Optimizing Key Updates in Sensor Networks

Yuksel, Ender; Nielson, Hanne Riis; Nielson, Flemming; Fruth, Matthias; Kwiatkowska, Marta

Published in:
2011 IEEE Sensors Applications Symposium (SAS)

Link to article, DOI:
10.1109/SAS.2011.5739805

Publication date:
2011

Optimizing Key Updates in Sensor Networks

Ender Yüksel*, Hanne Riis Nielson*, Flemming Nielson*, Matthias Fruth† and Marta Kwiatkowska†
* Informatics, Technical University of Denmark
Richard Petersens Plads 322, DK-2800 Kgs. Lyngby, Denmark
Emails: {ey, riis, nielson}@imm.dtu.dk
† Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
Emails: {matthias.fruth, marta.kwiatkowska}@comlab.ox.ac.uk

***Abstract*—Sensor networks offer the advantages of simple and low–resource communication. Nevertheless, security is of particular importance in many cases such as when sensitive data is communicated or tamper-resistance is required. Updating the security keys is one of the key points in security, which restrict the amount of data that may be exposed when a key is compromised. In this paper, we propose novel key update methods, and benefiting from stochastic model checking we propose a novel method for determining optimal key update strategies for custom network scenarios. We also present a case study where an application in commercial building automation is considered.**

I. Introduction

Protection of data in transit in sensor networks is often provided by encryption algorithms that rely on cryptographic keys. Algorithms are assumed to be known by the attackers as stated in Kerckhoffs's principle, whereas the keys should be kept secret. However, keys may get compromised over time, therefore networks secured by encryption usually have the notion of *key update* where the key in use is revoked, a new key is established and distributed.

Wireless sensor networks present several challenges in terms of cryptography. Most of the challenges arise from the constrained resources on the sensor devices. Memory and processing power limitations, as well as the demand for a long battery life generally eliminate the use of public key cryptography. Therefore, symmetric cryptography is widely used [1]. Besides, key types can also be affected by the limitations since pairwise keys (*e.g.* link keys) are more resource-consuming than group keys or network keys. In this study, we will focus on the symmetric *network key* updates.

Applications of quantitative analysis in the security domains are rare, as it is usually difficult to quantify or even simulate the underlying problems. We employ *stochastic model checking* techniques for reasoning about security, which allow deeper insights than qualitative techniques or simulation alone [2]. Using our beneficial modelling and analysis approach we ask questions and get quantitative answers. In particular, we focus on optimising key confidentiality, key recovery, and efficiency of the key updates.

In this work, we present three main contributions: 1) we propose four novel key update methods for wireless sensor networks which consider the trade-off between limited resources and demand for security, 2) we propose a novel method for determining the strategy and parameters for key update, and present a case study, 3) we present a novel application of stochastic model checking in the security domain. Besides, we develop generic formal models for key update and allow developers to assess individual security parameters for their application scenarios.

The rest of this paper is organized as follows: Section II defines the key update problem in a sensor network setting. We propose novel key update methods in Section III. In Section IV, we propose our method for determining the best key update strategy based on given requirements. Section V is an appetizer to our formalization and quantitative verification using stochastic model checking. We conclude after presenting a case study on wireless sensor networks in Section VI.

II. The Key Update Problem

Given the usual resource limitations in sensor networks, absolute security is often less important than quantifiable trade-offs between security and performance. As security is a qualitative concept, realistic analyses require results that are valid with respect to the full behaviour of the systems considered.

In the following sections we detail the *key update* problem in ZigBee, followed by a discussion on the status quo.

A. Setting up the Scheme

We will use *ZigBee* for the running example and the developments throughout the paper [3]. The latest specification of ZigBee (ZigBee-2007) specifies a suite of security services that includes methods for key establishment, key transport, etc. Although each revision and supporting specifications (such as stack and application profiles) roll out improvements, *key update* strategies and proper determination of related security parameters still remain gaps in the ZigBee standard [4].

Naturally, we want to ensure that the risk of using a compromised security key in a ZigBee network is as small as possible – and this calls for updating the key fairly often. On the other hand, this operation is computationally expensive and we would not like to perform it too often. Unfortunately, the ZigBee specification does not give any advice on this (i.e. how and when the key shall be updated) – it merely states that the security key shall be updated. The ZigBee Smart Energy Application Profile, being one of the most critical

and important profiles, goes one step further and states that *periodically* the trust center shall update the network key.

We present the key points that are necessary for a clear understanding of the development below, and omit all the details which are irrelevant to this study. A more detailed discussion of ZigBee security can be found in [4] and surely the ultimate source is the ZigBee documentation (i.e. the specification [3], stack profiles, and application profiles).

ZigBee uses symmetric encryption, AES standard with 128 bits keys, therefore all the security keys are symmetric. A *Network Key* (NK) is the mere mandatory key in a ZigBee network, which is shared amongst all the devices and used to secure broadcast communications. A *Trust Center* (TC), creates and distributes the NKs. TC is an application running on a ZigBee device, that is unique in every ZigBee network. As a key component of ZigBee security, the TC is assumed to run on a more powerful device (*e.g.* a coordinator) rather than a regular ZigBee end device. Two more types of security keys may exist in a ZigBee network depending on the security configuration: *Master Key* and *Link Key*. Unlike NK, those keys are pairwise shared. In this study we focus on NK as the key type and refer to it as the *key*, and we assume that the devices in the network already acquired the *key*.

The details of the *NK update* protocol is given in the specification and explained in [4]. However, this protocol does not specify *when* to update NK, which is what we are focusing on in this paper. We assume that when TC updates the key, all the devices in the network successfully update their keys. Compromise of a NK affects all the devices in a network. TC is fully responsible of creating and distributing the NK, therefore there is no role of the devices in NK establishment.

B. Status Quo and Related Work

The notion of key update is often assumed to be *periodical*, and even ZigBee – as a fairly promising standard – is no exception. From now on we will refer to the periodical updating method as Time-based key update (TB). TB does not consider any source of key compromise, its mere concern is the time passed. Therefore it cannot easily be fitted to a specific network that has properties related to number of messages, number of joining/leaving devices etc.

Now let us illustrate the relation between the key update period and the number of key compromises with an example. In Fig. 1, we present three instances using TB where we keep everything else but the update threshold constant. The labels on the left of the time-lines indicate the update period in months, *e.g.* `M=1` means the key is updated monthly; and the update points are denoted by little bars. Since the distributions are the same, we have the key compromise event in the same time point regardless of the key update threshold. The first key compromise named as C_1 is common to all three instances, as it happens in the first month after the network (or observation of the network) starts. Showing the period when the key is compromised by dotted lines, we observe that a key compromise lasts longer as the update threshold increases. Then in the second month, we observe another key compromise only in `M=1`. This key compromise is not visible in the other cases since their keys are already compromised. This little example shows that the decrease in the number of key compromises is misleading since the total period of time when the key is compromised gets larger.

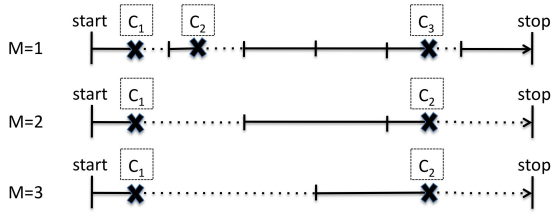

Fig. 1. Number of key compromise for different thresholds in TB

At this point we have to explain two standard notions of secrecy. *Forward secrecy* implies that compromise of the current key should not compromise any future key, whereas *backward secrecy* implies that compromise of the current key should not compromise any earlier key. While we have already assumed these properties in our study, Wallner *et al.* suggested that a shared key has to be updated on *every* membership change and redistributed to all authorized members securely to provide backward and forward secrecy [5]. This actually resembles the key update methods we will propose in the next section, however being an all-or-nothing approach lacks flexibility to be suitable to low-resource networks.

The effect of periodic key exchanges to the performance in an IEEE 802.15.4 network was discussed in [6]. They proposed a new key update method, which we will refer to as Message-based key update (MB) in this paper. However, their study was not about network keys but link keys, which are pairwise shared between devices. This approach counts the number of communicated messages and issues a key update after a certain number of messages communicated. MB does not consider leaving (and joining) devices, therefore is not aware of a device leaving the network while still holding a valid key. In this study, we assume MB for NKs (but not for link keys as in [6]) so that it can be comparable with other methods in the paper.

III. Proposed Methods for Key Update

In this section we will explain our four novel methods for key update. Including the two update methods we have previously explained (TB and MB), we will have six methods to be used in the case study.

A. Leave-based Key Update (LB)

This method is considering the leave events in the network. The key is updated after a predefined number of devices leave the network. In practice, when a device leaves the network it may still own a valid key, hence a device leave presents a security risk. To the best of our knowledge, this is a novel key update strategy that we propose in [7]. A counter in the Trust Center keeps track of the number of the devices left (or removed from) the network. When this number reaches

the predefined threshold value, all the keys in the network are updated and the counter is reset to zero. The idea here is to have a key update strategy that is inspired by the nature of the wireless sensor networks where the number of exchanged messages can be relatively low and devices can be tampered with by outside parties.

B. Join-based Key Update (JB)

This method is considering the join events in the network. The key is updated after a predefined number of new devices joins the network. A new device that joins the network presents a security risk since it may become a legitimate attacker. For example, the new device may have joined using an illegally obtained network key, or a legitimate device can try to decrypt old communication that it has captured before joining the network. This strategy is also our own proposal, and the idea is dual to LB. The key point here is, even though a device leaving the network with valid key is a risk itself, the bigger risk is when a new device joins after such a leave. Because the new device could have somehow captured the key from the previously leaving device. In this case, the counter would contain the number of joining devices, and the threshold would be set as a limit for this value.

C. Join-Leave-based Key Update (JLB)

In this method, the key is updated after a predefined number of devices join *or* leave the network. We consider each join and leave event as suspicious events and do not distinguish between them. JLB is powered by both JB and LB key update strategies therefore exhibiting the strength of these two. Since both join and leave events are considered, the threshold value is more sensitive than JB or LB.

D. Hybrid Key Update (Hy)

In the Hy method, we will employ multiple key update strategies, and issue the update whenever any of the update counters reaches its threshold. All the counters will be reset as the key is updated. In this way, we will be able to benefit from all useful key update strategies. Naturally, each key update strategy has a different strength *e.g.* performing well when too many leaves happen, or too many messages communicated, or the environment has less malicious activity, etc. In the hybrid key update strategy we will have all these strengths, with the cost of more computational power. Therefore we will suggest the hybrid key update to be used in networks where the coordinator device has sufficient resources. For example, if the coordinator device is a mobile computer or a powerful handheld then we can implement the hybrid strategy. Note that we did not include JLB in Hy, since it is limiting a real hybrid phenomenon and in Hy we want to observe leave and join events separately.

IV. Proposed Methodology for Determining Key Update Strategies

In this section, we will build a methodology that determines the best key update strategy based on the security and performance requirements of a network. We will first discuss key confidentiality and recovery from a key compromise, then continue with efficiency. After this discussion on what kind of requirements can be set and what kind of answers can be gathered, we will propose our methodology.

A. Optimizing Key Confidentiality

Using formal models, we can obtain the probability of being in a state where the key is compromised at a specific time instant. In this way, we also show that stochastic model checking can be efficiently used in determining the most appropriate key update threshold (*e.g.* time period) for an intended security level. As an example we may answer the question:

> *What is the probability that the key is compromised 6 months from now?*

by computing the transient probability of the key being compromised.

Obviously the answer depends on how often we are changing the key and we shall therefore pose the questions for different replacement strategies.

Naturally, we are also interested in the preservation of confidentiality in the *long-run* or *equilibrium*. In this case, the question to be asked is:

> *What is the probability of the key being compromised in the long run?*

Using stochastic model checking, we can obtain the steady-state probability of key compromise for different key update thresholds.

Surely the results of the two questions above shed light on how the key update strategy influences the risk of the key being compromised over time. But offhand they do not really give the solution to our problem. So let us ask a more direct question:

> *What is the best replacement strategy if it is acceptable that the probability of the key being compromised is 10%?*

We can answer such questions and return a solution set. However, if the solution set is not a singleton then we may need to narrow the solution set by other criteria as we define in the following sections.

We can define two metrics as we illustrated above: **average risk** and **maximum risk**.

B. Optimizing Mean Time To Recover

A security key may get compromised for several reasons, and eventually it will be updated. However, the time needed to recover from a compromised key needs to be optimized in such a way that the network is not without protection for a long time period. Thus, we are interested in the *mean time to recover* (MTTR) from a compromise of the key. Rather than formally defining it, we will illustrate by an example in Fig. 2.

Assume that we design a ZigBee network, where each device has a valid network key for secure communication.

Independent of the key update strategy, we can safely assume that the network key will be updated after some time. In fig. 2, we label the starting time of the network as `start` and a solid line following the start corresponds to the time where the network key is safe, i.e. not compromised. Then, we observe a thick cross on the solid line that represents a key compromise, we label this time point as C_1. After point C_1, the line is no more solid but dotted which means that the valid network key is compromised, therefore the communication might not be secure anymore. Note that any key comprising event after C_1 and before the soonest update won't make any changes because the key is already compromised. After some triggering event depending on the type of the key update strategy we will have a key update, labelled as `update`. At update points, compromised key is *revoked* and devices start using a new and *fresh* network key. The time periods we have labelled as R_1 and R_2 in Fig. 2, refer to the time period between the time point that a fresh key become compromised, and the time point that it is recovered by a key update a *recovery* time. Naturally, the summation of all these recovery times divided by the number of recoveries will give us the MTTR value.

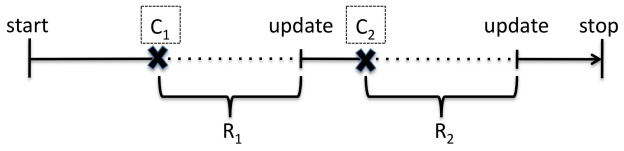

Fig. 2. Mean Time to Recover

Using our formal model and stochastic model checking we can compute the expected MTTR, up to a certain time bound. Which will result in the answer to this question:

> *What is the mean time to recover from a compromised key?*

It is important to have an optimum key update strategy and update threshold such that the MTTR value will be as small as possible. However, a too small MTTR value requires too high power consumption which might not be acceptable for networks that have devices with a limited power capacity.

No doubt, MTTR needs to be optimized in such a way that the network is not without protection for a long time period. If it is a short time then a potential malicious attacker will not have sufficient time to launch the attack whereas if it is a long time then the risk will be considerable higher for a successful attack. However, it is also important not to face a compromised key situation that is longer than a specified time bound. Thus, we are also interested in the risk that it takes more than a specific amount of time to recover from a compromised key. As an example we may answer the question:

> *What is the probability that the key has not been replaced 3 months after it was compromised?*

Thus, we define two metrics as we illustrated above: **MTTR** and **maximum recovery time**.

C. Optimizing Key Update Efficiency

One of the many important issues related to the key updates is the *power consumption*. Whenever we are replacing the key we are consuming power, and it is certainly interesting to compare the number of key updates performed over a certain period in different settings. Clearly, if we replace the key every 6 months then it is easy to see that we are updating the key twice a year. But how many times are we updating the key a year if we replace it whenever 10 devices have left the network? We can pose this question to our analysis:

> *How many key replacements can we expect to have performed a year from now?*

We succeed to compute the number of expected key updates in our analysis using the reward structures in our formal model. To demonstrate how this analysis can tighten the bounds of thresholds when looking for the best strategy, we will pose the question:

> *What is the best strategy if we on average can afford 1 key update per year?*

We also investigate the extent to which key updates triggered by the various threshold values are indeed needed, that is, will the key really be compromised when the updates occur. We shall classify the key updates as *useful* and *useless* updates. A *useful update* is a key update that is applied after a key gets compromised, therefore recovering the key. Similarly, a *useless update* is a key update that was not necessary because the key was not compromised. Naturally, we want the percentage of useless updates to be as small as possible because a key update is a costly action for ZigBee devices. In other words, the question we will try to answer is:

> *What is the best strategy if we expect to have at least 10% useful updates?*

We can define metrics as: **tolerated number of updates** and **percentage of efficient updates**.

D. Determining Optimum Method and Threshold

Obviously, it is not trivial to derive conclusion from the stochastic model checking results on the key update regarding confidentiality, recovery, and efficiency. For instance, the more efficient configuration is not the more secure one. To overcome such dilemmas, designers should decide on the priorities of the system and select appropriate security parameters. In this section, we propose a methodology for deciding the optimum key update strategy and key update threshold.

The first step is determining the characteristics of the application. Namely, we specify the application scenario, which depends on the type of the sensor devices and the objectives of the network. To choose the application scenario is fairly easy in ZigBee, because the ZigBee specification already has specialized application profiles that the designers and developers are supposed to make use of. This step will result in the values of parameters that we use in formal model.

Next comes the requirements about security and performance. Certainly, an extra key update causes an unwanted power consumption and therefore would drain the batteries

earlier than expected. After carefully specifying the requirements, we can exploit stochastic model checking on finding answers to our questions. At this point, picking a collection of different key update strategies and a set of threshold values would make everything easier. Model checking results will point us the appropriate threshold values if they exist, and of course different behaviours of different key update strategies.

After getting all the information above, it remains to evaluate all the solutions for all the requirements and conclude on the solution that satisfies all the requirements. Hence, a simple algorithm can be defined as:

I *Set the requirements using the metrics we defined*
II *Set the capabilities i.e. key update methods and thresholds*
III *Compute the solution set i.e. for each method find threshold values that satisfy all the requirements*
IV *If the solution set is empty or has multiple elements refine the requirements or capabilities and go to step III, else return the solution*

In Section VI, we present a case study that shows how we can get advice from stochastic model checking. In addition, it might be seen as a comparison between different key update methods and we observe how a method can beat another when different environment conditions and requirements exist.

V. Quantitative Verification

Stochastic model checking is an automatic technique for verification and performance analysis of stochastic systems. In the networking perspective, one application of this method is equivalent to a sufficient number of simulation runs, as it covers the full behaviour of the model and delivers provably correct results.

In the quantitative verification of all the key update methods considered, we did necessary abstraction as needed for formal verification. We formalized each method into a formal model, where *network* and *trust center* are modelled diligently. In the network module, we focus on events such as devices joining, leaving, and messaging. We consider that leaving devices and the messaging may cause key compromises. In the trust center module, we implement the key update strategy which is usually based on counting events and updating the key when a threshold is exceeded.

We designed our formal models to be stochastic, as we have events with stochastic delays such as device leave, device join, messaging etc. We developed continuous-time Markov chain (CTMC) models for each of the key update methods. The models we developed are both easy to customize, and compact enough to allow model checking. We specified the properties that we want to validate, in a stochastic temporal logic, CSL [8]. Thus, we are able to answer the questions by checking if the properties satisfied and getting quantitative answers. The stochastic model checking is fully automated using the well-established stochastic model checker PRISM [9].

All the models and properties that are relevant to this paper and sufficient to replicate the case study are available in [10].

VI. Case Study

In this case study, we focus on commercial building automation, specifically hotel security management. We consider a system where we use wireless sensors embedded in door locking cards which allow remote cancellation of cards, remote report of door lock status and door ajar alarms, etc.

The technical details of such a system in our perspective includes a maximum number of 50 devices in the network, aiming to keep the network at its maximum size as much as possible, replacing stolen or broken cards in two days on average, each device having non-stop operation for a year on average, each device sending one message a day on average, and the probability of a compromisation action by either devices leaving or by messages sent is one in ten thousand.

We use all the key update methods we mentioned in this paper, with the exception that the hybrid method was modified such that it does not include MB, so we denote it by Hy/mb.

To keep the case study tractable, we will set two solid requirements here:

R1: The key compromise probability should be below 5% in the long run.
R2: Maximum allowed number of key updates is 35 per year.

Key compromise in the long run: We start by computing the steady-state probabilities for a moderate set of thresholds. Actually, the aim here is to find the set of thresholds for each key update method that satisfies **R1**. As shown in Fig. 3, we assumed that the devices were capable of supporting a simple set of threshold values: {1,2,3,4,5} for TB, LB, JB, JLB; {500,1000,1500,2000,2500} for MB; and {(1-1-500),(2-2-1000),(3-3-1500),(4-4-2000),(5-5-2500)} for Hy/mb. To keep the graphical results more readable, the x-axis is shared by all the key update strategies such that for MB and Hy/mb it is used like an index to a threshold value. For example, the point 2.0 on the x-axis means threshold value of 2 for JLB, 1000 for MB, and (2-2-1000) for Hy/mb.

Even though it is evident in Fig. 3 which thresholds satisfy **R1**, we can still discuss the results to get more insight. In this sense, the interpretation of the first part of Fig. 3 is: *a*) TB has very high key compromise probability for the specified threshold set, *b*) JLB has the best results with only one exception where the threshold values are minimum (and number of key updates are boosted), *c*) the results of JB and LB are so close that the green colored line of JB is not visible under LB's blue line; however LB performs slightly better (approximately 0.015% difference), *d*) JB, LB, MB, and Hy/mb are in a competition where Hy/mb is winning and MB is losing as the threshold values get larger.

Number of key updates: In this criteria, we compute the expected number of key updates after 12 months. We present the results in the second part of Fig. 3. On the y-axis we have the expected number of updates, and naturally the number of key updates should decrease as the thresholds increase.

Interpretation of the second part of Fig. 3 is: *a*) JLB has high key update numbers for the specified threshold set, *b*) TB has minimum number of updates, *c*) again JB and LB perform very similar, and as expected this time JB performs slightly better, *d*) and again JB, LB, MB, and Hy/mb are in a competition but their results tend to converge as the threshold values get larger.

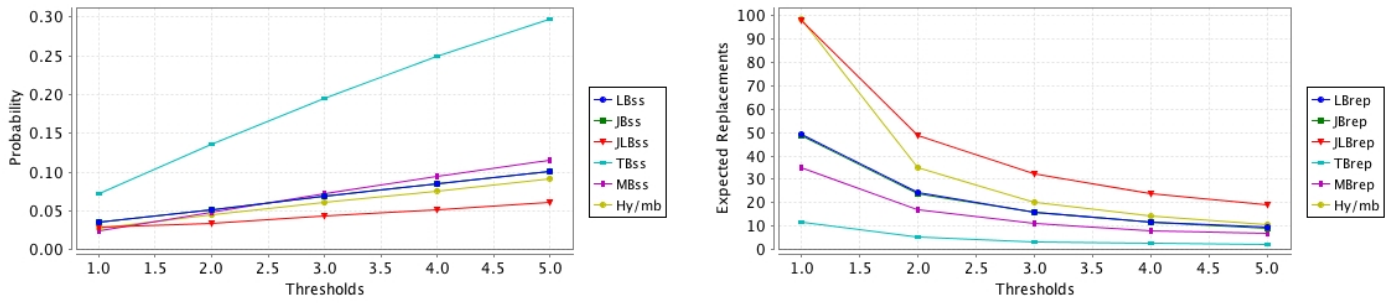

Fig. 3. Quantitative results, *Left*: Key compromise probability in the long run (R1), *Right*: Number of key updates in one year of time (R2)

Solution set: Now that we have our results above, we can compute the solution sets for each requirement:

S_{R1}={LB_1, JB_1, JLB_i, MB_j, Hy_j | $1 \leq i \leq 3,\ 1 \leq j \leq 2$}

S_{R2}={LB_i, JB_i, JLB_j, MB_i, Hy_j, TB_k | $2 \leq i \leq 5,\ 3 \leq j \leq 5,\ 1 \leq k \leq 5$}

Each key update strategy has a solution inside both R1 and R2, except TB which can not satisfy R1 at all. Then it is easy to find the intersection of the solution sets such that:

$S_{R1\&R2}$={JLB_3, MB_2}

As seen above, only two settings satisfied our requirements in the end: Join-Leave-based key update with a threshold of 3 devices, and Message-based key update with threshold of 1000 (notice that index 2 means threshold 1000 in MB). Now we can investigate further details, such as transient key compromisation probability. We know the steady-state probabilities of these two settings from Fig. 3 (0.044 for JLB_3, and 0.048 for MB_2), but we don't know the behaviour of those settings before they get stabilized, therefore transient probabilities will be very useful to know. In Fig. 4, we present the results for compromised key at monthly time instants for JLB_3 and MB_2. Fig. 4 supplies good insight and interesting results indeed, such as MB_2 has an alternating pattern where the maximum points increase for almost a year and then start to decrease; at the same time the minimum probabilities decrease for almost two years and then start to increase. In contrast with that, JLB_3 has a very consistent result and seems as the best choice considering that the deviation from the steady-state probability is almost negligible.

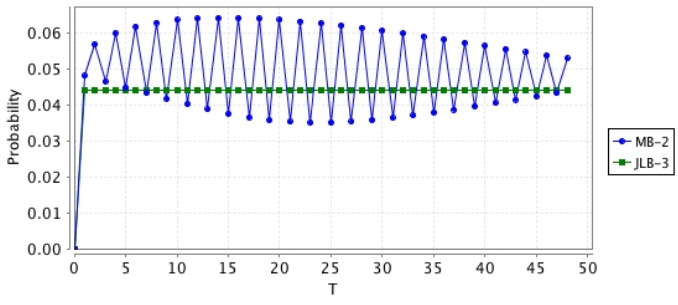

Fig. 4. Key compromise probability at *monthly* time instants

Although we are happy with the results of JLB_3 where we don't see jumps in the probability, we may also investigate more thoroughly. If we check daily time instants instead of monthly ones, the results verify that even in day level JLB_3 does not allow peaks, and still is very consistent (figure is omitted due to lack of space).

Result: Using our approach we conclude that the Join-Leave-based key update strategy with threshold value 3 is the optimum solution for this key update problem.

VII. Conclusion

In this paper, we discussed the problem of updating the cryptographic keys in wireless sensor networks. We have proposed new methods for key update, and a methodology for determining the optimum key update strategy in terms of confidentiality and power consumption. Furthermore, we have presented how stochastic model checking can be applied to the network security area and be useful in quantitative analysis of security. We believe that our study may assist sensor network applications in having higher security for lower costs.

References

[1] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security: A survey," in *Security in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds.)*. CRC Press, 2006.

[2] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication, and Software Systems*, ser. LNCS, M. Bernardo and J. Hillston, Eds., vol. 4486. Springer, 2007, pp. 220–270.

[3] *ZigBee Specification*, ZigBee Alliance Std. 053 474r17, 2008.

[4] E. Yüksel, H. R. Nielson, and F. Nielson, "Zigbee-2007 security essentials," in *Proceedings of the 13th Nordic Workshop on Secure IT Systems*, 2008, pp. 65–82.

[5] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key management for multicast: Issues and architectures," IETF, RFC 2627, Jun. 1999.

[6] J. V. Misic, F. Amini, and M. Khan, "Performance implications of periodic key exchanges and packet integrity overhead in an 802.15.4 beacon enabled cluster," *Int'l Journal of Sensor Networks*, vol. 3, no. 1, pp. 33–42, 2008.

[7] E. Yüksel, H. R. Nielson, F. Nielson, M. Fruth, and M. Kwiatkowska, "Optimizing zigbee security using stochastic model checking," Technical University of Denmark, Tech. Rep. IMM-Technical Report-2010-08, 2010.

[8] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton, "Verifying continuous time markov chains," in *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1996, pp. 269–276.

[9] The PRISM model checker website. [Online]. Available: http://www.prismmodelchecker.org

[10] The formal models for the key update methods. [Online]. Available: http://www2.imm.dtu.dk/people/ey/models/