Exploring Symmetric Cryptography for Secure Network Reprogramming *

Donnie H. Kim
Information Networking Institute
Carnegie Mellon University
donniekim@cmu.edu

Rajeev Gandhi
ECE Department
Carnegie Mellon University
rgandhi@ece.cmu.edu

Priya Narasimhan
ECE Department
Carnegie Mellon University
priya@cs.cmu.edu

Abstract

Recent secure code-update protocols for sensor networks have been based on asymmetric-crypto primitives such as digital signatures. Our approach, Castor, explores the feasibility of securing an existing code-update protocol, Deluge, using symmetric-crypto mechanisms that are more suited to the resource constraints of sensor nodes. Castor involves a synergistic combination of a one-way hash-chain, a one-way key-chain, and a sequence of MACs with delayed key-disclosure to enable sensor nodes to verify the update's authenticity. We guarantee that no correct node will ever install or forward a compromised part of an update, while addressing the performance issues related to delayed key-disclosure.

1 Introduction

Wireless sensor networks generally operate in deeply embedded environments. Given the long-lived characteristic of these networks, the need to automatically update the code running on previously installed sensors during their lifetimes is obvious. A number of *network-reprogramming* (or *code-update*) *protocols* (Deluge [4], MNP [7], Infuse [6], etc.) have emerged to support the reliable and efficient dissemination of a code-update (or program image) in sensor networks. These protocols employ the radio, the sensor's primary communication channel, and are, thus, referred to as "over-the-air" updates.

The propagation of fragments of a code-update's image is typically done in an epidemic fashion, i.e., sensors propagate received fragments to their immediate neighbors that, in turn, propagate them to their neighbors, and so on, until all of the sensors in the network have been reached. Preventing compromised anonymous nodes from disseminating arbitrary code to the entire network is critical, particularly because an adversary can hijack this epidemic propagation (through the injection of a corrupt code-update image) to adversely impact the entire network's functionality.

Recent research efforts ([1, 2] and our own previous work, Sluice [9]) on secure network-reprogramming protocols have leaned towards employing asymmetric cryptography, which comes with its associated computational costs. We seek to investigate *whether we can exploit symmetric cryptography to provide for lower end-to-end update latencies and lower computation/resource costs for update dissemination in sensor networks* (as compared to the use of asymmetric cryptography)? While it is well known that symmetric cryptosystems are computationally less expensive (and, therefore, possibly more suited to resource-constrained sensor networks) than asymmetric ones, there is more to their usage than this. With symmetric cryptography, a sender and receiver must set up a shared key in advance; an adversary that subverts even a single node can obtain access to the secret key and effectively compromise the entire code-update process and all of the innocent nodes. Thus, we need a way to be able to use symmetric cryptography in network reprogramming, without worrying about the exposure of the keys to an adversary due to compromised nodes in the network.

Our secure code-update protocol, Castor, leverages symmetric-crypto mechanisms that are likely to be more suited to the resource constraints of sensor nodes. Castor involves a synergistic combination of a one-way hash-chain, a one-way key-chain, and a sequence of MACs with the delayed disclosure of symmetric keys, to enable untrusted sensor nodes to verify the update's authenticity. We guarantee that no correct node will ever install or forward a compromised part of an update, while simultaneously addressing the performance issues related to delayed key-disclosure.

2 Background

• **Network-reprogramming protocols.** Deluge is distributed as a part of the popular TinyOS environment [11], and is currently the most widely available code-update protocol for sensor networks. In our implementation of Castor,

*Funded by ARO grant DAAD19-02-1-0389 to the Center for Computer and Communications Security at CMU.

we build upon Deluge primarily because of its wide availability – thus, this section describes Deluge's operation in detail. However, we emphasize that our approach can be easily extended to any network reprogramming protocol.

Deluge splits the code-update's binary image into fixed-size chunks called *pages*. Each page is further split into fixed-size transmission units called packets. Deluge uses pipelining or spatial multiplexing to reduce the end-to-end propagation latency, thereby making the update process more efficient. With pipelining, nodes are allowed to forward pages that they have already received without needing to wait to receive the entire set of pages that form the entire code-update image. Pages must be received sequentially, i.e., a node may not begin forwarding page N until it has first received pages $0, 1 \ldots, N-1$, inclusive. This constraint reduces the amount of state information that each sensor must maintain regarding the extent to which the code-update image has been locally received/transmitted.

• **One-way hash/key chains.** Lamport first proposed one-way chains [8] as a means of generating one-time passwords. The concept is based upon a one-way function $F()$ that is easy to compute, but that is computationally difficult to invert. A one-way chain $\{a_0, a_1, ..., a_{m-1}\}$ of length m is generated by repeatedly applying the function $F()$ to the last element, a_m, to generate a sequence where element $a_j = F(a_{j+1})$. The chain is generated in the order $a_{m-1}, a_{m-2}, ..., a_1, a_0$, but revealed in the reverse order. The head of the one-way chain, a_0, must be a trusted value because it serves as a commitment to the entire chain.

Due to the one-way chain property, (i) it is possible for every node to verify whether a received a_i is the ith element of the one-way chain by examining whether $F^i(a_0)$ equals a_i, (ii) it is not possible for a compromised node to derive a_{i+1} from a_i, for any $i > 0$, and (iii) even if intermediate elements are missing, they can be derived using later elements of the chain. Thus, a node that has access to an authentic a_0 can recursively verify the authenticity of the remaining elements of the chain.

Castor leverages the concept of one-way chains in two ways: (i) where $F()$ is a hash function, $H()$, and the elements of the chain are hashes, thereby producing a *one-way hash chain*, and (ii) where $F()$ is a pseudo-random function, $K()$, and the elements of the chain are symmetric keys, thereby producing a *one-way key chain*.

• **Symmetric-crypto authenticated broadcast.** Authenticated broadcast protocols (e.g., TESLA [17], μTESLA [18]) use symmetric keys to authenticate the sender of a message in the face of node compromise. The protocols involve a one-way key chain, $\{k_0, k_1, ..., k_m\}$, whose elements are used by a trusted sender to compute MAC over data, which is then broadcast (along with the MACs) to all of the nodes in the system. Upon the trusted entity's subsequent disclosure of the keys, the receivers can verify the received data. Disclosed keys are verified against k_0, the initial commitment to the key-chain, through the iterative application of a pseudo-random function. k_0, is assumed to be securely distributed to all of the receivers as a part of system initialization. In TESLA, k_0 is digitally signed by the sender and broadcast to all of the the receivers during bootstrapping. In μTESLA, the sender unicasts k_0 to each recipient using pair-wise symmetric keys, with one key between the sender and each receiver. Recent work [12] has attempted to overcome this scalability issue through the use of broadcast messages.

For a receiver to be able to verify a message's authenticity, it must receive the authenticated message *before* the corresponding key (i.e., the key with which the MAC is computed) is disclosed by the sender. The security of this delayed key-disclosure mechanism hinges upon loose time synchronization across the network, i.e., all of the receivers have an upper bound on the sender's local time and the sender knows when to divulge a key based on *a priori* knowledge of the worst-case end-to-end propagation time of a message from the sender to every node in the system.

We avoided using authenticated broadcast in Sluice[1] because of the intricacies involved in determining the proper key-disclosure interval for Deluge. Deluge ensures that the new code-update image will be reliably disseminated to all the nodes in the sensor network, but does not place a bound on the amount of time taken to disseminate the image to all of the nodes of the network. Without a bound on the end-to-end update latency, it is hard (as explained in Section 4.1) to choose the appropriate key-disclosure interval. Castor overcomes these problems and demonstrates the feasibility of symmetric-crypto authenticated broadcast for secure code updates.

3 Problem Statement

Castor aims to enhance existing code-update protocols with security, while upholding their efficiency mechanisms. Having first-hand experience with developing both an asymmetric-crypto (Sluice [9]) and a symmetric-crypto secure code-update protocol (Castor, described in this paper), we are uniquely positioned to compare our use of symmetric vs. asymmetric cryptosystems for securing code updates, and to evaluate their relative overheads.

3.1 System Model

We assume that individual sensor nodes are connected via an insecure wireless medium and are susceptible to compromise. An adversary can subvert an arbitrary number of sensor nodes in the system, and can gain control of their cryptographic material. Castor's design covers different kinds of

[1] Other research has independently followed a similar train of thought.

adversarial behavior, e.g., an adversary can inject an arbitrary number of corrupt packets or code-update images into the system, can withhold/delay/modify an arbitrary number of packets or images, and can eavesdrop on the communication of other sensors in the system. We place no restrictions on the number of malicious sensors, their locations, and the degree of connectivity or collusion between them.

We assume that there is a single trusted sender (i.e., a base station) that is the source of the code update. We assume that the base station and every sensor node is bootstrapped with the values of T (the key-disclosure interval) and k_0 (the head of Castor's key-chain) , and knows the one-way key function $K()$, in addition to the one-way hash function $H()$.

Because we exploit symmetric-crypto authenticated broadcast in Castor, we assume that the base station and the nodes in the network are loosely and securely time synchronized [3, 13, 19]. If the sensor-network application already uses time synchronization for its own purposes (e.g., TDMA-based applications, shooter localization), Castor can simply exploit this facility.

We do not currently address the confidentiality of an update, and threats (e.g., battery-drain attacks, denial-of-service) that can drain the memory buffers at each node.

3.2 Objectives

From a *security* viewpoint, we require that every node be able to verify whether a code-update image originates from the trusted source (authenticity property) and that this image is unmodified (data-integrity property). Furthermore, no uncompromised node should ever install or propagate any part of an unverified code-update image.

From a *performance* viewpoint, we desire that nodes perform the verification of the image incrementally, to start propagating pages that have been verified as correct and to halt pages that have been detected to be corrupt. Nodes should be permitted to leverage existing efficiency mechanisms, such as pipelining. We also desire our security enhancements to be sensitive to the resource constraints of sensor nodes and result in low computation and communication overheads with respect to the underlying insecure code-update protocol. We aim for the ideal "no-node-left-behind" condition, where every node in the network receives the appropriate cryptographic material in time to enable it to verify the authenticity of a code-update image.

4 Castor's Approach

Castor exploits a synergistic combination of multiple building-blocks: a one-way hash chain, a sequence of MACs, a one-way key chain, and delayed key-disclosure. Because Castor is currently built on top of the Deluge code-update protocol, we borrow terminology and concepts from Deluge in our description.

From Section 2, we recall that Deluge splits the code-update image (referred to as the "payload" because this is the data that is desired to be transmitted over the network) into fixed-size pages, and each page into fixed-size packets. Using Castor, the trusted base station (i) constructs a one-way key chain, (ii) constructs a one-way hash chain over the pages of the code-update image, (iii) computes a sequence of MACs, each of which is a MAC of the hash of the first page using a unique key from the one-way key chain, (iv) disseminates all of the MACs to the sensor nodes before starting the update process, (v) disseminates the pages of the image to the sensor nodes, and finally, (vi) discloses the keys, one by one, at predetermined intervals. The keys are disclosed one by one, in a delayed manner, to enable the untrusted nodes to verify the received MACs, and therefore, the code-update image. We describe the details of Castor's approach below.

4.1 Details of Approach

The trusted base station generates a one-way key-chain (containing p self-authenticating elements) by selecting key k_p at random and repeatedly applying a pseudo-random function $K()$ to k_p to generate the earlier keys $k_{p-1}, k_{p-2}, ..., k_1, k_0$, in that order. The base station divides time into fixed intervals of length T seconds each. We denote the ith such time interval by I_i, and the start of this interval by t_i for $i = 1, 2, ...$, where $t_i = (i-1)T$, with the Castor protocol starting at time $t = 0$.

For each new code-update image (i.e., each new version of the code or binary program image that is to run on every sensor in the network), the base station constructs an n-element one-way hash chain through page-level hashes, $h_0, h_1, h_2, h_3, ..., h_{n-1}$, where h_0 (the head of the hash chain) serves as the commitment to the entire hash chain. Figure 1 shows the sequence of n pages that results from this straightforward approach. The base station computes each page's hash using $H()$, and appends the generated hash to the previous page's payload, e.g., the hash $h_{n-1} = H(p_{n-1})$ is computed and appended to the payload of the previous page, thereby forming page p_{n-2}. The last page, p_{n-1}, does not contain a hash. The base station creates the first page, p_0, by concatenating the payload of p_0 with the hash, h_1, of page p_1. Since one of Castor's objectives is that a correct node should never forward an unverified code-update image, we require that each node verify the authenticity of p_0 (and, recursively, the entire code-update image through the one-way hash-chain property) before propagating p_0 to (or requesting any other page from) the other nodes of the network.

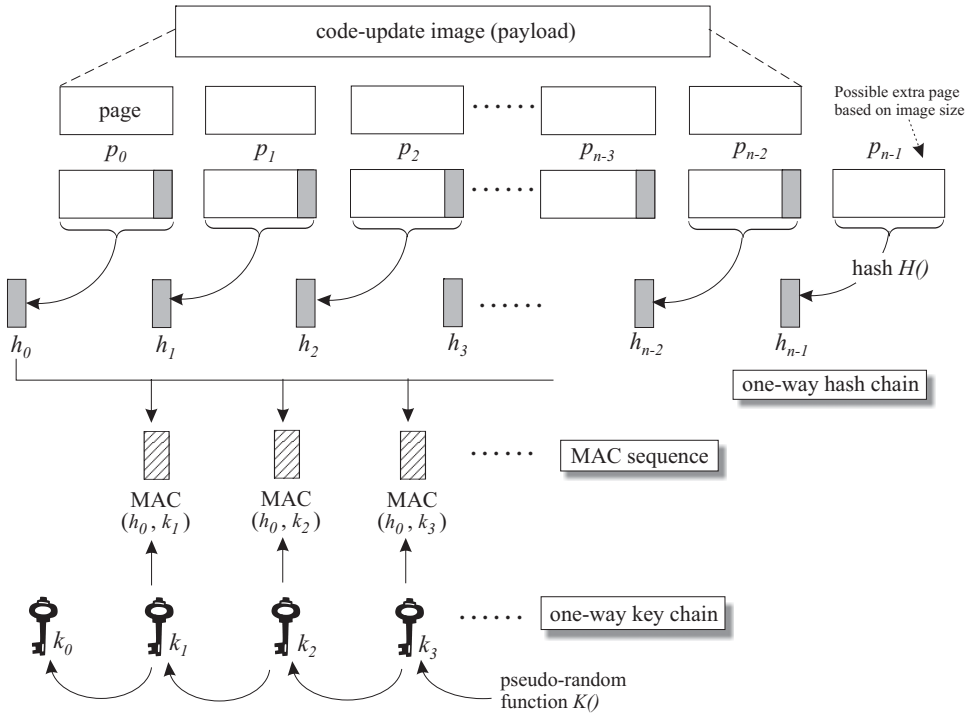

Figure 1. Castor's one-way key chain, one-way hash chain, and a sequence of MACs that together verify the authenticity of the code-update image.

If the base station has a new program image to disseminate in interval I_i, Castor enables a receiving node to authenticate p_0 by having the base station compute MAC(h_0, k_i)), i.e., the MAC of that image's h_0 using the as-yet-undisclosed key, k_i, from the one-way key chain. The base station then disseminates MAC(h_0, k_i), followed by the pages of the image, throughout the sensor network. If a node receives MAC(h_0, k_i) before the base station discloses k_i, then, that node can authenticate the source of h_0 (and, indirectly, the authenticity of the entire program image).

To accomplish this at every node in the network (i.e., the "no-node-left-behind" condition), the key-disclosure interval T must be appropriately selected by Castor. T should be large enough so that every node can receive MAC(h_0, k_i) before the base station discloses k_i. Because Deluge does not ensure the bounded end-to-end propagation time of any packet (including that containing MAC(h_0, k_i)), T will need to be a significantly large value. This, in turn, increases the end-to-end update latency because each node must wait T seconds to authenticate h_0 before starting to receive/propagate the rest of the update. On the other hand, a small T might provide a lower end-to-end update latency, but might "leave nodes behind", i.e., some nodes might be left unable to verify the authenticity of MAC(h_0, k_i) because they might not receive this MAC before the base station discloses k_i.

In Castor, we ensure that all of the nodes are able to verify the MAC (i.e., the "no-node-left-behind" condition) without adversely impacting the end-to-end update latency. We accomplish this by having the base station compute multiple MACs of h_0, each MAC using a unique key of the one-way key chain. For each fresh code-update image that requires dissemination, the trusted base station constructs the one-way page-level hash chain and the MAC sequence using as-yet-undisclosed keys in the one-way key chain. If the base station has a new image to disseminate in interval I_i, then, Castor requires the base station to compute MACs of h_0 not only with k_i, but also with some of k_i's successors[2] (i.e., $k_{i+1}, k_{i+2}, ...$).

The base station releases all of these MACs simultaneously, before the page-dissemination or the key-disclosure starts. The MACs then make their way through the network to all of the sensor nodes. The base station then disseminates pages $p_0, p_1, ..., p_{n-1}$, in that order, to the sensor network. The base station keeps k_i secret until time T_{i+1} (i.e., until the corresponding time interval, I_i, for key, k_i,

[2]Just how many of k_i's successor keys are used to generate MACs for a given code-update image is a tunable parameter that is set at deployment time to ensure the desired "no-node-left-behind" property for that specific sensor network. We discuss this further in our evaluation.

elapses), at which point the base station discloses k_i to all of the nodes in the sensor network. Figure 1 illustrates Castor's approach of using multiple MACs to verify the authenticity of h_0. Note that, even if intermediate keys are lost or perhaps not disclosed by the base station (because the base station did not generate any MACs during that time interval) to reduce overheads, they can be recomputed (if needed) using later received elements of the one-way key chain.

With this scheme, not every node needs to use the same MAC for verifying the code-update image. The purpose of the MAC sequence is to allow nodes closer to the base station to commence the update-verification process using an earlier element in the MAC sequence; nodes further away from the base station will wind up using later elements of the MAC sequence. Also, once the nodes closer to the base station have verified received pages, they can start to propagate verified-to-be-correct pages further into the system. Of course, the same rules apply as before: a node can use MAC(h_0, k_i) for verifying the code-update image only if it receives this MAC before the base station discloses k_i. We also assume that there are enough k_i's, and therefore, enough MACs, in the one-way key chain to facilitate our desired "no-node-left-behind" property. This scheme is similar to applying the technique described in [16] to the problem of securely disseminating code updates.

5 Implementation & Evaluation

We implemented Castor as an extension to Deluge, in order to evaluate Castor against Deluge's baseline performance. In Castor, the size of a page is 1104 bytes (as in Deluge), of which 20 bytes is reserved to encapsulate the hash of the next page of the code-update image. The trusted base station computes full 160-bit SHA1 digests [15] to construct the one-way hash chain across the pages of the image. We used 8-byte symmetric keys with the Matyas-Meyer-Oseas hash compression function (based on the RC5 block cipher) [14] as the pseudo-random function to generate the one-way key chain. We used 4-byte MACs with the CBC-MAC function (based on the RC5 block ciper) because the RC5 block cipher is available in TinySec [5], which is included in the TinyOS distribution.

We used TOSSIM [10] to assess Castor's overheads by determining the end-to-end latency of disseminating the secure code-update image in a sensor network. We evaluated the performance of Castor over grid topologies consisting of NxN sensor nodes (N varying from 2 to 15) spaced 15 feet apart. The grid topologies used for the empirical measurements were generated by `LossyBuilder`, which is provided with TOSSIM. We determined the end-to-end latency of code update by disseminating a standard 24-page update in the sensor network. We used a key-disclosure interval, T, of 60 seconds based on empirical observations that indicated the propagation time of the MAC to most nodes to be less than 60 seconds. To ensure that the "no-node-left-behind" condition is satisfied without undue communication overhead (due to transmitting extra MACs in the sequence), we restricted ourselves to disseminating MACs generated with only the first and the second ("as-yet-undisclosed-until-that-interval") keys of the one-way key chain in that interval; thus, there were two elements in the MAC sequence.

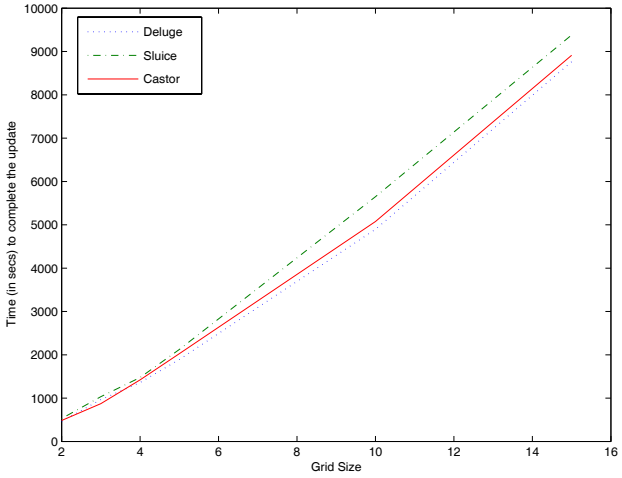

Figure 2. Comparing end-to-end update latency of Deluge, Sluice and Castor for sensor networks of different grid sizes.

Figure 2 shows Castor's end-to-end update latency relative to that of Sluice and Deluge. Castor provides lower update latency than Sluice because it uses a relatively inexpensive MAC verification operation to authenticate the update. Sluice requires a digital-signature verification operation, which we have measured to be as much as 35 seconds in our implementation of Sluice. Furthermore, this signature-verification time is cumulative in Sluice because each node has to verify the digital signature operation before it can disseminate the update further into the network. In Castor, on the other hand, each node receives the MAC packet within the end-to-end propagation time of any packet leaving the base station. Then, each node waits for the corresponding key-disclosure interval to expire and waits to receive the key (i.e., the key propagation-time) to verify the MAC. Castor, therefore, does not exhibit Sluice's cumulative effect of authenticating the first page of the update. We are currently in the process of generating further empirical results that benchmark the computational, communication and energy consumption overheads of Castor and Sluice.

6 Future Work

For a single code-update image, with the desired "no-node-left-behind" criterion in mind, Castor's MAC-sequence scheme can prove to be efficient and can result in lower end-to-end update latencies, as compared to a single-MAC scheme that likely uses a longer key-disclosure interval. However, given that updates (by their very nature as on-demand maintenance, rather than regular house-keeping, activities) are likely to be infrequent/bursty in sensor networks, Castor's scheme might be computationally inefficient across multiple distinct updates. For instance, suppose that the trusted base station uses key k_1 to compute the MAC of the first page of update version 1.0. As time passes, the base station generates keys $k_1, k_2, ...$, in that order, for potential use in generating MACs for any code-update images that are ready for dissemination. Assume that update version 2.0 becomes available at time $t = 999T$ seconds. The base station now generates the MAC(h_0, k_{1000}) (and also MACs with some of the successors of k_{1000}) of update version 2.0. However, by this time, the base station would have generated keys k_1 through k_{999} for generating secure code-update images, if needed. After the base station discloses k_{1000}, a receiving sensor node must perform 999 computations of the pseudo-random one-way function $K()$ on the received k_{1000} to ascertain whether the received k_{1000} is a legitimate element of the one-way key chain whose head is k_0.

Clearly, verifying successive distinct update images can become computationally expensive, a major drawback in a resource-constrained setting. To address this, we are currently designing and implementing a new version of Castor that employs an additional one-way key-chain.

7 Related Work

Several secure-code update protocols have recently emerged that use asymmetric cryptographic primitives to achieve their goals. Sluice [9] uses a combination of a one-way page-level hash chain and a digital signature to authenticate the source of a code-update image. Sluice uses a digital signature to authenticate the contents of the first page and the remaining pages are verified recursively using the page-level hashes embedded in the previous pages. Sluice upholds existing efficiency mechanisms, such as spatial multiplexing, and amortizes the cost of using a digital signature over the entire code-update image.

Secure Deluge [2] also uses a combination of one-way hash chains and a single digital signature amortized over the entire code-update image. However, in Secure Deluge, the one-way hash chain is created over packets rather than pages. As in Sluice, the first packet is digitally signed. The advantage of using a packet-level, rather than page-level, hashes is that each packet can be verified as soon as it is received. On the other hand, the spatial overhead of Secure-Deluge is greater than that of Sluice and requires packets (and not only pages) to be received in order.

Deng et al. [1] have proposed a similar protocol for secure code dissemination in sensor networks. Their approach uses a packet-level hashes similar to Secure Deluge, but allows for out-of-order packet arrivals within a page. This is accomplished by computing a hash tree of per-packet hashes and embedding the root of the hash tree within the corresponding page's payload. A one-way hash chain over pages is then computed similar to Sluice and Secure Deluge. The main advantage is that packets within a page can be received out of order, while simultaneously enabling a packet's verification upon its receipt. On the other hand, the spatial overhead of this scheme is greater than that of Sluice and Secure Deluge.

8 Conclusion

Secure code-update protocols for sensor networks have been based on asymmetric cryptographic primitives such as digital signatures. Even with only one signature used over an entire update image, these protocols exhibit high computation costs, making their usage on resource-constrained sensors undesirable.

Our new approach, Castor, uses symmetric cryptographic mechanisms that are more suited to the resource constraints of sensors. Castor involves a synergistic combination of a one-way hash-chain, a one-way key-chain, and a sequence of MACs with delayed key-disclosure to enable sensors to verify an update's authenticity. We guarantee that no correct node will ever install or forward a compromised part of an update, while addressing the performance issues related to delayed key-disclosure. We evaluate Castor's effectiveness by securing an existing code-update protocol, Deluge, running on the TinyOS platform, and benchmarking Castor's performance against that of Sluice, its counterpart that uses asymmetric cryptography.

References

[1] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *Proc. International Symposium on Information Processing in Sensor Networks*, pages 292–300, Nashville, TN, April 2006.

[2] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the Deluge network programming system. In *Proc. International Symposium on Information Processing in Sensor Networks*, pages 326–333, Nashville, TN, April 2006.

[3] S. Ganeriwal, S. Capkun, C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *Proc. ACM Workshop on Wireless Security*, pages 97–106, 2005.

[4] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. ACM International Conference on Embedded Networked Sensor Systems*, pages 81–94, Baltimore, MD, November 2004.

[5] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proc. ACM International Conference on Embedded Networked Sensor Systems*, pages 162–175, Baltimore, MD, November 2004.

[6] S. S. Kulkarni and M. Arumugam. Infuse: A TDMA based data dissemination protocol for sensor networks. Technical Report MSU-CSE-04-46, Department of Computer Science, Michigan State University, 2004.

[7] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. Technical Report MSU-CSE-04-19, Department of Computer Science, Michigan State University, May 2004.

[8] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[9] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *Proc. International Conference on Distributed Computing Systems*, page 53, Lisbon, Portugal, July 2006.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. ACM International Conference on Embedded Networked Sensor Systems*, pages 126–137, Los Angeles, CA, November 2003.

[11] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proc. Symposium on Networked System Design and Implementation*, pages 1–14, 2004.

[12] D. Liu and P. Ning. Multilevel μTESLA: Broadcast authentication for distributed sensor networks. *ACM Transactions on Embedded Computing Systems*, 3(4):800–836, 2004.

[13] M. Manzo, T. Roosta, and S. Sastry. Time synchronization attacks in sensor networks. In *Proc. ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 107–116, Alexandria, VA, November 2005.

[14] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[15] National Institute of Standards and Technology. Secure hash standard, April 1997.

[16] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium*, pages 35–46, San Diego, CA, February 2001.

[17] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[18] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.

[19] K. Sun, P. Ning, and C. Wang. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *Proc. ACM Conference on Computer and Communications Security*, pages 264–277, Alexandria, VA, October–November 2006.