Systematic Security and Timeliness Tradeoffs in Real-Time Embedded Systems*

Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
kang@cs.binghamton.edu

Sang H. Son
Department of Computer Science
University of Virginia
son@cs.virginia.edu

Abstract

Real-time embedded systems are increasingly being networked. In distributed real-time embedded applications, e.g., electric grid management and command and control applications, it is required to not only meet real-time constraints but also support the data confidentiality,integrity, and authenticity. Unfortunately, in general, cryptographic functions are computationally expensive, possibly causing deadline misses in real-time embedded systems with limited resources. As a basis for cost-effective security support in real-time embedded systems, we define a quantitative notion of Strength of Defense (SoD). Based on the SoD concept, we propose a novel adaptive security policy in which the SoD can be degraded by decreasing the cryptographic key length for certain tasks, if necessary, to improve the success ratio under overload conditions. Our approach is lightweight. The time complexity of our approach is linear and its amortized version has the constant overhead per SoD adaptation period. Moreover, our approach supports desirable security features requiring an attacker to do extra work to find the cryptographic key. In the performance evaluation, we show that our approach can considerably improve the success ratio due to controlled SoD degradation under overload.

1 Introduction

Real-time embedded systems, which used to be isolated, are increasingly being networked due to distributed real-time embedded (DRE) applications including, e.g., electric grid management, agile manufacturing, and defense applications. In these applications, DRE systems need to report the real world status, e.g., the battlefield or electric grid status, to the control center (CC) that prepares overall battle tactics or energy supply plans. It is important for DRE systems to report the real world status in a timely manner. At the same time, DRE systems need to ensure that an adversary cannot read the data, modify it, or claim a false identify by supporting the confidentiality, integrity, and authenticity requirements via cryptographic means.

Unfortunately, most cryptographic algorithms are computationally expensive, possibly causing many deadline misses in real-time embedded systems (RTESs) with limited resources. Note that resource over-provisioning may not be a viable solution due to the stringent cost, size, weight, and power constraints prevalent in these systems. On the other hand, it is not desirable to ignore security requirements or simply use a weak security scheme all the time. Thus, it is necessary to balance timing and security requirements. Despite the importance of the problem, relatively little work has been done.

To shed light on the problem, we present a novel approach called SSTT (Systematic Security and Timeliness Tradeoffs) in real-time embedded systems. More specifically, we aim to maximize the success ratio, while meeting the cryptographic security requirements in *soft real-time* applications such as battlefield monitoring and target tracking. To this end, we define a *quantitative metric* to measure the Strength of Defense (SoD) based on the cryptographic key length. Based on the SoD concept, we present a new *adaptive security policy* in which the SoD is degraded by decreasing the cryptographic key length for certain tasks, if necessary, to improve the success ratio under overload conditions. Although adaptive security support in real-time systems has previously been studied [1, 18, 5], very little work has been done to provide a quantitative SoD metric and adapt the cryptographic key length to improve the success ratio under overload. Moreover, SSTT incurs very little overhead, while providing desirable security and system features. (More details are given in Sections 2 and 3.)

To evaluate the performance, we compare our approach, via a simulation study, to a baseline approach that applies the EDF (Earliest Deadline First) scheduling algorithm [10], while always using the longest key for cryptographic security regardless of the current system status. SSTT always supports at least the minimum required SoD

*This work was supported, in part, by NSF grants IIS-0208758 and CCR-0329609.

and considerably improves the success ratio by systematically adapting the SoD, if necessary, to improve the success ratio when overloaded.

The remainder of this paper is organized as follows. In Section 2, an application scenario is discussed to motivate our work. Further, the scope of the work is described by discussing our security model. Our approach for systematic security and timeliness tradeoffs is discussed in Section 3. In Section 4, the performance of SSTT is evaluated via simulation. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and discusses future work.

2 Scope of the Work

In this section, a (simplified) application scenario and the security model are discussed to specify the scope of the work.

2.1 Application Scenario

In time-critical target tracking [11], for example, UAVs (Unmanned Aerial Vehicles) are required to perform soft real-time reconnaissance tasks for monitoring and transmitting the battle field status to the command and control center (CC). Similarly, DRE systems embedded to electric grids are required to report the local grid status to the CC across the network. In these applications, it is important for a RTES to support the security constraints, while meeting as many deadlines as possible. RTESs in these applications can be overloaded due to dynamic workloads. For example, when a UAV enters the current area of interest (AOI), it may be required to increase the frequency of the surveillance data transmission. In addition, a new target may enter the AOI, possibly overloading a RTES in a UAV. As another example, the CC of an electric grid may request the RTESs in the AOI showing abnormal electricity supply patterns report the status more frequently. Since real-time monitoring and cryptographic computations may compete for computational resources, it is important to effectively balance between the two conflicting requirements. When an individual RTES is under transient overload, our approach aims to improve the success ratio in a RTES by systematically adapting the key length. Although we expect our adaptive security policy can also improve the end to end delay, network QoS management is beyond the scope of this paper. A thorough investigation is reserved for future work.

2.2 Security Model

The security model considered in this paper is described as follows.

• We assume that isolated RTESs, e.g., RTESs in a UAV, are trusted and tamper-proof.

• We focus on cryptographic security issues in this paper. Other security issues such as denial of service attacks are reserved for future work.

• We consider a symmetric key system in which a CC and an individual RTES share a unique secret key, since the encryption/decryption in a public key system takes several orders of magnitude longer than that in a symmetric key system [17].

• It is assumed that a RTES uses a trusted cryptosystem, e.g., AES (Advanced Encryption Standard) [15], which does not have any known vulnerability. Thus, an adversary has to attack the cryptosystem in a *brute-force* manner trying to find the secret key via an exhaustive search in the key space as common in trusted cryptosystems [2, 17, 21].

• We assume that an encryption algorithm, e.g., AES, used in a RTES can support different key lengths.

• To minimize the overhead for key selection, we assume that the unique keys shared between the CC and a RTES are created and safely distributed to RTESs *a priori*. For example, keys of different lengths can be created *offline* and *predistributed* to RTESs in UAVs during the regular maintenance or before beginning a mission. When the RTES switches to a short key shared with the CC under overload, neither the CC nor RTES needs to create a new key. In this way, unnecessary deadline misses due to online key selection and exchange can be avoided. Dynamic key generation and distribution are reserved for future work.

3 Systematic Security and Timeliness Tradeoffs

In this section, the cryptographic security supported by our approach is described. The notion of the strength of defense is defined. Also, our approach for systematic security and timeliness tradeoffs is discussed.

3.1 Cryptographic Security Support

In our approach, we mainly consider three security goals: confidentiality, integrity, and authenticity as discussed before. To support the *confidentiality* requirement, when a RTES wants to send a message P to the CC, it first encrypts the plaintext P. Formally, the encrypted ciphertext message is:

$$C = E(P)_{\{K_e,\, Cnt\}} \tag{1}$$

where E is the encryption function such as AES [15], K_e is the unique encryption key shared between the RTES and CC, and Cnt is the current counter value that is incremented after each message encryption and transmission. By including the counter value, one can prevent a replay attack in which an attacker resends old messages [17]. It is important for a RTES to prevent replay attacks. Otherwise, an

adversary can deceive the CC by retransmitting old sensor data representing, e.g., the old battle field status.

To support the *integrity* and *authenticity* of the message, a secure message authentication code (MAC) M is computed over the message including C:

$$M = H(S|D|C|Cnt)_{K_m} \quad (2)$$

where H is a key-based, secure one way hash function supporting minimal collisions, $S|D|C|Cnt$ is the concatenation of the source S and destination D address, the ciphertext C (Eq 1), and the counter value. K_m is the unique message authentication key shared between the RTES and CC. Note that $K_m \neq K_e$ in our approach, since it is recommend to use different keys for different cryptographic functions [17]. Hence, a complete message sent by a RTES S to CC D (or vice versa) is:

$$S \rightarrow D: \ S, D, C, Cnt, M \quad (3)$$

where the ciphertext C supports the confidentiality of the original plaintext and the secure checksum M supports the integrity and authenticity of the message.

Upon the message reception, the receiver first computes the MAC M' on $S|D|C|Cnt$. If the computed MAC M' and the received MAC M match, the receiver knows that the message is not altered during the transmission. Also, it is confirmed that the message is actually from S, because an unauthorized adversary without the key K_m shared between the RTES and CC cannot generate the correct MAC M. In this way, one can support both the message integrity and authenticity [12, 17]. After successfully verifying the integrity and authenticity of the message, the receiver decrypts the ciphertext. Otherwise, it drops the message. Note that the sender can compute the secure one-way hash value via the same algorithm used for encryption without affecting security [17]. In this paper, we take this approach to reduce the storage requirements in resource constrained RTESs.

3.2 Strength of Defense

The strength of a scrutinized symmetric key system, which has no known shortcut to break it, is often estimated by the difficulty of finding the key via brute-force attacks as discussed before. The speed of a brute-force attack is mainly determined by the number of possible key values to be tested and the speed of a potential attacker. When the currently used key is l bits long and the adversary can test m keys per second, it will take him, in average, $2^{l-1}/m$ seconds to find the key. In 2001, Bond et al. showed that their DES (Digital Encryption Standard) [17] key cracker, which is one of the fastest, low cost DES key cracking machines, can test 2^{25} keys/second, finding a 56 bit DES key in less than 5 hours in the best case [3]. A U.S. government organization NIST (National Institute of Standards and Technology) presumes that a possible attack might be able to test 2^{55} keys/second, finding a DES key in only one second [14]. Due to the insecurity of the DES, NIST announced a new standard algorithm for encryption, i.e., AES [15], which uses a 128, 192, or 256 bit key. Based on the observation, we require that:

$$l_i \geq l_{min} \quad (4)$$

for the safety of the cryptosystem. For example, a security officer can set $l_{min} = 128$ following the NIST recommendation discussed above. Note that our algorithm is not tied to a specific encryption algorithm, key length, or speed of a potential attacker, but it is generally applicable to dynamic balancing between timing and security requirements as long as an encryption algorithm supports variable key lengths.

Given that our approach always supports at least the minimum strength of the cryptosystem by satisfying Eq 4, we can define the SoD via the normalized average key length used by real-time tasks in a RTES:

$$SoD = \frac{1}{N} \sum_{i=1}^{N} l_i / l_{max} \quad (5)$$

where N is the number of tasks currently in the system, l_i is the length of the key used by an arbitrary task T_i, and l_{max} is the maximum key length supported by the RTES. Note that $l_{min}/l_{max} \leq SoD \leq 1$; that is, the SoD metric in Eq 5 succinctly indicates the currently supported SoD compared to the maximum possible SoD that can be supported by a RTES. Although encryption and secure one-way hashing using a longer key are safer, it usually takes more time [2] incurring deadline misses. For example, the execution of the AES algorithm using 128, 192, and 256 bit keys in a low-end microprocessor takes approximately 2ms, 3ms, and 4ms in average [4]. Therefore, we propose to use a longer key under light loads, while switching to a shorter key when the system suffers transient overloads. For example, a RTES in a UAV can normally use a 256 bit AES key to support the strong cryptographic security of mission critical data, while switching to a 128 bit key when overloaded. (Recall that keys are assumed to be generated and distributed offline to avoid the overhead due to online key selection and exchange as discussed in Section 2.)

3.3 Security Adaptation under Overload

In our model, a soft real-time task T_i is associated with a relative deadline D_i. If it is a periodic task, we assume its deadline is equal to the period. An aperiodic soft real-time task is also associated with a relative deadline. The estimated execution time of T_i is: $C_i = C_{i,c} + C_{i,e(l_i)}$ where

$C_{i,c}$ is the estimated real-time function execution time and $C_{i,e(l_i)}$ is the estimated time for data encryption and secure one-way hashing when the current key length used by T_i is l_i. Thus, the estimated utilization of a real-time task T_i is: $U_i = C_i/D_i$.

We aim to maximize the success ratio:

$$\text{Maximize } Success\ Ratio = N_t/N_s \tag{6}$$

where N_t and N_s represent the number of the timely tasks that finish within the soft deadlines and the number of the tasks submitted to the system, respectively. The success ratio maximization is subject to Eq 4 and:

$$\sum_{i=1}^{N} U_i \leq B \tag{7}$$

where B is the utilization bound, e.g., 100% in EDF (Earliest Deadline First) [10]. In our approach, Eq 4 is always enforced as discussed in Section 3.2. Therefore, via key length adaptation, we can improve the success ratio under overload, while always supporting at least the minimum required SoD as follows.

1. Initialize the step size $\alpha = 1$.
2. Measure the current utilization U at every sampling period, e.g., 1 second.
3. If U is higher than the specified upper threshold U_h (e.g., 90% in EDF), decrease the key length for α tasks that have the earliest deadlines among the tasks using the key longer than l_{min}. Also, double α.
4. Repeat Steps 2 and 3 until $U \leq U_h$ or $l_i = l_{min}$ for every task T_i in the system.

Figure 1. Key Length Reduction under Overload

In our approach shown in Figure 1, we consider that a RTES is overloaded if the current utilization is higher than the specified upper threshold U_h. By requiring $U_h < B$, e.g., by setting $U_h = 90\%$ in EDF, SSTT can start adapting the key length before the system becomes saturated. Specifically, the key length is reduced for α tasks when the current utilization is higher than U_h. Note that, in SSTT, the key length is reduced for the earliest deadline tasks first to maximize the success ratio under overload. Also, our approach can quickly react to a transient overload by exponentially increasing α, if necessary, to handle severe overload.

Our algorithm has little overhead. Since the EDF queue is already sorted in nondecreasing order of deadlines, finding the first α tasks incurs no additional overhead for sorting. Specifically, the time complexity of this algorithm is $O(N)$ in the worst case. To amortize the overhead, SSTT considers the key length adaptation for only the first k tasks in the EDF queue per adaptation period where k is a predefined constant, e.g., 10. In this way, the overhead of our approach becomes **constant**, i.e., O(1). Under severe overload, it is possible that $\alpha > k$. In this case, it will take SSTT more than one adaptation period to switch α tasks to a shorter key. Hence, there is a tradeoff between the overhead and speed of overload management. In this paper, we take the constant overhead approach for the performance evaluation discussed in Section 4. In addition to improving the timeliness under transient overload by degrading the SoD with little overhead, SSTT has several desirable security and system features as follows:

- By switching between several independent keys, we can require an adversary to spend more time to find the current key. Also, the key may not be used anymore when the adversary eventually finds it. In this way, we can further confuse an adversary forcing him spend more time and resources to break the cryptosystem.
- Storing a constant number of keys with different lengths in a RTES does not significantly increase the storage requirement, which is desirable in resource constrained RTESs. We can further reduce the storage requirement by using the same algorithm for encryption and secure one-way hashing. In addition, switching to a different key incurs little overhead due to the offline key selection and distribution.

In our approach, a RTES S can request the CC D to use a shared shorter key by setting the flag F to 1 in a regular message of the the following format, which extends the message format described in Section 3.1.

$$S \rightarrow D:\ S, D, C, Cnt, F, M_s \tag{8}$$

Note that F is part of a regular message; therefore, a RTES does not have to transmit a separate message for key length synchronization. Thus, key length synchronization incurs little extra communication overhead. To support the integrity and authenticity of the message, the secure checksum M_s is computed for $S|D|C|Cnt|F$. The CC D returns the following acknowledgment (ACK) message to the RTES S:

$$D \rightarrow S:\ D, S, ACK, Cnt, F, M_d \tag{9}$$

when it receives the request (Eq 8) and the integrity and authenticity of the request message are successfully verified. $M_d = MAC(D|S|ACK|Cnt|F)$ to support the integrity and authenticity of the ACK message. The Cnt is

included in both the messages exchanged between S and D to avoid replay attacks. The RTES starts using the shorter key when it receives and successfully verifies the ACK message (Eq 9). We assume the underlying networking protocol, e.g., TCP (Transmission Control Protocol) [20], reliably delivers each message such that a message from one node is delivered without error to the destination. Furthermore, we assume that the packet delivery is handled by a separate network interface card without affecting the success ratio of the real-time tasks running in a RTES. Finally, note that we do not exchange the key. Thus, an adversary can extract the key from the messages exchanged between the RTES and CC by no means but brute-force attacks.

4 Performance Evaluation

In this section, we describe our simulation settings and compare the performance of SSTT to the baseline approach that applies the EDF scheduling algorithm and always uses the longest key. Note that we do not consider another extreme in which a RTES always uses the shortest key even when the system is underutilized. Since RTESs are often used in mission critical applications, they are required to support the strong confidentiality, integrity, and authenticity unless many deadlines are missed. Further, we intentionally do not apply other overload management techniques such as admission control to clearly show the performance improvement, if any, achieved by SSTT. In our experiments, a simulation run executes for 10 (simulated) minutes. For each performance data, we take an average of 10 simulation runs using different seed numbers.

Table 1. Simulation Settings

Parameter	Value
$C_{i,c}$	$Uniform(3ms, 8ms)$
$C_{i,e(l_i)}$	1ms, 2ms, or 3ms for a short key and 4ms for a long key
$AppLoad$	60%, 70%, 80%, ..., 160%
Slack Factor	(8, 12)
#Keys in a RTES	2
|Short Key|/|Long Key|	0.5

Table 1 summarizes the simulation settings. To generate workloads, we create multiple workload sources that generate tasks whose inter-arrival times are exponentially distributed. By increasing the number of the sources, we can increase the load applied to the simulated RTES. Specifically, we increase the AppLoad from 60% to 160%. A source S_i is associated with the estimated execution time $C_i = C_{i,c} + C_{i,e(l_i)}$. S_i is also associated with the relative deadline $D_i = slack \times C_i$ where the slack is uniformly distributed between 8 and 12. The total execution time C_i and slack factor are similar to [22] that models air traffic control workloads. Without losing the general applicability of our approach, we assume that a RTES stores two keys where the length of the short key, e.g., 128 bits in the AES algorithm, is a half of the length of the long key, e.g., 256 bits. The encryption time $C_{i,e(l_i)}$ is taken from [4] measured for the AES using a 128 bit and 256 bit key, respectively. Hence, the task execution time is reduced by 2ms when the key length is decreased by half. In addition, we consider the cases in which the execution time reduces by 1ms and 3ms to show the performance results given the nominal variances. Note that we do not model the time for keyed one-way hashing. This omission only favors the baseline approach that always uses the long key. If the key length for secure one-way hashing is also reduced under overload, SSTT can further improve the success ratio.

We set the tunable parameters of our algorithm discussed in Section 3 as follows: $U_h = 90\%$ and $k = 10$. SSTT has constant overhead per adaptation period, because it only considers the 10 earliest deadline tasks for SoD degradation per adaptation period. In addition, we consider two alternative adaptation periods, i.e., 1 second and 5 seconds. We have observed that the system generally becomes more reactive to overloads when the short adaptation period is used. On the other hand, the SoD can be increased by decreasing the key length less often when the longer adaptation period is used. Due to space limitations, we only show the performance evaluation results for the 1 second adaptation period in the following. (A more detailed discussion of the performance results is given in [6].)

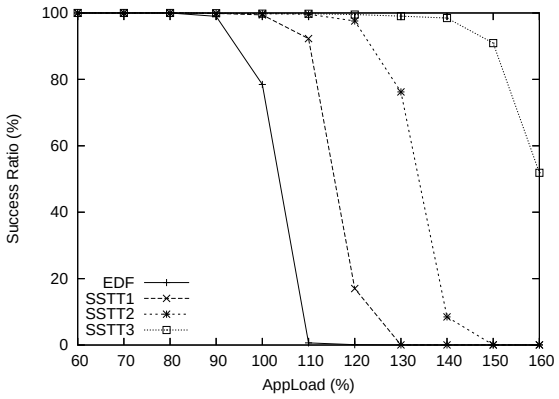

Figure 2. AppLoad vs. Success Ratio

Figure 2 shows the success ratio of the tested approaches for different AppLoads. Specifically, SSTT1, SSTT2, and SSTT3 represent the success ratios when the encryption time is reduced by 1ms, 2ms, and 3ms due to the key size reduction. EDF's success ratio is nearly 100% up to 90% AppLoad, while it significantly drops to near 0% when AppLoad = 110% due to the domino effect in EDF schedul-

ing [10]. Note that the success ratio of EDF is below 80% when AppLoad = 100%, because our simulator generates workload sources in a stochastic manner, stopping the source generation when the estimated workload is equal to or *higher* than the required AppLoad. In fact, when AppLoad = 100%, the actual stochastic workloads generated for the 10 simulation runs ranged between approximately $102\% - 109\%$. SSTT significantly improves the success ratio as shown in the figure. SSTT1 achieves the near 100% success ratio up to 100% AppLoad. Its success ratio is over 90% when AppLoad = 110%, while achieving the approximately 17% success ratio for AppLoad = 120%. Thus, we observe that SSTT1 is more resilient to overloads than EDF due to security and timeliness tradeoffs. The success ratio of SSTT2 is over 95% when AppLoad = 120%. Also, it is over 75% when AppLoad = 130%. SSTT2 achieves the better success ratio than SSTT1, because it can reduce more workloads by degrading the SoD for a task. For similar reasons, SSTT3 achieves the best performance among the tested approaches, showing the approximately 90% success ratio when AppLoad = 150%, while its success ratio is over 50% when AppLoad = 160%. Hence, we observe that SSTT can significantly improve the success ratio under overload by delaying the occurrence of the domino effect.

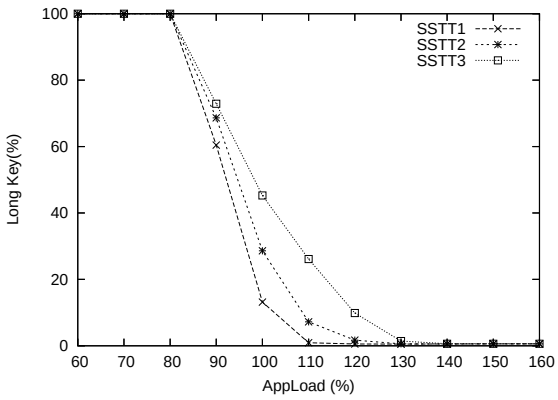

Figure 3. AppLoad vs. Fraction of Tasks Using the Long Key

Figure 3 shows the fraction of the tasks executed using the long key for the tested AppLoads. We do not plot the fraction for EDF, since it always uses the long key. Every task executed in SSTT1, SSTT2, and SSTT3 uses the long key up to AppLoad = 80%. Thus, the $SoD = 1$ (Eq 5) in this case. When AppLoad = 90%, SSTT begins to degrade the SoD: Approximately 60%, 68%, and 72% of the tasks executed in SSTT1, SSTT2, and SSTT3 use the long key. This is because SSTT begins to degrade the SoD when the current utilization is higher than the threshold $U_h = 90\%$. Further, the actual workloads generated stochastically are often higher than 90% when AppLoad = 90% as discussed before. When AppLoad = 100%, approximately 13%, 28%, and 45% of the executed tasks use the long key in SSTT1, SSTT2, and SSTT3, respectively. Almost all the tasks executed in the three approaches use the short key when AppLoad = 130% to improve the success ratio under overload, achieving the $SoD \approx 0.5$. We observe that the key length of SSTT3 decreases relatively slowly compared to SSTT1 and SSTT2, because it can reduce more workloads than SSTT1 and SSTT2 by decreasing the key length for a certain number of tasks.

5 Related Work

Cryptographic security support has rarely been studied in the context of real-time systems. QRAM [8, 16] selects an appropriate encryption key length based on the importance of an application and its resource requirements. However, the selection of the key length only occurs at the start of an application, e.g, a video conference, unlike our approach. Online application of QRAM may incur high overheads due to the complex QoS optimization procedure. As a result, many deadline can be missed. In contrast, our amortized SoD adaptation procedure discussed in Section 3 has the constant time complexity.

Miyoshi et al. [13] have developed a novel access control scheme using the resource control lists to protect time-multiplexed resources such as the CPU and network bandwidth against some DoS (Denial of Service) attacks. Their work is complementary to our work. For example, we can use the resource control lists to protect real-time embedded systems against some DoS attacks, while balancing the timing and cryptographic security requirements.

The access control problem in the multilevel security model has been studied in the real-time database literature [1, 18, 5, 7]. A majority of these work including [1, 18] temporarily allow a covert channel, which can be used by an adversary to enable an illegal information flow between different security levels, to improve the timeliness under overload conditions. George et al. [5] propose a secure real-time concurrency control protocol to avoid a covert channel. Kang et al. [7] propose an approach to preventing covert channels similar to [5], while aiming to support the desired average and transient deadline miss ratio in real-time databases. However, none of these work considers issues related to cryptographic security support.

Although the notion of Quality of Protection has been introduced in [9] to integrate the security and QoS support, it is not clearly known yet how to measure the quality of general security service. Generally, the quality of security service can only be measured *qualitatively* unlike real-time performance. Spyropoulou et al. [19] have proposed the notion of QoSS (Quality of Security Service). Ideally, a sys-

tem administrator and a security officer can select an appropriate security scheme to optimize the cost-benefit relation, when a quantitative model showing the computational cost and benefit of a security service is given. However, they give no specific model that can be used for the cost-benefit analysis. Also, they do not consider real-time constraints. In this paper, we suggest to use the key length as the *quantitative* SoD metric in the context of real-time embedded systems. Generally, real-time system security is a challenging open problem with many remaining issues to explore. Our work is an initial attempt to tackle the problem focused on systematic cryptographic security and timeliness tradeoffs in real-time embedded systems.

6 Conclusions and Future Work

A number of RTESs are employed in important applications, e.g., electric grid management or defense applications. In these systems, it is essential to meet deadlines, while supporting the security requirements considered in this paper. However, cryptographic security support in RTESs has rarely been explored. To address this problem, we propose a novel approach for systematic security and timeliness tradeoffs based on the concept of the strength of defense. Our approach is not only lightweight but also provides desirable security and system properties in RTESs. In the simulation study, our approach significantly improves the success ratio under overload. In the future, we will further investigate efficient cryptographic security support in RTESs. We will also investigate other security issues such as detection of (distributed) denial of service attacks.

References

[1] Q. Ahmed and S. Vrbsky. Maintaining Security in Firm Real-Time Database Systems. In *14th Annual Computer Security Applications Conference*, 1998.

[2] M. Blaze, W. Diffe, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security, A Report by An Ad Hoc Group of Cryptographers and Computer Scientists, January 1996. Available at http://www.crypto.com/papers/.

[3] R. Clayton and M. Bond. Experience using a low-cost FPGA design to crack DES keys. In *Cryptographic Hardware and Embedded System*, 2002.

[4] J. Deamen and V. Rijmen. Efficient Block Ciphers for Smartcards. In *USENIX Workshop on Smartcard Technology*, 1999.

[5] B. George and J. R. Haritsa. Secure Concurrency Control in Firm Real-Time Databases. *Distributed and Parallel Databases*, 5:275–320, 1997.

[6] K. D. Kang and S. H. Son. Systematic Security and Timeliness Tradeoffs in Real-Time Embedded Systems. Technical Report CS-TR-06-KD01, Department of Computer Science, SUNY Binghamton, 2006. Available at www.cs.binghamton.edu/∼kang.

[7] K. D. Kang, S. H. Son, and J. A. Stankovic. STAR: Secure Real-Time Transaction Processing with Timeliness Guarantees. In *The 23rd IEEE International Real-Time Systems Symposium*, Dec. 2002.

[8] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On Quality of Service Optimization with Discrete QoS Options. In *the 4th IEEE Real-Time Technology and Applications Symposium*, 1998.

[9] J. Linn. Generic Security Service Application Program Interface. IETF Request for Comments: 1508, 1993.

[10] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[11] J. Loyall, R. Schantz, D. Corman, and J. P. andS. Fernandez. A Distributed Real-Time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets. In *The 11th IEEE Real-Time Embedded Technology and Applications Symposium*, 2005.

[12] W. Mao. *Modern Cryptography*. Prentice Hall, 2004.

[13] A. Miyoshi and R. Rajkumar. Protecting Resources with Resource Control Lists. In *IEEE Real Time Technology and Applications Symposium*, 2001.

[14] National Institute of Standards and Technology. Advanced Encryption Standard Questions and Answers, 2002. Available at http://csrc.nist.gov/CryptoToolkit/aes/aesfact.html.

[15] National Institute of Standards and Technology, Federal Information Processing Standards Publication 197: Announcing the Advanced Encryption Standard, Nov. 2001.

[16] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical Solution for QoS-based Resource Allocation Problems. In *IEEE Real-Time Systems Symposium*, December 1998.

[17] B. Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.

[18] S. H. Son, R. Mukkamala, and R. David. Integrating Security and Real-Time Requirements using Covert Channel Capacity. *IEEE Transactions on Knowledge and Data Engineering*, 12(6), Dec 2000.

[19] E. Spyropoulou, T. Levin, and C. Irvine. Calculating Costs for Quality of Security Service. In *15th Computer Security Applications Conference*, 2000.

[20] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[21] M. J. Wiener. Efficient DES Key Search. Technical Report TR-244, Carleton University, May 1994.

[22] M. Xiong, K. Ramamritham, J. A. Stankovic, D. Towsley, and R. Sivasankaran. Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1155–1166, September/October 2002.