Castor: Secure Code Updates using Symmetric Cryptosystems

Donnie H. Kim Rajeev Gandhi Priya Narasimhan

May 31, 2007
CMU-CyLab-07-007

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Castor: Secure Code Updates using Symmetric Cryptosystems

Donnie H. Kim
Information Networking Institute
Carnegie Mellon University
Pittsburgh, PA, USA
donniekim@cmu.edu

Rajeev Gandhi
ECE Department
Carnegie Mellon University
Pittsburgh, PA, USA
rgandhi@ece.cmu.edu

Priya Narasimhan
ECE Department
Carnegie Mellon University
Pittsburgh, PA, USA
priya@cs.cmu.edu

Abstract

We present Castor, a secure code-update protocol for sensor networks that exploits symmetric cryptoystems. Through a synergistic combination of a one-way hash-chain, two one-way key-chains with the delayed disclosure of symmetric keys, and multiple message authentication codes (MACs), Castor enables untrusted sensor nodes to verify an update's authenticity and guarantees that no correct node will ever install or forward a compromised part of a code-update image. We describe an implementation of Castor that hardens the TinyOS-based update protocol, Deluge, against node compromise. We experimentally compare Castor's computational and communication costs with those of Deluge and with those of a contemporary secure update protocol, Sluice, that uses asymmetric cryptosystems (digital signatures) instead. Our results demonstrate that Castor incurs reasonable overheads as compared to Deluge, and lower resource usage as well as lower end-to-end update latency as compared to Sluice.

1 Introduction

Sensor networks are often used in distributed embedded environments to address long-term data-collection and sensing needs for diverse purposes such as facility management, home surveillance, patient monitoring, etc. These sensor-network installations are intrinsically long-lived and are, therefore, likely to warrant code updates to the sensors over their lifetimes. Given the scale and deployment of sensor networks, it is infeasible to manually update the code running on every sensor. *Network-reprogramming* (code-update) *protocols*, e.g., the Deluge protocol [3] packaged with TinyOS [8], have emerged as a way to remotely re-flash the code image running on all of the sensors in the network.

These "over-the-air" update protocols employ the radio, the sensor's primary communication channel, and are epidemic in nature. The code-update image typically originates from a trusted source (often a base station). The trusted base station sends sequential fragments of this image to sensors within its "earshot". These sensors forward the received fragments of a code-update image (henceforth called *update-image* or *image*) to their neighboring sensors that, in turn, forward the fragments to their neighbors, and so on, effectively disseminating the entire image to all of the sensors in the network. A malicious sensor node can hijack this epidemic dissemination to inject a corrupt code-image and thereby infect the entire network. Thus, it is critical to secure the code-update protocol so that correctly functioning nodes are not adversely impacted by the presence of compromised ones. Secure update protocols (such as [1, 2] and our own previous work, Sluice [6]) have typically addressed this need through the judicious use of asymmetric cryptosystems, such as digital signatures.

Although computationally less expensive (and, therefore, more suited to resource-constrained sensor networks) than their asymmetric counterparts, symmetric cryptosystems have been largely disregarded for secure network reprogramming. With symmetric cryptosystems, the same key is used for both generating, and later verifying, authentication information ; this requires the sender and the receiver to set up the shared key in advance. An adversary that subverts even a single node can obtain access to, and abuse, the cryptographic materials to subsequently compromise the entire update process and all of the innocent nodes. Thus, to enjoy the benefits of symmetric cryptosystems for secure code updates, we need to develop a way to authenticate the image at correct nodes even in the face of the exposure of keys by compromised nodes in the network.

Contributions. To the best of our knowledge, we are the first to exploit symmetric cryptosystems to secure code updates in sensor networks. Our contributions include:

• *The Castor*[1] *symmetric-crypto secure update protocol.* Castor leverages symmetric cryptosystems to enable untrusted sensor nodes to verify an image's authenticity and to guarantee that no correct node will ever install or forward a compromised part of an image even if the symmetric keys are exposed by compromised nodes in the sensor network. We describe Castor's design, and its current implementation that secures the TinyOS-based update protocol, Deluge.

• *Experimental evaluation of symmetric vs. asymmetric cryptosystems for secure updates.* We evaluate Castor's computational and communication costs with those of Deluge and with those of Sluice, an asymmetric-crypto secure up-

[1] Existing code-update protocols (Deluge, Trickle, Sprinkler and Infuse) evoke a water-based theme. Beavers (genus *Castor*) use lightweight branches and twigs to build dams that control the flow of water. A sluice serves the same purpose as a beaver dam, but is constructed from concrete, steel, etc. In this vein, our new Castor protocol uses lightweight (symmetric crypto) mechanisms to secure the flow of code updates, while our previous Sluice protocol uses more heavyweight (asymmetric crypto) mechanisms.

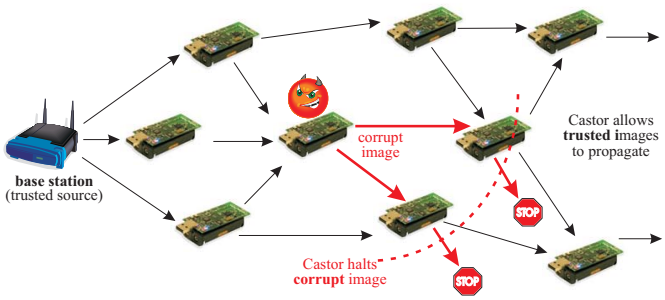


Figure 1. Halting the flow of corrupt images using Castor.

date protocol. Castor exhibits reasonable overheads as compared to Deluge, and lower resource usage and lower end-to-end update latency as compared to Sluice.

2 Problem Statement

We seek to investigate the feasibility of using symmetric cryptosystems for hardening code-update protocols against node compromise. We aim for our resulting Castor protocol to uphold its security guarantees even in the face of its symmetric keys being exposed by compromised nodes. We also seek to benchmark the performance of our new Castor protocol against an asymmetric-crypto counterpart (Sluice). Having developed both Sluice and Castor, we are well positioned to conduct this benchmarking effort.

Security. We require that every node be able to verify whether an image originates from the trusted source (authenticity property) and that this image is unmodified (data-integrity property). Furthermore, no correct node should ever install or propagate any part of an unverified image.

Performance. We desire that nodes perform the verification of the image incrementally, i.e., that they start forwarding pages that have been verified as correct without waiting to receive the remaining pages and suppress pages that have been detected to be corrupt, as shown in Figure 1. Nodes should be permitted to leverage existing update-related efficiency mechanisms, e.g., Deluge's pipelining, if it is available (discussed in Section 3.1). We desire our security enhancements to be sensitive to the resource constraints of sensor nodes and to incur low computational and communication overheads with respect to the insecure version of the update protocol. We aim for the ideal *orphanless* outcome where no node is left behind, i.e., every node in the network receives the appropriate cryptographic material in time to enable it to verify an image's authenticity.

3 Castor

In this section, we outline the design and implementation of Castor, describing the existing building blocks in Section 3.1 and the system model in Section 3.3. We then discuss the ideas underlying authenticating a single update in Section 3.4 and multiple updates in Section 3.5, followed by the details of our implementation and its empirical evaluation in Section 4.

3.1 Building Blocks

Code-update protocols. Deluge, packaged with the TinyOS environment [8], is the most widely available code-update protocol for sensor networks. Castor currently builds upon Deluge primarily because of the latter's widespread availability. We emphasize, though, that Castor's approach is generic enough to secure other update protocols.

Deluge divides the update-image into fixed-size fragments called *pages*. Each page is further divided into fixed-size transmission units called *packets*. In Deluge, the sensor nodes use a three-phase (maintain-request-transmit) handshake to disseminate pages between nodes. In its maintain phase, every node broadcasts an advertisement that contains the version number of the image and the number of the highest page available for forwarding. A receiving nodes uses the advertisement to detect the availability of a newer version of the image and then asks for the pages of the newer image in the node's request phase. In its transmit phase, a node that has already received some/all of the pages of the newer image disseminates any requested pages to its neighbors. Deluge uses a pipelining mechanism where nodes can start to forward received pages without needing to wait to receive all of the pages of the image. In-order receipt of pages is enforced, i.e., a node can neither accept nor forward page N until it has first received pages $0, 1 \ldots, N-1$, inclusive. This requirement reduces the amount of state that each node must maintain to track how much of the image has been locally received/transmitted.

One-way hash/key chains. The one-way chain concept [5] involves a one-way function $G()$ that is easy to compute, but computationally difficult to invert. A one-way chain $\mathbb{a}$ of length $m+1$ is generated by repeatedly applying $G()$ to some (randomly selected) initial value, a_m, to generate a sequence where $a_j = G(a_{j+1}) = G^{m-j}(a_m), 0 \leq j \leq m-1$. The chain is generated in the order $a_m, a_{m-1}, a_{m-2}, \ldots, a_1, a_0$, but revealed in the reverse order. The head, a_0, of $\mathbb{a}$ must be a trusted value because it serves as a commitment to the entire chain. If a node confirms a_0 to be the authentic head of a one-way chain, it can easily determine whether a given a_r is the rth element of the chain by verifying whether a_0 results from applying $G()$ successively r times to a_r. Thus, a node holding an authentic a_0 can recursively verify the authenticity of the remaining elements of $\mathbb{a}$.

Thus, due to the one-way chain property, (i) every node can verify whether a received a_i is the ith element of $\mathbb{a}$ by examining whether $G^i(a_i)$ equals a_0, where a_0 is authentic, (ii) in no way can a node derive a_{i+1} from a_i, for any $i > 0$, and (iii) intermediate missing elements of $\mathbb{a}$ can be derived using later elements of $\mathbb{a}$.

Castor uses one-way chains in two ways: (i) where $G()$ represents a hash function, $H()$, and the elements of the chain are hashes, thereby generating a *one-way hash chain*, $\mathbb{h}$ and (ii) where $G()$ represents a pseudo-random function, $F()$, and the elements of the chain are symmetric keys, thereby generating a *one-way key chain*.

Symmetric-crypto authenticated broadcast. Authenticated broadcast protocols (e.g., TESLA [13], μTESLA [14]) use symmetric keys to authenticate the sender of a message in the face of node compromise. The protocols involve a one-way key chain, $\{k_0, k_1, \ldots, k_m\}$, whose elements are used by a trusted source to compute message authentication codes

(MACs) over a transmission unit called a message. The message is then broadcast, along with the MACs, to all of the nodes in the network. The sender keeps the keys secret until all of the nodes have received the MACs. Only upon the trusted source's disclosure of the keys can the receivers verify the received message. Disclosed keys are verified against k_0, the initial commitment to the key-chain. k_0 is securely distributed to all of the nodes as a part of system bootstrapping. In TESLA, k_0 is digitally signed by the trusted source and broadcast to all of the the nodes during bootstrapping. In μTESLA, the source unicasts k_0 to each nodes using pair-wise symmetric keys, with one key shared between the source and each node. Recent work [10] overcomes the attendant scalability issues with μTESLA by using broadcast instead of unicast.

For a node to be able to verify a message's authenticity, it must receive the message *before* the corresponding key (i.e., with which the MAC of the message is computed) is disclosed by the source. This delayed key-disclosure strategy hinges upon loose time synchronization across the network, i.e., all of the nodes must have an upper bound on the trusted source's local time. Time synchronization allows a node to verify whether the key used to authenticate a message has already been disclosed by the source at the time of the message's receipt. In the latter case (i.e., key disclosure by the time of message receipt), the node can no longer rely on the message's authenticity even if the key used to authenticate the message is a valid element of the one-way key chain. The trusted source determines the key-disclosure interval based on its *a priori* knowledge of the worst-case end-to-end propagation time of a message from itself to *all* of the nodes in the network.

While Deluge ensures that a new image is reliably disseminated, it does not bound the amount of time required to disseminate the image to every node in the network. Without this upper bound, it is difficult for the source to choose an appropriate key-disclosure interval. An incorrectly chosen key-disclosure interval could either result in number of "orphan" nodes that are unable to authenticate the image or can increase the *end-to-end update latency* (the latency to propagate the image from the source to every node in the system). Due to the issues in tuning the key-disclosure interval, previous secure-update protocols (including our own Sluice) avoided the use of symmetric cryptosystems. Castor overcomes these problems, as described in Section 3.4.

3.2 Rationale for Approach

To authenticate an image, Castor starts with the fundamental idea of using a hash-chain, $\mathbb{h}$, across the pages of the image, with h_0 (the head element of $\mathbb{h}$) being the hash of the first page of the image. The remainder of Castor's approach then revolves around ensuring that h_0 is indeed a trusted value that can serve as a legitimate commitment to all of $\mathbb{h}$. This is often done through mechanisms that allow a sensor node to verify h_0's authenticity.

In Castor, these mechanisms take the form of (i) an m-element *one-way verification key-chain*, $\mathbb{k}$, with delayed key-disclosure, and (ii) an s-element sequence, $\mathbb{M}$, of MACs of hash h_0, where each MAC is computed over h_0 using a distinct element of $\mathbb{k}$. While key-chain $\mathbb{k}$ serves to authenticate

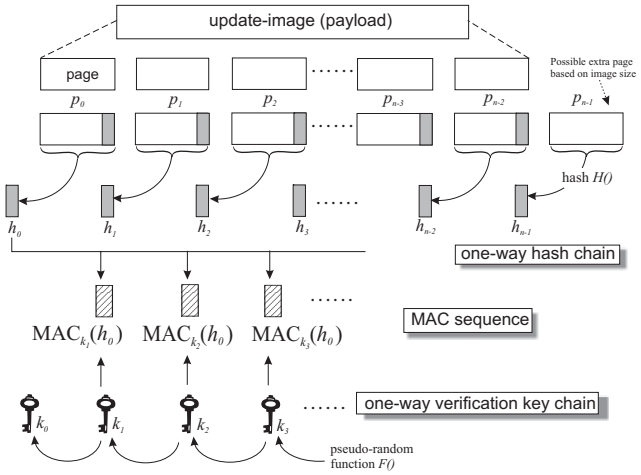


Figure 2. Castor's one-way hash chain, $\mathbb{h}$, one-way verification key-chain, $\mathbb{k}$, and MAC sequence, $\mathbb{M}$, that collectively verify a single update' authenticity (Section 3.4.)

a single update-image, Castor also employs a second key-chain called the *one-way backbone key-chain*, $\mathbb{K}$, to mitigate the computational costs associated with authenticating successive update-images. The verification and backbone key-chains have their associated key-disclosure intervals, T and T', respectively. The synergistic operation of these mechanisms is described in detail in Section 3.5.

3.3 System Model

We assume that sensor nodes are connected via an insecure wireless medium and are susceptible to compromise. An adversary can subvert an arbitrary number of nodes, and can gain control of their cryptographic material. Castor's design covers different kinds of adversarial behavior, e.g., an adversary can inject an arbitrary number of corrupt packets or corrupt images into the system, can withhold/delay/modify an arbitrary number of packets or images, and can eavesdrop on the communication of other nodes in the network. We place no restrictions on the number of malicious nodes, their respective locations, or the degree of connectivity or collusion between them. We do not currently address the confidentiality of an update or threats (e.g., battery-drain attacks, denial-of-service) that might drain the memory buffers at each node.

We assume that there is a single trusted sender (e.g., a base station) that acts as the source of the images. We assume that the base station and every node is bootstrapped with the values of T (the key-disclosure interval for $\mathbb{k}$), T' (the key-disclosure interval for $\mathbb{K}$), k_0 (the head element of $\mathbb{k}$), K_0 (the head element of $\mathbb{K}$), and knows the one-way pseudo-random function $F()$ and the one-way hash function $H()$.

Because we exploit authenticated broadcast in Castor, we assume that the base station and the nodes in the network are loosely and securely time synchronized [15]. If the sensor-network application already uses time synchronization (e.g., TDMA-based applications), Castor can simply exploit this facility.

3.4 Authenticating a Single Update

This section describes how the one-way hash chain, $\mathbb{h}$, the one-way verification key-chain, $\mathbb{k}$, and the MAC sequence, $\mathbb{M}$, work together to enable a node to verify the authenticity of h_0 for a given update-image.

For each image, the base station constructs $\mathbb{h}$ out of the 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. The base station computes the hash of each page using the hash function $H()$, and embeds the generated hash in the previous page's payload, i.e., the base station computes the hash $h_{n-1} = H(p_{n-1})$ and appends the hash to the payload of the previous page, p_{n-2}, to form the new page $p_{n-2} = p_{n-2} \mid h_{n-1}$, where the symbol $\mid$ donates concatenation ($a \mid b$ denotes that b is concatenated with a, or more specifically, appended to a). The base station computes the next hash $h_{n-2} = H(p_{n-2} \mid h_{n-1})$ to generate page $p_{n-3} = p_{n-3} \mid h_{n-2}$, and so on (note that the last page, p_{n-1}, will not contain a hash). To guarantee that a correct node will never forward an unverified part of an image, Castor requires each node to first verify the authenticity of p_0 before the node can forward p_0 or any of the following pages, $p_1, p_2, ...$ to other nodes.

To authenticate p_0, Castor uses MACs of h_0 computed with the keys that are the elements of $\mathbb{k}$. The base station generates $\mathbb{k}$ by selecting a key, k_m, at random and repeatedly applying a pseudo-random function, $F()$, to k_m to generate the earlier keys $k_{m-1}, k_{m-2}, ..., k_1, k_0$, in that order. Key k_0 serves as a commitment to the entire key-chain, $\mathbb{k}$, and is assumed to be known to all of the nodes. The base station divides time into fixed intervals of length T seconds each. We denote the ith such time interval by $I_i = [(i-1)T,\ iT]$ for $i = 1, 2, ...$, and we assume that Castor starts execution at time $t = 0$. The base station uses the key, k_i (or a later key), to compute the MAC of any data transmitted during the interval I_i that needs to be authenticated, and discloses k_i only at the end of the interval I_i (i.e., at $t = iT$ seconds). Each node can verify the authenticity of a given MAC that is computed with k_i by verifying that (i) k_i is a valid element of $\mathbb{k}$, and (ii) this node receives the specific MAC before the base station discloses k_i.

Assume that the base station has a new image to disseminate at the beginning of the time interval I_i (if the new image is available in the middle of I_i, the base station can wait until the beginning of the next time interval to initiate the dissemination). The base station computes $MAC_{k_i}(h_0)$, i.e., the MAC of that image's h_0 using the as-yet-undisclosed key, k_i, that is an element of $\mathbb{k}$. The base station then broadcasts $MAC_{k_i}(h_0) \mid h_0$ to all of the nodes in the sensor network. In turn, receiving nodes blindly forward the unauthenticated MACs[2] to their neighbors, and so on.

[2]Suppose that nodes forwarded authenticated MACs instead. Recall that once a node authenticates a MAC, the MAC has effectively expired and become useless in securing the code-update at other nodes, rendering them "orphans". Thus, forwarding only authenticated MACs hinders us in accomplishing our objective of an orphanless outcome (Section 2). However, we recognize that forwarding unauthenticated MACs makes Castor vulnerable to denial-of-service attacks, where an attacker can consume network bandwidth by injecting malformed/bogus MACs.

If a node receives $MAC_{k_i}(h_0) \mid h_0$ before the base station discloses k_i, then, that node can authenticate h_0 (and, hence, p_0) once it receives k_i. Once a node authenticates p_0, it can authenticate the remaining pages of the image using the one-way property of $\mathbb{h}$, i.e., h_1 contained in p_0 authenticates p_1, h_2 contained in p_1 authenticates p_2, and so on. Thus, we require every node to authenticate p_0 (through $MAC_{k_i}(h_0) \mid h_0$) before initiating the "real" update process, i.e., starting to request the pages, $p_0, p_1, p_2, ...$, of the image. Once a node authenticates p_0, there are no temporal requirements on the propagation delay of any other page. Thus, Castor does not require the timely dissemination of any data other than the packet containing $MAC_{k_i}(h_0) \mid h_0$.

We note that Castor still requires that each node receive this packet (containing $MAC_{k_i}(h_0) \mid h_0$) before the key k_i is disclosed by the trusted base station. Thus, the key-disclosure interval, T, needs to be selected carefully to ensure that every node is able to verify the authenticity of the image by authenticating h_0. T should be large enough so that every node can receive $MAC_{k_i}(h_0) \mid h_0$ before the base station discloses k_i. T will necessarily be large for sensor networks consisting of hundreds of nodes. A large T will increase the end-to-end update latency because even if a node receives $MAC_{k_i}(h_0) \mid h_0$ quickly, it must still wait for T seconds to authenticate h_0 before the node can start to receive/propagate the rest of the image. On the other hand, while a small T might provide a lower end-to-end update latency, it can prevent some nodes from authenticating $MAC_{k_i}(h_0) \mid h_0$ because these nodes might not receive the MAC before the base station discloses k_i. Because Deluge does not place a bound on the propagation time of a packet within the network, Castor (which builds on Deluge) must use a relatively large T to ensure that all of the nodes are able to authenticate the image.

We overcome the problem of fine-tuning the value of T in Castor by ensuring that the base station computes multiple MACs of h_0 (henceforth called a *MAC sequence*, $\mathbb{M}$) and disseminates these MACs simultaneously. Thus, if the base station has a new image to disseminate at the beginning of the interval I_i, then, the base station computes MACs of h_0 not only with k_i, but also with some of k_i's successors[3] (i.e., $MAC_{k_{i+1}}(h_0)$, $MAC_{k_{i+2}}(h_0)$, $.....MAC_{k_{i+s-1}}(h_0)$). Figure 2 illustrates Castor's approach of using multiple MACs to verify the authenticity of h_0.

For each image, the base station constructs $\mathbb{h}$ over the pages of the image and computes $\mathbb{M}$ using several as-yet-undisclosed elements of $\mathbb{k}$. The base station releases all of the elements of $\mathbb{M}$ simultaneously, in the form of a specially designated *multi-MAC structure*[4], within the interval I_i, *before* disseminating any page of the image. The base station keeps k_i secret until the end of the time interval I_i and discloses k_i only after $t = iT$ seconds (similarly, the base station discloses k_{i+1} at $t = (i+1)T$, and so on). As soon as a node verifies one of the MACs in $\mathbb{M}$, it can start requesting

[3]For simplicity, Castor currently uses MACs computed with consecutive keys of k_i. Just how many, and exactly which, of k_i's successors in $\mathbb{k}$ are used to generate MACs for a given image is a parameter that should be tuned to ensure the orphanless outcome, i.e., that all of the nodes in the network are able to verify the MAC.

[4]Contains $MAC_{k_i}(h_0) \mid MAC_{k_{i+1}}(h_0) \mid ... \mid MAC_{k_{i+s-1}}(h_0)$

the following pages; in turn, the base station disseminates pages $p_0, p_1, ..., p_{n-1}$ in that order, into the network. The use of multiple MACs allows nodes closer to the base station to commence the verification of the image by using an earlier element in $\mathbb{M}$; nodes further away from the base station will likely end up using later elements of $\mathbb{M}$. Furthermore, once nodes closer to the base station have verified pages of the image, they can start to propagate these verified-to-be-correct pages further into the network. Of course, the same rules apply as before: a node can use $MAC_{k_r}(h_0)$ for verifying the image only if it receives this MAC before the base station discloses k_r.

The number of MACs disseminated, and the keys used to compute these MACs can be chosen by the base station to meet two objectives: (i) nodes closer to the base station can authenticate the image using an earlier element of $\mathbb{M}$ and can start processing the image quickly, and (ii) nodes further away from the base station can verify the image's authenticity by using a later element of $\mathbb{M}$. The base station only needs a coarse estimate of the end-to-end propagation latency of the multi-MAC structure to all of the nodes in the network. To compute the last MAC in $\mathbb{M}$, the base station can choose a key whose disclosure time is beyond the estimated propagation latency of the multi-MAC structure. The use of multiple MACs to allow nodes closer to the source to authenticate data has been proposed in [13]. We use the multiple-MAC concept to overcome Deluge's unbounded end-to-end propagation latency.

3.5 Authenticating Multiple Updates/Versions

For a single image, Castor's use of multiple MACs can be efficient and result in a lower end-to-end update latency, as compared to a single-MAC scheme that likely uses a longer key-disclosure interval. However, the process of verifying whether a key is part of $\mathbb{k}$ might be computationally inefficient across updates especially if updates are infrequent. For instance, suppose that the base station uses key k_i to compute the MAC of the first page of update version 1.0 in the interval I_i, and that update version 2.0 becomes available in the time interval I_{i+1000}. The base station uses key k_{i+1000} to generate $MAC_{k_{i+1000}}(h_0)$ for update version 2.0. After the base station discloses k_{i+1000}, a receiving node must perform 1000 computations of the pseudo-random function, $F()$, on the received k_{i+1000} (i.e., until these computations yield k_i) in order to verify whether the received k_{i+1000} is an element of $\mathbb{k}$[5].

To address this "cross-update" inefficiency, Castor adopts a two-level key-chain scheme (we note that the concept can be easily extended to use multiple levels of key chains). Essentially, Castor leverages an additional *one-way backbone key-chain*, $\mathbb{K}$, which is used to periodically (with period $\gg T$) authenticate the intermediate elements of key-chain $\mathbb{k}$ (whose elements are used to generate the MACs for the update-image). Because $\mathbb{K}$ authenticates intermediate elements of $\mathbb{k}$, sensor nodes do not need to perform many pseudo-random function computations to verify sparsely used elements of $\mathbb{k}$.

To illustrate this, let $K_0, K_1, \ldots, K_r$ denote the elements of $\mathbb{K}$, where K_0 serves as the commitment to $\mathbb{K}$ and is assumed to be known to all of the nodes. The base station periodically discloses the elements of $\mathbb{K}$ with a period $T' = qT$ (for simplicity, we chose the key-disclosure frequency for $\mathbb{K}$ to be a multiple of that for $\mathbb{k}$, although this is not necessary). Suppose that the base station uses the backbone key K_i for authenticating the intermediate verification-key k_{iq} during the interval $I' = [(i-1)T',\ iT']$, and discloses the key, K_i, at time $t = iT'$. The base station sends periodic messages that contain a MAC of a future element of $\mathbb{k}$ which is computed using a future element of $\mathbb{K}$.

Assume that, at time $t = iT' = qiT$, the base station broadcasts a backbone packet P_i containing $K_i \mid k_{qi} \mid MAC_{K_{i+1}}(k_{q(i+1)})$ to all of the nodes in the network. This packet discloses the backbone key K_i, the verification key k_{qi} (where both keys are revealed during their respective disclosure intervals $t = iT' = qiT$), and a MAC of the future verification key $k_{q(i+1)}$ computed using the future backbone key K_{i+1}. Each node can authenticate k_{qi} by using the contents of backbone packets P_{i-1} and P_i (assuming that the node receives P_{i-1} before $t = qiT$ and receives P_i before $t = q(i+1)T$). To authenticate the intermediate verification key k_{qi}, each node (i) verifies whether K_i is an element of $\mathbb{K}$ by verifying whether K_{i-1} equals $F(K_i)$, and (ii) authenticates k_{qi} by computing $MAC_{K_i}(k_{qi})$ using the keys k_{qi} and K_i embedded in P_i, and then verifying the results against $MAC_{K_i}(k_{qi})$, which is embedded in P_{i-1}. Note that these backbone packets represent out-of-band traffic, with respect to the normal data packets that form the pages of the update-image.

The advantage of using $\mathbb{K}$ to authenticate the intermediate elements of $\mathbb{k}$ is that fewer computations need to be performed at the nodes to verify whether a received verification key is a valid element of $\mathbb{k}$. Each node needs to apply the pseudo-random function, $F()$, recursively to a received verification key until the process yields the authenticated intermediate element of $\mathbb{k}$. Section 4.3 further discusses the computational advantages of using our two key-chain scheme, as opposed to using a single key-chain.

3.6 Accommodating "Latecomers"

The use of MACs whose trustworthiness/utility expires at the end of a time interval (i.e., the key-disclosure interval) poses a problem in sensor networks where *latecomers* can exist. Latecomers are nodes that join the network at any time or those that awaken after having effectively left the system while in an extended sleep mode. To ensure that such latecomers are able to receive a secure code-update, the base station might need to periodically broadcast a MAC that enables the authentication of h_0. However, this will undoubtedly increase communication costs for secure updates.

An alternative approach is to exploit Castor's backbone key-chain to authenticate h_0. Specifically, the base station modifies the backbone packet, P_i, to include the MAC of h_0, i.e., $P_i = h_0 \mid K_i \mid k_{im} \mid MAC_{K_{i+1}}(h_0 \mid k_{(i+1)m})$. Latecomers can authenticate the image using P_i without significantly increasing Castor's communication costs because P_i is sent

[5]To reduce communication costs, the base station need not disclose any of the intermediate keys $k_{i+1}, ..., k_{i+999}$. However, even if the base station discloses these keys, nodes will end up performing 1000 computations of $F()$ to verify whether $k_{i+1}, ..., k_{i+999}, k_{i+1000}$ are valid elements of $\mathbb{k}$.

rather infrequently (due to the length of the backbone key-disclosure interval). The drawback of using the backbone key-chain in this manner is that latecomers will take longer to finish an update as compared to the non-latecomer nodes in the network because latecomers will have to wait for an average of $T'/2$ seconds before they can authenticate the first page of the update-image. Another strategy to accommodate latecomers is to embed the digital signature of h_0 (as Sluice and SecureDeluge do) with the MACs in the advertisement. Non-latecomer nodes can simply use the MACs to verify the authenticity of the image with lower computational costs, while latecomers can use the digital signature instead for the same purpose (with the anticipated greater costs of verifying a signature operation).

3.7 Implementation in TinyOS

The base station in Castor divides an image into fixed-size pages of 1104 bytes (the default Deluge page-size), of which 20 bytes are reserved by Castor for encapsulating the hash of the next page of the image. The base station computes full 160-bit SHA1 digests [12] to construct $\mathbb{h}$. We use 8-byte symmetric keys and the Matyas-Meyer-Oseas hash compression function [11], which is based on the RC5 block cipher, as the pseudo-random function, $F()$, to generate $\mathbb{k}$. We compute 4-byte MACs using the CBC-MAC function, which is also based on the RC5 block cipher. Our use of these specific cryptographic algorithms was motivated by the availability of the RC5 block cipher in TinySec [4], which is packaged with the TinyOS distribution.

We modified Deluge to include Castor's mechanisms for authenticating images. Specifically, we extended Deluge's advertisements to embed Castor's multi-MAC structure and the verification keys disclosed by the base station. Furthermore, we also embed h_0 in the advertisement to allow a node to verify the MAC as soon as the node receives the key used to compute the MAC (rather than having the node wait to receive the entire page, p_0, and then compute the hash of that page). Denoting Deluge's original advertisement as Adv, Castor's advertisement, Adv', now encapsulates
$Adv \mid h_0 \mid MAC_{k_i}(h_0) \mid MAC_{k_{i+1}}(h_0) \mid ... \mid MAC_{k_{i+s-1}}(h_0)$.

The key disclosed by the base station replaces the MAC (that is computed using the key) in the advertisement since a MAC expires the moment that the base station discloses the key used to compute that MAC. Castor's current implementation uses 10 bytes of metadata (primarily the index of the embedded key within $\mathbb{k}$), 20-byte hashes, 8-byte symmetric keys, and 4-byte MACs. To reduce communication costs, every sensor node removes any expired MACs from the advertisement before forwarding it to other nodes.

4 Empirical Evaluation

We used TOSSIM [7] to simulate Castor's secure dissemination of update-images. Our simulation topologies consisted of sensor nodes spaced 15 feet apart in an $N \times N$ grid, with N ranging from 10 to 20. We used `LossyBuilder`, provided with TOSSIM, to generate these grid topologies. In the interests of space, we discuss only our results and experiments for the 10×10 sensor network in this paper; our results for the other topologies lead to similar conclusions.

Because TOSSIM does not model execution time, we modified TOSSIM's event queue to insert appropriate time delays to simulate the time taken by each node to perform the cryptographic operations. Specifically, we inserted a delay of 0.5ms for verifying a MAC and for computing a pseudo-random function (to verify whether a received key is part of a key-chain), and 200ms for computing a hash [4]. TOSSIM intentionally uses randomized and staggered times to initiate the boot sequence of the nodes in the network in order to prevent artificial time synchronization [7] between the nodes. We modified TOSSIM to eliminate this randomness in order to simulate time synchronization between all the nodes in the sensor network. Because TOSSIM models a previous generation of the TinyOS platform, our results are useful for benchmarking Castor against Deluge and Sluice, rather than for measuring Castor's absolute performance.

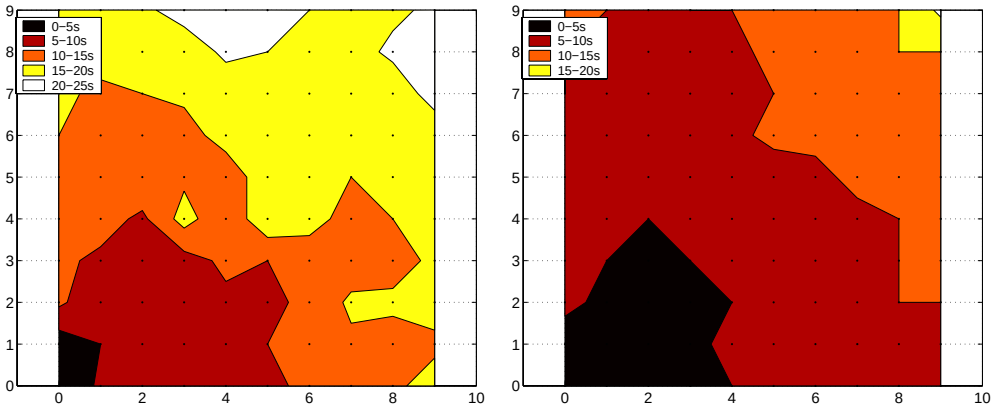
Num of MACs	Spatial Overhead (bytes)	Percentage of runs completed in			
		< 30s	$30-60$s	$60-90$s	$90-120$s
1	34	90%	10%	0	0
2	38	80%	20%	0	0
3	42	20%	20%	40%	20%
6	54	10%	30%	30%	30%

Table 1. Advertisement's propagation latency for varying advertisement-packet size in a 10×10 network.

We benchmark different metrics – end-to-end update latency, communication cost and computational cost – for Castor, Sluice and Deluge. We evaluate the impact on Castor's performance of different choices of parameters such as the key-disclosure interval (T). We also vary the number of MACs embedded in the advertisement, by using either 1 MAC (called the 1-MAC advertisement) or 6 MACs (called the 6-MAC advertisement) in the multi-MAC structure. Our experiments begin by introducing a new 4-page image from a desktop computer to a base-station node attached to the desktop computer. We measure the image's propagation time relative to the first advertisement from the source. To enable benchmarking across the three protocols (Castor, Sluice, Deluge), we intentionally select a program image that will result in 4 pages, regardless of the protocol; basically, the image is padded such that Castor's/Sluice's cryptographic material does not affect the number of pages in the image, as compared to Deluge. No concurrent services, other than those required by Castor execute during the experiments. The results presented here do not consider the computational or communication overhead of time synchronizing the sensor nodes or ensuring that the nodes stay synchronized.

4.1 Advertisement's Propagation Latency

The key-disclosure interval, T, is an important parameter in Castor as it enables the desired orphanless outcome and influences the end-to-end update latency. The correct choice of T depends on the latency to propagate the advertisement from the base station all of the nodes in the network and on the number of MACs embedded in the advertisement. We measured the advertisement's propagation latency while varying the number of MACs embedded in the advertisement. Figures 3(a) and 3(b) show the advertisement's propagation latency for Castor and Deluge, respectively.



(a) (b)

Figure 3. Advertisement's propagation latency in a 10×10 network: (a) Castor with 6 MACs and T =30s, (b) Deluge.

The isolines show the temporal wavefront of the advertisement, e.g., the isoline for 5-10s graphically "joins" all of the nodes that receive the advertisement within 5-10s of the base station sending the advertisement. To study the relationship between the advertisement's propagation latency and the advertisement's size, we conducted 10 additional simulations where we vary the number of MACs embedded in the packet. The results of these simulations, shown in Table 1, indicate that in 90% of the simulations, the 1-MAC advertisement reached all of the nodes in less than 30s. In constrast, the 6-MAC advertisement reached all of the nodes of the network in less than 30s, in but 10% of the simulations.

4.2 Communication Costs

Castor's communication overhead, with respect to Deluge, is primarily due to the multiple MACs and the keys used to compute these MACs. Because Castor's current implementation embeds the multi-MAC structure and the keys in the advertisement, Castor's communication overhead effectively amounts to the extra number of bytes in the advertisement.

As shown in Table 1, the 6-MAC advertisement with unexpired MACs has a spatial overhead of 54 bytes (due to the embedded 10-byte metadata, the 20-byte hash, $6 \times 4 = 24$ bytes for the 6 unexpired MACs). On the other hand, the 1-MAC advertisement with an unexpired MAC has a spatial overhead of 34 bytes. In our current implementation, the base station always embeds the 10-byte metadata and the last symmetric key disclosed by the base station in the advertisement. This causes Castor's advertisement to be 18 bytes larger than that of Deluge, even in the steady state (i.e., when no new images need to be disseminated, but advertisements for the current image are still being sent).

We compare the average number of bytes sent by a sensor node in Castor and Deluge in Figure 4(a) for the 1-MAC advertisement with $T = 180$s, and in Figure 4(b) for the 6-MAC advertisement with $T = 30$s. From these graphs, Castor exhibits a constant communication overhead even after all of the MACs used to authenticate the image have expired because the advertisement still carries the metadata and the last verification key disclosed by the base station. These graphs also indicate the cost of the multi-MACs structure and the keys embedded in the advertisement. Advertisements are frequently used by the underlying Deluge protocol (even in the steady state) to announce the current version of the image. Thus, an increase in the advertisement's size causes a proportional increase in Castor's communication costs. We are currently investigating strategies to reduce these costs by omitting the key from the advertisement during the steady state or by using a separate packet to carry the cryptographic materials alone.

4.3 Computational Costs

To examine the possible computational benefits of using symmetric over asymmetric cryptosystems in securing code updates, we first calculate the additional number of cryptographic operations in Castor. We then use the actual TinySec hash- and key-computation latencies [4] (Castor exploits TinySec's hash and pseudo-random functions) to estimate Castor's computational overheads. In comparing Castor and Sluice, we focus on the computational cost of verifying authenticity of h_0 rather than that of the whole image because the protocols primarily differ in their authentication of h_0 .

The number of cryptographic operations needed in Castor depends on the inter-arrival times of successive update versions. Assume that the base station disseminates version 1.0 at time $t = 0$ and that version 2.0 becomes available at time $t = t_{diff}$. For simplicity, we assume that t_{diff} is an integral multiple of T. We express t_{diff} as $uT' + vT$, where $u = \lfloor t_{diff}/T' \rfloor$, $v = (t_{diff} - uT')/T$, and $\lfloor\ \rfloor$ represents the floor function. Since version 1.0 is disseminated at $t = 0$ and version 2.0 is disseminated at time $t_{diff} = uT' + vT = (un + v)T$, the base station uses k_1 to authenticate version 1.0 and k_{un+v+1} to authenticate version 2.0.

Starting with the simplest case, we first determine the number of Castor computations at the sensor nodes with a 1-MAC advertisement and without the backbone keychain. Suppose that the base station does not disclose keys $k_2, k_3, ..., k_{un+v}$ because it did not use these keys to generate any MACs (we note that the computational costs are independent of whether or not the keys are disclosed by the base station). The base station generates $MAC_{k_{un+v+1}}(h_0)$ for version 2.0; after the base station discloses k_{un+v+1}, a receiv-

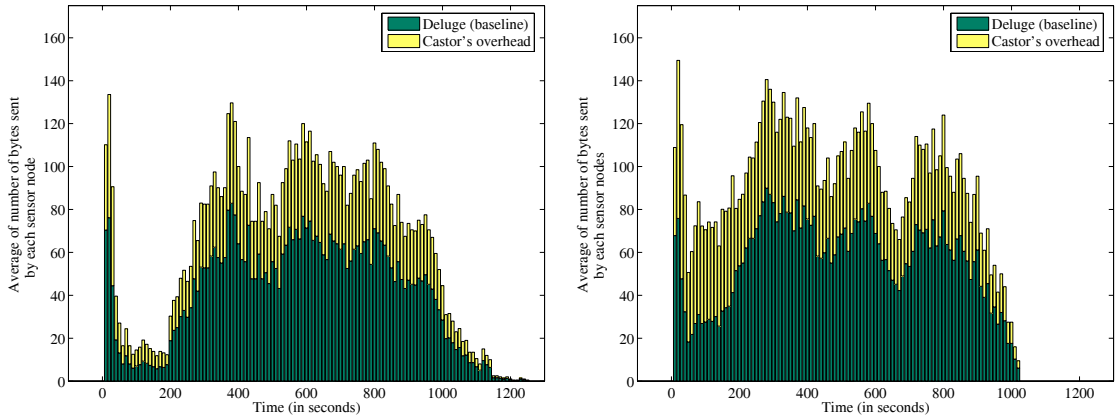


Figure 4. Average number of bytes sent by a node in Deluge and Castor for Castor's (a) 1-MAC ($T = 180$s) case, and (b) 6-MAC ($T = 30$s) case.

ing node must perform $un + v$ computations of the pseudo-random function, $F()$, to verify whether the received k_{un+v+1} is an element of the the one-way verification key chain, $\mathbb{k}$. Thus, in the absence of the backbone key-chain, $\mathbb{K}$, Castor requires $un + v$ pseudo-random computations and one MAC-verification operation to be performed at the nodes.

The use of $\mathbb{K}$ reduces the number of pseudo-random computations; only u computations of $F()$ are needed to verify that k_{un} is an element of $\mathbb{k}$. Nodes then need only $v+1$ additional computations of $F()$ to verify if k_{un+v+1} is also an element of $\mathbb{k}$. Thus, Castor requires at most $u+v+1$ cryptographic operations (in our case, RC5) for each image, resulting in a computational overhead of $\sim 0.26(u+v+1)$ms on Mica2 sensor nodes [4]. On the other hand, the use of digital signatures (specifically, TinyECC [9]) instead of MACs requires at least 2.418s on MicaZ sensor nodes (that use the same processor as the Mica2 nodes) and 5.056s on TelosB nodes for signature verification.

Castor's use of multiple MACs slightly increases the computational overhead as compared to using only a single MAC. If the base station computes s MACs, i.e., $MAC_{k_{un+v+1}}(h_0)$, $MAC_{k_{un+v+2}}(h_0)$,..., $MAC_{k_{un+v+s}}(h_0)$, Castor requires an additional $s-1$ pseudo-random function computations and one MAC-verification operation for each node, beyond the single-MAC case. Although some nodes might use a MAC computed with a key that has an earlier disclosure time, they must still verify the other keys before propagating them to the other nodes of the network; this results in increased computational costs.

4.4 End-to-End Update Latency

Castor's update latency increases with T. Figure 5(a) shows Castor's latency to disseminate each of the following – the advertisement packet containing the MACs, the key used to verify the MAC, the first page and the entire 4-page image – to all of the nodes in the network. In our experiments, we use a 1-MAC advertisement and vary the value of T, starting with $T = 30$s (from Table 1, this causes the MACs to take 30s to reach every node).

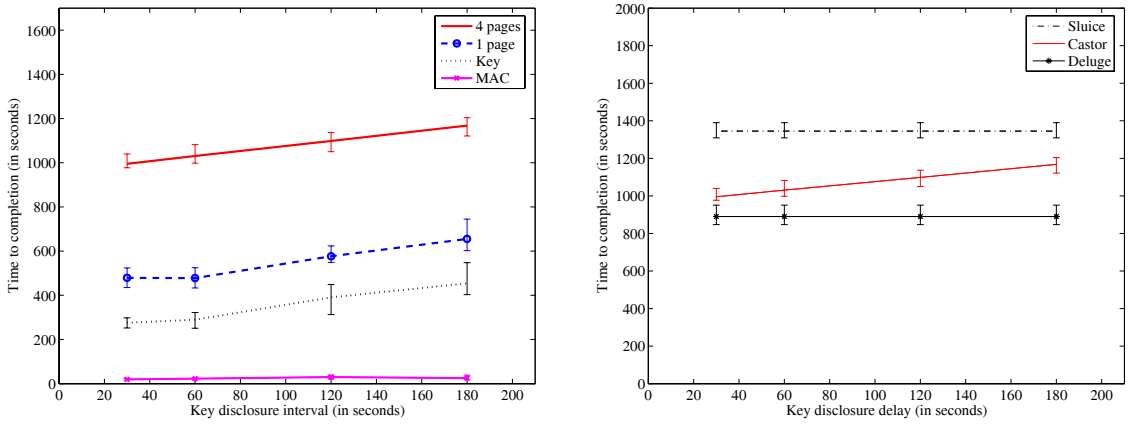
The graphs shows that the update latency increases with T. With a 1-MAC advertisment, nodes closer to the base station must wait until every other node in the network has received the advertisement. Effectively, this increases the time taken to disseminate the first page, and therefore, increases Castor's end-to-end update latency.

Castor's update latency is lower than Sluice's. Figure 5(b) shows the time taken to disseminate the 4-page image using Deluge, Sluice and Castor. We see that Castor exhibits a lower end-to-end update latency than Sluice because Castor's MAC verification takes less time (200ms) than Sluice's digital-signature verification (which is 5s, based on measurements from TinyECC [9]).

Using multiple MACs reduces Castor's update latency. Figures 6(a) and 6(b) show Castor's latency to disseminate various items – the multi-MAC structure, the packets containing the corresponding keys and the first page, and the entire image. For comparison, the graph also shows Deluge's and Sluice's respective latencies to disseminate the entire image. Castor's latency is shown for 1, 2, 3 and 6 MACs, for T set to 30s, 15s, 10s and 5s, respectively, in Figure 6(a) and T set to 180s, 90s, 60s and 30s, respectively, in Figure 6(b). We ran our experiments until we had 5 successful simulations where the advertisement reached all of the nodes before the last key is disclosed.

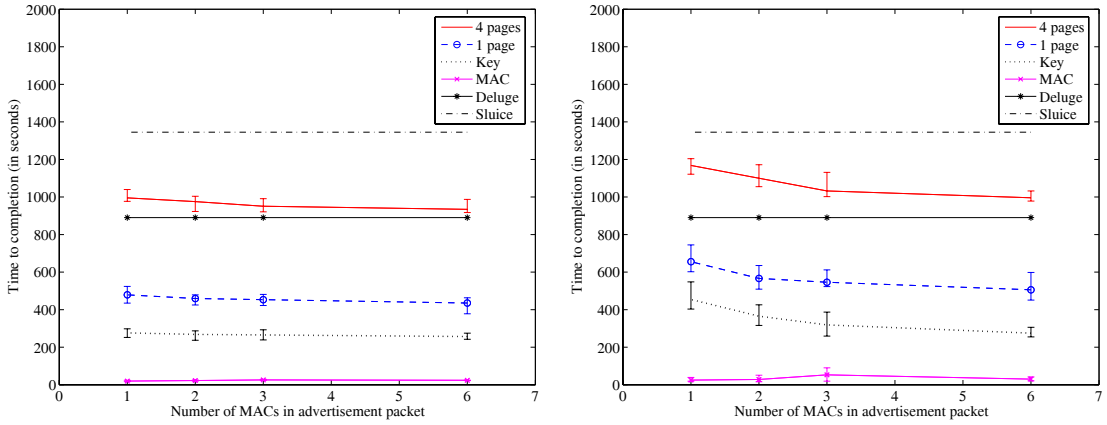
The benefit of using multiple MACs in lowering the update latency is evident in both graphs, but more prominent in Figure 6(b), where Castor uses a larger T, allowing it to cover larger networks. From Figure 6(b), using the 6-MAC advertisement to authenticate the image reduces the end-to-end update latency by 15% (from 1168s to 988s), as compared to the case of the 1-MAC advertisement.

Multiple MACs with a smaller T *vs.* single MAC with a larger T. Figure 7(a) shows the percentage of nodes that receive the (authenticated) first page over time for Deluge, Sluice and Castor. For Castor, we cover the cases of the 1-MAC and 6-MAC advertisements with $T = 30$s, the 6-MAC advertisement with $T = 5$s. Figure 7(b) compares the characteristics of Deluge, Sluice and Castor (Castor with 6 MACs



(a) (b)

Figure 5. (a) Castor's latency to disseminate various items (1-MAC advertisement, the disclosed key, the first page and the 4-page image), as T varies, and (b) latency to disseminate the 4-page image in Deluge, Sluice and Castor, as T varies.



(a) (b)

Figure 6. Castor's latency to disseminate the multi-MAC advertisement, the disclosed key, the first page and the 4-page image for 1, 2, 3 and 6 MACs with (a) T set to 30, 15, 10, 5s, respectively, and (b) T set to 180, 90, 60, 30s, respectively.

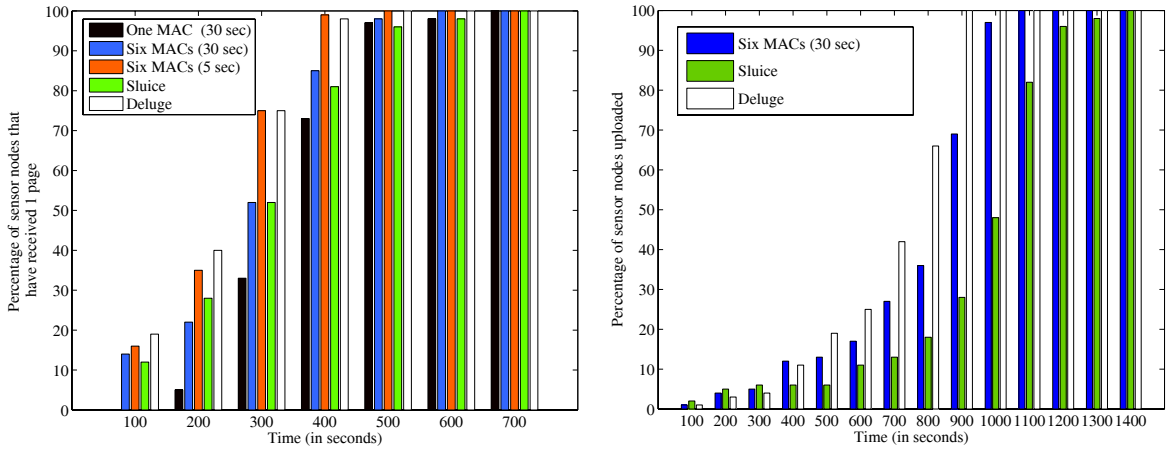
and $T = 30$s) to disseminate the 4-page image to all of the nodes in the network. Figure 7(a) reveals the advantages of using multiple MACs – with a single MAC, Castor stalls every node from initiating the update until the corresponding key is disclosed after 30s, while with multiple MACs, Castor relaxes this restriction and reaches performance closer to that of Deluge. Furthermore, examining Castor's performance in the 6-MAC case for T of 5s and 30s indicates that as T reduces, Castor's performance becomes closer to that of Deluge. Thus, one approach to reduce the end-to-end update latency as the network scales, is to use multiple MACs with a smaller T rather than a single MAC with a larger T.

5 Related Work

Several secure-code update protocols have recently emerged that use asymmetric cryptosystems to achieve their goals. Sluice [6] uses a combination of a one-way page-level hash chain and a digital signature to authenticate the source of an 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 image.

Secure Deluge [2] also uses a combination of one-way hash chains and a single digital signature amortized over the entire 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 SecureDeluge is greater than that of Sluice and requires packets (and not only pages) to be received in order.

Deng et al. [1] proposed a similar protocol that uses 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



(a) (b)

Figure 7. Benchmarking Deluge, Sluice and Castor: Percentage of nodes that have received and verified (a) the first page of the image, and (b) the entire 4-page image.

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.

6 Conclusion

Secure code-update protocols (such as Sluice and Secure Deluge) for sensor networks have been based on asymmetric cryptosystems such as digital signatures. Our new Castor protocol avoids using digital signatures altogether and instead exploits symmetric cryptosystems that are more suited to the resource constraints of sensors. Castor involves a synergistic combination of a one-way hash-chain, a verification key-chain, a backbone key-chain, and a sequence of MACs with delayed key-disclosure to enable untrusted sensors to verify an update-image's authenticity. We describe an implementation of Castor that secures the TinyOS code-update protocol, Deluge. We also empirically benchmark Castor's end-to-end update latency, computation costs and communication costs against both Deluge and Sluice.

7 References

[1] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *IPSN*, pp 292–300, Nashville, TN, Apr 2006.

[2] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the Deluge network programming system. In *IPSN*, pp 326–333, Nashville, TN, Apr 2006.

[3] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, pp 81–94, Baltimore, MD, Nov 2004.

[4] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *SenSys*, pp 162–175, Baltimore, MD, Nov 2004.

[5] L. Lamport. Password authentication with insecure communication. *CACM*, 24(11):770–772, 1981.

[6] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *ICDCS*, Lisbon, Portugal, July 2006.

[7] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *SenSys*, pp 126–137, Los Angeles, CA, Nov 2003.

[8] 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 *NSDI*, pp 1–14, 2004.

[9] A. Liu, P. Kampanakis, and P. Ning. TinyECC: Elliptic curve cryptography for sensor networks (v 0.3), Feb 2007.

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

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

[12] National Institute of Standards and Technology. *Secure hash standard*. April 1997.

[13] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *NDSS*, pp 35–46, San Diego, CA, Feb 2001.

[14] 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.

[15] K. Sun, P. Ning, and C. Wang. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *ACM CCS*, pp 264–277, Alexandria, VA, Oct–Nov 2006.