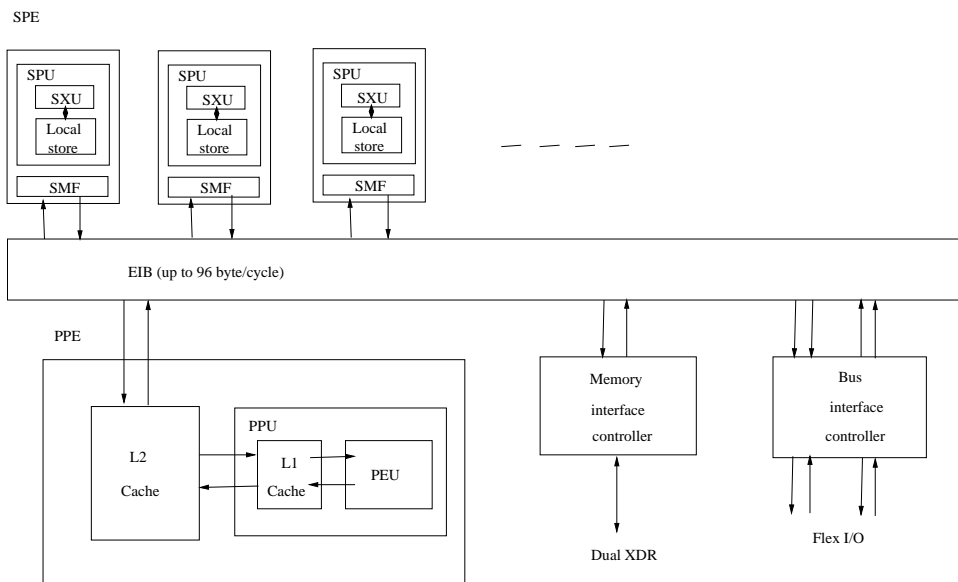


Fig. 1. Cell BE processor system architecture

## II. ARCHITECTURE

Our paper is based on the platform of the Cell BE processor system architecture, which is shown in Fig. 1. The processor is a heterogeneous, multi-core chip capable of massive floating-point processing optimized for computation-intensive workloads and rich broadband media applications [5]. It consists of one 64-bit *power processor element* (PPE), eight specialized co-processors called *synergistic processor elements* (SPEs), a high-speed memory controller, and a high-bandwidth bus interface, all integrated on-chip. The PPE and SPEs communicate through an internal high speed *element interconnect bus* (EIB).

The design of the SPE and its architectural specification lowered the required complexity and area. It also enabled high-frequency design with modest pipeline depths [7]. This is achieved without requiring the mechanisms that typically allow efficient instruction pipelining (register renaming, highly accurate branch predictors, and so on). This reduces architectural complexity where feasible, subject to latencies from basic resource decisions such as the large register file (2 Kbytes) and local store (256 Kbytes) [4].

The PPE is built on IBM's 64-bit Power Architecture with 128-bit vector media extensions and a two-level on-chip cache hierarchy. It is fully compliant with the 64-bit Power Architecture specification and can run 32-bit and 64-bit operating systems and applications. The SPEs are independent processors, each running an independent application thread. But one application may employ multiple SPEs. The SPE design is optimized for computation-intensive applications. Each SPE includes a private local store for efficient instruction and data access, but also has full access to the coherent shared memory, including the memory-mapped I/O space. Both types of processor cores share access to a common address space, which includes main memory, and address ranges corresponding to each SPE's local store, control registers, and I/O devices.

The PPE and SPEs are highly integrated. The PPE provides common control functions, runs the operating system, and provides application control, while the SPEs provide the bulk of the application performance. The PPE and SPEs share address translation and virtual memory architecture, and provide support for virtualization and dynamic system partitioning. They also share system page tables and system functions such as interrupt presentation. Finally, they share data type formats and operation semantics to allow efficient data sharing among them.

Each SPE consists of the SPU (*synergistic processor unit*) and the *synergistic memory flow* (SMF) controller. Each SPU consists of an SXU (*synergistic execution unit*) and a local store (256 Kbytes). The SMF controller moves data and performs synchronization in parallel to SPU processing and implements the interface to the element interconnect bus, which provides the Cell BE with a modular, scalable integration point.

## III. KEY ESTABLISHMENT IN MULTI-CORE PROCESSOR SYSTEMS

### A. Background

Secure communication between different hardware components is needed to ensure that the exchanged information remains confidential and that only authorized persons can access the devices [8]. The public-key cryptosystem can provide such security protections, but with some caveats. Although RSA has been the most widely used public-key cryptosystem for 20 years, its computational demands are prohibitive for mobile and lightweight devices. Doubling the size of the RSA key leads to an approximate eightfold increase in computation time as the computational effort grows proportionally to the cube of the key size. However, an increase in RSA key size is becoming necessary because the NIST and RSA Laboratories

recommend using the current 1,024-bit key size to protect data only until the year 2010 [8]. In securing multi-core processors, public-key technique becomes too expensive especially when there are frequent data exchanges among the cores.

The communication among different cores employed by the same application can be protected by symmetric key techniques [9]–[13]. A shared secret key among the cores will be needed. Establishing such a secret key can obviously be achieved with public-key techniques. However, we argue that public-key techniques are too expensive because of the overall number of public keys that each core needs to carry and the overall computation intensive encryption/decryption operations need to be performed in each key establishment process. Such a key establishment process will be needed frequently when different applications use clusters of the cores with frequent context-switch.

We term the secret key shared by all the cores working on the same application “group keys”. We propose a comprehensive technique to achieve such efficient group key establishment and use the group key to secure the communication in multi-core processors. Our scheme is based on key pre-distribution that has been developed for wireless sensor networks [14]–[20].

There are intrinsic relations and similarities of the key establishment in multi-core processor and that in wireless sensor networks: The multiple cores can be treated as sensor nodes. Both of them require low power and have limited on-board memory. Packaging and thermal cooling costs are the biggest drivers for reducing power in such chips, especially chips manufactured in large quantities for price-sensitive products [21]; The secure communication problem is similar; The sharing of the communication medium is similar as well [1].

### B. Key Pre-Distribution

In key pre-distribution, we need to address the question of how the cores should select and store some keys before the processor is used by a number of applications and security protection is needed for these applications [14], [16], [17], [22]–[26].

*Problem 1: (Key Pre-Distribution) How can we pre-distribute the key information in the cores such that some clusters of cores can derive secret group keys efficiently later on to secure their inter-core communication?*

Due to the dynamic nature of the core assignment process and different application’s computing requirements, different clusters of cores may be used at a certain time. Therefore, it is impossible to decide the group keys prior to the formation of clusters. There are a number of ways to solve the group key agreement problem in the absence of prior knowledge of groups. Blundo et al. proposed a method to allow each member of any group of users of a given size to compute a common secure group key [27]. However, the method is unsuitable for multi-core security provision because of large amount of memory needed.

A naive solution to the group key agreement problem is to let all cores carry a *master* secret key with themselves. Once

groups are formed, cores in a group can use this global master secret key to conduct key agreement and obtain a new group key. There is a great danger with this approach: if one core is compromised by attackers through side-channel eavesdropping or any other types of attacks, the security of the entire multi-core processor will be compromised. Some existing studies suggest storing the master key in tamper-resistant hardware to reduce the danger, but this increases the cost and energy consumption for each core. Furthermore, tamper-resistance hardware might not always be safe [28].

Another solution to the group key agreement problem is to let each core carry  $N - 1$  secret keys, each of which is known to this core and one of the other  $N - 1$  cores (assuming  $N$  is the total number of cores, e.g.,  $N = 1024$  in kilocore [29]). The main problem with this solution is memory usage: if the key size is 1,024 bits [8] and the number of cores is 1,024 [29], each core needs to use  $1M$  bits of memory just to store the keys. This is impractical for a core that has a very limited amount of local store (current size of about 256 KBytes).

Eschenauer and Gligor proposed a novel key distribution method for wireless sensor networks in [14]. In this method, instead of carrying all  $N - 1$  keys, each sensor randomly selects some keys from a pre-defined key pool. It has been shown that, with high probability, sensors can find some neighbors sharing at least one key with themselves. One node may share different keys with different neighbors. Since some neighboring nodes may not share any keys [14], [30], [31], multi-hop secure paths can be identified to deliver secret toward the neighbors in order to establish secure communication. Similar to any key pre-distribution schemes, if any sensor is compromised, then the keys it carries are all disclosed.

This scheme was later extended by Chan et al. to require sensor nodes to share at least  $q$  keys in order to establish secure communication [17]. Such a unique requirement improves the security performance of the scheme against compromise. It is shown that adversary needs to compromise much more nodes in order to gain any meaningful payoff.

In this paper, we will employ a scheme that is similar to the one proposed in [17]. Therefore, each core will store  $m$  keys from a large key pool. When two cores need to communicate securely, they need to find at least  $q$  common keys. If enough common keys cannot be found, other techniques should be performed to allow the cores to establish the secret between them [17], [30], [31].

### C. Common Key Discovery

After key pre-distribution, cores working in the same application need to discover the common keys among themselves so that a group key can be established by the group head and be delivered to them. Group head can be the largest of the core IDs in the group.

*Problem 2: (Common Key Discovery) Once a group of cores are recruited by one application, the cores need to find out the common keys among them. What techniques should be used to identify such common keys among these nodes efficiently and securely?*

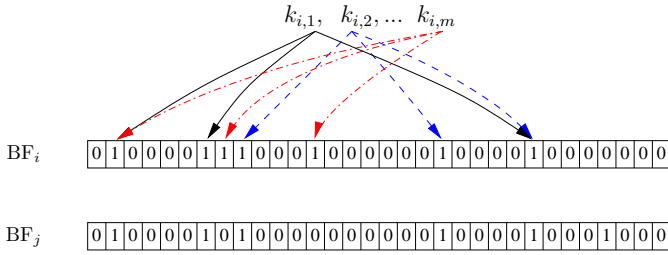


Fig. 2. Bloom filter illustration. On core  $i$ , each key is mapped through Bloom filter onto  $BF_i$ .  $BF_j$  for core  $j$  is also shown in the figure. By calculating the mapping bits of all the carried keys and comparing them with  $BF_j$ , core  $i$  can quickly decide that core  $j$  also carries key  $k_{i,1}$  and  $k_{i,2}$ , but not  $k_{i,m}$ . This is because the corresponding bits mapped from key  $k_{i,m}$  include “0” in  $BF_j$ . Since  $BF_i$  is much shorter than the shared keys, all other cores can carry  $BF_i$  so that they can identify common keys with core  $i$  quickly.

A straightforward way for key discovery is to ask cores to broadcast their key identification numbers on the data bus. Then these cores will have a full view of the security connectivity graph and the group head can find the best way to deliver its group key to all cores in its group. But this is insecure as the key identification numbers are disclosed to the data bus and inefficient especially with multi-core processors working on different applications and having to switch between applications frequently. We propose to use Bloom filter to perform efficient and secure common keys discovery among the cores in the same group in this paper.

Bloom filter is a space-efficient probabilistic data structure that can be used to test whether an element is a member of a set [32]–[35]. False positives are possible, but false negatives are not. Elements can be added to the set, but cannot be removed. As more and more elements are added to the set, the probability of false positives increases [35].

Bloom filter is basically a one-way hashing function that maps a set of items to a much shorter array (see Fig. 2). Initially the entire array contains all 0’s. Then each item (in our case, the key identification numbers of each core) will be mapped to a few bits in the array with 1. After all items are mapped to the array, some bits in the array are 1’s while others are 0’s. This array can be called Bloom filter results,  $BF_i$ , of the key carried by core  $i$ . In our scheme, each core carries the Bloom filter results of all cores for later usage. Such a calculation and data storage can be performed in the chip manufacturing process or in the power-up phase of the multi-core processor [6].

Now suppose two cores,  $i$  and  $j$ , need to find out their common keys. Core  $i$  performs the bloom filter function for each of the keys that it carries and compares the result pattern with  $BF_j$  which records the Bloom filter results of core  $j$ . If all the corresponding bits are one, then it is expected that core  $j$  also carries this key (with a certain possibility of false positive). If there are some corresponding bits in  $BF_j$  that are 0, core  $j$  does not carry this key with certainty. In this way, core  $i$  can find out the list of shared keys with core  $j$  quickly.

Since false negative is impossible but false positive is in Bloom filter, care must be taken to filter the keys that are not

carried by core  $j$  as indicated by Bloom filter. Note that such false positives are controllable with the mapping function and the size of the array.

#### D. Group Key Delivery

With the use of the random key pre-distribution, it is still possible to have partitioned groups, defined as those groups that the group head has no way to deliver the generated group key to all group members. We introduce *Multi-Core Graph* to discuss partitioned groups.

A Multi-Core Graph  $G_{mc}(V, E)$  is constructed in the following way. Let  $V$  consist of all the cores in the group. For any two cores  $v_i$  and  $v_j$  in  $V$ , draw an edge between them if and only if  $v_i$  and  $v_j$  have at least  $q$  common keys, i.e., two cores are connected by an edge if and only if they can establish a secure point-to-point channel between themselves.

If  $G_{mc}(V, E)$  is already connected, then key agreement can be fairly easy: the group head generates a secret group key  $K$  and uses the connection (i.e., the secure channel) in  $G_{mc}$  to deliver the key to each of the group members. Some cores can get  $K$  directly from the group head, while some cores get the key from other cores in the group. As long as the graph is connected, every core within the group can get the key, and because the key is distributed via a secure channel, the key is only disclosed to these cores.

As we will see in Fig. 3 of Section IV, the chance of connected graph is low when there are only a few cores in a group or the key space is large. Therefore, expecting a connected  $G_{mc}$  is unrealistic; the challenge is how to conduct key agreement when  $G_{mc}$  is partitioned. This leads to the following problem:

**Problem 3: (Connecting Partitioned Groups)** *The cluster of cores in a group may not be connected as the corresponding multi-core graph is partitioned. How should these cores be connected securely and efficiently?*

We present two schemes that can help to establish connected  $G_{mc}$ . In the first scheme, the PPE is recruited to connect the partitioned graph and it is assumed that the PPE shares a secret key with each of the cores in a multi-core processor. In the second scheme, the assumption of PPE’s sharing keys with all cores connectivity is relaxed. Instead, a two-phase key agreement scheme is proposed to recruit additional cores into the group to perform key establishment.

We assume that  $G_{c,0}, G_{c,1}, \dots, G_{c,\tau}$  are  $\tau + 1$  components within  $G_{mc}$ . Without loss of generality, we assume that the group head  $H_{mc}$  is in component  $G_{c,0}$ . Our goal is to establish connections between  $G_{c,0}$  and the other  $\tau$  components.

#### PPE-Assisted Key Agreement Scheme

If PPE can participate in the key agreement process, we can exploit it to add edges to  $G_{mc}$  to make the graph connected. Our goal is to establish connections between  $G_{c,0}$  and the other  $\tau$  components with the help from PPE. We also assume that PPE has a record of the keys that are carried by each of the cores. We call  $K_0$  the key shared by PPE and the group

head. Our algorithm is described in the following.<sup>1</sup>

- 1)  $H_{mc}$  randomly picks one core from each component. Let  $v_1, \dots, v_\tau$  represent these cores.  $H_{mc}$  then sends  $(v_1, \dots, v_\tau)$  to PPE. Such communication should be encrypted using the key  $K_0$  shared between PPE and  $H_{mc}$ .
- 2) PPE randomly generates a secret key  $K_b$  (the edge key), and encrypts it using  $K_1, \dots, K_\tau$ , respectively, where  $K_i$  is the key shared between  $v_i$  and PPE. We use  $K_i(K_b)$  to represent the encrypted key. PPE then sends  $(K_0(K_b), K_1(K_b), \dots, K_\tau(K_b))$  to  $H_{mc}$ .
- 3)  $H_{mc}$  delivers  $K_i(K_b)$  to  $v_i$ , for  $i = 1, \dots, m$ .
- 4)  $H_{mc}, v_1, \dots, v_\tau$  decrypt the messages and get  $K_b$ . With the edge key  $K_b$  shared between  $H_{mc}$  and each of the components, the Multi-Core Graph  $G_{mc}$  is now connected.

### Two-Phase Key Agreement Scheme

If getting help from PPE during the key agreement phase is difficult or considered insecure, a localized solution should be developed. As we will see from Fig. 3 in Section IV, in order to improve connectivity, we need to increase the number of cores in a group or lower the size of the key space pool. However, the number of key pool size affects the resilience of a key establishment scheme against compromise [14]. In addition, the number of cores is usually fixed by application or PPE. The only way to increase the number of cores in a group is to recruit additional cores temporarily into the group. We propose the following two-phase heuristic key agreement scheme.

- 1)  $H_{mc}$  randomly picks one core from each component. Let  $v_1, \dots, v_\tau$  represent these cores.  $H_{mc}$  then computes the bloom filter similarity scores of each of the  $v_1, \dots, v_\tau$  cores with each of all other cores in the processor. The bloom filter similarity score of two cores  $i$  and  $j$  is computed as

$$SS(i, j) = \mathbf{BF}_i \cdot (\mathbf{BF}_j)^T, \quad (1)$$

where  $i \in \{v_1, \dots, v_\tau\}$ ,  $j \in \mathcal{C} \setminus \{v_1, \dots, v_\tau\}$ ,  $\cdot$  represents matrix product, and  $T$  represents transpose of a matrix.

- 2)  $H_{mc}$  chooses the core with the highest bloom filter similarity score between any pair of cores, termed  $u_1$ . Remove the row and column of all bloom filter similarity scores and repeat the same process for all other rows/columns.
- 3) Then  $H_{mc}$  uses these cores to connect the cores employed by the application.

Note that the additional cores get to know the secret key. A challenge is how to reduce the number of additional cores to the group before making the new graph connected. This will limit the exposure of the group key generated by the group head.

<sup>1</sup>We use a single shared key, such as  $K_0$  shared between PPE and the group head, to illustrate our PPE-Assisted Key Agreement Scheme. In order to make sure that this process is as secure as those between regular cores, they need to share  $q$  keys.

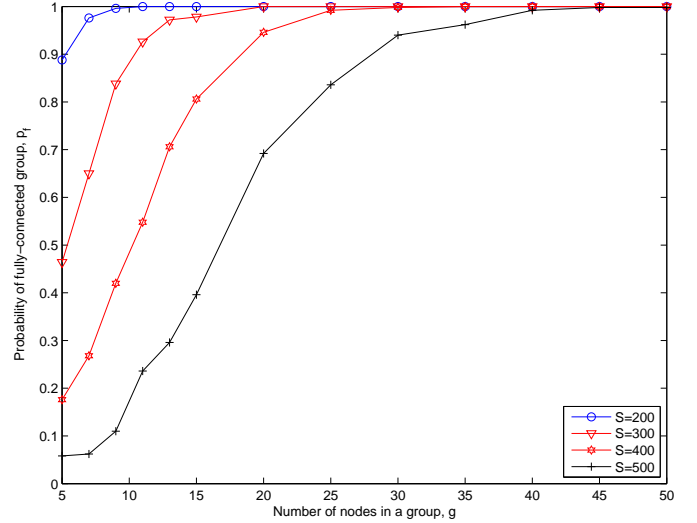


Fig. 3. Connectivity probability. We simulated and computed the probability of the cores in a group to be connected through secure connections, i.e., cores sharing more than  $q = 2$  keys. Each core randomly selects  $m = 20$  keys from a large key pool of size  $S$ .

## IV. PERFORMANCE EVALUATION

We evaluated the efficiency of our proposed scheme for multi-core processors through high-level simulations built on Matlab. In our simulations, we investigated the chance of groups to be fully-connected, i.e., all cores in the same group are securely connected, and the number of partitions in such groups if they are not fully-connected.

Figure 3 shows our simulation results of the probability that the cores in a group form a connected graph  $G_{mc}$ . The connectivity probability is measured as the chance a group of  $g$  cores forms a non-partition graph. Each of the core is assumed to randomly choose  $m = 20$  keys from a key pool of size  $S$ . At least  $q = 2$  keys need to be shared before two cores are declared connected. It can be observed from Fig. 3 that as  $g$  increases, connectivity improves. This is because more cores increase the chance of eventually finding common keys. Furthermore, as  $S$  increases, connectivity lowers because of the larger space to choose keys from.

We investigated the number of components (partitions) in typical cases and the results are shown in Figure 4. For example, when  $S = 300$ , and when there are 15 cores in a group, the average number of components is 2, which means that PPE needs to connect, on an average, 2 components in a group to form a connected graph in  $G_{mc}$ .

In Fig. 5, we present the comparison of number of partitions as a function of  $m$  with  $S = 300$  and  $g = 8$ . The number of partitions lowers as  $m$  increases, because of better chance of sharing keys. When  $q$  increases, the number of partitions is larger.

We compare the number of partitions as a function of  $m$  with  $S = 300$  and  $q = 2$  in Fig. 6. In this figure, cores need to share at least  $q = 2$  keys in order to be considered securely connected. While all curves drop as  $m$  increases, the rates of

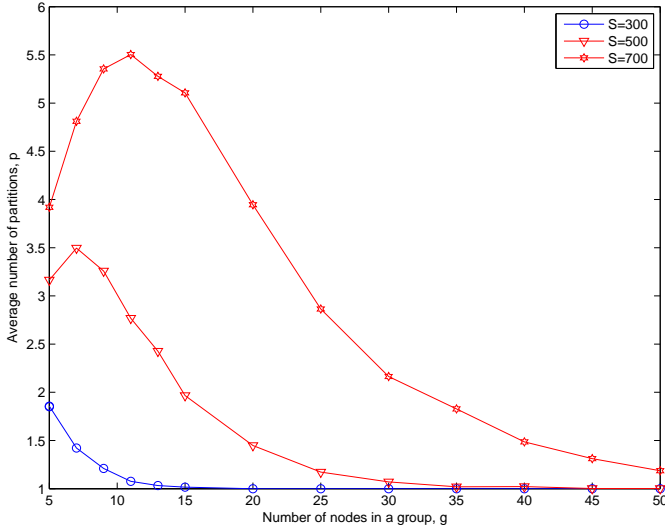


Fig. 4. Average number of partitions in  $G_{mc}$ . We simulated the average number of partitions in a group of  $g$  cores, each of which randomly chooses  $m = 20$  keys from a large key pool (size of  $S$ ). Cores need to share at least  $q = 2$  keys in order to be considered securely connected.

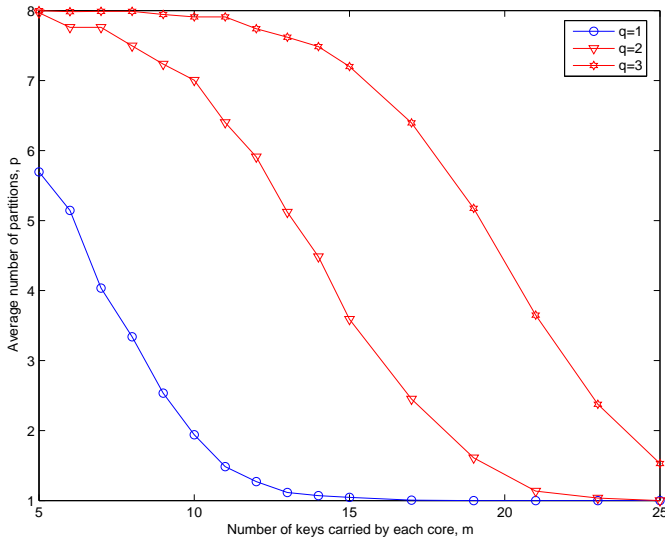


Fig. 5. Average number of partitions in  $G_{mc}$ . We simulated the average number of partitions in a group of  $g = 8$  cores, each of which randomly chooses  $m$  keys from a large key pool (size of  $S = 300$ ). Cores need to share at least  $q$  keys in order to be considered securely connected.

such drops are higher when  $g$  is larger. This means that the benefit of carrying more keys in the cores is higher when the number of cores in a group is larger.

We compare the number of partitions as a function of  $m$  with  $S = 300$  and  $q = 1$  in Fig. 7. In this figure, cores need to share just one key ( $m = 1$ ) in order to be considered securely connected. A similar trend and comparison can be seen, but the curves cross each other at different values of  $m$ . This is because of the different required  $q$ . For example, the  $g = 16$  curve drops much faster than other curves, before it approaches the value of 1 partition (fully-connected group).

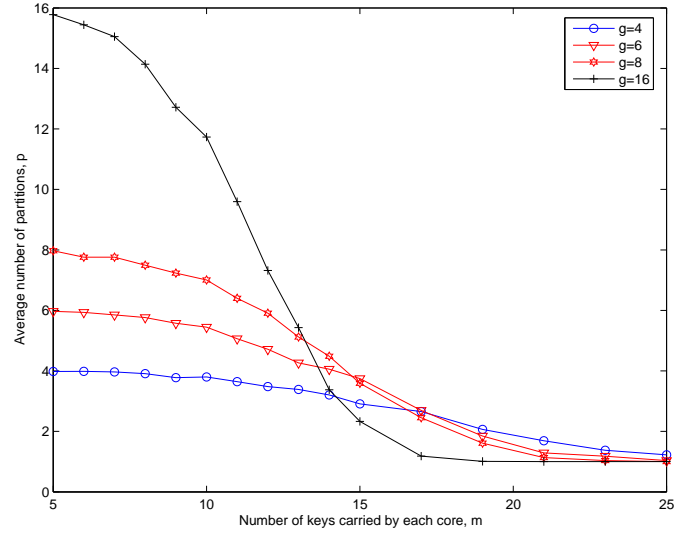


Fig. 6. Average number of partitions in  $G_{mc}$ . We simulated the average number of partitions in a group of  $g$  cores, each of which randomly chooses  $m$  keys from a large key pool (size of  $S = 300$ ). Cores need to share at least  $q = 2$  keys in order to be considered securely connected.

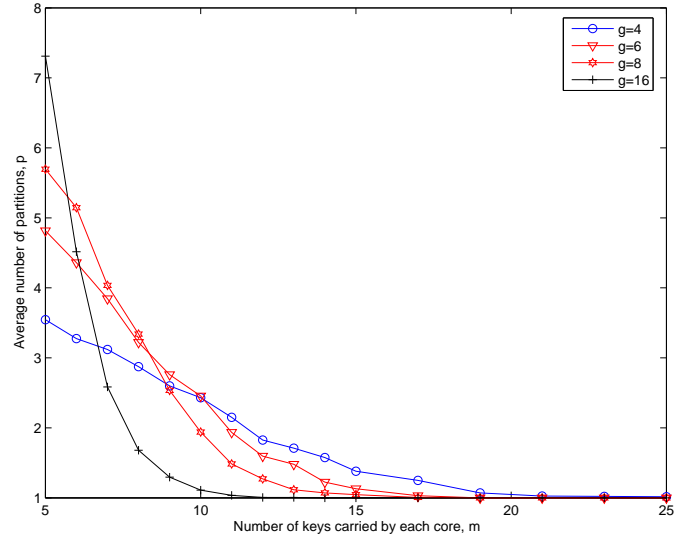


Fig. 7. Average number of partitions in  $G_{mc}$ . We simulated the average number of partitions in a group of  $g$  cores, each of which randomly chooses  $m$  keys from a large key pool (size of  $S = 300$ ). Cores need to share at least  $q = 1$  key in order to be considered securely connected.

## V. CONCLUSIONS

Scientists and engineers are trying to design processors to satisfy the ever-increasing demand for higher computation power and lower cost. The new trend is the design and application of multi-core architectures with large number of cores. Building modular processors with *multiple cores* is far more cost-effective than building monolithic processors.

In all multi-core processors, a major technological challenge is designing the internal, on-chip communication network. We focus on the problem of securing the multi-core communication system in this paper. We have employed random key pre-

distribution to establish secure communications among cores that are working in the same application. In our scheme, each of the cores chooses a number of keys from a large key pool. When a group of cores are used in an application, they discover their shared keys and establish a group key for secured communication. The key discovery and the process to connect partitioned groups use space-efficient Bloom filter to ensure the security and efficiency of the key establishment process.

Performance evaluations through simulations have been shown to prove that, with parameters carefully designed and chosen, our scheme can achieve efficient and successful secure connectivity for the group of cores on multi-core processors. Such parameters include the number of keys that each core carries, the number of keys that two cores must share before considered connected, and others. In our future work, we will implement our scheme on multi-core hardware and evaluate its performance under more realistic settings.

## REFERENCES

- [1] S. Borkar, "Thousand core chips: a technology perspective," in *ACM DAC'07*, Jun. 2007, pp. 746–749.
- [2] C. Sun, L. Shang, and R. P. Dick, "Three-dimensional multiprocessor system-on-chip thermal optimization," in *ACM CODES+ISSS'07*, Sep. 2007, pp. 117–122.
- [3] S. Borkar et al., "Platform 2015: Intel processor and platform evolution for the next decade," in *Intel Corporation, Tech. Rep.*, Mar. 2005.
- [4] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture," *IEEE Micro*, pp. 10–24, Mar.-Apr. 2006.
- [5] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessors communication network: Built for speed," *IEEE Micro*, pp. 10–23, May-Jun. 2006.
- [6] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *IEEE Design and Test of Computers*, pp. 570–580, Nov.-Dec. 2007.
- [7] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, and C. Johns et al., "The design and implementation of a first generation Cell processor," in *Proc. Intl Solid-State Circuits Conf. Tech. Digest*, 2005, pp. 184–185.
- [8] H. Eberle, S. Shantz, V. Gupta, N. Gura, L. Rarick, and L. Spracklen, "Accelerating next-generation public-key cryptosystems on general-purpose CPUs," *IEEE Micro*, pp. 52–59, Mar.-Apr. 2005.
- [9] P. Ning, An Liu, and W. Du, "Mitigating dos attacks against broadcast authentication in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 1, pp. 1:1–35, 2008.
- [10] P. Ning D. Liu and R. Li, "Establishing pairwise keys in distributed sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 1, pp. 41–77, 2005.
- [11] W. Yu, Y. Sun, and K. J. Liu, "Optimizing the rekeying cost for contributory group key agreement schemes," *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 3, pp. 228–242, 2007.
- [12] W. Gu, X. Bai, S. Chellappan, D. Xuan, and W. Jia, "Network decoupling: A methodology for secure communications in wireless sensor networks," *IEEE Trans. on Dependable and Secure Computing*, vol. 18, no. 12, pp. 1784–1796, 2007.
- [13] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili, "A pairwise key predistribution scheme for wireless sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 228–258, 2005.
- [14] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proc. of the 9th ACM conference on Computer and communications security*, Washington, DC, USA, November 18-22 2002, pp. 41–47.
- [15] W. Du, J. Deng, Y. S. Han, and P. Varshney, "A witness-based approach for data fusion assurance in wireless sensor networks," in *IEEE 2003 Global Communications Conference (GLOBECOM)*, San Francisco, CA, USA, December 1–5 2003.
- [16] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, Washington, DC, USA, October 27-31 2003, pp. 52–61.
- [17] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proc. of IEEE Symposium on Security and Privacy*, Berkeley, California, May 11-14 2003, pp. 197–213.
- [18] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A key predistribution scheme for sensor networks using deployment knowledge," *IEEE Trans. on Dependable and Secure Computing*, vol. 3, no. 1, pp. 62–77, 2008.
- [19] D. Liu and P. Ning, "Improving key predistribution with deployment knowledge in static sensor networks," *ACM Transactions on Sensor Networks*, vol. 1, no. 2, pp. 204–239, 2005.
- [20] D. Huang and D. Medhi, "Secure pairwise key establishment in secure pairwise key establishment in large-scale sensor networks: An area partitioning and multigroup key predistribution approach," *ACM Transactions on Sensor Networks*, vol. 3, no. 3, pp. 16:1–34, 2007.
- [21] D. Stasiak, R. Chaudhry, D. Cox, S. Posluszny, J. Warnock, S. Weitzel, D. Wendel, and M. Wang, "Cell processor low-power design methodology," *IEEE Micro*, vol. 25, no. 6, pp. 71–78, 2005.
- [22] J. Deng and Y. S. Han, "Using mds codes for the key establishment of wireless sensor networks," in *Proc. of the International Conference on Mobile Ad-hoc and Sensor Networks (MSN '05)*, X. Jia, J. Wu, and Y. He, Eds., Wuhan, P. R. China, December 13-15 2005, vol. 3784 of *Lecture Notes in Computer Science (LNCS)*, pp. 732–744, Springer-Verlag.
- [23] D. Huang and D. Medhi, "A byzantine resilient multi-path key establishment scheme and its robustness analysis for sensor networks," in *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium*, Colorado, USA, April 4-8 2005, pp. 240b–240b.
- [24] R. Blom, "An optimal class of symmetric key generation systems," *Advances in Cryptology: Proceedings of EUROCRYPT 84 (Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, eds.)*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 209, pp. 335–338, 1985.
- [25] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proc. of ACM Conference on Computer and Communications Security (CCS '03)*, Washington, DC, USA, October 27-31 2003, pp. 42–51.
- [26] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *Proc. of the 23rd Conference of the IEEE Communications Society (Infocom '04)*, Hong Kong, China, March 7-11 2004, pp. 586–597.
- [27] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," *Lecture Notes in Computer Science*, vol. 740, pp. 471–486, 1993.
- [28] R. Anderson and M. Kuhn, "Tamper resistance — a cautionary note," in *Proceedings of the Second Usenix Workshop on Electronic Commerce*, November 1996, pp. 1–11.
- [29] "A thousand processors on one chip," <http://blogs.zdnet.com/emergingtech/index.php?p=207>, Apr. 4th, 2006.
- [30] J. Deng and Y. S. Han, "Multi-path key establishment for wireless sensor networks using just enough redundancy transmission," *IEEE Transactions on Dependable and Secure Computing*, in press.
- [31] J. Deng and Y. S. Han, "Babel: Using a common bridge node to deliver multiple keys in wireless sensor networks," in *Proc. of IEEE Global Telecommunications Conference - General Symposium (GLOBECOM '07)*, Washington, DC, USA, November 26-30 2007, pp. 161–165.
- [32] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.
- [33] R. Karp and M. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research Development*, vol. 31, no. 2, pp. 249–260, 1987.
- [34] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 613–620, October 2002.
- [35] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: a survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2005.