Authenticated In-Network Programming for Wireless Sensor Networks

Ioannis Krontiris and Tassos Dimitriou

Athens Information Technology,
P.O.Box 68, 19.5 km Markopoulo Ave.,
GR- 19002, Peania, Athens, Greece
{ikro,tdim}@ait.edu.gr

Abstract. Current in-network programming protocols for sensor networks allow an attacker to gain control of the network or disrupt its proper functionality by disseminating malicious code and reprogramming the nodes. We provide a protocol that yields source authentication in the group setting like a public-key signature scheme, only with signature and verification times much closer to those of a MAC. We show how this can be applied to an existing in-network programming scheme, namely Deluge, to authenticate code update broadcasts. Our implementation shows that our scheme imposes only a minimal computation and communication overhead to the existing cost of network programming and uses memory recourses efficiently, making it practical for use in sensor networks.

1 Introduction

The process of programming sensor nodes typically involves the development of the application in a PC and the loading of the program image to the node through the parallel or the serial port. The same process is repeated for all the nodes of the sensor network before deployment. However, after deployment, there is often the need to change the behavior of the nodes in order to adapt to new application requirements or new environmental conditions. This would require the effort of re-programming each individual node with the updated code and relocate it back to the deployment site. Network programming saves this effort by propagating the new code over the wireless link to the entire network, as soon as that code is loaded to only one node. Then, nodes reprogram themselves and start operating with the updated code.

As network programming simplifies things for legitimate users, it also simplifies things for attackers that want to disrupt the normal operation of the network or operate them for their own advantage. In currently deployed networks the nodes do not authenticate the source of the program; therefore an attacker could easily approach the deployment site and disseminate her own malicious/corrupted code in the network.

This possibility makes sensor networks deployments susceptible to outsider attacks. Besides loosing control of the network or getting back altered measurements, it is even possible that the network is reprogrammed with malicious code that has the same functionality with the legitimate code but also reports data to the adversary. In such a case legitimate users would never know that something is wrong. Hence, it is important that the sensor nodes can efficiently verify that the new code originates from a trusted source, namely the base station.

2 Problem Definition and Contribution

The goal of this work is to provide an efficient source authentication mechanism for broadcasting a program image from the base station to the sensor network. While the authentication mechanism should still allow efficient dissemination procedures, such as *pipelining*, it should also block malicious updates as early as possible.

By now, what have been studied extensively in sensor networks are point-to-point authentication mechanisms. Using a shared key, two nodes can exchange authenticated messages by appending a message authentication code (MAC) to each packet, computed using that key. Due to its low computational overhead, MACs are an attractive tool for securing communication in sensor networks. However, in order to use it for broadcast authentication, all nodes should share the same key. But then, anyone who could physically capture a node and retrieve that key could impersonate the source. A solution to that problem has been given by Perrig et al. in [1], which is based on delayed disclosure of keys by the sender. The shortcoming of this approach is that it requires time synchronization between the nodes, while current dissemination protocols for in-network programming do not place such bounds.

The most natural solution for authenticated broadcasts is asymmetric cryptography, where messages are signed with a key known only to the sender. Everybody can verify the authenticity of the messages by using the corresponding public key, but no one can produce legitimate signed messages without the secret key. However, public key schemes should be avoided in sensor networks for multiple reasons: long signatures induce high communication overhead of 50 - 1000 bytes per packet, verification time places a lower bound on the computational abilities of the receiver, and so on.

However our goal is not to authenticate just messages, since here we are dealing with *streams*, rather than simple messages. The size of program images that will be sent over the radio is usually between a few hundreds of kilobytes and a few thousands. This fact can allow the use of public key schemes if we manage to reduce the size of the public key and also make signature size to be only a small percentage of the total transmitted stream. Furthermore, if we reduce the verification time down to the order of that of a symmetric scheme, we will have proved that public key cryptography is an attractive solution for such problems.

Therefore, our goal and the contribution of our work is to provide an efficient authentication scheme for a finite stream of data based on *symmetric* cryptography primitives while at the same time having the properties of asymmetric cryptography.

2.1 Design Goals

The solution that we present in this work was designed having the following requirements in mind:

1. **Low computational cost**. As we mentioned above, asymmetric cryptography involves high computational cost and is not preferable for use in sensor networks. Our scheme should impose public-key properties but at the same time minimize the computational cost for sign verification at the receivers (sensor nodes).
2. **Low verification time**. The rate at which a code segment is transmitted to the receiver should not be delayed.
3. **Low communication overhead**. The signature transmitted with data should constitute a small percentage of the total bytes, imposing a low communication overhead.
4. **Low storage requirements**. Any cryptographic material that needs to be stored in the sensor nodes should be as small as possible, given their limited memory resources.

Moreover, since we are providing an authenticated broadcast protocol we need to assure the following:

1. **Source authentication**. A mote must be able to verify that a code update originates from a trusted source, i.e., the base station. This means that an attacker should not be able to send malicious code in the network and reprogram the nodes.
2. **Node-compromise resilience**. In case an attacker compromises a node and read its cryptographic material, she must not be able to reprogram any other non-compromised node with malicious code.

Even though we do not address protection against DoS attacks, our protocol must provide some resilience against such attacks in the following sense: In case an attacker is trying to transmit malicious code to the network, any receiving node should be able to realize this as soon as possible and stop receiving it or forwarding it to other nodes. This means that nodes should not authenticate the code *after* its reception but rather *during* that process.

3 Related Work

A recent work that proposes a solution for secure dissemination of code updates in sensor networks is described in [2]. The authors first suggested the use of hash chains to efficiently authenticate each page of the program image. However, they make the assumption that there exists a public key scheme to authenticate the initial commitment of the hash chain, without giving any specific solution.

Another work on the same problem is described in [3], where the authors set the additional goal of DOS-resilience and therefore they need to authenticate each packet separately. To do that they construct a signed hash tree scheme (similar to a Merkle tree) for *every* page in the program image, and they transmit these trees before the actual data. This increases considerably the overhead of packets sent and received by the motes. Moreover, due to memory constrains in the motes, these values need to be stored and loaded from the EEPROM, which is a very energy consuming operation.

In [4] the use of a reverse hash chain computed over the program pages is also used, as in [2] and in our scheme. However the authors use the RSA digital signature scheme for signature verification at the motes, which we have excluded from our design goals. On the other hand, an authentication scheme for broadcasting messages in a sensor network that uses only symmetric primitives is described in [5]. The authors keep the memory and computational overhead of their algorithm efficiently low. However they are concerned about the problem of authenticating broadcasted queries, which are normally less harmful messages with very small size, so their requirements are different than in our case.

4 Overview and Useful Tools

Throughout this paper we are considering Deluge [6] as a paradigm of in-network programming. However other data dissemination protocols like MOAP[7], MNP[8] and INFUSE[9] are following similar principles, and the algorithms presented here should be applicable to those protocols as well.

Deluge propagates a program image by dividing it first into fixed-size pages and then using a demand-response protocol to disseminate them in the network. As soon as a node receives a page, it makes it available to any of its neighbors that also need it. At the same time it sends a request to the sender in order to receive the subsequent pages.

To sign a program image we are following the approach by Gennaro and Rohatgi in [10] for signing digital streams. What they proposed is to divide the stream into blocks and embed some authentication information in each block. In particular, their idea is to embed in each block a hash of the following block. In this way the sender needs to sign just the first block and then the properties of this signature will propagate to the rest of the stream through the "chaining" technique.

So, given a program image divided into N fixed-size pages $P_1, P_2, \ldots, P_N$ and a collision-resistant hash function H, we construct the hash chain

$$h_i = H(P_{i+1}|h_{i+1}), \qquad i = 0 \ldots N-2$$

and we attach each hash value h_i to page P_i, as shown in Figure 1. For the last hash value, $h_{N-1} = H(P_N)$. According to this scheme, we need to authenticate only h_0, which we will sign and release before the transmission of any page. The signing and verification of h_0 constitutes the main overhead of the security protocol, which our goal is to minimize.

Towards this goal, our main design principle is based on the fact that real world software updates in sensor networks do not constitute an every-day operation but rather they are performed occasionally. Therefore, we do not need to

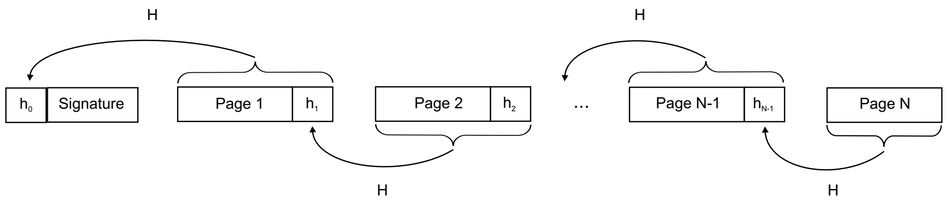

Fig. 1. Applying the hash chaining technique to the pages of a program image. Only h_0 needs to be signed by the sender.

authenticate an unlimited number of broadcasts. We only need to be able to do so for a sufficiently large number of times. This fact allow us to use *one-time signature schemes*, which exhibit fast verification times. Despite their name, there exist one-time signature schemes that can be used r-times instead of just once, r being a design parameter adjustable to our needs.

4.1 One-time Signature Schemes

One-time signatures were first introduced in [11, 12]. They are based on the idea of committing a secret key via one-way functions, decreasing dramatically the signing and verification time compared to asymmetric primitives. In the rest of the paper we will describe an efficient one-time signature scheme appropriate for sensor networks and how this can be used for authenticating broadcasts of program images. We believe this technique to be interesting on its own, apart from its usage in devices with limited capabilities.

In one-time signature schemes the signer is generating a set of secrets prior to signing a message along with a set of public commitments to this set which are given to the verifier in an authenticated manner. To sign a message, the signer reveals a subset of these secrets, which is determined by the message content. The verifier authenticates the message by checking the correspondence of these secrets to the commitments that were given earlier. Since a part of the signer's secrets is now revealed, a new key must be generated for the next message.

Although one-time signatures have been known for a relatively long time, they have been considered to be impractical for two main reasons: First, they can be used to sign a message only once and then a new key must be generated; Second, the signature size is relatively long in comparison with common public-key signatures and MACs.

Recently, this area was revisited and some one-time signature schemes were proposed that seem attractive for sensor networks, mainly because they allow the reuse of the same key more than once, but also because they try to reduce the verification time. For example, Reyzin and Reyzin [13] introduced HORS, an r-time signature scheme with efficient signature and verification times. This scheme was further improved by Pieprzyk et al. [14]. Both of these "r-times signatures" can sign several messages with the same key with reasonable security before they can get compromised.

However there are still some drawbacks that prevent us from applying those schemes to sensor networks. The main one is the size of the public/secret key pair and the size of the generated signatures. The public key must be stored on all sensor nodes so its size must be minimized as much as possible. Moreover, the signature is transmitted by the radio and received by nodes, which have to verify it. The larger the signature size, the more energy a node has to spend in order to receive it and verify it.

4.2 Merkle Trees

As we described so far, all verifiers need an authenticated copy of the public commitment to the one-time signature in order to verify the validity of that signature. Merkle [15] introduced a scheme that enables the verification of a large number of public commitments using low storage requirements, i.e. a single hash value. This is done by using a technique called Merkle hash tree. A Merkle hash tree is a complete binary tree where each node is associated with a value, such that the value of each parent node is the hash function on the values of its children:

$$v(parent) = H(v(left)|v(right))$$

where $v()$ here stands for the value of a node and H for a hash function.

If we put the public commitments to the leaves of a Merkle tree, then the root can serve as a short public commitment to all the one-time signatures. Then we only need to give the root to the verifier in a secure and authenticated way. To verify a one-time signature the receiver does not need to know the whole Merkle tree. Instead, the only thing that the signer needs to provide to the verifier is the authentication path, i.e., the values of all the nodes that are siblings of nodes on the path between the leaf that represents the public commitment and the root.

Given that authentication path, a leaf may be authenticated as follows: First apply the one-way hash function to the leaf and its first sibling in the path, then hash the result and the next sibling, etc., until the root is reached. If the computed root value equals the published root value then the signature's commitment is authentic.

4.3 HORS

Our open question so far is the efficient signing of the first value h_0 of the hash chain. This will enable the authentication of the whole chain and therefore the authentication of the program image. To sign h_0 we will modify the HORS scheme so that the sizes of the signature and the public key are reduced to a magnitude proper for use in sensor networks. Here we briefly review the HORS scheme.

First, the signer generates a secret key SK that consists of t random values. The public key PK is computed by applying a one-way function f to each of the values of the secret key and then distributing them to the intended receivers in an authenticated way. A message m is signed according to the following steps:

1. Use a cryptographic hash function H to convert the message to a fixed length output. Split the output into k substrings of length $\log_2 t$ each.
2. Interpret each substring as integer. Use these integers to select a subset σ of k values out of the set SK.
3. This σ is the signature of the message m.

The verifiers recompute the hash value of the message m, re-produce the same indices and pick the corresponding values of the set PK (instead of SK). Then they verify that the hash value of each member of the signature equals to the corresponding member of the public key PK. The signature is accepted if this is true for all k values.

Note that for each message that we sign, a part of the secret key is leaked out. Some typical values for HORS are $l = 80$, $k = 16$ and $t = 1024$. In this case, assuming a hash output of 20 bytes, the public key will be $1024 \times 20 = 20,480$ bytes or 20 KB, which is not suitable for sensor nodes. Moreover, the security of the scheme and the size of the public key are directly related to the number of messages that we can sign. For the above example, we can sign just 4 messages with acceptable security, meaning 4 program image updates in our case. However, to make the scheme practical this number must be higher.

Let r be equal to the number of messages that we allow to be signed with the current instance of the secret key. For an analysis (see also [13]) we assume that the hash function H behaves like a random oracle and that an adversary has obtained the signatures of r messages using the same setting of secret/public key. Then the probability that an adversary can forge a message is simply the probability that after rk values of the secret key have been released, k elements are chosen at random that form a subset of the rk values. The probability of this happening is $(rk/t)^k$. If we denote by Σ the attainable security level in bits, by equating the previous probability to $2^{-\Sigma}$, we see that Σ is given by

$$\Sigma = k(\log_2 t - \log_2 k - \log_2 r). \tag{1}$$

As an example, for $t = 1024$, $k = 16$ and $r = 4$ we get $\Sigma = 64$ bits of security. For $t = 65536$, $k = 8$ and $r = 32$ we get the same level of security but we can sign a lot more messages with the same key. However, since the number t of PK values determines the public/secret keys sizes, it is directly limited by the restrictions imposed by sensor networks capabilities. As a first step we can use equation (1) to solve for t for any desirable security level. Thus we get

$$t = 2^{\Sigma/k} kr. \tag{2}$$

5 Our *r*-times Signature Scheme

The reason that makes HORS inappropriate for sensor networks is that the public key grows unacceptably high if we want to sign more messages and keep security at an acceptable level. So we need to effectively reduce the public key size. One way to do this (also proposed in [16]) is to use a Merkle tree, which we discussed in section 4.2. Given the secret key values, we apply a one-way function f to each one and we place the results to the leaves of the Merkle tree. The root of the resulting Merkle tree is the public key.

Even though we have reduced the size of the public key down to a hash output size, we increased the size of the signature to the size of the authentication path. This also results in a corresponding increase to the signature verification time. So, this solution is also not attractive for applications in sensor networks.

Our solution goes one step further and distributes the values of the secret key into *many* Merkle trees, thus achieving a tradeoff between public key size and signature size. First we show how this can be done.

Let f be an $l-$bit one-way function. The generation of the key pair is done by the following algorithm:

Secret Key Generate t random $l-$bit quantities for the secret key: $SK = (s_1, \ldots, s_t)$.

Public key Compute the public key as follows: Generate t hash values $(u_1, \ldots, u_t)$, where $u_1 = f(s_1), \ldots, u_t = f(s_t)$. Separate these values into d groups, each with t/d values. Use these values as leaves to construct d Merkle trees. The roots of the trees are the public key of our scheme.

In this way we have reduced the public key size down to a few hash values that constitute the roots of the Merkle trees. These values need to be passed to all sensor nodes in an authenticated way. This can be done for example during initialization of sensor nodes. Now, a message is signed according to the following steps (see also Figure 2):

1. Use a cryptographic hash function H to convert the message to a fixed length output. Split the output into k substrings of length $\log_2 t$ each.
2. Interpret each substring as integer. Use this integers to select a subset σ of k values out of the set SK.
3. The signature of the message m is made up by the selected secret values along with their corresponding authentication paths.

The verifiers recompute the hash value of the message m, re-produce the same indices and pick the corresponding values of the set PK. Then they evaluate each authentication path of the signature to reproduce the root of the Merkle tree and compare it with the corresponding member of the public key PK. The signature is accepted if this is true for all k values. The detailed description of the algorithm is shown in Figure 3.

As an example, let's apply to our scheme the same values we did for HORS, i.e., $l = 80$, $k = 16$ and $t = 1024$, and assuming a hash output of 20 bytes. If we construct 32 Merkle trees with 32 leaves each (so all 1024 secret values are covered), we will get 32 roots of trees, i.e., 640 bytes that will constitute our public key, compared to 20 KB we got from HORS. These values will provide 64 bits of security for $r = 4$ messages (images).

If we choose now $l = 80$, $k = 8$ and $t = 65536$, and $r = 32$ we get the same level of security. In this case, by constructing 64 Merkle trees of 1024 leaves each,

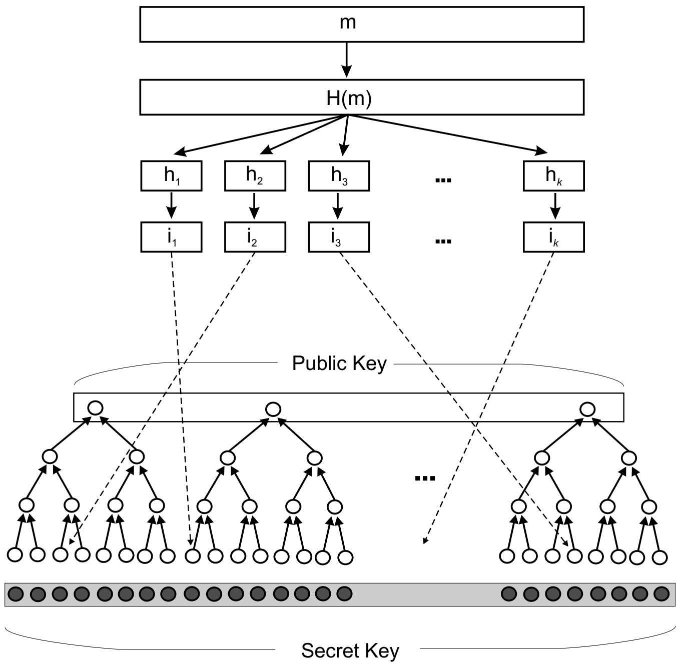

Fig. 2. Signature construction for message m using multiple Merkle trees.

the public key will become 1024 bytes, which is still an attractive value for use in sensor nodes.

5.1 Tradeoffs

The public key stored in each sensor node is given by the hash values residing at the roots of the trees. The more the number of the trees, the bigger the public key becomes but the smaller the signature size becomes. To see why, notice that signature size depends on the length of the authentication paths, which are ultimately related to the height of the Merkle trees. More trees means less secret values per tree and hence smaller height. To find this tradeoff between public/signature size let T denote the number of trees. Hence the public key size is simply

$$S_{PK} = |h|T, \tag{3}$$

where $|h|$ is the output of the hash function in bits since every root contains a hash value of its children. For example, $|h|$ can be equal to 128 bits in the case of MD5 or 160 bits in the case of SHA-1.

As the number of trees is T, there can be at most t/T values stored at the leaves of each tree. Thus the height of each tree (and the length of each

```
Key Generation
  Input: Parameters $l$,$k$,$t$
    Generate $t$ random $l$-bit strings $s_1, s_2, \ldots, s_t$
    Let $u_i = f(s_i)$ for $1 \leq i \leq t$
    Group $t$ hash values $u_1, u_2, \ldots, u_t$ into $d$ groups of $t/d$ values
    Place each group at the leaves of a Merkle tree, constructing $d$ Merkle trees
    Let $w_1, w_2, \ldots, w_d$ be the roots of the Merkle trees
  Output: $PK = (k, w_1, w_2, \ldots, w_d)$ and $SK = (k, s_1, s_2, \ldots, s_t)$

Signing
  Input: Message $m$ and secret key $SK = (k, s_1, s_2, \ldots, s_t)$
    Let $h = H(m)$
    Split $h$ into $k$ substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each
    Interpret each $h_j$ as an integer $i_j$ for $1 \leq j \leq k$
    Let $\mu_{i_j} = (s_{i_j}, AP(s_{i_j})))$, i.e. the secret ball along with its authentication path
  Output: $\sigma = (\mu_{i_1}, \mu_{i_2}, \ldots, \mu_{i_k})$

Verifying
  Input: Message $m$, signature $\sigma = (\mu'_1, \ldots, \mu'_k)$ and public key $PK = (k, w_1, \ldots, w_t)$
    Let $h = H(m)$
    Split into $k$ substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each
    Interpret each $h_j$ as an integer $i_j$ for $1 \leq j \leq k$
    Compute which Merkle tree corresponds to $i_j$: $M_j = i_j/(t/d)$ for $1 \leq j \leq k$
    Hash the values in each $\mu'_k$ to produce the corresponding root $w'_{M_j}$
  Output: "accept" if for each $j, 1 \leq j \leq k, w'_{M_j} = w_{M_j}$; "reject" otherwise
```

Fig. 3. Our proposed signature scheme. f is a one-way function and H is a hash function. Both f and H may be implemented using a standard hash function, such as SHA-1 or MD5.

authentication path) is simply $\log_2 t/T$ or $\Sigma/k + \log_2(kr) - \log_2 T$ using equation (2). The signature consists of k such authentication paths, where each path is a sequence of hash values. Thus the signature size is given by

$$S_{sig} = |h|(\Sigma + k\log_2(kr) - k\log_2 T). \tag{4}$$

From this equation it should be obvious that increasing the number of trees T (and hence the public key size) results in a decrease in the signature size.

This equation can be simplified further if we recall how the k secret values are selected (Figure 2). The message m to be authenticated is first hashed to obtain $H(m)$, a value that is $|h|$ bits long. Then these $|h|$ bits are broken into k parts, where each part references one of the secret values. Thus the number of secrete values t must be equal to $2^{|h|/k}$, or equivalently

$$|h| = k\log_2 t. \tag{5}$$

Combining with equations (2) and (4), we find that the signature size is given by

$$S_{sig} = |h|(|h| - k\log_2 T). \tag{6}$$

In the same manner, the security level becomes

$$\Sigma = k(|h|/k - log_2 k - log_2 r). \tag{7}$$

In the figures below we tried to keep the public key size equal to approximately 1 KByte so that it fits well in the memory of typical Mica nodes. Assuming $h = 128$, i.e. using MD5 to produce the hash values, we find that the number of trees T should be equal to 64, by equation (3).

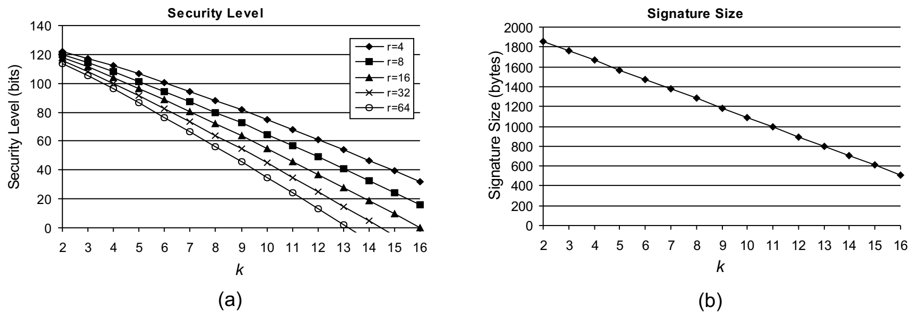

Fig. 4. Signature size and security level as a function of k.

6 Implementation Details and Evaluation

We implemented our authentication protocol in order to measure its efficiency. Our implementation was built on Deluge 2.0, which we slightly modified in order to include our scheme. At the end we were able to download authenticated images on the sensor nodes, and reprogram them by using Deluge. The whole security mechanisms were kept transparent from the end users, unless they tried to inject a corrupted or malicious program image.

In our implementation design we faced several issues, some of which we address here. First of all, one primary goal was for a mote to be able to authenticate each page separately and stop the downloading of the image as soon as a non-authentic page is received. However, by that time earlier pages that passed the verification procedure will have been propagated to the rest of the network wasting energy of the motes. This may be considered as a kind of DoS attack, if exploited properly by an adversary. However it is a price that must be paid in order to support pipelining. If one is interested in optimizing the protocol from a security point of view, then it must modify it to exclude the pipelining, so that only complete images that have been authenticated can be further forwarded.

Another issue to be considered here is the memory optimization of the protocol. While it is not possible to avoid storing the public key in the mote's memory, we can do so for the signature. This is because the signature is made up by authentication paths and the authentication of each path can be done independently by the others. Referring to Figure 5, the mote first receives the hash value of Image 1. This will provide the indices to the public values. Then, the first authentication path of the signature will be received. The verification of that path evolves only a few hashing operations and a comparison of the result with the corresponding public value. This can be done fast enough by the mote (see Section 6.1) so that the path has been verified before the next path starts coming in. So, only a temporary storing is needed, equal to the size of a path (dependent on the height of the Merkle trees at the base station).

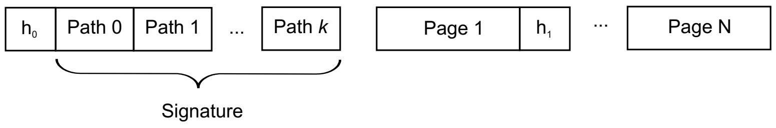

Fig. 5. The order at which a mote receives the signature, the pages and their hash values. Verification of the signature is possible by storing only one path at the time.

So the only extra memory requirement that we impose is the buffering of the pages, since we want to compute their hash value and compare them with the corresponding commitments in the hash chain. For example, the page size in Deluge is 1104 bytes, which is a large percentage of the available memory in a mote (usually at the order of a few kilobytes). Nevertheless, it is possible to buffer one page of that size at the time, as long as we do not require any more memory of that order, which is true for our scheme.

6.1 Evaluation

Following the discussion of Section 5.1, we used our secure version of Deluge to measure the verification time of the signature attached to the program image. Our implementation was done on the mica2 platform, which exhibits low memory capabilities (4 KB of RAM). For all of our experiments we set a security level equal to 60 bits (although this can be modified accordingly), which is a satisfactory value for most security applications.

For that security level, Figure 6(a) shows the verification time of the signature for different number of Merkle trees T (determining the public key size) and different values of r (number of images that can be signed with the same keys). So, for example, if we take $k = 8$ and want to sign $r = 64$ images using the same secret-public key pair while keeping the public key size down to 1 KB (i.e. $T = 64$), we get a verification time equal to $186.3ms$. This is just the computational time and does not include the time to transmit the stream. So,

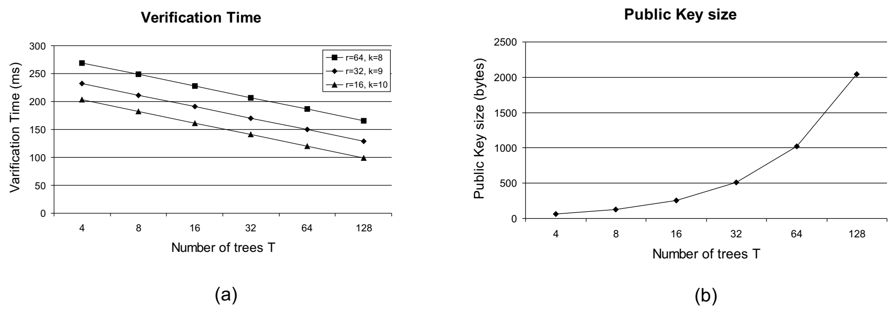

Fig. 6. Verification time and public key size as a function of T

it defines the computational overhead that our algorithm imposes on Deluge. Notice also that this time uses standard implementations of hash functions, so it can be improved even further using optimized code.

Figure 6(b) shows how the size of the public key changes as a function of the number of Merkle trees T. If the secret values are distributed over more Merkle trees, the public key increases but the verification time decreases accordingly. So, this is a tradeoff that must be decided at design time, depending on the available memory on the sensor nodes, which will determine how big the public key can be.

7 Conclusions

In this paper we presented an efficient and practical scheme for authenticated in-network programming in sensor networks. Our solution imposes asymmetric cryptography properties using *symmetric* cryptography primitives. It minimizes the public key and signature sizes to values that are appropriate for sensor networks. The verification procedure at the motes is also time and computational efficient, since it involves only hashing and comparison operations. Our scheme also provides node compromise resilience, preventing an attacker who captures a node to reprogram any other node in the network. Furthermore, images are authenticated at a per-page basis, which enables a node to stop the downloading of a new image as soon as a page fails the verification procedure.

We implemented our solution and integrated it in Deluge, showing that it can easily adapt to an existing in-network programming protocol. We tested our secure Deluge version and measured the verification time of the signature at the mote's side. This showed that the computational overhead imposed by our scheme is at the order of one to two hundreds milliseconds, which is very efficient for applications running on sensor nodes.

References

1. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.D.: SPINS: Security protocols for sensor networks. Wireless Networks **8**(5) (2002) 521–534
2. Lanigan, P., Gandhi, R., Narasimhan, P.: Secure dissemination of code updates in sensor networks. In: Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys '05). (2005) 278–279
3. Deng, J., Han, R., Mishra, S.: Secure code distribution in dynamically programmable wireless sensor networks. Technical Report CU-CS-1000-05, Department of Computer Science, University of Colorado, Boulder, CO (2005)
4. Dutta, P., Hui, J., Chu, D., Culler, D.: Securing the deluge network programming system. In: Proceeding of the 5th International Conference on Information Processing in Sensor Networks (IPSN 2006). (2006)
5. Benenson, Z., Pimenidis, L., Hammerschmidt, E., Freiling, F.C., Lucks, S.: Authenticated query flooding in sensor networks. In: Proceedings of the 21st IFIP International Information Security Conference (SEC 2006). (2006)
6. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd international conference on Embedded networked sensor systems. (2004) 81–94
7. Stathopoulos, T., Heidemann, J., Estrin, D.: A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing (2003)
8. Kulkarni, S.S., Wang, L.: MNP: Multihop network reprogramming service for sensor networks. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05). (2005) 7–16
9. Arumugam, M.: Infuse: a TDMA based reprogramming service for sensor networks. In: Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04). (2004) 281–282
10. Gennaro, R., Rohatgi, P.: How to sign digital streams. Information and Computation **165**(1) (2001) 100–116
11. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International Computer Science Laboratory, Palo Alto (1979)
12. Merkle, R.: A digital signature based on a conventional encryption function. In: Advances in Cryptology – CRYPTO '87. Volume 293 of Lecture Notes in Computer Science., London, UK, Springer-Verlag (1988) 369–378
13. Reyzin, L., Reyzin, N.: Better than BiBa: Short one-time signatures with fast signing and verifying. In: Proceedings of the 7th Australian Conference on Information Security and Privacy (ACISP '02), London, UK, Springer-Verlag (2002) 144–153
14. Pieprzyk, J., Wang, H., Xing, C.: Multiple-time signature schemes against adaptive chosen message attacks. In: Selected Areas in Cryptography (SAC 2003), Springer (2003) 88–100
15. Merkle, R.C.: A certified digital signature. In: Proceedings on Advances in cryptology (CRYPTO '89), Springer-Verlag New York, Inc. (1989) 218–238
16. Seys, S., Preneel, B.: Power consumption evaluation of efficient digital signature schemes for low power devices. In: Proceedings of the 2005 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (IEEE WiMob 2005). Volume 1. (2005) 79–86