A Forward & Backward Secure Key Management in Wireless Sensor Networks for PCS/SCADA

Hani Alzaid[1], DongGook Park[2], Juan González Nieto[1], Colin Boyd[1], and Ernest Foo[1]

[1] Information Security Institute, Queensland University of Technology
Brisbane QLD 4000 Australia
{h.alzaid, juanma, c.boyd, e.foo}@isi.qut.edu.au
[2] School of Information Technology, Sunchon University, Korea
dgpark6@sunchon.ac.kr

Abstract. Process Control Systems (PCSs) or Supervisory Control and Data Acquisition (SCADA) systems have recently been added to the already wide collection of wireless sensor networks applications. The PCS/ SCADA environment is somewhat more amenable to the use of heavy cryptographic mechanisms such as public key cryptography than other sensor application environments. The sensor nodes in the environment, however, are still open to devastating attacks such as node capture, which makes designing a secure key management challenging. In this paper, a key management scheme is proposed to defeat node capture attack by offering both forward and backward secrecies. Our scheme overcomes the pitfalls which Nilsson et al.'s scheme suffers from, and is not more expensive than their scheme.

Key words: wireless sensor network, forward and backward secrecy, key management, process control systems, supervisory control and data acquisition.

1 Introduction

Process Control Systems (PCSs) or Supervisory Control and Data Acquisition (SCADA) systems are used to monitor and control a plant or equipment in industries such as energy, oil and gas refining and transportation. These systems encompass the transfer of data between the network manager and a number of Remote Terminal Units (RTUs), sensor nodes, etc. A SCADA system gathers critical information (such as where a leak in a pipeline has occurred) and then transfers this information back to the network manager. The network manager is responsible for alerting the home station about the leak and carrying out necessary analysis such as determining whether the leak is critical or not.

The owners and operators of SCADA systems aim to increase the monitoring sensitivity of their systems and reduce the day to day running cost wherever it is possible. Due to the intelligent monitoring capabilities of the Wireless Sensor Networks (WSNs), integration between SCADA and WSNs can be one way to achieve these aims. WSNs facilitate the monitoring process by performing specific tasks such as sensing physical phenomena at a remote field and then reporting them back to the network manager. They can form the "eyes and ears" of SCADA systems. Nodes, which are capable of performing functions such as gas detection and temperature sensing, provide information that can tell an experienced operator how well oil/gas pipelines are performing.

Roman et al. highlighted the role that WSNs can play in SCADA [14]. They argued that WSNs can aid SCADA's functionalities by providing monitoring, alerts, and information on demand. However, vulnerabilities related to WSNs can be introduced to SCADA. One of those potential vulnerabilities is the security compromise of sensor nodes given the lack of tamper resistance packaging [4]. An adversary can gain control of one or more sensor nodes and readily access sensitive information such as keys or passwords. The adversary therefore can easily get access to the plain text of the encrypted messages that are routed through the controlled nodes – this compromises the data confidentiality. The adversary may also inject their own commodity nodes into the network by fooling nodes so that they believe that these commodity nodes are legitimate members of the network. Another adversary activity is launching a selective forwarding attack where the node, that is under the control of the adversary, selectively drops legitimate packets in order to affect the overall performance of the system [5].

In this paper, we focus on strengthening the security level at the weakest component of the SCADA system which exists in remote fields [1]. The remote field has the weakest physical security requirements and consists of substations and intelligent electronic devices such as sensors (will be discussed in a later section). We propose a new key management protocol that updates the shared symmetric key between the network manager and a sensor node or between the network manager and a group of sensor nodes.

The rest of this paper is organized as follows. Section 2 provides an overview of SCADA systems. Section 3 explains the different types of the adversarial model. Section 4 discusses some of the related work. Section 5 explains the proposed key management protocol. Finally, the paper is concluded in Section 6.

2 SCADA

To best understand the added value of the proposed scheme, some understanding of SCADA is in order. Today's SCADA systems (the third generation) are a combination of legacy and modern technology [9]. It has become an open system architecture rather than a vendor controlled architecture as in the second generation of SCADA. It uses open standards and protocols which facilitate distribution of the functionalities of SCADA. We refer the readers interested in the differences between these generations to the paper by McClanahan [9]. Figure 1 shows a simplified SCADA system architecture which is composed of the following components:

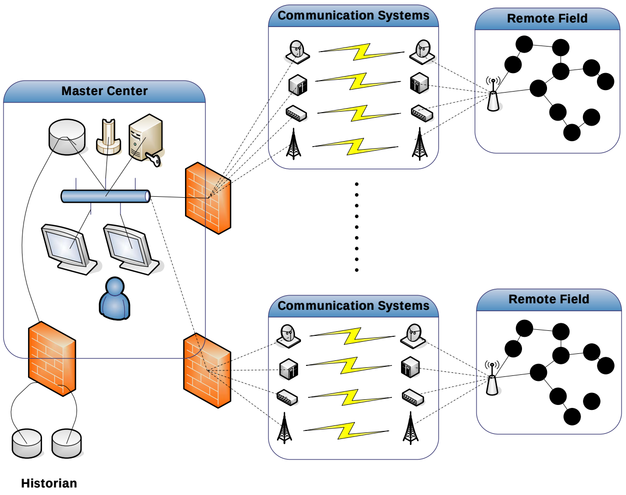

Fig. 1. The simplified version of PCS/SCADA

Master Center The master center component contains the network manager, human machine interaction, database storage, processing server, etc. It has the highest physical security level compared to other components. Generally speaking, it receives monitoring information from remote fields (through the communication system component), processes it, and then makes decisions.

Historian The historian is a backup for the SCADA system data which is often located in a separate subnet different to the one where the master center component exists. The master center component is able to access the historian in order to backup the data of the SCADA system.

Remote Fields The remote fields are composed of substations (gateways) and intelligent electronic devices (IEDs) [1] which can be physically distant from the SCADA master center and in many cases are not physically secured due to the largeness or remoteness of the coverage area. The substation connects IEDs with the master center component through the communication component. It has a high degree of complexity and might have better physical security than IEDs. The IEDs can be sensor nodes, remote terminal units, or relays to name a few.

Communication System The communication systems are responsible for transferring monitored data (control data) from remote field components (master center) to master center component (remote field components). This communication can be done via fiber optics, radio, satellite, etc.

3 Adversary Model and Security Concerns

When designing a key management protocol for WSNs, the most challenging and unique security threat would be *node capture*. With limited resources in sensor nodes, defeating this type of threat is very hard. Node capture will translate into compromise of all the credentials stored in the sensor node. Furthermore, the adversary can compromise all the *software codes* installed within the sensor node, especially random number generation functions. For example, he can modify the codes or replace them with his own codes to mislead functions related to SCADA/PCS, use a fixed number for random numbers for input to security protocols, or launch a selective forwarding attack. However, the computation power of the adversary falls short of compromising the network manager and gateways which have reasonable physical security. Their physical security increases in proportion to the importance of the domain where a SCADA/PCS is deployed.

Our purpose in this paper is to design a key management scheme which is resilient to node capture: i.e., a scheme that enables sensor nodes to recover its secure status even after they have been captured and then released back. Consequently, we are interested in what the adversary can do both when a node is captured, and after it is released back. Key disclosure is technically simple [4]; what else should be done by the adversary to keep control of the node after he put it back to the field? He will hope that the node uses values of his choice for all cryptographic keys or keying materials. For this purpose, he may try to modify software components (especially the random number generation part), and monitor all or part of the subsequent key update messages. In this regard, we use the following criteria to classify the adversaries.

- The adversary can read and modify all the software codes and configurations, including secret keys, installed in the sensor node.
- The adversary can carry out seamless monitoring of all the subsequent key update protocol exchanges.

According to the above two criteria, we divide the adversaries into four distinct types as shown in Figure 2. Type I is the weakest adversary: neither seamless monitoring nor software compromise; Type IV is the strongest: seamless monitoring and software compromise. Type IV is so much powerful that it is unlikely to devise any practical cryptographic countermeasure for WSNs. The use of tamper-proof technology will be needed to cope with this type of adversary, but it is outside the scope of this paper. Our goal in the paper is to have a new key management scheme which is resilient to all the other three types of attackers only with cryptographic countermeasures.

One interesting point here is that the assumption of software modification is equivalent to that of software-based random number generation, in terms of their consequence in the context of cryptographic protocols. Software algorithm-based random number generation does not give true random numbers, which can only be obtained from a strong physical source of randomness. One consequence of this equivalence is that it makes no sense to use expensive tamper-proof technologies while true random number generation not used. Put a different way, we do not

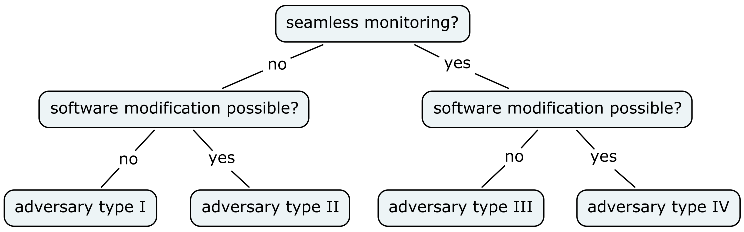

Fig. 2. Classification of Adversaries. "Seamless monitoring" means the adversary keeps monitoring *every* subsequent key update message after compromising a sensor node; "software modification" includes alteration of any software installed in the node, especially the random number generator.

have to bother with true random number generation when software modification is assumed to be an easy work for the adversary.

Having identified different types of adversaries, we have the following concerns with regard to node capture and the consequent disclosure of all the internal data of the captured node:

- Past key secrecy: The past keys should not be compromised.
- Future key secrecy: The future keys should not be compromised.

The requirement of resilience to node capture rules out the use of any long-term keys; the keys must change or evolve continuously over time, with old prior keys deleted securely. In other words, we require a *key evolution* scheme in order to achieve past/future key secrecy against the threat of node capture.

Terminology. To the best of our knowledge, the terms "past/future key secrecy" have never been used in previous literature. Similar terminology include "(perfect) forward secrecy" and "backward secrecy", which has always been quite confusing. The term "(perfect) forward secrecy" goes back to Günther [3]. The original term assumes a long-term key and session keys established by the key, and means that the current session key is not compromised by the "future" (thus, the expression "forward") exposure of the long-term key. This terminology, somehow, seems to have got a slightly different usage in the context of group key communication; it concerns about the contamination of a group key at a particular time by the compromise of an older/newer group key. The inherent ambiguity has brought a twin terminology: "backward secrecy". Some authors choose the term "backward secrecy" to mean "forward secrecy" called by other authors, and vice versa. To avoid all this confusion, we will use a new more concrete expression: "past/future key secrecy".

The notation to be used in the rest of the paper can be found in Table 1.

4 Related Work

There are several papers dealing with key management designs for SCADA systems such as [2,12]. However, these designs either use heavy cryptographic mech-

Table 1. Notations for the proposed scheme

Name	Description
M	Network manager.
N	Sensor node.
K_{MN}	Shared pairwise key between M and N.
s_0, t_0	Pre-installed global secret data in every N.
K_G^i	The i-th group key ($i \geq 0$).
r_X	Random nonce chosen by entity X.
(K_M^{-1}, K_M)	Asymmetric key pair of network manager.
$\{m\}_K$	Encryption of message m under the key K.
$h(\cdot)$	A cryptographic hash function.
$\mathrm{MAC}_K(m)$	A message authentication code function on m using the key K.

anisms, which do not suit resource constrained devices, or do not consider the integration of WSNs within SCADA.

To the best of our knowledge, the only existing key management in the wireless control environment, that considers the integration between SCADA/PCS and WSNs, has been proposed by Nilsson et al. [10]. They designed two key update protocols: the first one updates the pairwise symmetric key between the network manager M and a sensor node N (as described in Protocol 2) while the other scheme updates the global or group key among M and the whole group G of sensor nodes (as described in Protocol 1). They claimed that the protocols provide both forward and backward secrecy (or in our newly defined terminology, they provide both past and future key secrecy). It is unfortunately not the case.

Protocol 1 Group key update protocol from [10]

M: generates a new group key $K_G^{'}$ and a random number r_M

1. $M \rightarrow N$: $\{K_G^{'}, r_M\}_{K_{MN}}$
2. $M \leftarrow N$: $MAC_{K_G^{'}}(N, r_M)$

Protocol 2 Pairwise key update protocol from [10]

N: generates a random number r_N

1. $M \leftarrow N$: $\{r_N\}_{K_M}$, $MAC_{K_{MN}}(\{r_N\}_{K_M})$

M, N: compute the new pairwise key $K_{MN}^{'} = h(K_{MN}, r_N)$

To initiate the group key update protocol, M generates a new group key, $K_G^{'}$, randomly. It then encrypts it with another random number, r_M, and sends it over the network to the target group. No node in the group has any clue whether the received key is fresh or not. In other words, the freshness property, from the viewpoint of N does not hold since the two values (the new group key $K_G^{'}$ and the random number r_M) are random values chosen by M. It is both impractical and insecure for each sensor node to maintain a list of keys that have been used. Thus, an external adversary will be able to record a rekeying message and then re-inject it into the network, which leads to updating the group key with an old key. Consequently, the group enters a key mismatch phase where the key version that the group of sensors uses and what M has are different.

One good security practice is to minimize the damage caused by a compromised node. However, the authors did not consider common attacks in WSNs that an adversary is capable of launching attacks such as selective forwarding [5] or node compromise [4]. If a single sensor node has the ability to affect the operation of a good number of sensor nodes, then the adversary will try to compromise that node. For example, if an adversary compromised a sensor node (say, node N_b) in a multi-hop path, then it would be able to enforce all other nodes downstream to enter the key mismatch phase. The adversary simply drops the rekeying message from M for the group key, and then use the new group key to calculate MACs on their identities and the received nonce, which results in a successful impersonation attack. We can easily fix the problem by replacing the MAC data with another one: e.g., $MAC_{K_{MN}}(K_G^{'}, r_M)$.

Moreover, to initiate the pairwise key update protocol, N generates a random number, r_N, and encrypts it with K_M. It subsequently computes the MAC on the encryption result and sends this MAC and the encryption result over the network to M. The new pairwise key can be calculated, at the sender N and at the receiver M, by hashing r_N with the previous pairwise key. This means that the new pairwise key is always determined by N. The adversary consequently is able to know all the future keys once he compromised N. A closer look at the protocols, Protocol 1 and Protocol 2, reveals more serious defects of them.

- Defect I. The whole value of the new group key are directly carried by the protocol messages, encrypted under the pairwise key K_{MN}. The consequence is that compromise of the pairwise key for just one node leads to compromise of the group key for the whole group. This is a more serious problem than it might appear, because the pairwise key compromise does not necessarily require node capture.
- Defect II. The value of the new pairwise key $K_{MN}^{'}$ is only determined by the sensor node. When the adversary of Type II or IV (he can compromise the key generation codes stored in the node) captures the node, all the future pairwise keys for the node can be pre-determined by the adversaries. Namely, physical compromise of the node immediately leads to compromise of all the future pairwise keys if the adversary can modify the codes installed in the node. This, in turn, leads to compromise of all the future group keys as well because, as mentioned in Defect I, the group key is delivered encrypted under the pairwise key. Hence, contrary to their claim, the scheme does not provide

"future key secrecy", against node compromise, for either the pairwise key or the group key.
- Defect III. Although not explicitly shown in the protocol descriptions above, the key input r_N for the new pairwise key K'_{MN} is not really random in their scheme; it is in fact a function of a pre-installed secret key and a counter value stored in the node. This means that, when the node is captured and all the installed data including keys exposed to the adversary, all the past pairwise keys as well as the future keys can immediately be computed even without recording a single key update message! In fact, this disaster is not just because of Defect III, but also due to Defect II. Note that, due to Defect III combined with Defect II, the adversary does not have to modify the node's software at all in order to extract all the past and future pairwise keys. Hence no minimum level of past or future key secrecy against node compromise in their scheme. Moreover, the adversary can extract any group key in the past or future if he has got the records of the corresponding group key update message. Note also that, for this, "seamless" monitoring is not needed by the adversary. What does this mean? The scheme is, in terms of either kind of key, neither forward nor backward secure against node compromise for all the types of adversary I, II, III and IV (see Figure 2).

As for past key secrecy, we note two proposed schemes in the WSN context: Klonowski et al. [6] and Mauw et al. [8]. Both schemes use hash functions in order to achieve key evolution. Both schemes, however, are intended to be used not for group key update but for updating *pairwise* keys for node-to-node [6,13] or node-to-base station communication [8].

On the other hand, as for future key secrecy, Mauw et al.'s protocol does not provide this property. The protocol is based on a hash chain scheme originally proposed for RFID security [11]. In RFID environments, protecting secret tag information from tampering in the future is a big concern while it does not seem to be such a prime concern in WSNs. This is because it is more authentication and integrity than privacy that really matters in WSNs, especially SCADA/PCS. Hence, future key secrecy is more valued than past key secrecy. On the other hand, the protocol proposed by Klonowski provides future key secrecy in a "weak" sense; namely, it will be computationally hard for the adversary to compute a future key from the current compromised key if he fails to record, say ten, subsequent evolution steps [13].

5 The Proposed Scheme

Devising a key management for WSNs is not trivial and in particular may not be successfully accomplished by simple adaptation of security solutions designed for wired networks. This is because of the limited resources such as limited energy lifetime, slow computation, small memory, and limited communication capabilities which exist in WSNs [16,17]. In this section, we describe a key management scheme which secures communication between remote fields (where the WSN resides) and the master center (where the network manager resides) by considering vulnerabilities that are associated with WSNs .

5.1 Key Management Protocols

This paper focuses on updating two types of keys, which are the group key and the pairwise key, in the wireless process control environments. A pairwise key is shared between the network manager M and each sensor node N, while the group key is shared among M and the whole group of sensor nodes.

Group Key Update Protocol Our solution for group key rekeying also exploits the idea of key evolution using a hash chain in order to achieve past key secrecy. The protocol uses a hash chain, $h^i(s_0)$, where s_0 is a pre-installed key component at the pre-deployment phase and $i \geq 0$ denotes the index for key update phases.

As for future key secrecy, we use the *reverse hash chain* technique, which was first introduced by Lamport [7]. The network manager prepares in advance a hash chain of length n, starting from a random seed t_{n-1} and ending with the final value t_0:

$$t_{n-1},\, t_{n-2} := h(t_{n-1}),\, t_{n-3} := h(t_{n-2}),\, \ldots\,,\, t_1 := h(t_2),\, t_0 := h(t_1)\,.$$

For reasons of convenience which will become clearer shortly, we write $h^{-i}(t_0)$ instead of t_i although h is not an invertible function and $h^{-1}(x)$ can only mean the set of all preimages of x in a strict sense. Roughly speaking, $h^{-i}(t_0)$ is the i-th preimage of t_0 in the reverse hash chain. The secret data, t_0, will be pre-installed into sensor nodes together with another key component s_0.

Now, with two secret key components s_0 and t_0 pre-installed within all sensor nodes, using Protocol 3, the group key K_G^i evolves as follows:

$$K_G^i = h^i(s_0) \,\oplus\, h^{-i}(t_0)\,,\ i \geq 0\,,$$

where we define $h^0(s_0) = s_0$ and $h^0(t_0) = t_0$. Figure 3 explains the key evolution in the protocol.

Protocol 3 The protocol for group key update

1. $M \rightarrow N$: i, $\{h^{-i}(t_0)\}_{K_{MN}}$
2. $M \leftarrow N$: $h_{K_{MN}}(K_G^i)$

M, N: update the value of the group key (i.e., $K_G^i = h^i(s_0) \,\oplus\, h^{-i}(t_0)$).

Any sensor node can easily compute the i-th hash image $h^i(s_0)$ from $h^{i-1}(s_0)$ while only the network manager knows the value of the i-th preimage $h^{-i}(t_0)$.

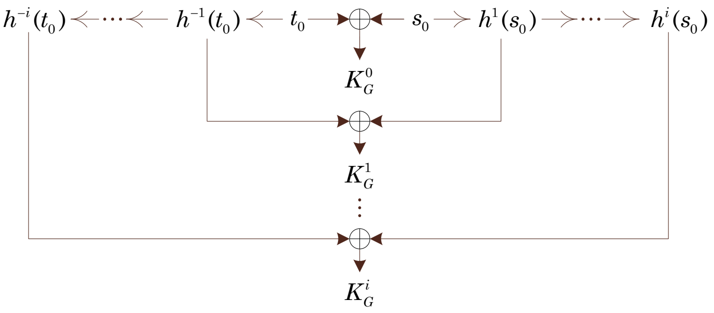

Fig. 3. Key evolution in the proposed protocol

Thus, it is only the network manager who can release the preimage into the sensor field. As a consequence, the first message in the protocol provides the sensor node with a weak form of signature from the network manager: the message could have been generated only by the network manager, not by any sensor nodes including the node itself. The check of the preimage (i.e., $h(h^{-i}(t_0)) = h^{-(i-1)}(t_0)$) also makes sure that the key update message is fresh.

After the i-th key update, the sensor node stores the index i and the secret data: $h^i(s_0)$, $h^{-i}(t_0)$ and K_G^i. Considering the highly lossy communication environment of sensor networks, the sensor node may sometimes fall behind the group key update schedule. The sensor node, however, will soon be able to catch up at the next rekeying: it can compute the correct value of the new group key simply by checking the difference of two index values – the received and the stored – and applying the corresponding number of hash operations.

Now let's assume that the adversary has somehow extracted the current value of the group key, K_G^i. However, he cannot extract from this the previous key K_G^{i-1} because he cannot compute the value of $h^{i-1}(s_0)$. Note that this holds even when the adversary has recorded all the previous key update messages, and compromised all the previous manager-to-node pairwise keys. In fact, the node capturing and extracting all the stored secret data does not surrender the past group key to the adversary. This is because the previous values for $h^i(s_0)$ were never exchanged over the air, and were deleted after group key computation. Hence we can say that the protocol provides past key secrecy for any kind of compromise: group key compromise, pairwise key compromise, and the compromise of the node itself.

The protocol also provides future key secrecy in the sense that the adversary, just with knowledge of the current group key K_G^i, cannot predict the next group key K_G^{i+1}. The computation of K_G^{i+1} requires knowledge of $h^{-(i+1)}(t_0)$, which has not yet been exchanged. In the next step of the key update, the adversary, without knowledge of the pairwise key K_{MN}, will not be able to obtain the value of $h^{-(i+1)}(t_0)$ from the protocol message. In fact, the pairwise key compromise alone does not lead to the future group key compromise; it will only happen when the adversary captures a sensor node, thereby extracting the hidden component

$h^i(s_0)$. Hence, the protocol satisfies future key secrecy in the face of group key and/or pairwise key compromise; simple delivery of the encrypted value of the new group key, as in [10], cannot provide this kind of resilience. The protocol will fail to provide future key secrecy only when the node is physically captured. Even in the case of capture, the adversary should listen to the key update message to extract the future group key. Furthermore, when the pairwise key is updated, any adversary of type I, II, or III will not be able to have any knowledge of the new pairwise key. This, in turn, leads to the adversary's failure to have any knowledge of the new group key established using the new pairwise key. Hence, we achieve the future group key secrecy even after node capture, as far as the adversary has no ability to modify the software codes stored in the node.

The protocol uses the pairwise key K_{MN} to encrypt the i-th preimage $h^{-i}(t_0)$ in the first message, and also to provide key confirmation by computing keyed hash of the new group key. This is in order to rule out any compromised or suspicious sensor nodes from group key update.

Our protocol, however, has one limitation: it is vulnerable to a kind of collusion attack. Assume that a sensor node was captured at a key update phase i, and another node was subsequently captured again at the phase $i + 10$. Then, the adversary can extract all the group keys for the phases i to $i+10$. Of course, this compromise is limited to the past keys, not the future keys. We call this attack "*sandwich attack*" which will be considered in our future work.

Protocol 4 The protocol for pairwise key update

1. $M \rightarrow N$: i, $\{h^{-i}(t_0), g^{r_M}\}_{K_G^{i-1}}$ # *broadcast* message
2. $M \leftarrow N$: $\{g^{r_N}\}_{K_{MN}}$, $h_{K_{MN}}(g^{r_M}, g^{r_N})$

N: keeps the hashed value of the current pairwise key: $K^1_{MN} = h(K_{MN})$.

M, N: increment the group key index from $i-1$ to i, and update the values of the pairwise key (i.e., $K_{MN} := g^{r_M r_N}$) and the group key (i.e., to K^i_G).

Pairwise Key Update Protocol Protocol 4 shows the rekeying protocol for the pairwise key shared between the network manager and the sensor node. This protocol is based on Diffie-Hellman protocol which has recently become not only feasible on resource constrained nodes, but attractive for WSNs [15]. The network manager M first generates a secret random number r_M, and computes the Diffie-Hellman component g^{r_M}. It then *broadcasts* Message 1, which includes the index i of the next group key, and ciphertexts of the next group key component $h^{-i}(t_0)$ and a Diffie-Hellman component g^{r_M}, encrypted under the current group key, K_G^{i-1}.

The inclusion of the group key index i in the first message enables each sensor node to check if it keeps the current value of the group key; if not, the node can request the network manager to send the latest key component $h^{-i}(t_0)$. Thus, the group key rekeying protocol exchange as described in Protocol 3 can be inserted between Messages 1 and 2 of the protocol in the case of group key index mismatch.

After retrieving the plaintext of Message 1 using the group key, the node checks the preimage if $h(h^{-i}(t_0)) = h^{-(i-1)}(t_0)$. This check provides evidence for the node that M has really started the pairwise key update session. Considering that Message 1 is a broadcast message encrypted using the "group" key, it would be simply impossible to achieve this evidence without using the preimage as used here. Of course, using digital signature/verification is a different story.

Now the node constructs the second message of the protocol: it generates its own Diffie-Hellman component g^{r_N}, encrypts it, and generates the keyed hash of both Diffie-Hellman components under the current pairwise key K_{MN}. After sending the message to M, the node computes the new group key, $K_G^i = h^i(s_0) \oplus h^{-i}(t_0)$, increments the group key index from $i-1$ to i, and computes the Diffie-Hellman key $g^{r_M r_N}$ to be used as the new pairwise key, while keeping the hash $h(K_{MN})$ of the old pairwise key and safely deleting the old key.

On receiving Message 2, M decrypts g^{r_N}, and verifies the keyed hash from N. The inclusion of g^{r_M} and g^{r_N} in the hash provides M with confidence about the freshness and authenticity, respectively, of the message.

Use of Diffie-Hellman key agreement for the pairwise key update provides the past and future pairwise key secrecy; the key inputs are temporary randoms, and thus no relation to the previous or next key inputs. Even after node compromise, if the attacker is not able to modify the software codes in the node (i.e., the adversary of type I or III), or if he fails to record the key update messages (i.e., the adversary of type I or II), the node will escape from the control of the adversary to recover the secure status. Thus, our scheme satisfies past pairwise key secrecy for all the adversary types, and future pairwise key secrecy for any adversary type except type IV, even against node capture and its compromise.

Impersonation attack. If the adversary is in full control of a compromised node, in which he installed his own malicious attacking software, then the adversary's node can still impersonate M to some other victim node, succeeding in causing the victim to receive a fake Diffie-Hellman component, say g^x. But the attack is limited to that. The attacking node has only two options when receiving Message 2 of the victim node: (1) forward the message verbatim to M, or (2) cut out the message. In the former case, M will get not the expected hash $h_{K_{MN}}(g^{r_M}, g^{r_N})$ but a strange one $h_{K_{MN}}(g^x, g^{r_N})$. In the latter case, M will see no response from N. In both cases, M will issue Message 1 again through the *unicast* channel to N, which will finally lead to key agreement between M and N.

Delivery failure management. The delivery failure in the WSNs will lead to key mismatches of group keys and/or pairwise keys. With no long term key available in our key update protocols, key mismatch is a big concern and should be handled carefully. Simple retransmission of the protocol messages is not a solution; it may open the door to replay attacks. Moreover, it may require the sensor node to go back to the old key even after it has successfully updated the pairwise key. Consequentially, the node must keep two keys at the same time: the old key and the new updated key.

Our solution is to use key evolution here again. With no response from the node N, the manager M initiates Protocol 5 over the unicast channel to N.

Protocol 5 The protocol to handle delivery failure

1. $M \rightarrow N$: i, j, $\{h^{-i}(t_0), g^{r_M}\}_{K^j_{MN}}$ # *unicast* message
2. $M \leftarrow N$: $\{g^{r_N}\}_{K^j_{MN}}$, $h_{K^j_{MN}}(g^{r_M}, g^{r_N})$

M, N: update the values of the pairwise key (i.e., $K_{MN} := g^{r_M r_N}$)
N: increments the indice i and j, and updates the values of the pairwise key (i.e., $K_{MN} := g^{r_M r_N}$) and the group key (i.e., to K^i_G), and then keeps the hashed value of old key: $K^{j+1}_{MN} := h(K^j_{MN})$

Here, $K^j_{MN} = h^j(K_{MN})$ is a hashed copy of the current key from M's viewpoint. For the time of the first protocol run, the index j is set to 1; it will be incremented by one whenever the protocol is retried. On receipt of Message 1 over the unicast channel, the sensor node N compares the received group key indice i, j with the stored indice i', j', and executes the required action as follows:

- Case 1: $i = i'$ and $j \geq j'$. For simplicity, consider the case $j = j' = 1$. The pairwise key update protocol (Protocol 4) has just been run, but the reply message of the protocol failed to arrive at M. The node N has been keeping the hashed copy $K^1_{MN} = h(K_{MN})$ of the old pairwise key, which is applied to the ciphertext for Message 1 of Protocol 5. The retrieved value of $h^{-i}(t_0)$ ensures the authenticity of the message; the entity other than N, in possession of $h^{-i}(t_0)$ and K^1_{MN}, should be M. The node decrypts the encrypted part of Message 1 using K^1_{MN}. Then, N follows exactly the same step as in 4 except that it uses the hash of the old pairwise key instead of the current pairwise key. At the end of the protocol run, N will end up with a new pairwise key, and the hash of K^1_{MN}, i.e., K^2_{MN}; now $j = 2$. The current pairwise key is simply deleted. One or more failure again will be followed by reinitiation of the protocol by M with j incremented. It could also happen that Message 1 itself fails to arrive at N, and subsequently M retries the protocol. This will lead to the case $j > j'$.
- Case 2: $i = i'$ and $j < j'$. This cannot happen; otherwise it is simply a bogus message from another sensor node. N should ignore Message 1.
- Case 3: $i > i'$. This happens when the node N has never been involved in the pairwise key update protocol due to delivery failure of Message 1 of Protocol 4. In this case, N applies the hash to the current pairwise key j times, and uses the resulting value as the description key for Message 1.
- Case 4: $i < i'$. This is another case of replay attack. N should ignore Message 1.

Now, the old key does not need to be kept just for handling key mismatch; Protocol 5 does not come with any breach of security.

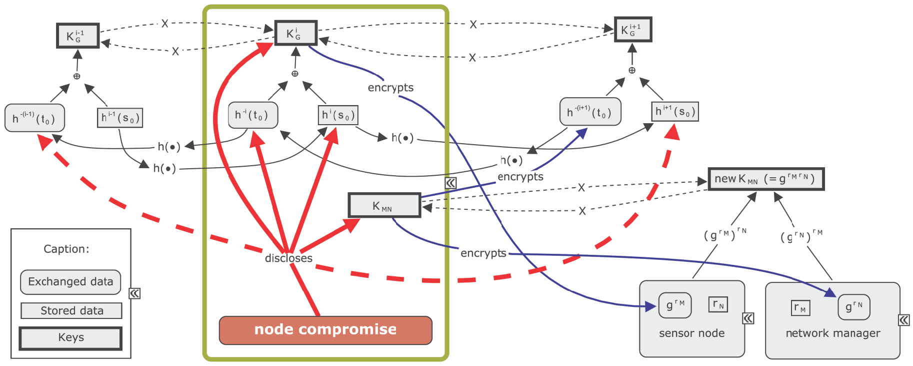

Fig. 4. Relations between keys and keying materials and the significance of node compromise

5.2 Putting it all together

In our scheme, the pairwise key is used for secure delivery of the group key update information in Protocol 3; the group key, in turn, encrypts the Diffie-Hellman components to establish a new pairwise key in Protocol 4. This combination helps the sensor networks to recover its security quickly after some sensor nodes are captured and their keys are compromised.

Figure 4 illustrates how all the keys and keying data are related to each other as they evolve over time. Note that no keys are delivered over the air; only their keying materials, such as $h^{-i}(t_0)$, are exchanged or even never exchanged over the air (e.g., $h^i(s_0)$). Thus, unlike the scheme of Nilsson et al. (see Defect I in Section 4), the pairwise key compromise alone does not lead to the group key compromise, and vice versa.

Using the inverse hash chain as well as the hash chain, we achieve both past/future group key secrecy at the same time; furthermore the group key update message provides an inherent message authenticity.

Both M and N contribute their Diffie-Hellman inputs to the computation of the new pairwise key, and thus the adversary can not determine the future values of the pairwise key even after node capture and the resulting compromise of the built-in software, which was not the case in the scheme of Nilsson et al. (see Defect II in Section 4).

Carefully designed with node capture in mind, our scheme does not surrender all the key components required to retrieve the past/future group/pairwise keys. Only the adversary equipped with seamless monitoring and software compromise (i.e., the type IV adversary) can keep the control of the once-captured node.

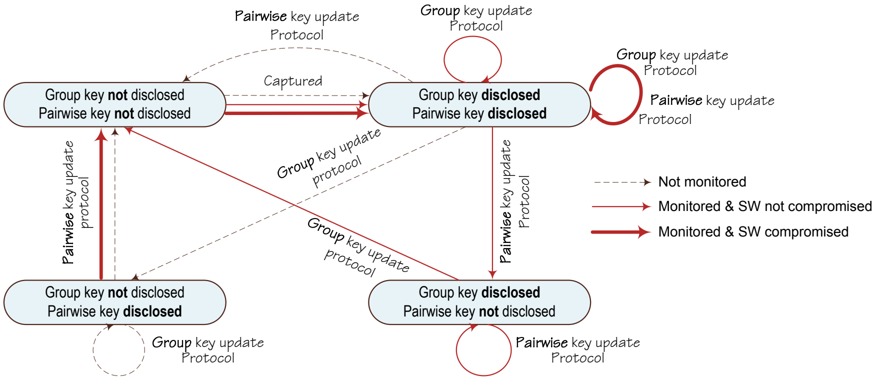
Pairwise key update Protocol
Group key update Protocol
Group key update Protocol
Pairwise key update Protocol
Captured
Group key not disclosed Pairwise key not disclosed
Group key disclosed Pairwise key disclosed
Group key update protocol
Pairwise key update protocol
Pairwise key update Protocol
Group key update Protocol
Group key not disclosed Pairwise key disclosed
Group key disclosed Pairwise key not disclosed
Group key update Protocol
Pairwise key update Protocol
Not monitored
Monitored & SW not compromised
Monitored & SW compromised

Fig. 5. State diagram of key disclosure

Figure 5 shows how the node recovers its secure state with the help of the key update protocols, after it has been captured and all the keys in it are compromised. Without seamless monitoring (i.e., adversary types I and II), the adversary will soon lose all the control of the keys. Even with adversary type III (i.e., seamless monitoring but no software compromise), the node will eventually recover the secrecy of both keys. Only for adversary type IV (i.e., both seamless monitoring and software compromise), there is no path available back to the original secure state. We argue that a non cryptographic countermeasure such as tamper-proof technology is additionally required to fight against the strongest adversary of type IV.

6 Conclusion

Wireless sensor networks (WSNs) has brought a devastating security threat: node capture. The threat is so powerful that almost all existing key management protocols are just helpless because it overthrows the fundamental assumption for cryptographic system design: long term secret keys are securely stored. This is why so called forward secrecy and backward secrecy are required in cryptographic key management protocols for WSNs. Both terminologies are rather misleading and confusing, and so we propose more proper ones: future key secrecy and past key secrecy.

Nilsson et al. [10] have recently proposed a key management scheme for WSN applications in PCS/SCADA environments, which was incorrectly claimed to provide future and past key secrecies. Some proposals (only for pairwise key update) provide past key secrecy, but not future key secrecy [8,6].

We noticed that any cryptographic countermeasure alone cannot prevent the most powerful adversary in the WSN context; he can capture a node to extract all confidential data, modify any built-in codes, and seamlessly monitor to keep control of the node. This kind of attackers can only be fought by using tamper-proof technologies as well as cryptographic ones. The assumption regarding this type of adversaries, however, is by no means the most usual or reasonable assumption. Seamless monitoring requires the adversary not to lose every single session for group key or pairwise key update. The task of modification of random number generation codes will add another burden to that.

In order to measure the resilience of key management protocols, we derived four different types of adversaries varying in their capability with regard to seamless monitoring and software manipulation. As shown in Section 3, Nilsson et al.'s scheme, contrary to their claims, turned out to provide neither past key secrecy nor future key secrecy against node compromise by any type of adversaries.

We applied Lamport's reverse hash chain as well as usual hash chain to provide both past and future key secrecies. Our scheme avoids the delivery of the whole value of new group key for group key update; instead only the half of the value is transmitted from the network manager to the sensor nodes. This way, the compromise of a pairwise key alone does not lead to the compromise of the group key, which was not the case in the scheme by Nilsson et al. The new pairwise key in our scheme is determined by Diffie-Hellman based key agreement. As for the scheme of Nilsson's et al., it uses key transport, not key agreement, where the new pairwise key is determined by the sensor node and then delivered to the network manager by using public key encryption. This has brought a vital flaw to their scheme.

In short, our scheme provides a very strong resilience; both past and future key secrecies against node capture by all the adversary types except the strongest one, Type IV. A sensor node attacked by the adversary of Type IV, in theory, cannot be quarantined by a cryptographic method alone; a non-cryptographic countermeasure such as tamper-proof protection is needed together.

References

1. Cheryl Beaver, Donald Gallup, William Neumann, and Mark Torgerson. Key management for SCADA. Technical Report SAND2001-3252, Sandia National Laboratories - Cryptography and Information Systems Surety Department, March 2002.
2. Robert Dawson, Colin Boyd, Ed Dawson, and Juan Manuel González Nieto. SKMA: a key management architecture for SCADA systems. In Rajkumar Buyya, Tianchi Ma, Reihaneh Safavi-Naini, Chris Steketee, and Willy Susilo, editors,

ACSW Frontiers, volume 54 of *CRPIT*, pages 183–192. Australian Computer Society, 2006.

3. Christoph G. Günther. An identity-based key-exchange protocol. In *EUROCRYPT*, pages 29–37, 1989.
4. Carl Hartung, James Balasalle, and Richard Han. Node compromise in sensor networks: The need for secure systems. Technical Report CU-CS-990-05, University of Colorado at Boulder - Department of Computer Science, January 2005.
5. Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1(2-3):293–315, 2003.
6. Marek Klonowski, Miroslaw Kutylowski, Michal Ren, and Katarzyna Rybarczyk. Forward-secure key evolution in wireless sensor networks. In Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, editors, *CANS*, volume 4856 of *Lecture Notes in Computer Science*, pages 102–120. Springer, 2007.
7. Leslie Lamport. Password authentification with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
8. Sjouke Mauw, Ivo van Vessem, and Bert Bos. Forward secure communication in wireless sensor networks. In John A. Clark, Richard F. Paige, Fiona Polack, and Phillip J. Brooke, editors, *SPC*, volume 3934 of *Lecture Notes in Computer Science*, pages 32–42. Springer, 2006.
9. Robert McClanahan. SCADA and IP: is network convergence really here? *Industry Applications Magazine, IEEE*, 9(2):29–36, Mar/Apr 2003.
10. Dennis K. Nilsson, Tanya Roosta, Ulf Lindqvist, and Alfonso Valdes. Key management and secure software updates in wireless process control environments. In Virgil D. Gligor, Jean-Pierre Hubaux, and Radha Poovendran, editors, *WISEC*, pages 100–108. ACM, 2008.
11. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to privacy-friendly tags. In *In RFID Privacy Workshop*, 2003.
12. Ludovic Pietre-Cambacedes and Pascal Sitbon. Cryptographic key management for SCADA systems-issues and perspectives. *International Journal of Security and its Applications*, 2(3):31–40, July 2008.
13. Michal Ren, Tanmoy Kanti Das, and Jianying Zhou. Diverging keys in wireless sensor networks. In *ISC*, pages 257–269, 2006.
14. Rodrigo Roman, Cristina Alcaraz, and Javier Lopez. The role of wireless sensor networks in the area of critical information infrastructure protection. *Information Security Technical Report*, 12(1):24–31, March 2007.
15. Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In Roberto Verdone, editor, *EWSN*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2008.
16. Marcos August M. Vieiral, Claudionor N. Coelho. Jr, Di6genes Cecilio da Silva Junio, and Jose M. da Mata. Survey on wireless sensor network devices. volume 1, pages 537–544, September 2003.
17. John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. *Wireless sensor network security: A survey*, chapter 17 in Security in Distributed, Grid, and Pervasive Computing. Auerbach Publications, CRC Press, 2006.