SDTP+: Securing a Distributed Transport Protocol for WSNs using Merkle Trees and Hash Chains

Amit Dvir, Levente Buttyan, Ta Vinh Thong
Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics, Hungary

***Abstract*—Transport protocols for Wireless Sensor Networks (WSNs) are designed to fulfill both reliability and energy efficiency requirements. Distributed Transport for Sensor Networks (DTSN) [1] is one of the most promising transport protocols designed for WSNs because of its effectiveness; however, it does not address any security issues, hence it is vulnerable to many attacks. The first secure transport protocol for WSN was the secure distributed transport protocol (SDTP) [2], which is a security extension of DTSN. Unfortunately, it turns out that the security methods provided by SDTP are not sufficient; some tricky attacks get around the protection mechanism. In this paper, we describe the security gaps in the SDTP protocol, and we introduce SDTP$^+$ for patching the weaknesses. We show that SDTP$^+$ resists attacks on reliability and energy efficiency of the protocol, and also present an overhead analysis for showing its effectiveness.**

***Index Terms*—Distributed Transport Protocol; WSN; DTSN; SDTP; Hash Chain; Merkle-Tree;**

I. Introduction

Numerous transport protocols specifically designed for WSN applications, requiring particularly reliable delivery and congestion control (e.g., multimedia sensor networks) have been proposed [3]. Unfortunately, existing transport protocols for WSNs do not include sufficient security mechanisms or totally ignore the security issue. Hence, many attacks have been found against existing WSN transport protocols [4]. Broadly speaking, these attacks can be classified into two groups: attacks against reliability and energy depleting. Reliability attacks aim to mislead the nodes so that loss of a data packet remains undetected. In the case of energy depleting attacks, the goal of the attacker is to perform energy-intensive operations in order to deplete the nodes' batteries [4].

Buttyan and Grilo [2] presented the Secure Distributed Transport Protocol (SDTP), the first secure transport protocol for WSNs, which was based on the Distributed Transport for Sensor Networks (DTSN) protocol [1]. We argue that the security methods provided by SDTP are not sufficient and are still vulnerable. In this paper we propose a novel secure distributed transport protocol (SDTP$^+$) for WSNs that patches the security holes found in SDTP. Specifically, we revise the security methods used by SDTP [2] and propose additional security extensions. Our SDTP$^+$ protocol contains the following proposed mechanisms: (1) Application of Merkle-trees [5] and Hash chains [6]; (2) Status timer; (3) Aggregate timer; (4) Retransmission timers; (5) Sending predeleted packets; (6) Sending and forwarding ACK packets after a certain number of duplications; (7) Limiting EAR responding; and (8) Limiting the retransmission number.

The rest of the paper is as follows: Section II discusses the related works. In Sections III-IV the operation and security of SDTP are given. Our SDTP$^+$ protocol is described in Section V, while its security and overhead analysis are in Sections VI-VII. In this paper we only focus on our main result; additional explanations can be found in our technical report [7].

II. Related Works

In [4] the authors mentioned that the main vulnerabilities of reliable transport protocols for wireless sensor networks include the possibility for an attacker to replay, as well as inject fake or modified control packets. These can lead to unrecoverable data loss or energy depleting. Using a forged or altered ACK packet, an attacker can give the sender the impression that data packets arrived safely when they may actually have been lost. Similarly, forging or altering $NACK$ packets to trigger futile retransmissions can lead to draining of the node's batteries. While futile retransmissions do not directly harm the reliability of service, it is still undesirable. One of the solutions that would prevent these attacks can be authentication in lower layers. However, in our case, intermediate nodes also cache, modify, and verify the control packets. Hence, as already mentioned in [4], using a broadcast authentication method at the routing layer is not suitable; instead, the problem needs to be solved at the transport layer. For more details about why we should focus on the transport layer please refer to [2].

Based on the conclusion in [4], Buttyan and Grilo [2] have proposed the first security extension of the DTSN protocol. Their method is based on symmetric key cryptographic primitives, and hence, it is efficient in the WSN context. Unfortunately, it turns out that the proposed extension in [2] is still vulnerable to some kinds of attacks. The SDTP$^+$ protocol proposed in this paper improves the security extension proposed in [2] and presents new security extensions to eliminate the security holes. The security mechanisms proposed in this paper are based on the application of hash-chains [6] and Merkle-trees [5], which have been broadly used in designing security protocols. Hash chains have been used in many secure routing protocols (e.g., Ariadne [8]). Similarly, Merkle trees have been used in securing WSN protocols [9]. In this paper, we apply hash-chains and Merkle-trees in a new context, namely, for securing WSN transport protocol.

III. The SDTP protocol

The SDTP [2] protocol preserves the main advantage of DTSN [1] that allows intermediate nodes to cache and retransmit data packets, thus reducing the average number of hops a retransmitted packet has to traverse. Moreover, to reduce caching overhead each intermediate node only stores a packet with some probability p. As in the DTSN protocol, SDTP uses three types of control packets: Explicit Acknowledgment Requests (EARs), Positive Acknowledgments (ACKs), and Negative Acknowledgments ($NACK$s).

The source sends an EAR packet in one of the following cases: after the transmission of a certain number of data packets; output buffer becomes full; if during a predefined timeout period the application has not requested the transmission of any data; or expiration of the EAR timer (EAR_timer) [1]. An EAR may take the form of a bit flag piggybacked on a data packet or an independent control packet.

Upon receipt of an EAR packet, depending on the gaps in the sequence of received data packets, the destination sends a control packet (ACK or $NACK$). An ACK refers to a packet sequence number n indicating that all packets with sequence number smaller than or equal to n have been received by the destination. A $NACK$ refers to a base sequence number n along with a bitmap of set/unset bits, where each set bit represents a different sequence number starting from the base sequence number n. The meaning of the base sequence number n is the same as in the ACK case, while the packets corresponding to the set bits in the bitmap are required to be retransmitted.

The main security solution of the SDTP protocol is as follows [2]: each (sequentially numbered) data packet is extended with an ACK MAC (Message Authentication Code) and a $NACK$ MAC, which are computed over the whole packet with two different keys, an ACK key (K_{ACK}) and a $NACK$ key (K_{NACK}). Both keys are known only to the source and the destination and are specific to the data packet; hence, these keys are referred to as per-packet keys.

Upon receiving a data packet, the destination can check the authenticity and integrity of each received data packet by verifying the two MAC values. Upon receipt of an EAR packet, the destination sends an ACK or a $NACK$ packet, depending on the gaps in the received data buffer. In case an ACK packet refers to a data packet with sequence number n, the destination reveals its ACK key; similarly, when it wants to signal that this data packet is missing, the destination reveals its $NACK$ key. Any intermediate node storing the relevant packets can verify if the ACK or $NACK$ message it receives is authentic by checking if the appropriate MAC is verified correctly with the included key. For each verification of the $NACK$ key, the intermediate node retransmits the corresponding data packet (if stored), unsets the bit, and removes the corresponding key. In case the bitmap becomes clear, the intermediate node sends an EAR message and the $NACK$ becomes an ACK message. In addition, each intermediate node and the source maintain for each session the largest verifiable acknowledged sequence number so far, which we denote by $MaxSN$, to protect against replaying control packets.

IV. Security issues and design objectives

A. Attacker Model

As most external attacks can be defeated by link layer authentication schemes such as TinySec, we focus here on the case of internal attackers (i.e., compromised nodes) that want to avoid discovery of the node compromise by the network operator; that is, stealthy attackers [10] are assumed. The main goal of the attacker is deceiving the honest nodes that data packets are delivered while in reality they are lost, while the secondary goal is to mount attacks where the nodes are coerced into spend more energy than actually needed. In particular, we are not interested in (and actually cannot really prevent) "brute force" Denial-of-Service type attacks. Instead we are interested in attacks where compromised nodes misbehave in more sophisticated ways that are more difficult to discover while causing huge damage.

B. Security issues in the SDTP protocol

SDTP is believed to be secure because the shared secret S is never leaked, and hence, only the source and the destination can produce the right ACK and $NACK$ master keys and per-packet keys [2]. Since the source never reveals these keys, the intermediate node can be sure that the control information has been sent by the destination. In addition, because the per-packet keys are computed by a one-way function, when the ACK and $NACK$ keys are revealed, the master keys cannot be computed from them; hence, the yet unrevealed ACK and $NACK$ keys cannot be derived. Surprisingly, SDTP is still vulnerable to some tricky attacks that we discuss below.

In the first attack, called a *creating fake packets* attack, colluding attackers can cause data packets be deleted from the caches of intermediate nodes, although they should not be. Let us consider the following scenario: Let S, I, D be the source, intermediate, and destination nodes, respectively; and let $A1$ and $A2$ be the two cooperative compromised nodes. Assume that the topology is such that there are symmetrical links between $(S, A1)$, $(A1, I)$, $(I, A2)$, and $(A2, D)$ pairs. First, $A1$ creates a data packet m containing a MAC value computed with fake ACK and $NACK$ keys, then node I stores the packet without being able to verify the MAC values. Later, even without the presence of D, $A2$ generates fake ACK, $NACK$ packets with the corresponding fake keys, which will match the MAC values of the stored m at node I. Hence, node I considers these fake acknowledgment packets to be valid. Therefore, node I deletes all the stored packets with a sequence number that is less than m (including m) from its cache and updates its $MaxSN$ to be m. As a consequence, intermediate nodes can be easily misled to believe that data packets have been delivered, although the destination has not received them.

Other vulnerabilities are found in the following cases: the destination in SDTP sends ACK or $NACK$ packets upon reception of an EAR. Attacks aiming at replaying or forging EAR information, where the attacker always sets the EAR flag to 0 or 1, can have harmful effect. Always setting the EAR flag to 0 prevents the destination from sending an ACK or $NACK$ packet, while always setting it to 1 makes the

destination send control packets unnecessarily. In SDTP the attackers may replay $NACK$ packets to force futile retransmissions of data packets. Finally, by unsetting some or even all the bits of the bitmap in $NACK$ packets, an attacker can prevent the intermediate nodes from retransmitting packets.

V. The SDTP$^+$ Protocol

SDTP$^+$ aims at enhancing the authentication and integrity protection of control packets, and is based on an efficient application of asymmetric key crypto and authentication values, which are new compared to SDTP. Our proposed mechanism is tailored for the problem of authenticating and verifying the ACK and $NACK$ packets; hence, it is more effective in the context of WSN than general purpose schemes. The general idea of SDTP$^+$ is the following: two types of "per-packet" *authentication values* are used, ACK and $NACK$ authentication values. The ACK authentication value is used to verify the ACK packet by any intermediate node and the source, whilst the $NACK$ authentication value is used to verify the $NACK$ packet by any intermediate node and the source. The ACK authentication value is an element of a hash chain [6], whilst the $NACK$ authentication value is a leaf and its corresponding sibling nodes along the path from the leaf to the root in a Merkle-tree [5]. Each data packet is extended with one Message Authentication Code (MAC) value (the MAC function is HMAC), instead of two MACs as in SDTP. Unlike SDTP where the MAC is computed over the whole packet, our MAC is computed over the entire packet except for the EAR and RTX flags in the header, due to the fact that the EAR and RTX flags are legitimately modified by the intermediate nodes during the protocol run, and they should not be embraced in the MAC, otherwise, this may lead to verification failure at the destination. Unfortunately, by not authenticating the flags an attacker can modify them, which will increase the overhead caused by the operations that nodes perform due to the faked flag values. We will discuss the solution of this problem in section VI-B. Finally, besides these security mechanisms we also propose the application of timers (Status, Aggregate, and Retransmission timer) and transmission limitation mechanisms for mitigating the effects of the energy depleting attempts.

A. SDTP$^+$ - Building Blocks

We discuss the two main building blocks used in SDTP$^+$ that are new approaches compared to SDTP.

1) Hash Chain [6]: A hash chain is a sequence of hash values that are computed by iteratively calling a one-way hash function on an initial value. Let us denote an initial value by x, the hash function by h, and the initial value of the hash chain as $v_m = h(x)$. Then, the i-th element v_i of the hash chain is computed as $v_i = h(v_{i+1}) = h^{(i)}(v_m)$. An important property of the hash chain is that its elements can be easily computed in one direction, but not in the reverse direction. In other words, if someone knows v_i, then she can compute any $v_j = h^{(i-j)}(v_i)$ for any $j < i$, but she cannot compute any v_k for $k > i$. This property stems from the one-way property of the hash function.

2) Merkle-tree [5]: A property that limits the application of hash chains in some applications is that the elements can only be revealed sequentially. Merkle-trees overcome this problem by allowing for the pre-authentication of a set of values with a single digital signature and for the revelation of those values in *any* order. The operation of a Merkle-tree can be summarized as follows: Let the set of values that we want to authenticate be $v_1, v_2, \ldots, v_{2^\ell}$. First, we hash each value v_i into v_i' with a one-way hash function. Then, we assign the hashed values to the leaves of a binary tree. Moreover, to each internal vertex u of this tree, we assign a value that is computed as the hash of the values assigned to the two children of u. Either in hash chain or Merkle-tree, when a digital signature is used, each node has to know the public signature verification key. A sensor node is very costly to digitally sign and verify; therefore, it is useful to use the signing procedure only for bootstrapping. The signing procedure is important only for secure distributiion of the root of the hash chain and the Merkle-tree root.

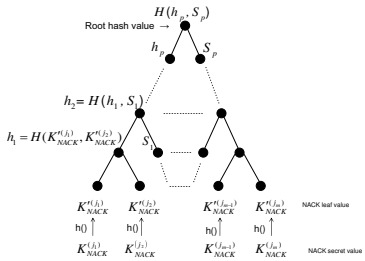

Fig. 1. The structure of Merkle-tree used in our method. Each internal node is computed as the hash of the ordered concatenation of its children.

B. SDTP$^+$ - Authentication Value Hierarchy

In SDTP$^+$ we adopt the notion and notations of the pre-shared secret S, and ACK, and $NACK$ master secrets K_{ACK}, K_{NACK} given in [2]. These keys are computed in exactly the same way as the SDTP protocol. However, in our method the generation and management of the per-packet keys $K_{ACK}^{(n)}$, $K_{NACK}^{(n)}$ is based on the application of hash-chains and Merkle-trees, which is different from SDTP.

1) The ACK Authentication Values: The authentication value associated with the ACK packet referring to the i-th data packet is the i-th element of a hash chain as explained in section V-A1. At the beginning of each session, the source generates the ACK master secret K_{ACK} and calculates a hash chain of size $m+1$ with the initial value $K_{ACK}^{(m)} = h(K_{ACK})$, where m is the number of data packets that the source wants to send in the session. Each element of the calculated hash-chain represents a *per packet ACK authentication value* as follows: $K_{ACK}^{(m)}, K_{ACK}^{(m-1)}, \ldots, K_{ACK}^{(1)}, K_{ACK}^{(0)}$, where $K_{ACK}^{(i)} = h(K_{ACK}^{(i+1)})$ and h is a one-way hash function. The value $K_{ACK}^{(0)}$ is the root of the hash-chain, and $K_{ACK}^{(i)}$ represents the ACK authentication value corresponding to the packet with sequence number i.

2) The $NACK$ Authentication Values: Unlike ACK value, because several NACK authentication values are revealed at

a time in any order, hash-chain is not applicable. Hence, for authenticating the $NACK$ packets we propose the application of a complete binary Merkle-tree based on section V-A2. First, the so-called *NACK secret values*, which are the set of values that we want to authenticate, are computed as follows: $K^{(n)}_{NACK} = PRF(K_{NACK}, \text{"per packet } NACK \text{ secret"}, n)$, where n is a sequence number of a data packet. Afterwards, we hash each *NACK secret value* and assign the hashed values to the leaves of the Merkle-tree that we called *NACK leaf value*: $K^{'(n)}_{NACK} = h(K^{(n)}_{NACK})$. The internal nodes of the Merkle-tree are computed based on these per-packet NACK leaf values. The Merkle-tree leaves corresponding to the $NACK$ secret values can be found in Fig. 1.

C. SDTP$^+$ - Source Mechanism

When a session is opened, first, the source computes the session master secret K, the ACK master secret K_{ACK}, and the $NACK$ master secret K_{NACK}. Then, based on the information about the number of data packets in the session and the master secrets, the source calculates the ACK authentication values and the $NACK$ authentication values. That is, a hash-chain and a Merkle-tree are generated for the session. Afterwards, the source sends an open session message with the following parameters: the root of the hash chain ($K^{(0)}_{ACK}$), length of the hash chain ($m+1$), number of Merkle-tree roots (t), $SessionID$, source ID, destination ID, and the roots of the t Merkle-trees (we choose t = 1 for efficiency purpose, for the reason see [7]). Finally, the source digitally signs the whole message (we choose ECC [11] for a simple illustration, for more options see [7]). Taking into account the possibility that the open session packet will be lost and will not reach the destination, the source may need to retransmit the open-session packet. Upon sending an open session packet, the source launches a timer (open-session timer), and when the time has elapsed without any feedback about the successful reception of the open session packet, the source retransmits the open session packet. Besides setting a timer, the source also limits the retransmission number of the open session packet.

After receiving an ACK message about the open session packet from the destination, the source starts to send data packets. Each data packet is extended with a MAC, computed over the whole packet except for the flags in the header using the shared secret with the destination. When the source node receives an ACK packet that includes the ACK authentication value ($K^{(i)}_{ACK}$) corresponding to the packet of sequence number i, it hashes iteratively the ACK authentication value i times. If the result is equal to the root hash value $K^{(0)}_{ACK}$, then the ACK packet is accepted and the source removes all the packets with sequence numbers smaller than or equal to i from its cache, and updates the value of $MaxSN$ i; otherwise, it ignores and drops the ACK packet.

When the source node receives a $NACK$ packet that includes the ACK authentication value $K^{(i)}_{ACK}$, $NACK$ authentication values (secret values $K^{(i+1)}_{NACK}$,..., $K^{(i+j)}_{NACK}$, and their corresponding sibling values), it first checks the ACK authentication value and performs the same steps as for ACK authentication. Then the source continues with verifying the $NACK$ authentication values. For each set bit in the bitmap, the node verifies the $NACK$ authentication values. Upon success, the $NACK$ packet is accepted and the source retransmits the required packets.

D. SDTP$^+$ - Destination Mechanism

When the destination node receives an open-session packet, it verifies the signature computed on the packet. Upon success, the destination starts to generate the session master secret, the ACK master secret, the $NACK$ master secret, the hash-chain, and the Merkle-tree. Finally, the destination sends an ACK for the open session packet to the source.

Upon receiving a data packet with sequence number i, first, the destination checks the authentication data field using the secret shared between the source and the destination. Upon success, the destination delivers the packet to the upper layer, otherwise, the packet is ignored and dropped. Upon the receipt of a packet with a set EAR flag, the destination sends an ACK or a $NACK$ packet depending on the gaps in the received data packet stream. The ACK packet that refers to sequence number i is extended with the ACK authentication value ($K^{(i)}_{ACK}$). For this purpose the structure of ACK packets is extended with an ACK authentication value field. Similarly, the $NACK$ packet with base sequence number i is extended with the ACK authentication value ($K^{(i)}_{ACK}$), as the semantics of the base sequence number in $NACK$ packets is the same as that of the sequence number in ACK packets. In addition, if the j-th bit is set in the bitmap, then the $NACK$ packet is further extended with the $NACK$ secret value $K^{(i+j)}_{NACK}$ and its sibling authentication values. To locate these authentication values, the format of $NACK$ packets is extended with an ACK authentication value field and a variable number of $NACK$ secret and sibling value fields.

In the DTSN/SDTP protocol, the destination sends an ACK or a $NACK$ packet upon receipt of an EAR. In order to mitigate the effect of EAR replay or EAR forging attacks where the EAR flag is set/unset by an attacker(s), SDTP$^+$ uses two new mechanisms: status timer (dynamic or static) and limiting the number of responses to EARs. The status timer is set at the destination and its duration could be a function of the source EAR timer. To counter that attackers always set the EAR bits, the destination limits the number of responses on receiving a set EAR flag. In the period of the destination's EAR timer the destination will not send more than X control packets, where X can be dynamic or static. The detailed discussion about the timer duration can be found in our report [7].

E. SDTP$^+$ - Intermediate Node Mechanism

Upon receipt of an open session packet and the corresponding ACK packet, an intermediate node verifies the signatures computed on them, and in case of success it stores the hash chain root value, the tree root values, and the $SessionID$ included in the open session packet, and forwards the packet towards the destination. Otherwise, an intermediate node changes its probability to store packets in the current session to zero. Upon receipt of a data packet, an intermediate

node stores with probability p the data packet and forwards the data packet towards the destination. Note that the intermediate and source nodes have the same steps in the case of receiving ACK or $NACK$ control messages.

In the following we describe some extra steps for the intermediate node mechanism in order to mitigate and prevent some attacks. After successfully verifying ACK/$NACK$ packets, an intermediate node does not forward them immediately but only after a certain period of time. Specifically, intermediate nodes set a timer, called an aggregate-timer, which can be either dynamic or static. If in this period of time more than certain number of ACK/$NACK$ packets arrive, the intermediate node tries to merge the verification information from the control packets into one control packet (i.e., a form of aggregation); otherwise, it will send the original ACK/$NACK$ packets. The sum of the aggregate timers set by intermediate nodes should not be larger than the source EAR timer. Hence, the timer value may be a function of the source EAR timer and the maximum number of nodes in the path between a source and a destination.

In order to mitigate the effect of a $NACK$ replay attack where the attacker replays old $NACK$ packets to force futile retransmissions of data packets, we introduce the following extensions. The transmission rate of a retransmitted data packet with sequence number i is limited by using a timer called the retransmission-timer (denoted by Re_timer_i). In addition, the number of retransmissions (denoted by $Retransmission_i$) per stored data packet with sequence number i is limited as well. With these extensions, on receiving a replayed $NACK$ packet, after a certain number of retransmissions of the same data packet, the intermediate node only forwards the $NACK$ packet without retransmission, although it stored the data packet. In the worst case, however, an intermediate node still can be made to retransmit unnecessarily up to the limitation. We refer the reader to the report [7] for further analysis about the overhead of futile retransmissions in the worst case.

The attacker can also attempt to prevent intermediate nodes from retransmitting packets by deleting some or even all the bits of the bitmap in $NACK$ packets. Note that the attacker does not modify or delete the ACK value but only the bitmap in $NACK$ packets. To mitigate the impact of this attack, after receiving a certain number T of ACK packets with the same acknowledge value equal to $MaxSN$, the intermediate node automatically retransmits the first packet in its buffer that has a sequence number greater than $MaxSN$. In addition, these ACK control packets are forwarded to the source. Unfortunately, we cannot entirely prevent, but only mitigate, this attack; however, as explained in Section IV-B, the period of time for this kind of attack is relatively small because each node has an aggregate timer and the source will filter the irrelevant control packets. For more details about the impact of the T value please refer to [7].

When an intermediate node needs to delete a packet from its buffer because its buffer is full, it sends the first packet in its buffer that has a sequence number greater than the stored $MaxSN$ to the destination with probability q, which can be a different probability from the caching probability, and then deletes this packet. Note that q needs to be set close to zero in case the attacker is able to inject fake packets with a high sequence number, which can cause a retransmission of a fake packet. However, due to the fact that injecting fake packets has limited effectiveness, as explained in section IV-B, q can be larger than 0.5. Finally, when an intermediate node receives some packet with a new session number but the same (*source, destination, application ID*) tuple as the packets already stored in its buffer, these stored packets will not be deleted from the buffer. For the detailed processing of the $NACK$ packets see [7].

VI. Security Analysis

We synthesized some typical attacks against the reliability and energy efficiency properties of the protocol to which both DTSN and SDTP are vulnerable. We analyze SDTP$^+$ and illustrate the advantage of applying hash-chain and Merkle-trees in preventing these attacks. For further detailed attacks and explanations see [7].

A. Analyzing the resistance against reliability attacks

The two basic attacks against reliability are concerned with forging ACK and $NACK$ packets. Let n be the ACK value, and the attacker wants to modify it to a greater m. In this case the attacker has to include a correct ACK authentication value $K_{ACK}^{(m)}$. However, computing $K_{ACK}^{(m)}$ based on $K_{ACK}^{(0)}$,...,$K_{ACK}^{(n)}$ is hard because the hash function used in generating the hash-chain is one-way. In the $NACK$ case, to be successful an attacker has to include a valid $NACK$ authentication value ($NACK$ secret values and their siblings) in the $NACK$ packet. Note that the $NACK$ authentication value is the leaves of the Merkle-tree, and computing the leaves based on the upper level hash values is hard because the hash function used to generate the Merkle-tree is one-way.

We recall the *creating fake packets* attack presented in Section IV-B, which is one of the main weaknesses of SDTP. This attack is possible because in SDTP the authenticity of ACK/$NACK$ packets is based on the verification of ACK/$NACK$ MACs with the corresponding keys. However, the authenticity of the keys is not checked; hence, bogus MACs with self-created keys can be created easily. The attack is not feasible in SDTP$^+$ because the authentication values are the elements of hash-chain and Merkle-tree, and computing the unrevealed ACK authentication value of packet m ($K_{ACK}^{(m)}$) from the already revealed $K_{ACK}^{(n)}$ ($m > n$) is hard due to the property of one-way functions. Computation of the unrevealed NACK authentication values from the already revealed ones is hard for the same reason. Moreover, attackers cannot send a bogus self-generated hash-chain and Merkle-trees to honest nodes because they cannot forge the digital signature of the source.

B. Analyzing the resistance against energy depleting attacks

In the following, we discuss how SDTP$^+$ resists the attacks aimed at increasing energy consumption.

Due to the fact that the EAR flag is not protected in the SDTP$^+$ packets, its modification cannot be detected. Adding

the status timer forces the destination to send control packets to the source even without any trigger from the source, which mitigates the effect of this attack. Moreover, by limiting the number of control packets sent by the destination in a given time, we can control the overhead by setting this number.

To mitigate the effect of the attack attempt, where in order to cause futile retransmissions the attacker intercepts a $NACK$ packet, and generates lots of valid "shorter" $NACK$ packets based on the combinations of (subset of) the authentication values found in the original $NACK$, each intermediate node has a period of time to aggregate the received $NACK$s into one $NACK$, and only handles this $NACK$.

The next scenario would cause the EAR sending attempts [1] at the source to exceed the limit and lead to session close: an attacker intercepts a $NACK$ and unsets some bit(s) along with the corresponding $NACK$ authentication values and passes it on. To alleviate the impact of this attempt, as described in Section V-E, honest nodes automatically send and forward a data packets after receiving certain number of the same ACK packets.

VII. Overhead Analysis

We evaluate the overhead of our new security scheme in a wireless sensor network assuming MICAz motes. First, based on [12], [11], we estimate the time for each building block and, finally, calculate for each node the time overhead of the security scheme.

Regarding the hash chain, the source and destination have to generate a hash chain with chain length $m+1$ [12]. Only the source has to sign, using Elliptic Curve DSA [11], the first message. Each intermediate node and destination have to verify the signature once and each intermediate node has to verify the hash element per ACK message. This requires one signing operation ECC_G and $(m+1) \cdot SHA$ time at the source ($ECC_G + (m+1) \cdot SHA$), and one signing operation ECC_V and $(m+1)$ hashing operations at the destination, i.e., $(m+1) \cdot SHA$ time. At intermediate nodes, one verification operation ECC_V and $\frac{(2 \cdot m - I + 1) \cdot I}{2}$ hashing operations are required in the worst case, where I is the number of ACKs that the intermediate node receives. Thus, intermediate node needs $ECC_V + \frac{(2 \cdot m - I + 1) \cdot I}{2} \cdot SHA$ time in the worst case.

For the Merkle-tree, the source generates a complete binary Merkle-tree of height d (recall that we choose t = 1 for efficiency purpose [7]). Furthermore, for creating the leaves the source and destination require $m \cdot SHA$ time [12], while to distribute the tree-roots in a secured manner it requires two signing operations ECC_G and ECC_V from the source and destination, respectively. For generating the Merkle-tree, the source hashes at each level of the tree, which takes $2^m - 1$ hash operations that takes $(2^m - 1) \cdot SHA$ time. For each bit in the bitmap of a $NACK$ message an intermediate node that stored the message needs to verify the $NACK$ authentication value, which requires d hash operations. With a given loss probability, the nodes on the path between the source and the destination have to retransmit $m \cdot loss\_propability$ times, which requires $d \cdot loss\_propability \cdot SHA$ time.

Regarding message overhead, for the ACKs, it is the hash value size; for the $NACK$s, it is the number of the set bits in the bitmap times the authentication value size (which is d times the hash value size). Hence, intermediate node needs to handle $d \cdot loss\_probability \cdot SHA$ message overhead.

VIII. Conclusion

In this paper, we proposed security extensions to SDTP in order to patch the weaknesses of the protocol. The security extensions are based on a minimal amount of asymmetric key cryptography, hash chains, and Merkle-trees. We showed that with these proposed mechanisms, SDTP$^+$ resists attacks against the reliability and energy efficiency requirements, including the attacks found against SDTP. In our future work, we will develop a formal and automated security proof for SDTP$^+$, as well as implement the proposed theoretical mechanisms to examine their efficiency in practise.

Acknowledgments

The work presented in this paper has been carried out in the context of the CHIRON Project (www.chiron-project.eu), which receives funding from the European Community in the context of the ARTEMIS Programme (grant agreement no. 225186). Amit Dvir has also been supported by the Marie Curie Mobility Grant, OTKA-HUMAN-MB08-B 81654. The last two authors are also partially supported by the grant TAMOP - 4.2.2/B-10/1-2010-0009 at the Budapest University of Technology and Economics.

References

[1] F. Rocha, A. Grilo, P. R. Pereira, M. S. Nunes, and A. Casaca, "Performance evaluation of DTSN in wireless sensor networks," in *EuroNGI - Network of Excellence Workshop*, Barcelona, Spain, Jan. 2008, pp. 1–9.

[2] L. Buttyan and A. M. Grilo, "A Secure Distributed Transport Protocol for Wireless Sensor Networks," in *IEEE International Conference on Communications*, Kyoto, Japan, June 2011, pp. 1–6.

[3] J. Yicka, B. Mukherjeea, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, Aug. 2008.

[4] L. Buttyan and L. Csik, "Security analysis of reliable transport layer protocols for wireless sensor networks," in *Proceedings of the IEEE Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS)*, Mannheim, Germany, March 2010, pp. 1–6.

[5] R. C. Merkle, "Protocols for Public Key Cryptosystems," in *Symposium on Security and Privacy*, California, USA, April 1980, pp. 122–134.

[6] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," in *Fourth Conference on Financial Cryptography*, Southampton, Bermuda, March 2002, pp. 102–119.

[7] A. Dvir, L. Buttyan, and T. V. Thong, "Sdtp+: A secure distributed transport protocol for wireless sensor networks," http://www.crysys.hu/members/tvthong/SDTP/DvirBTh12SDTPTech.pdf.

[8] Y. C. Hu, A. Perrig, and D. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," *Wireless Networks*, vol. 11, no. 1-2, pp. 21–38, Jan. 2005.

[9] Y. C. Hu, D. B. Johnson, and A. Perrig, "Secure efficient distance vector routing in mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, July 2003.

[10] A. Herzberg and H. Shulman, "Stealth DoS Attacks on Secure Channels," in *Network and Distributed System Security Symposium*, California, USA, Feb 2010, pp. 1–19.

[11] R. Roman, C. Alcaraz, and J. Lopez, "A Survey of Cryptographic Primitives and Implementations for Hardware-Constrained Sensor Network Nodes," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 231–244, Oct. 2007.

[12] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, "Analyzing and modeling encryption overhead for sensor network nodes," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, CA, USA, Sep. 2003, pp. 151–159.