

Secure code distribution in wireless sensor networks

Yee Wei LAW, James SING, Braden KIDD, Slaven MARUSIC¹
*Department of Electrical Engineering and Electronic Engineering,
The University of Melbourne, Parkville, VIC 3052, Australia*

Email: {yee.wei.law, jamesbsing, bradenkidd}@gmail.com, s.marusic@ee.unimelb.edu.au

Abstract: After a WSN is deployed, there are occasions where the software (or interchangeably, firmware) in the sensor nodes needs to be updated. Due to the large number of nodes, it is impractical to update the software manually. Instead, software updates are sent to the nodes wirelessly, that is, the nodes are reprogrammed in situ. This feature or process is called remote network reprogramming, or simply network reprogramming. This paper is about the securing of this process. We believe the best approach to the problem is not piecing several components together, but a concerted effort at integrating the components and optimizing the components for each other. We first divide the problem into three design spaces: (1) data dissemination protocols, (2) cryptographic protocols for data dissemination, (3) code verification. We explore each of these design spaces, by reviewing state-of-the-art proposals. In our review, we perform detailed analysis, and offer new insights into the rationales and inner-workings of these designs. Our end goal is to integrate and cross-optimize these three components. Our conclusions for this work are as such. For data dissemination protocols, our deduction is that to surpass Deluge or any of its subsequent derivatives, revolutionary techniques are needed, otherwise incremental improvements might be achievable by looking into the cross-layer optimization between the TDMA-based media access control layer and the data dissemination protocol layer. For cryptographic protocols, we propose exploring the relative merits of Deng et al.'s scheme and the so-called intermediate scheme a bit further. For code verification algorithms, we propose exploring energy-storage trade-off in the latest implementations, and optimizing the implementations for emergent 8051-based SoC platforms.

Keywords: Wireless sensor networks, secure code dissemination, secure network reprogramming, data dissemination, elliptic curve cryptography

1. Introduction

The mission of WSNs is not only to provide flexibility in space but also flexibility in time. Once a network has been deployed, there must be a facility to update its functionality according to the needs of the time. Remote network reprogramming allows us to do just that. It allows us to update new code containing new business rules to the sensor nodes in the field, and ‘instantaneously’ transform the functionality of the network. In general, network reprogramming consists of three stages from the networking point of view [1]:

- target selection, where specific nodes are selected to receive the update;
- code dissemination, where the code is delivered to the selected nodes;

¹ The authors gratefully acknowledge sponsorship by the Australian Research Council Research Network on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), and the DEST International Science and Linkage Grant.

- completion verification, where the nodes that have successfully received the update report to a central authority.

The need for target selection and completion verification is very much application-dependent, hence outside the scope of this paper. Beyond the networking point of view, and from a security point of view, there is another stage in between code dissemination and completion verification, that is, code verification. Code verification is performed after the nodes receive the new code. Before the nodes load the new code from their external storage (typically Flash memory) to their internal memory (also typically Flash memory), they need to verify cryptographically if the code is trustworthy.

Our objective in this paper is to explore the techniques for code dissemination and code verification. The primary criteria for these techniques are energy efficiency and security. The popular approach to securing code dissemination is to assume an existing *data dissemination protocol*, which is typically Deluge, and construct a cryptographic protocol on top of it. We advocate a completely integrated approach where we address the energy-efficiency and the security of the protocol together. Thus, our starting point is consistent with the convention, i.e., a data dissemination protocol. The reason for using a data dissemination protocol and not simple flooding is energy-efficiency. Authenticated broadcast protocols like μ TESLA [18] are inadequate, as they are purely cryptographic protocols and do not specify any mechanism at all for disseminating data energy-efficiently.

The paper is organized as follows. Section 2 reviews existing data dissemination protocols and offers insight on potential improvements of existing schemes. Section 3 reviews the cryptographic protocols proposed to secure data dissemination. Section 4 reviews current elliptic curve cryptosystem (ECC) implementations because ECC is the most energy-efficient public-key algorithm to date. In all these sections, we not only describe the most influential proposals, but also perform detailed analysis, and offer new insights into the rationales and inner-workings. Finally, we conclude in Section 5. The symbols and notations used in this paper are partially summarized in Table 1. Omitted symbols should be clear from the context.

Table 1: Partial list of symbols and notations

Symbol	Meaning	Symbol	Meaning
$Sign(\bullet, \bullet)$	Signature function taking a key as the first parameter and a text as the second parameter	PK	Public key of the administrator; stored in every node
$h(\bullet)$	Hash function	SK	Secret key of the administrator
p_i	Page i of disseminated data	m_i	Packet i in a page
$w (w > 1)$	Maximum number of hashes per packet	σ	Number of bits in a hash
α	Number of index packets in a page	β	Number of bits in all the index packets of a page
\parallel	Concatenation operator		

2. Review of Data Dissemination Protocols

Data dissemination protocols are the facilitators of code dissemination. Some well-known examples are (in approximate chronological order) SPIN [8], MOAP [20], Infuse [1], Trickle [14], Deluge [9], Firecracker [13], Aqueduct [19], MNP [11], Sprinkler [16], Freshet [10] and Stream [17]. We summarize the salient points as follows.

In the reprogramming state, use push: The intuition behind this is that, if we know what to transfer, then we would not waste energy transferring something that is not needed. This intuition comes in the form of *negotiation*. Negotiation is a push paradigm pioneered by SPIN and is used by most if not all later protocols. A negotiation round consists of the exchange of an advertisement, a request and data in a three-way handshake. By paying a little price in the overhead, we can avoid sending data that are actually not needed. A

beneficial bi-product of this three-way handshake is that the advertiser can make sure it has a bi-directional link with the requester, so that the data do end up in the requester.

In the steady state, use pull: When a new node joins a network, it will request for the latest software version from its neighbours. Existing nodes do not periodically advertise their software version, thus saving unnecessary transmissions.

Negative acknowledgements: After negotiation, a requester receives data from the advertiser. Instead of sending an acknowledgement after every data transfer, the requester renews its request. These new requests serve as selective negative acknowledgements. This idea is actually borrowed from reliable multicast protocols.

Sender selection: This approach was started by Trickle, and was later adopted and adapted by other protocols. The idea is to divide time into rounds, and in each round, a node decides probabilistically whether or not to broadcast an advertisement. MNP's adaption is to select nodes that received more distinct requests than the others, to make advertisements later.

Rate adjustment: This approach also originated in Trickle, and was later adopted and adapted by other protocols. The rationale is to bound the frequency of advertisements based on the number of overheard advertisements.

Scheduled sleeping: Freshet is the protocol that takes the paradigm of saving-energy-by-sleeping to the extreme. The first phase, (Blitzkrieg phase), of the protocol involves propagating information about the data and the network topology through the network so that nodes can determine approximately when they will need to be active to receive data.

Spatial multiplexing or pipelining: This is a measure for improving throughput. Basically, when one data transfer is taking place, other non-interfering nodes can engage in parallel data transfers. Without pipelining, propagating s amount of data across d hops consumes $O(ds)$ time units, whereas with pipelining, propagating the same amount of data across the same number of hops consumes only $O(d + s)$ time units. Deluge and MNP are two protocols that support pipelining.

Collision avoidance: Infuse and Sprinkler are two protocols that use time division multiple access (TDMA) to avoid collisions. Sprinkler takes the exploitation of TDMA a step further but trying to construct an optimal schedule using a connected dominating set (CDS) algorithm. A dominating set for a graph $G(V,E)$ is a subset V^* of V such that every vertex not in V^* is adjacent to at least one member of V^* — a dominating set can thus be employed as a backbone of a network. The problem of finding the minimal CDS is NP-complete [2], and hence some non-optimal heuristics have to be used. The CDS algorithm of Sprinkler is localized and has a complexity of $O(1)$, and thus scalable. However in practice, since a TDMA schedule is normally optimized for convergecasts (the predominant traffic pattern) in the steady state, the schedule would have to be re-optimized for global broadcasts every time the network goes into the reprogramming state.

Minimizing data volume: The philosophy stems from Stream, a meta-protocol that can be used on top of a compatible protocol like Deluge, MNP and Freshet. In fact, the design of Stream is motivated by the code bloat problem that plagues Deluge, MNP and Freshet. The essence of the idea is to divide the firmware image into two components: the reprogramming support and the application support. Flash memories are usually partitioned into banks, so the two components can reside in different banks. In the steady state, all nodes run the application support component. In the reprogramming state, all nodes switch to the reprogramming support component. The reprogramming support does not need to be updated during the reprogramming phase, thus saving bandwidth and energy.

Depending on the organization of the operating system (OS), an update may vary greatly in size. If the OS is a monolithic OS like TinyOS² or Mantis³, an update is the whole

² <http://www.tinyos.net>

firmware image, containing a mixture of both the OS kernel and the applications. If the OS is a modular OS like SOS⁴ or Contiki⁵, an update can be just an application module that needs to be updated or added. The choice of OS though is an orthogonal issue. For practical purposes, TinyOS remains the de facto standard. Also, the code should be compressed with the best compression algorithm available before it is disseminated.

There are a few lessons we can learn from the summary above. First, Deluge has been regarded as a benchmark since its publication – because it has either pioneered or adsorbed most of these techniques. There are things that can be improved on Deluge, for example, sender selection, sleep scheduling, collision avoidance and data volume minimization. MNP has improved on sender selection and sleep scheduling. Freshet has taken a drastic approach on improving sleep scheduling. What remains are collision avoidance and data volume minimization. Deluge is independent of the underlying MAC protocol, so improving media access benefits Deluge and its derivatives equally, unless there is cross-layer optimization involved. In terms of data volume minimization, if we choose to use a modular OS to reduce the size of the update code, the result will not only benefit Deluge but other protocols as well, and so again, switching to a modular OS is an independent improvement. After all, TinyOS is the de facto standard. The deduction is that to surpass Deluge or any of its subsequent derivatives, revolutionary techniques are needed, otherwise incremental improvements might be achieved by looking into the cross-layer optimization between the TDMA-based MAC layer and the data dissemination protocol layer.

3. Review of Cryptographic Protocols for Data Dissemination

Due to the dominance of Deluge as a data dissemination protocol, most if not all proposals that try to secure data dissemination so far are more or less cryptographic ‘add-ons’ targeted at Deluge-like systems. The specific aspect of Deluge that these protocols make use of is the data structure. A piece of data that needs to be disseminated is broken down into pages, and these pages are further broken down into packets. The problem boils down to delivering all the packets in all the pages securely to all the nodes.

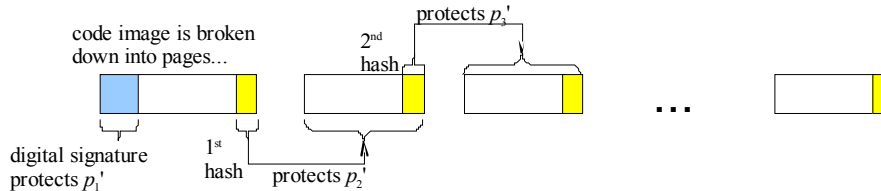


Figure 1: Signing and hash chaining of code pages

Let us denote pages by p_1, \dots, p_n , where n is the number of pages; and define other symbols as in Table 1. From p_1, \dots, p_n , p_1', \dots, p_n' are generated, in the reverse order, as follows (for symbols see Table 1, for illustration see Figure 1):

- $p_n' = p_n$
- $p_{n-1}' = p_{n-1} \parallel h(p_n')$
- $p_{n-2}' = p_{n-2} \parallel h(p_{n-1}')$
- ...
- $p_1' = \text{Sign}(SK, p_1 \parallel h(p_2')) \parallel p_1 \parallel h(p_2')$

p_1', \dots, p_n' are what get disseminated to the nodes, in sequence. Notice the reverse-order generation establishes a forward-order chain of dependency (Figure 1). The signature is generated using the administrator’s private key (the administrator can also be the code

³ <http://mantis.cs.colorado.edu>

⁴ <http://nesl.ee.ucla.edu/projects/sos>

⁵ <http://www.sics.se/contiki>

distributor), because had a symmetric key been used, the symmetric key would have to be stored in every node in the network but a node is generally not tamper-resistant. We explain why the pages have to be received in sequence by using an example. Suppose we have received an alleged p_i before p_{i-1} , when we receive another alleged p_i , we cannot determine, without p_{i-1} , which of the two p_i 's is the bona fide p_i . So before p_{i-1} is received, we have no choice but to buffer all alleged p_i 's, offering an attacker the opportunity to launch denial-of-service (DoS) attacks using fake p_i 's. Therefore, p_i 's must be received in sequence.

This general approach originates in Lanigan et al.'s Sluice [12]. Sluice was designed for Deluge, although it can be used for any network reprogramming scheme that partitions code into pages. As such, Sluice imposes some restrictions on the system: (1) all pages must be delivered in strict order; (2) the next page cannot start if the current page has not yet finished; and (3) a node will have to buffer a whole page before verifying it. Subsequent proposals are incremental improvements of Sluice. Dutta et al. [5] propose to operate at the packet level instead of at the page level. Dutta et al.'s proposal is Deluge-specific.

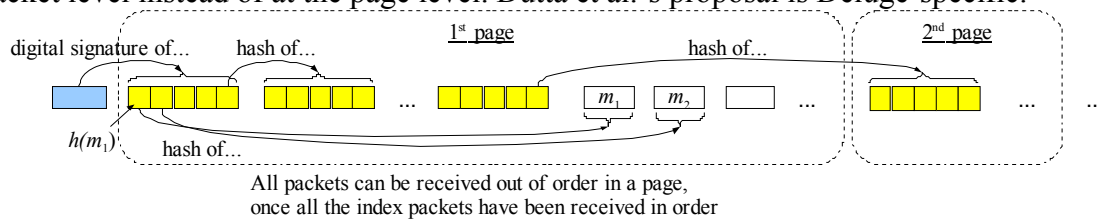


Figure 2: Intermediate scheme (not in the literature)

Deng et al. [4] note that hash chaining forces the packets to be received in order. It is only possible to receive the packets out-of-order if their corresponding hashes are received in advance. If there are n packets, then we need to verify n hashes in advance. Using only one hash for all n packets, e.g. $h(m_1||m_2||\dots||m_n)$, is vulnerable to DoS attacks, because it takes only one bit of error to make all n packets useless. There are two approaches:

1. We can sign each of the hashes using the administrator's private key, but signing each of the hashes is too energy-consuming.
2. We can chain the hashes as in Figure 2 and let the hashes arrive before the packets, but if we chain the hashes, we still need to enforce the hashes to arrive in order, even though the packets themselves can arrive out-of-order. We call this scheme the *intermediate scheme* for ease of discussion later.

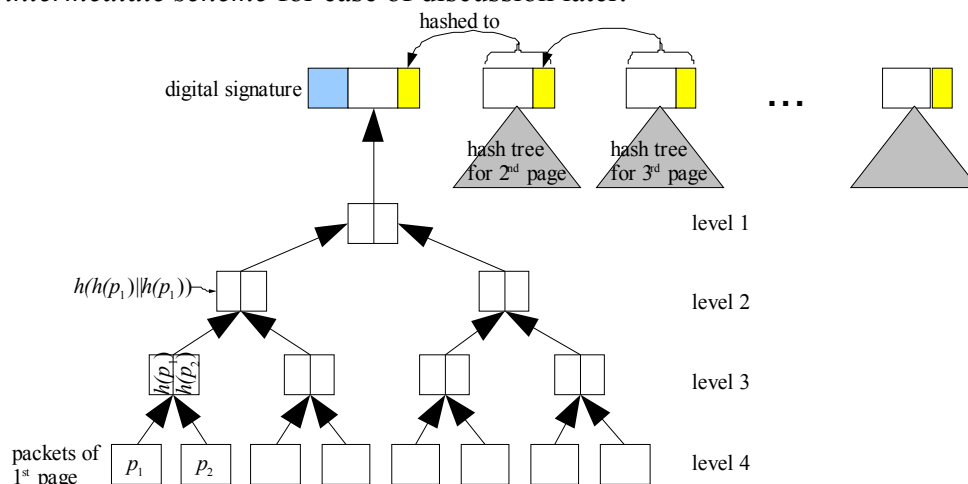


Figure 3: Deng et al.'s scheme [4]; example shows a packet can store at most two hashes

A common approach, (also Deng et. al.'s approach), is to build a tree on top of the n hashes until we get a single hash at the root of the tree, and sign the root of the tree instead (Figure 3). This hash tree is not a Merkle tree because it is Deng et. al.'s intention to create a hash for each packet at the bottom level of the tree. As many hashes are placed in a packet as

allowed by the maximum transmission unit; such packets are called *index packets*. Pages are chained, and hence have to be received in order. The levels also have to be received in order, but within each page and within each level of the hash tree, the packets can be received out-of-order. The index packets at level $l - 1$ can be deleted once all index packets at level l have been received and verified. Deng et al. [4] show that, using Tmote nodes, under a packet loss rate of 5%, while the chain scheme (Figure 1) incurs a cost of 308% more messages, their scheme (Figure 3) only costs 43% more messages. The advantage of being able to receive packets out-of-order is apparent.

The number of index packets measures the overhead and can easily be calculated as follows. Say there are n number of packets in a page, and we can cram a maximum of w hashes into an index packet, and there are L levels in the hash tree (taking the root as level 0), then there are w^L hashes at the bottom of the tree. To accommodate all packets in a page, we must have $w^L \geq n$, or $L = \lceil \log_w n \rceil$. The number of index packets per n nodes, α , is thus

$$\alpha = 1 + \sum_{l=1}^L w^{l-1} = 1 + (w^L - 1)/(w - 1) \quad (1)$$

If we denote the number of bits in a hash as σ , then the number of bits in the index packets per n nodes, β , is

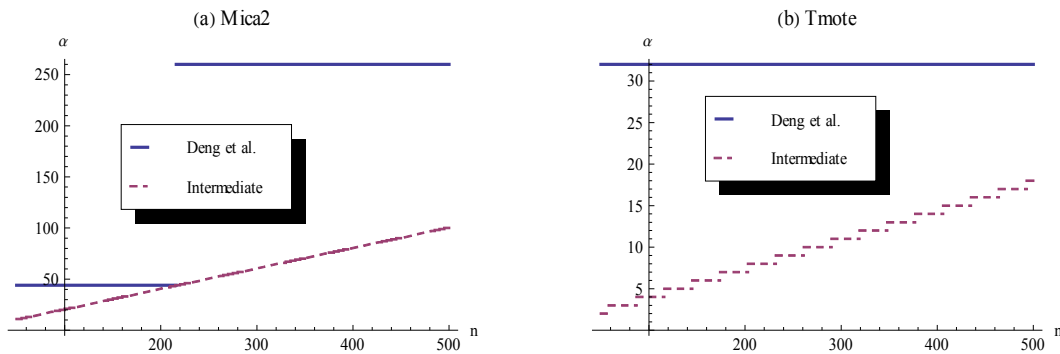
$$\beta = \sigma \left[1 + \sum_{l=1}^L (w^l - \lfloor w^l - w^{l-L} n \rfloor) \right] \quad (2)$$

It is a simple counting exercise to obtain Equation (2). There is some flexibility in determining the page size n depending on whether α or β is to be minimized.

There is a proposal [26] for using a little-known and patented public-key scheme called “combined public key” to secure Advertisement and Request messages. The obvious problem with the approach is the resource requirement. Securing the Advertisement and Request messages is the problem of a larger scale, i.e., key management – we shall not address it here. The most important issue at hand is the security of the code that gets downloaded into all of the nodes and which eventually becomes the “souls” of the nodes.

Pages and packets are the prevalent code structure. The concept of pages was initiated by Deluge’s philosophy of (1) limiting the amount of state a receiver must maintain while receiving data, (2) enabling efficient incremental upgrades, and (3) allowing pipelining. This means the page/packet structure has nothing to do with the organization of the code itself. What needs to be explored is how the page size – which is determined by whether α or β is to be minimized – impacts the underlying protocol.

3.1 Further Analysis



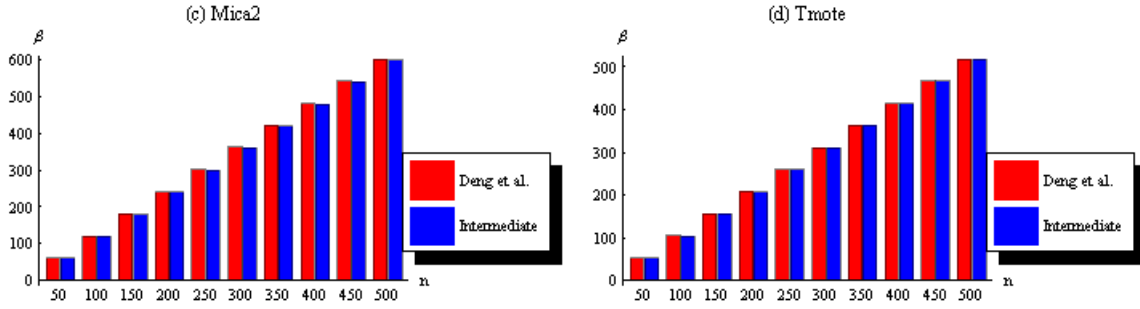


Figure 4: Comparison of Deng et al.'s scheme to the intermediate scheme

By comparing Deng et al.'s scheme with the intermediate scheme, there are some interesting discoveries. For Deng et al.'s scheme, α_{Deng} and β_{Deng} can be computed using (1) and (2) respectively. For the intermediate scheme,

$$\alpha_{intmd} = \lceil n / (w-1) \rceil \quad \beta_{intmd} = n + \lceil n / (w-1) \rceil \quad (3)$$

From (1) and (3), when $w^L \geq n + 1$, we have

$$\alpha_{Deng} = 1 + (w^L - 1) / (w - 1) = (w^L + w - 2) / (w - 1) \geq n / (w - 1) + 1 \geq \lceil n / (w - 1) \rceil = \alpha_{intmd} \quad (4)$$

In other words, when n is not a power of w , Deng et al.'s scheme uses more index packets than the intermediate scheme. It is more difficult to compare the β 's of the two schemes analytically, so we resort to the plots in Figure 4. The plots are based on these parameters: for Mica2 motes, $w = 6$, whereas for Tmote motes, $w = 30$. The plots show that the β 's of the two schemes are virtually identical. Since the intermediate scheme incurs lesser index packets most of the time, the intermediate scheme delivers more hash bits per hash packets (consistent with intuition). The disadvantage of the intermediate scheme is that for every n packets, $\lceil n / (w-1) \rceil$ of the packets have to be received in order, whereas in Deng et al.'s scheme, only $\lceil \log_w n \rceil$ levels of packets have to be received in order. It is difficult to evaluate the impact of this difference analytically due to the dynamics of the underlying data dissemination protocol. We propose simulations for this purpose as future work.

4. Review of Code Verification Algorithms

When the code arrives at a node, the code has to be cryptographically verified. The public-key algorithm should of course be energy-efficient, and the signature that the algorithm produces should be able to fit in a single packet. Since we are only using the signature verification part of the algorithm, significant savings can be made on the code size. So far the most efficient type of algorithm is ECC [22] – to put things into perspective, one of the earliest benchmark results of RSA-1024 on ATmega128L measures the time for an exponentiation at between 30s and 80s depending on the size of the exponent [26]. There have been notable advances in ECC implementation for WSNs in recent years (Table 2).

Table 2: ECC implementations for sensor node platforms (in approximate chronological order)

Authors	Platform	Algorithm	Operation	Time (s)
Guajardo et al. [6]	MSP430x33x 3MHz	ECC-160	Point multiplication	1.10
Gura et al. [7]	ATmega128L 8MHz	ECC-160	Point multiplication	0.81
Malan et al. [15]	ATmega128L 8MHz	ECC-160	Private key generation	0.23
			Public key generation	34.16
Liu et al. (TinyECC ⁶)	MSP430F1611 4MHz	ECC-160	Signature verification	2.44
	ATmega128L 8MHz	ECC-160	Signature	4.04

⁶ <http://discovery.csc.ncsu.edu/software/TinyECC>

			verification	
Ugus et al. [22]	ATmega128L 8MHz	ECC-160	Point multiplication	1.03
			Point multiplication (2 pre-computed points)	0.57
Wang et al. [24]	MSP430F1611 4MHz	ECC-160	Point multiplication (prime field)	3.13
Szczechowiak et al. (NanoECC) [21]	MSP430F1611 4MHz	ECC-160	Point multiplication (prime field)	0.72
			Point multiplication (binary field)	1.04

Most publications use the time for one point multiplication as the benchmark, but our interest is primarily in the energy for signature verification. Nevertheless, it is apparent that NanoECC represents the best result so far. We propose a few areas to explore further:

- If software update is done very infrequently, then code size can be more important than energy-efficiency. NanoECC uses about 30KB of code size, or over 60% of the Flash memory space available on a MSP430F1611, so future work should explore energy-storage trade-off in the implementation.
- Several exciting architectures have emerged in the market, for example Chipcon's ZigBee SoC offerings with low-power 8051 cores. These chips feature a low 0.3 μ A standby mode current, and hence are potential platforms to explore.

After the signature is verified, the code should also be checked for consistency and safety, such as the presence of potentially dangerous instructions, before it is loaded.

5. Conclusions and Future Work

Network reprogramming is a tool for instant network morphosis. The security of the 'tool' is vital to the commercial success of the technology. The best approach is not piecing several components together, but a concerted effort at integrating the components and optimizing the components for each other. In this paper, we have explored the three design spaces: (1) data dissemination protocols, (2) cryptographic protocols for data dissemination, and (3) code verification. Our end goal is to integrate and cross-optimize these three components. The following summarizes our agenda/recommendation. For data dissemination protocols, our deduction is that to surpass Deluge or any of its subsequent derivatives, revolutionary techniques are needed, otherwise incremental improvements might be achieved by looking into the cross-layer optimization between the TDMA-based media access control layer and the data dissemination protocol layer. For cryptographic protocols, we propose exploring the relative merits of Deng et al.'s scheme and the so-called intermediate scheme further. For code verification algorithms, we propose exploring energy-storage trade-off in the latest implementations, and optimizing the implementations for emergent 8051-based SoC platforms.

References

- [1] M. Arumugam, "Infuse: a TDMA based reprogramming service for sensor networks", in Proceedings of the 2nd international Conference on Embedded Networked Sensor Systems (SenSys '04), poster abstract, ACM, 2004.
- [2] B. Clark, C. Colbourn and D. Johnson, "Unit disk graphs", *Discrete Mathematics*, 86:165–177, 1990.
- [3] C.-C. Han, R. Kumar, R. Shea and M. Srivastava, "Sensor network software update management: a survey", *Int. J. Netw. Manag.*, 15:4, pages 1099-1190, pages 283-294, John Wiley & Sons, Inc., 2005.
- [4] J. Deng and R. Han and S. Mishra, "Secure code distribution in dynamically programmable wireless sensor networks", in *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 292-300, ACM, 2006.

- [5] P.K. Dutta, J.W. Hui, D.C. Chu and D.E. Culler, "Securing the Deluge network programming system", in IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks, pages 326-333, ACM, 2006.
- [6] J. Guajardo, R. Bluemel, U. Krieger, C. Paar, "Efficient implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x family of microcontrollers", in PKC 2001, LNCS, vol. 1992, Springer, Heidelberg, 2001.
- [7] N. Gura, A. Patel, A. Wander, H. Eberle, S. Chang Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs", CHES 2004, Volume 3156 of LNCS, pages 119-132, 2004.
- [8] W.R. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks", in Proceedings of the 5th Annual ACM/IEEE international Conference on Mobile Computing and Networking (MobiCom '99), pages 174-185, ACM, 1999.
- [9] J.W. Hui and D. Culler, "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 (SenSys '04), pages 81-94, ACM, 2004.
- [10] M.D. Krasniewski, R.K. Panta, S. Bagchi, C.-L. Yang and W.J. Chappell, "Energy-efficient on-demand reprogramming of large-scale sensor networks", ACM Trans. Sen. Netw., 4(1):1-38, 2008.
- [11] S. S. Kulkarni and L. Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks", in Proc. 25th IEEE Int. Conf. on Distributed Computing Systems (ICDCS 2005), pages 7-16, IEEE, 2005.
- [12] P.E. Lanigan, R. Gandhi and P. Narasimhan, "Sluice: Secure Dissemination of Code Updates in Sensor Networks", in Proc. 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), pages 53-62, IEEE Computer Society, 2006.
- [13] P. Levis and D. Culler, "The firecracker protocol", in Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, ACM, 2004.
- [14] P. Levis, N. Patel, S. Shenker and D. Culler, "Trickle: A Self-Regulating Algorithm for Code Propagation and maintenance in Wireless Sensor Networks", in Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.
- [15] D.J. Malan, M. Welsh, M.D. Smith, "A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography," in 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004), pages 71-80, IEEE, 2004.
- [16] V. Naik, A. Arora, P. Sinha and H. Zhang, "Sprinkler: a reliable and energy efficient data dissemination service for wireless embedded devices", in Proc. 26th IEEE International Real-Time Systems Symposium (RTSS 2005), IEEE Computer Society, 2005.
- [17] R.K. Panta, I. Khalil and S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks", 26th IEEE Int. Conf. on Computer Communications (INFOCOM 2007), pp.928-936, 2007.
- [18] A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D. Tygar, "SPINS: Security Protocols for Sensor Networks", in Proceedings of the 7th Ann. Int. Conf. on Mobile Computing and Networking, pages 189-199, ACM Press, 2001.
- [19] L. A. Phillips, "Aqueduct: Robust and efficient code propagation in heterogeneous wireless sensor networks", Master's thesis, University of Colorado at Boulder, 2005.
- [20] T. Stathopoulos, J. Heidemann and D. Estrin, "A remote code update mechanism for wireless sensor networks", CENS Technical Report 30, 2003.
- [21] P. Szczechowiak, L.B. Oliveira, M. Scott, M. Collier and R. Dahab, "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks", in EWSN 2008, volume 4913 of LNCS, pages 305-320, Springer-Verlag, 2008.
- [22] O. Ugus and A. Hessler and D. Westhoff, "Performance of Additive Homomorphic EC-ElGamal Encryption for TinyPEDS", <http://www.ist-ubisecsens.org/publications/EcElgamal-UgHesWest.pdf>
- [23] A.S. Wander, N. Gura, H. Eberle, V. Gupta and S. Chang Shantz, "Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks", in The Third IEEE International Conference on Pervasive Computing and Communications (PERCOM'05), pages 324—328, IEEE, 2005.
- [24] H. Wang, B. Sheng and Q. Li, "Elliptic Curve Cryptography based access control in sensor networks", International Journal of Security and Networks (IJSN), Special Issue on Security Issues on Sensor Networks 1(3/4), pages 127–137, 2006.
- [25] Q. Wang, Y. Zhu and L. Cheng, "Reprogramming wireless sensor networks: challenges and approaches", IEEE Network, 20(3):48-55, 2006.
- [26] R.J. Watro, D. Kong, S.-F. Cuti, C. Gardiner, C. Lynn and P. Kruus, "TinyPK: securing sensor networks with public key technology", in SASN 2004: 2nd ACM Workshop on Security of ad hoc and Sensor Networks, Washington, DC, pp. 59–64, ACM, 2004.
- [27] Y. Zhang, X.-S. Zhou, Y.-M. Ji, Z.-Y. Fang and L.-F. Wang, "Secure and DoS-Resistant Network Reprogramming in Sensor Networks Based on CPK", 4th IEEE International Conference on Wireless Communications, Networking and Mobile Computing, 2008.