Towards Security and QoS Optimization 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

A number of real-time embedded systems (RTESs) are used to manage critical infrastructure such as electric grids or C^4I systems. In these systems, it is essential to meet deadlines, for example, to avoid a power outage or loss of a life. The importance of security support is also increasing, because more RTESs are being networked. To securely transmit sensitive data, e.g., a battle field status, across the network, RTESs need to protect the data via cryptographic techniques. However, security support may cause deadline misses or unacceptable QoS degradation. As an initial effort to address this problem, we formulate the security support in RTESs as a QoS optimization problem. Also, we propose a novel adaptive approach for security support in which a RTES initially uses a relatively short cryptographic key to maximize the QoS, while increasing the key length when the security risk level is raised. In this way, we can make a possible cryptanalysis several orders of magnitude harder by requiring the attacker to search a larger key space, while meeting all deadlines by degrading the QoS in a controlled manner. To minimize the overhead, we derive the appropriate QoS levels for several key lengths via an offline polynomial time algorithm. When the risk level is raised online, a real-time task can use a longer key and adapt to the corresponding QoS level (derived offline) in O(1) time.

Keywords

Real-Time Embedded Systems, Security, Deadlines

1. INTRODUCTION

A number of real-time embedded systems (RTESs) are used to manage critical infrastructure such as electric grids and C^4I (Command, Control, Communications, Computers, and Intelligence) systems. In these systems, meeting deadlines is essential, since a deadline miss may cause a catastrophic result, e.g., a power outage or loss of a life. The importance of security support is also increasing, since more RTESs, e.g., C^4I systems, are being networked. Although cryptographic security and real-time systems have been well studied *separately*, very little work has been done to meet both timing and cryptographic security requirements.

Simultaneously supporting security and timing constraints is challenging, since these requirements can compete for computational resources. A simplistic approach, e.g., always using the longest key, may incur severe QoS degradation in resource constrained RTESs, often based on cheap, low-end microprocessors and/or microcontrollers. (It is known that the time for encryption increases as the key length increases [13].) In addition, sensor and control data have to be protected for a relatively short time, since they change frequently. Unless the current risk is high, too hard an armor could be excessive, wasting precious resources. Thus, we aim to handle potentially time-varying risks in a cost-effective manner.

In this paper, we first formulate the security support in RTESs as a QoS optimization problem. In addition, we present a novel *adaptive* and *risk-aware* approach to efficiently satisfying security and timing constraints in resource constrained RTESs. When a network intrusion detection system raises the level of the security risk, e.g., due to the detection of attempts for illegal data access, a RTES begins to use a longer cryptographic key to better protect sensitive data. As a result, a possible cryptanalysis trying to find the cryptographic key is forced to take several orders of magnitude longer. At the same time, the RTES may have to degrade the QoS of certain real-time tasks to avoid potential deadline misses due to the increased security cost.

To minimize the adaptation overhead, we perform an offline analysis to optimize the QoS for several different key lengths. In this way, we ensure all deadlines will be met if the total utilization is smaller than or equal to the schedulable utilization bound, e.g., 1 in EDF, while maximizing the QoS for given key lengths. Given the current risk level, our approach can dynamically select the appropriate key length and QoS levels of real-time tasks from the precomputed settings in O(1) time. Consequently, our approach can support desirable security and real-time features with minimal overheads.

The remainder of the paper is organized as follows. The scope of the work is described by discussing the basic assumptions and formulating the problem of security and real-time performance management in Section 2. The crypto-

graphic security support is described in Section 3. Our online and offline algorithms for security and timing assurance are presented in Section 4. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and discusses the future work.

2. SCOPE OF THE WORK

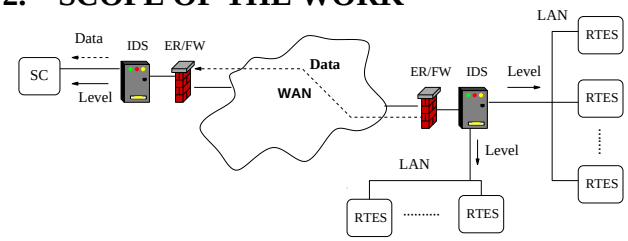

Figure 1: System Model: Example

2.1 Application Scenario and System Model

Figure 1 shows a high level diagram of example RTESs and a service center (SC) that operate as follows. A RTES monitors and controls, e.g., (part of) an electric grid or a battle field. A RTES may transmit sensor or control data to a SC to report, e.g., an overload of a power generator. RTESs in a specific area are connected via a local area network (LAN) such as a control area network (CAN), while communicating with a SC across the wide area network (WAN) through an edge router (ER) that connects the LAN to the backbone network.

We assume that each task in a RTES is associated with a hard deadline; however, the transmission delay between a RTES and SC does not require a hard guarantee. We also assume that RTESs are independent of each other to avoid a deadline miss due to an unpredictable delay for communication. We assume that the communication medium access, transmission, and propagation delay in the LAN, e.g., CAN, is predictable and the delay is included in the schedulability analysis to meet all deadlines, similar to [14]. We also assume a RTES transmits data in asynchronous mode, in which control is immediately returned to a real-time task without waiting for the completion of the transmission.

An ER is equipped with a firewall (FW) to filter suspicious packets and connection requests, e.g., insecure file transfer protocol (FTP) connection requests. Note that a firewall does not completely solve the network security problem, because an adversary, who has access to the wide area network, may eavesdrop or inject packets to the network. For example, a terrorist may try to find the weakest point of an electric grid by eavesdropping. Or, he may attempt to send false data to the SC. Hence, it is necessary to encrypt and authenticate data to support the confidentiality, integrity, and authenticity of the network messages.

An intrusion detection system (IDS) monitors incoming and outgoing packets and periodically broadcasts the current risk level to the connected RTESs or a SC in a specific area. (We assume the intrusion detection and broadcast task is sporadic and can subsequently be mapped as a periodic task. A thorough investigation is reserved for future work.) A SC is in charge of a specified set of RTESs. It takes care of, e.g., long-term plans for energy supply and military tactics, based on the information received from RTESs (and other sources, if any). We assume that a SC is also protected via a firewall and IDS, similar to RTESs.

2.2 Threat Model

We assume that an isolated RTES is trusted. Thus, a RTES only has to encrypt data before transmitting them across the network. We also assume that an IDS is not compromised.

We consider a symmetric key system in which a SC and RTES share a secret key, since the encryption/decryption in a public key system takes several orders of magnitude longer than that in a symmetric key system [13]. We support the confidentiality, integrity, authenticity of a message via encryption and cryptographic one-way hash function. Therefore, an adversary cannot eavesdrop, corrupt, or inject messages without being detected. In addition, we support semantic security and replay protection.

Each RTES shares a pair of secret keys with a SC to encrypt and authenticate a message to support the message confidentiality, integrity, and authenticity. We use different keys for encryption and authentication, since the rule of thumb is to use separate keys for different applications [13]. Note that only the SC can decrypt and check the integrity and authenticity of the message transmitted by the RTES using the shared keys.

Finally, it is assumed that main attacks against a scrutinized cryptosystem without known vulnerability, e.g., DES (Digital Encryption Standard) [13] or AES (Advanced Encryption Standard) [12], are *brute-force attacks* that try to find the secret key via an exhaustive search in the key space. Note that this is a common assumption in trusted cryptosystems [13]. (A more detailed discussion of the specific security support considered in this paper is given in Section 3.)

2.3 Problem Formulation

In this paper, we assume that a RTES consists of a set of N periodic tasks $< T_1, T_2, ..., T_N >$. We assume the deadline of a task is equal to its period. The (worst case) execution time of a task T_i is: $C_i = C_{i,c} + C_{i,e(l)}$ where $C_{i,c}$ is the real-time function execution time and $C_{i,e(l)}$ is the data encryption (and transmission) time for the key length l, respectively. The utilization of a task T_i is: $U_i = C_i/P_i$ where P_i is the period of T_i. Further, each task is associated with $k(\geq 1)$ discrete QoS levels. The QoS monotonically increases as the QoS level (and the corresponding execution time) increases, similar to the milestone approach [8]. Ideally, we aim to optimize the aggregate QoS of the real-time tasks $\{T_1, T_2, ..., T_N\}$ defined in terms of both security and real-time performance, while addressing the current security risk indicated by an IDS:

$$\text{Maximize } QoS = S(l)\frac{\sum_{i=1}^{N} Q(i)}{\sum_{i=1}^{N} max\ Q(i)} \leq 1 \qquad (1)$$

where $Q(i)$ and $max\ Q(i)$ are the current and maximum possible QoS of a task T_i and

$$S(l) = \begin{cases} 1 \text{ if } S(l) \geq R(t) \\ 0 \text{ otherwise} \end{cases} \qquad (2)$$

where $S(l)$ indicates the strength level of the currently used cryptosystem, which is determined by the current key length l measured in number of bits. For example, a security officer can set $S(80) = 1, S(112) = 2$, and $S(128) = 3$ for the DES algorithm. $R(t)$ in Eq 2 denotes the current risk level estimated by the IDS, e.g., 1 (low), 2 (medium), and 3 (high). Specifically, we select the key lengths to defend the cryptosystem against a potential cryptanalysis for a certain estimated period of time. Also, note that our QoS optimization is subject to

$$\sum_{i=1}^{N} U_i \leq UB \tag{3}$$

where the UB is the utilization bound of an employed real-time scheduling algorithm such as EDF. When the risk level $R(t)$ is raised by the IDS, a task T_i has to use a longer key. As a result, C_i and U_i may increase, and therefore, the QoS manager should reallocate resources to re-optimize the QoS. Otherwise, the QoS may become suboptimal, possibly incurring deadlines misses if Eq 3 cannot be satisfied.

3. CRYPTOGRAPHIC SECURITY SUPPORT

This section describes our cryptographic security scheme followed by a discussion of the strength of a symmetric key system.

3.1 Confidentiality, Integrity, and Message Authentication

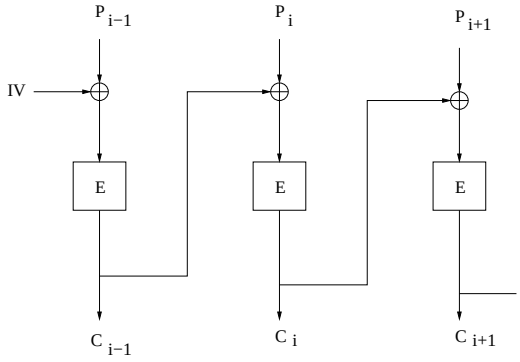

Figure 2: Cipher Block Chaining

In our approach, when a RTES wants to send a message P to a SC, it first encrypts the plaintext P to support the *confidentiality* of the message. The encrypted ciphertext message is:

$$C = E(P)_{\{K_e,\, Cnt\}} \tag{4}$$

where E is the encryption function such as DES or AES, K_e is the encryption key shared between a RTES and SC, and Cnt is the current counter value that is incremented after each message encryption and transmission. As shown in Figure 2, we use the cipher block chaining (CBC) mode [2, 13] in which the encrypted previous block is fed into the encryption of the current block; that is, the input to the encryption function E using the key K_e is XOR(C_{i-1}, P_i) where C_{i-1} is the previous ciphertext block and P_i is the current plaintext block to be encrypted. By using the counter value Cnt as the initialization vector (IV) in CBC mode, we can support the desirable security features as follows.

- *Semantic security*: The notion of semantic security [2, 5] ensures that a view of a ciphertext does not help a cryptanalyst to find any information about the corresponding plaintext. In our approach, if one message is encrypted more than once, it will be encrypted differently, because the counter, i.e., IV, is incremented after each encryption. It is known that semantic security can be supported when the counter is used as the IV in CBC mode [13].
- *Replay protection*: An adversary cannot replay an old message, since the IV is updated after each message.

Semantic security and replay protection are supported as long as the counter is not wrapped around yet. Generally, it is a good practice to distribute new keys before the counter wraps around.[1] By using several keys of different lengths, we can safely delay the expensive key distribution process in two ways. First, a separate counter can be associated with each key of a certain length in a RTES to support semantic security and replay protection. After encrypting (and transmitting) a message using a specific key, only the corresponding counter is incremented. Since we use a constant number of keys, the related overhead of maintaining the separate counters for different lengths of keys can be limited. Furthermore, when the risk level is increased, we reduce the frequencies of certain control tasks, if necessary, to use a longer key without missing any deadline. Accordingly, the frequency of the data transmission between a RTES and SC is reduced. Thus, the key distribution is safely delayed due to the corresponding delay of the counter increment.

To check the integrity of the message and authenticate the claimed sender, a message authentication code (MAC) is computed over the encrypted message C:

$$MAC = H(C|Cnt)_{K_m} \tag{5}$$

where H is a key-based one way hash function, $C|Cnt$ is the concatenation of the ciphertext (Eq 4) and the counter value, and K_m is the message authentication key shared between a RTES and SC.[2] Using a key-based one way hash function, one can support the following desirable security features [10, 13].

- *Data Integrity*: A one way hash function works in one direction; that is, computing the hash value from the pre-image, i.e., $C|Cnt$ in Eq 5, is easy. However, generating a pre-image that hashes to a specific value is hard. Also, a good one way hash function avoids a collision;that is, it ensures different messages hash to different values. Thus, it is very hard for an adversary to modify or fabricate a message that hashes to the original MAC value. Upon the reception of the message, the receiver computes the MAC of the $C|Cnt$ using the hash function (Eq 5) to check its integrity.

[1] A detailed discussion of key distribution is beyond the scope of this paper. A thorough investigation is reserved for future work.

[2] In fact, the MAC is computed over the entire message including the source and destination address. We intentionally omit these parts for the clarity of presentation.

If the message body including the $C|Cnt$ has not been altered during the transition, the computed MAC will be equal to the received one.

- *Message Authentication*: Unfortunately, a one way hash function itself cannot authenticate a message. For example, an adversary can generate a false message and hash the message. The message will be verified successfully by the receiver. A key-based hash function using a secret authentication key (K_m in Eq 5) can handle this problem. Only the sender with the authentication key can generate a valid hash value that can only be verified by the receiver sharing the key with the sender [13]. Thus, the receiver can verify whether or not the message is sent from the claimed sender by checking the MAC. Once the integrity and authenticity of the message is verified, the receiver decrypts the ciphertext. Otherwise, it drops the message.

To implement a secure one way hash function for message authentication, we use the block cipher in CBC mode shown in Figure 2. The final encryption output C_n (where n is the number of data blocks to be authenticated) is the hash value. We take this approach, since it is a very efficient method to implement a secure one way hash function [13]. Further, the CBC MAC is provably secure [3].

A message sent by a RTES A to SC B (or vice versa) includes the encrypted message and MAC in addition to other information such as source and destination address (not shown here):

$$A \rightarrow B: \ C, Cnt, MAC \tag{6}$$

where the ciphertext C supports the confidentiality of the original plaintext and MAC supports the integrity and authenticity of the message. In addition, we can support semantic security and replay protection by using the IV in CBC mode, while safely delaying the key (re)distribution procedure when the risk is raised as discussed before.

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 [13, 19]. The speed of a brute-force attack, in which an adversary tests all possible keys, is mainly determined by the number of possible key values to be tested. Wiener [19] designed a specialized parallel hardware to estimate the average time for a brute-force attack to break the DES algorithm. Note that this approach is not limited to the DES, but generally applicable to other encryption algorithms such as AES [13]. Table 1 shows the average time estimates for hardware brute-force attacks in 2005. (These estimates extrapolate the previous results [13, 19] according to Moore's law.)

Since a RTES mainly deals with time-varying sensor and control data, using a short, e.g., 80 bit, key might be acceptable. However, a malicious third party could attack a cryptosystem with less budget as the hardware technology develops fast. As a result, today's safe key length may not be safe tomorrow. This is an important observation, since RTESs can be used for a long time after the deployment, for example, to continuously monitor and control critical infrastructure. Our approach is also useful in this case; a RTES can simply switch to a longer key, while dropping a short key that is not considered safe anymore due to the advance of the hardware technologies decreasing the time and cost for brute-force attacks.

Table 1: Estimated Average Times for Parallel Brute-Force Attacks

	Length of Keys		
Hardware Cost	80 bits	112 bits	128 bits
\$10K	7000 years	10^{13} years	10^{18} years
\$100K	700 years	10^{12} years	10^{17} years
\$1M	70 years	10^{11} years	10^{16} years
\$10M	7 years	10^{10} years	10^{15} years
\$100M	245 days	10^{9} years	10^{14} years

4. SECURITY AND TIMELINESS ASSURANCE

Our online security and QoS adaptation algorithm is summarized in the following.

1. Periodically receive the current risk level $R(t)$ at time t from the IDS.

2. If the current strength of defense $S(l) < R(t)$, use a new key that is l' bits long where $S(l') = R(t)$ and $l' > l$.

3. Degrade the QoS of certain real-time tasks, if necessary, to meet all deadlines.

Our offline precomputation algorithm for a specific key length is summarized and discussed as follows.

1. **Input:** Key length l and QoS function $g_i : u_i \rightarrow q_i$ for each task T_i where $1 \leq i \leq N$ (#tasks in the system) and u_i and q_i are the utilization and corresponding QoS, respectively.

2. **Output:** $Q\_List[l][i]$

3. **for** $(i = 1;\ i \leq N;\ i++)$
For a task T_i, derive the convex hull $H_i = \{< u_{i1}, q_{i1} > , ..., < u_{ij}, q_{ij} >\}$ where $j \leq L$ (#max QoS levels).

4. /* Merge sort in nondecreasing order of utilization */
$H\_List = Merge(H_1 - \{< u_{11}, q_{11} >\}, H_2 - \{< u_{21}, q_{21} >\}, ..., H_N - \{< u_{N1}, q_{N1} >\})$

5. $Q\_List[l] = \emptyset$ (empty set)

6. $U^a = UB$

7. /* Support at least the minimum QoS */
for $(i = 1;\ i \leq N;\ i++)$
{
a. **if** $(U^a < u_{i1})$
b. **print**("Error: Cannot Support Key Length l")

c. **exit**(-1)
d. $Q\_List[l][i] = q_{i1}$
e. $U^a = U^a - u_{i1}$
}

8. /* Initial resource allocation */
for $(i = 1;\ i \leq N;\ i++)$ $\quad U[i] = u_{i1}$

9. /* QoS optimization */
for $(i = 1;\ i \leq |H\_List|;\ i++)$
{
a. $id = H\_List[i].id$
b. $\delta U = H\_List[i].u - U[id]$
c. **if** $(\delta U > U^a)$ **exit**(0)
d. $Q\_List[l][id] = H\_List[i].q$
e. $U^a = U^a - \delta U$
f. $U[id] = H\_List[i].u$
}

In our approach, we assume a RTES stores the secret keys in an array $key[1..|R|]$ where $|R|$ is the number of risk levels considered by the IDS. For each key length, as shown in Steps 1 and 2, we precompute the near optimal QoS levels of the real-time tasks offline to minimize the overhead for online security and QoS adaptation, if necessary. For a specific key length l, task T_i can simply switch to the QoS level stored in $Q\_List[l][i]$ when the system needs to use a key of length l to ensure that the level of defense is higher than or equal to the risk level.

In Step 3, as shown in Figure 3, we remove suboptimal discrete QoS levels by deriving the convex hull for each task $T_i (1 \leq i \leq N)$, similar to [7]. In this way, a list of near optimal $< utilization, QoS >$ pairs, i.e., $\{< u_{i1}, q_{i1} >, < u_{i2}, q_{i2} >, ..., < u_{ij}, q_{ij} >\}$, can be derived for each task T_i in increasing order of utilization where u_{ij} is the utilization required to support task T_i's QoS level j.

In Steps $4-7$, we initialize $Q\_List[l]$ to support at least the minimum required QoS when the key is l bits long. In Step 4, all the $< utilization, QoS >$ pairs of all tasks except $< u_{i1}, q_{i1} >$ are merged in nondecreasing order of utilization. If two pairs have the same utilization, the one with the higher QoS is added to the list first. A tie, in which two pairs have the same utilization and QoS, is broken in a random manner. In Step 6, the available utilization U^a is initialized. In Steps 7.a−7.c, we check whether U_a is enough to support the deadline and the minimum QoS of T_i. If this test fails, the RTES cannot support the specific key length l and the algorithm terminates with an error. If the required utilization u_{i1} to support the minimum QoS of T_i is available, we set $Q\_List[l][i] = q_{i1}$ and subtract u_{i1} from U^a in Steps 7.d and 7.e. The procedure is repeated for the next task T_{i+1}, if any.

In Step 8, the required minimum utilization of each task T_i $(1 \leq i \leq N)$ needed to support at least T_i's minimum QoS is initialized. In Step 9.a, we remove the first task in the H_List, i.e., T_{id}, which requires the smallest utilization. δU, i.e., the extra utilization needed to improve the QoS of T_{id}, is computed in Step 9.b. If $\delta U > U^a$, the algorithm terminates normally in Step 9.c. (We can allocate remaining resources in a greedy manner; however, we do not take the greedy approach to reduce possibly excessive QoS fluctuations when the risk level changes.) Otherwise, the QoS of T_{id} is raised to the next higher level in Step 9.d. In Steps 9.e and 9.f, the available utilization U^a and the utilization allocated to T_{id} are updated, respectively. The whole procedure is repeated for every key length used by a RTES. Consequently, the algorithm generates a table $Q\_List[1..\#keys][1..\#tasks]$ that can be used by our online algorithm to select an appropriate key length and QoS levels considering the current risk level in O(1) time. Finally, the time complexity of our offline algorithm is $O(KLNlogLN)$, which becomes $O(NlogN)$ when the number of keys (K) and QoS levels (L) are constant.

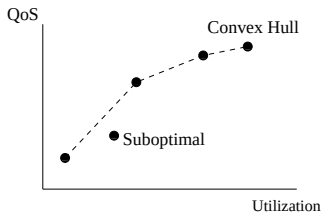

Figure 3: Example QoS Function

5. RELATED WORK

Cryptographic security support has rarely been studied in the context of real-time systems. Lee et al. [7] propose to consider security as a QoS dimension and select an appropriate encryption key length based on the importance of an application and its resource requirements. However, they do not discuss these security vs. performance issues in detail. They neither discuss the important security features and implications discussed in this paper. Further, the selection of an encryption key length and QoS levels only occurs at the start of an application, e.g, a video conference, unlike our approach. Son et al. [16] developed an adaptive security manager in a real-time database. When the real-time database is overloaded, a weaker encryption algorithm is used to improve the deadline miss ratio. However, their approach does not consider potential security risks to make the decision for adaptation. Their work neither deals with hard deadlines.

The notion of Quality of Protection has been introduced in [9] to integrate the security and QoS support. Although the concept has existed more than a decade, it is not clearly known yet how to measure the quality of general security service. Spyropoulou et al. [17] have proposed the notion of QoSS (Quality of Security Service). Ideally, a system 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 general, the cost-benefit analysis of security services is an open problem. We have taken a first step to solving this problem focused on cryptographic security support in real-time embedded systems.

Miyoshi et al. [11] 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 systems against some DoS attacks, while supporting timing, stability, and cryptographic security requirements. In general, real-time system security is a challenging open problem with many remaining issues to study. We discuss a specific instance of the problem in this paper.

The access control problem in the multilevel security model has been studied in the real-time database literature [1, 15, 16, 4, 6]. A majority of these work [1, 15, 16] 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. [4] propose a secure real-time concurrency control protocol to avoid a covert channel. Similarly, Kang et al. [6] use a secure concurrency protocol to eliminate a covert channel, while supporting the desired average and transient deadline miss ratio via feedback control. Teo et al. [18] have developed a dynamic access control scheme based on the quantified levels of security risk received from an intrusion detection system, similar to our approach. However, they consider neither cryptographic security nor real-time constraints. Generally, our work is different from these work in that we focus on supporting cryptographic security and hard deadlines.

6. CONCLUSIONS AND FUTURE WORK

A number of real-time embedded systems are used to manage critical infrastructure, e.g., electric grids or C^4I systems. In these systems, it is essential to meet deadlines, while supporting the confidentiality, integrity, and authenticity of network messages. Despite the importance, security support in RTESs has rarely been supported. To address this problem, we formulate the problem as a QoS optimization problem and propose offline and online algorithms for security and QoS adaptation to address dynamic security risks, while optimizing the QoS. In this way, our approach efficiently supports essential cryptographic security features in resource constrained RTESs. In the future, we will further investigate efficient cryptographic support in RTESs considering different task models such as sporadic or aperiodic models. Further, we will investigate other security issues closely related to RTESs, e.g., detection of (distributed) denial of service attacks.

7. 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. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *38th IEEE Annual Symposium on Foundations of Computer Science*, 1997.

[3] M. Bellare, J. Kilian, and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362–399, Dec. 2000.

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

[5] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[6] 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.

[7] 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.

[8] K. J. Lin, S. Natarajan, and J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Real-Time System Symposium*, December 1987.

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

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

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

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

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

[14] J. W. S.Liu. *Real-Time Systems*. Prentice Hall, 2000.

[15] 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.

[16] S. H. Son, R. Zimmerman, and J. Hansson. An Adaptable Security Manager for Real-Time Transactions. In *Euromicro Conference on Real-Time Systems*, pages 63–70, Stockholm, Sweden, June 2000.

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

[18] L. Teo, G.-J. Ahn, and Y. Zheng. Dynamic and Risk-Aware Network Access Management. In *ACM Symposium on Access Control Models and Technologies*, 2003.

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