

Secure Wireless Sensor Networks Against Attacks

Well-established principles, appropriate protocols and ciphers, and the randomness inherent in the physics of thermal noise add up to fortify systems to ensure they are both secure and efficient.

The Internet of Things (IoT) is growing rapidly, and wireless sensor networks (WSNs) are critical to extending the reach of the Internet infrastructure to “everything.” In fact, WSNs are already in use in critical monitoring and control applications around the world. Any loss of security in these systems may have real and direct consequences on efficiency and safety. Fortunately, the literature on securing wireless systems is readily available, and best practices are well known.

Despite this knowledge, the news is filled with reports documenting successful attacks on wireless in general and WSNs in particular. Surprisingly, many products on the market do not embrace even the most basic aspects of system security, and many other products with well-intended security fall short of the mark. Wireless security is not trivial, but with rigorous attention to detail, it is possible to build systems that are not vulnerable to wireless attack.

GOALS & CONSEQUENCES

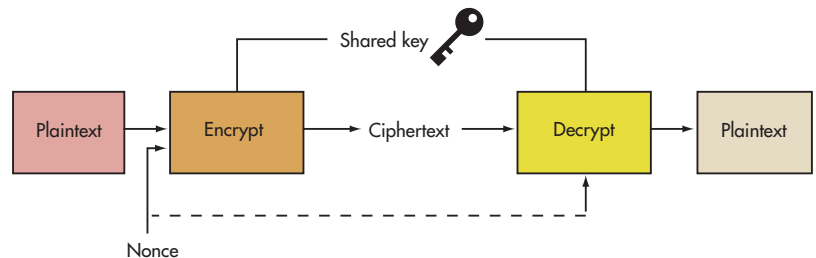
Today’s security issues are not limited to wireless systems. Indeed, Internet attacks big and small are so common today that they are barely newsworthy. There is a perception that wireless systems are more vulnerable to attack, because anyone with the appropriate radio can communicate with a wireless device from some distance. Of course, on the Internet, anyone with a computer can launch an attack from distances far longer than any radio signal will propagate. The bottom line is that all cyber-physical systems, whether wired or wireless, need to take careful precautions against attack. The primary goals of security in WSNs are based on providing three elements:

- Confidentiality: Data being transported in the network cannot be read by anyone but the intended recipient.
- Integrity: Any message received is known to be exactly the message that was sent, without additions, deletions, or modifications of the content.
- Authenticity: A message that claims to be from a given source is, in fact, from that source. If time is used as part of the authentication scheme, authenticity also protects a message from being recorded and replayed later.

Confidentiality is required for not only security-related applications but also for common everyday applications. For example, sensor information regarding production levels or equipment status may have some competitive sensitivity. Sensor data should be encrypted so only the intended recipient can use it.

Both sensing and command information must arrive intact. If a sensor indicates “the tank level is 72 cm” or the controller states “turn the valve to 90 degrees,” losing one of the digits in either one of those numbers could be catastrophic.

Having confidence in the source of a message is critical. Either of the two messages above could have very bad consequences if



1. A source node uses a block cipher, a nonce, and a secret key to encrypt a plain text message, turning it into ciphertext. Only someone in possession of the key can decrypt the message. The nonce may be sent in the clear, or it may be implicit from the structure of the protocol, such as a timestamp.

it were sent by a malicious attacker. An extreme example is a message like “here’s a new program for you to run.”

The consequences of poor security are not always easy to anticipate. For example, a wireless temperature sensor or thermostat might seem like a product with little need for security. But consider the consequences to product sales due to a newspaper headline describing how criminals used a radio to detect the “vacation” setting on the thermostat and robbed those houses while the owners were gone. The safest course is to encrypt all data.

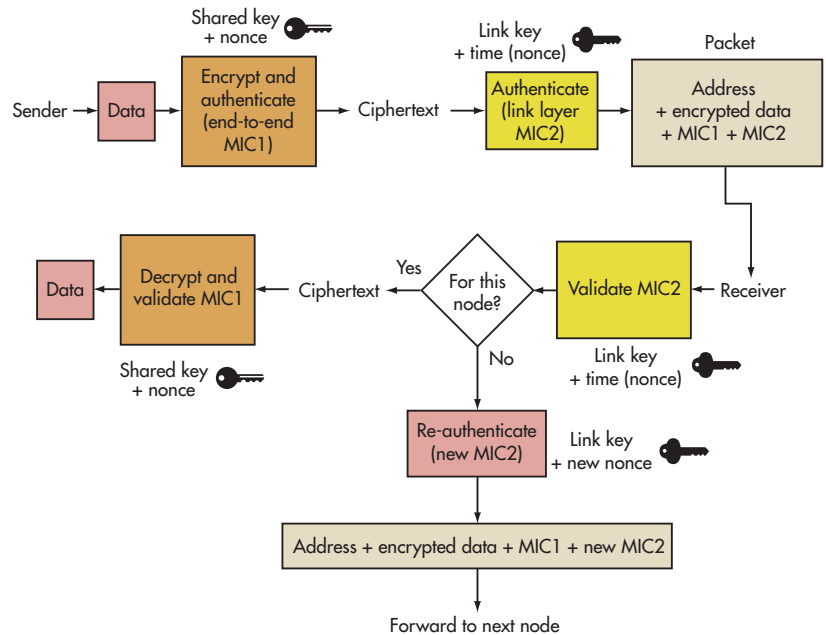
In the early days of ZigBee, most networks were run with no security. As a result, in a multi-vendor interoperability demonstration in front of many potential customers, a number of ZigBee networks failed dramatically because they interpreted a command from a different network to be a coordinator realignment message that told them to change channels. There was no way for the ZigBee networks to determine that the messages were coming from a device that was not in their network. This disastrous behavior was not the result of an actual attack, but rather a lack of authentication, which led to interpretation of packets from a completely different network.

In industrial process automation, the consequences of an attack may be much more dire than the loss of a customer. If faulty process control information is delivered to the control system, an attacker could cause physical damage. For example, a sensor feeding data to a motor or valve controller indicating that the motor speed or tank level is too low could result in a catastrophic failure, similar to what happened to the centrifuges in the Stuxnet attack.¹

On a purely practical level, even a failed attack or an academic revelation of a potential weakness is likely to lead to a loss of sales, urgent engineering effort, and a major public relations challenge. Fortunately, there are very powerful tools for building secure, robust wireless communications networks. It takes diligence and attention to detail, but there is nothing fundamentally hard about it.

SECURITY TOOLS

The most basic cryptographic tool is the block cipher. As an example, AES-128 is a particular block cipher that takes a 16-byte message (the plaintext) together with a 128-bit key and generates a 16-byte encrypted version of the message (the



2. Most secure WSN systems use two levels of keys: end-to-end session keys and one or more link keys. Each pair of nodes with an end-to-end communication channel will have a unique shared key for encryption and authentication. In addition, a link key is used for hop-by-hop authentication.

ciphertext). Anyone with the same key can decrypt the ciphertext to get back the plain text. Anyone without the key cannot get back the plain text.

The AES cipher is easy to implement in software and commonly available in hardware on many radio and microprocessor chips. As far as anyone knows, AES-128 is unbreakable. Given the ciphertext, there is absolutely no way to figure out the plain text without the key. Indeed, the U.S. National Security Agency chose this cipher for the encryption of secret documents. In all of the reported attacks on WSN security, no one has ever claimed that the AES cipher provided the weak link.

The only known attack on AES-128 is a so-called “brute force” attack, meaning that the attacker tries every possible key to see which one gives a reasonable message. Trying every possible 128-bit key is a big task. If you had a billion computers, and each computer could check a billion keys every second, and you ran all of those computers for a billion years, you would only try about 0.1% of all of the possible 128-bit keys. There are more than 300 billion billion billion billion different 128-bit keys.

A block cipher lets the source encrypt a message so only the destination (with the same key) can decrypt it (Fig. 1). If the messages are something simple like “turn the light on” or “turn the light off,” then even if the messages are encrypted to seemingly meaningless strings of bits, anyone intercepting a few messages will quickly figure out that there are only two different messages.

A solution to this problem is to have a message counter and then number each message sent. Due to the nature of the cipher, any change in the message plain text will result in a different ciphertext. Two messages sent at different times, such as “Msg 1: turn the light on” and “Msg 53: turn the light on,” will look completely different to anyone who doesn’t have the key. As long as the message counter never repeats, the ciphertext will also never repeat. This concept of a message counter that never repeats is called a nonce, for “number used once.”

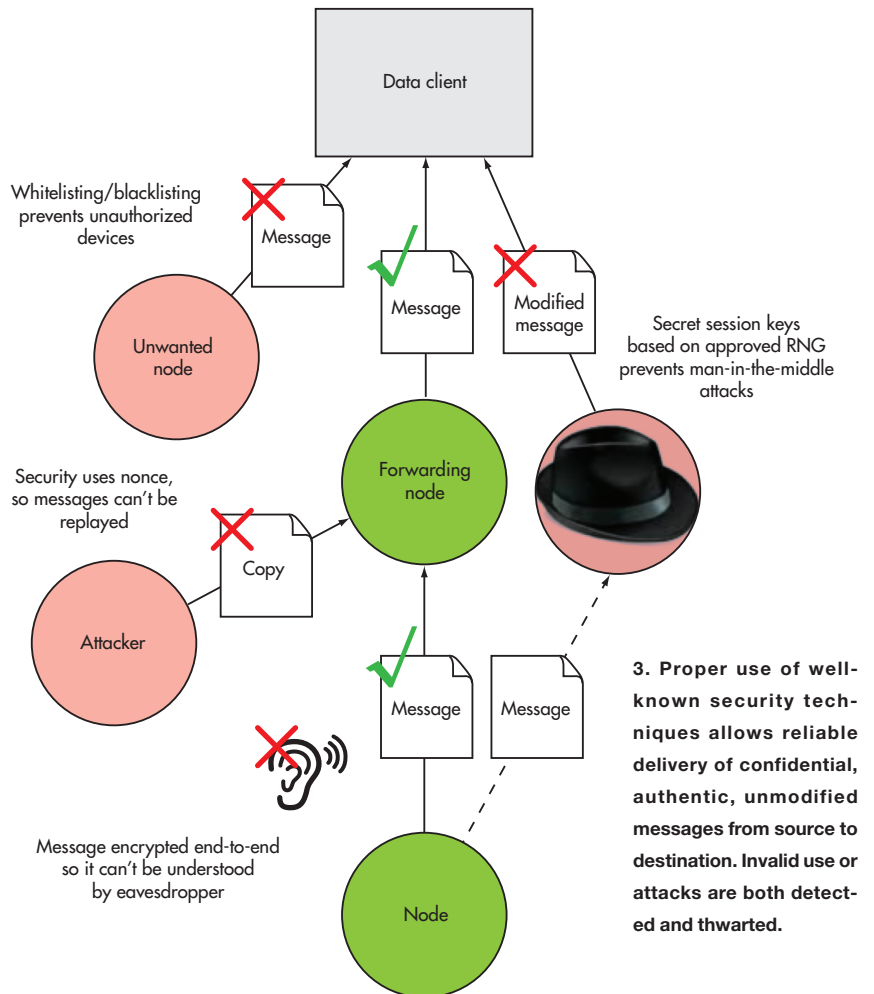
The message integrity check (MIC), also sometimes called a message authentication code (MAC), is a cryptographic checksum of the message. By sequentially running all parts of a message through a block cipher with a particular key, the sender of the message creates a short encrypted summary of the entire message, called the message integrity check.

This MIC is then appended to the message and sent along with it. The receiver, using the same key, can perform the same function on the message, calculate its own MIC, and verify that the result matches the MIC that was received. Any changes to the message, even a single bit, will cause the MIC to change and therefore cause the message to be rejected by the recipient.

Also, a person can generate the encryption keys in a WSN, but this is typically impractical and ultimately insecure, as we will see below. Ideally, we’d ask computers to generate the keys for us. We don’t want anyone to be able to guess the keys, so we’d like them to be random, and that requires a random number generator (RNG).

Usually people are happiest with computers when they are completely deterministic, and random behavior is frowned upon. Making a computer truly random is not a trivial task, and it always involves interaction with something that’s non-digital.

Fortunately, radios are intrinsically non-digital, and it has taken a century of progress from the days of Marconi to get them to the point where they deliver digital messages reliably. Any well-designed WSN system will use the radio or some other source of thermal noise as an integral part of its RNG and will generate truly random numbers.



Finally, even a legitimately obtained yet incorrectly deployed device could confuse a control system not expecting an additional input. Access control lists (whitelists, blacklists) provide an additional layer of control to ensure that unwanted devices can’t disrupt a network.

LACK OF UNDERSTANDING

The single most common mistake in WSN security is not appreciating the magnitude of the problem until it is too late. Building and deploying a wireless lighting control system without security may not sound like a problem until the local college students start making light shows out of your customers’ office spaces. Even those who realize that security is important may not appreciate the widespread sophistication, software and hardware tools, and skills that are available and regularly applied on the dark side of this conflict.

Several WSN companies boast that their channel-hopping protocols have some security benefit, as if an attacker will not be able to buy a multichannel receiver and transmitter. Others

seem to think that millions or billions of keys is enough to prevent a successful attack, when in fact even billions of billions is not enough.² If there is the possibility of a brute force attack revealing a key, then a key-rotation schedule that is a small fraction of the brute force time will deter the average attacker, but might not stop a very lucky one.



4. Linear Technology's SmartMesh LTC5800 (system-on-chip) and LTP5900 (module) families are the industry's lowest-power IEEE 802.15.4E compliant wireless sensor networking products. SmartMesh ICs and modules enable tiny sensor "motest" to be designed with a battery life of over 10 years, while companion network manager components enable the development of highly robust and secure WSNs.

SHARED KEYS AND SOFTWARE REVERSE ENGINEERING

Assuming that a proper cipher has been chosen, and nonces are used, the simplest system will use a single shared key for all cryptographic operations. This approach is fine as long as the key remains secret, but that is a difficult goal to achieve.

An extreme example is the reported vulnerability of a Bluetooth-controlled toilet/bidet combo, in which the default pairing key of all zeros was used.³ This is really more an example of "no security" than poor security, but illustrates the point that the best protocols are no defense against poorly chosen keys, or even a random key that becomes widely known on the Internet. Bluetooth has excellent security tools. But if you don't use them properly, they are worthless once someone publishes your ill-advised product-wide key on the Web.

The next level is to have a single unique key for each network that is delivered or installed, or a new key each time a network is formed. If you have a good RNG, and you control all of the hardware in your network, then this approach is fine. But if any one node in the network is compromised, then the entire network is open to attack. If users are allowed to write their own software on the nodes in the network, then it is quite difficult to prevent a malicious user from finding the network key.

Even if the node software is closed, it is quite difficult to prevent attackers from reading out the code in a microprocessor if they have possession of the hardware. The security literature is filled with examples of such attacks, which often go like this: obtain a legal version of the hardware and break into it to get the code; reverse engineer the code to figure out where the key is stored (this can be as simple as comparing the code from two different networks to see which bits are different); and use this information to either figure out how the key was calculated (see "Poor Quality RNG" below) or to make it much quicker to get the key out of hardware captured from the actual network under attack.

DVD security has fallen victim to such attacks, both with the original DVD Content Scrambling System (CSS) and the HD-DVD/Blu-ray Advanced Access

Content System (AACs). Hackers examining player code and exposing and publishing several of the processing keys protecting that material compromised both.⁴

With very rare exceptions, you must assume that a determined attacker will be able to obtain your hardware, read out your code, and reverse-engineer your algorithms and software. As a result, a well-designed security system must not depend on the algorithms and software remaining secret, and it must not rely on the key or keys in any one device remaining secret. An attacker that has one of the network nodes must be assumed to be able to gain complete control of that node. In a well-designed system, the compromise of a single node must not affect security in the rest of the network.

The simplest solution to this reverse-engineering problem is to ensure that every communication session (or flow of data between two endpoints) has its own unique keys that are unknown to any other nodes in the network. In this case, even a compromised node in the network cannot snoop, manipulate, or impersonate the data or commands from any other node in the network.

KEY DISTRIBUTION

Assuming that appropriate protocols and ciphers are used, a network with unique random keys for each end-to-end session protects the confidentiality, integrity, and authenticity of network communication (Fig. 2). However, arriving at this situation presents vulnerabilities in some systems.

It is usually inconvenient to pre-program every node in the network with all of the unique keys that it will need for all future sessions, so keys need to be distributed after network formation. In some systems, this has been done by sending the initial session keys "in the clear" (not encrypted), under the assumption that the network is then only vulnerable to an attack for a brief period during network formation. Unfortunately, an attacker may well be able to set up snooping equipment and wait patiently for a network reset, or in fact cause a net-

MORE ON WIRELESS SENSORS

- Go to *electronicdesign.com* and see:
- Advanced Smart Sensor Networks Open Up A Multitude Of Applications
 - Wireless Vibration Sensors Enable Continuous And Reliable Process Monitoring
 - Energy Harvesting Powers Wireless Sensor Networks In Industrial Apps

work reset by power cycling the network controller or gateway, or through some other method.

A simple solution is to install a single unique key on each node in the network at manufacturing time and have a single trusted security manager in the network, which is given those keys, allowing a secure session between each node and the security manager. The security manager then generates the required keys for all other sessions and sends them via its secure channels to each of the devices involved. Alternatively, there is another suite of tools using public key infrastructure that provides similar functionality as well as other benefits.⁵

POOR QUALITY RNG

Among those who take security seriously, perhaps the most common mistake is the use of an RNG with poor randomness. Even with all of the proper protocols and ciphers, the network is only as challenging to attack as the keys are difficult to guess. The use of non-cryptographic random number generators or cryptographic random number generators with seeds (initial values) that are non-random is a common mistake.

Random numbers are useful in many different applications in computer science, so many operating systems have a “rand()” function built in. For example, the original UNIX rand() function maintained an internal 32-bit state and computed the next random number and next state based on the current state. A user could seed this RNG with a 32-bit number, and then each call to rand() would generate the next value in a sequence of 4 billion values. It wasn't a great RNG, but it was good enough for most non-cryptographic applications.

Today, however, it would be a simple homework assignment to generate a table in a single desktop computer that contained all 4 billion possible random numbers and their location in the sequence. No amount of randomizing

the seed will help. The RNG itself is not sufficiently sophisticated.

Cryptographic RNGs use much more internal state—typically at least 128 bits. With 128 bits, as discussed above, even billions of computers operating for billions of years are extremely unlikely to find a pattern in the sequence of numbers. The implementation and test procedures of good cryptographic RNGs are well documented.⁶

Even the best RNG algorithm is only as random as the seed it was provided. IOActive pointed out a common mistake in two WSN security systems by reverse engineering the software binaries of both products to discover that they were using a very non-random seed.⁷ Both products used the time function (in seconds) as the seed of their random number generator. Since there are only a few tens of millions of seconds per year, even a modest laptop computer can generate all possible keys by a quick search of the last few decades.

SECURE NETWORKS

While the news is full of examples of failed wireless security, the world is filled with wireless networks that are, in fact, secure. Secure networks just aren't newsworthy. As discussed, a secure network requires both a secure protocol and a secure implementation (*Fig. 3*).

Some examples of well-designed security protocols in WSNs include the Wireless HART and ISA100.11a industrial automation protocols, as well as the ZigBee Smart Energy protocols. All of these protocols have undergone extensive review by security experts, and many implementations have sailed through similar review.

In particular, the Wireless HART protocol is the basis of secure networks deployed in critical infrastructure applications all over the world, from the Arctic Circle to the Arabian Desert. End users of this technology trust WSNs to supply process control information reliably and confidentially between authenticated end

MORE CONTRIBUTED ARTICLES


For more articles that address common design problems written by engineers just like you, go to <http://electronicdesign.com/learning-resources/design-solutions>. To submit your own article, go to <http://electronicdesign.com/submit-articles>.

points. Customers in such industries, as well as the vendors who supply them, have confidence in their networks because of deep analysis and testing of the protocols and implementations that underlie them.

In WSNs for industrial process automation, it was understood from the beginning that security was critical, and the protocols and implementations reflect that reality. As new protocols emerge, especially for the Internet of Things, some hard lessons will need to be relearned in application environments where it may not be as obvious that security is critical. As the preceding examples have shown, there are some who have not yet learned these lessons. Fortunately, it is just as easy to provide “industrial quality” security in Internet Protocol (IP) applications as in industrial applications.

End users have deployed SmartMesh IP embedded wireless mesh sensor networks from Linear Technology’s Dust Networks line in some of the toughest RF environments, including industrial process plants, data centers, smart parking applications, railcars, and mining. Built on Linear’s ultra-low-power LTC5800 802.15.4 system-on-chip, SmartMesh networks are embedded systems complete with hardware and networking software that deliver secure mesh sensor networks with >99.999% data reliability and ultra-low power (*Fig. 4*). SmartMesh networks have several layers of security to address confidentiality (through encryption), integrity (of a message), and authenticity (verifying that a message is from the stated sender).

CONCLUSION

The consequences of poor security in wireless sensor networks are severe. It is unfortunate that there has not been a serious attempt to secure many products on the market today, or that those that have done so have failed in that effort. Fortunately, by using well-established principles, appropriate protocols and ciphers, and the randomness inherent in the physics of thermal noise, it is possible to build systems that are both secure and efficient. Many such protocols and implementations exist, and there are secure wireless networks all over the world. Everyone in the wireless sensor networking space will benefit when all of the rest of the networks are secure too. 

REFERENCES

1. “The Real Story of Stuxnet,” IEEE Spectrum, David Kushner, <http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet>
2. “Cracking DES,” Electronic Frontier Foundation, O’Reilly Media, 1998.

IndustryTrends

3. TrustWave SpiderLabs Security Advisory TWSL2013-020: Hard-Coded Bluetooth PIN Vulnerability in LXL Satis Toilet, <https://www3.trustwave.com/spiderlabs/advisories/TWSL2013-020.txt>
4. AACS encryption key controversy, https://en.wikipedia.org/wiki/AACS_encryption_key_controversy
5. Public-key infrastructure, https://en.wikipedia.org/wiki/Public_key_infrastructure
6. "Annex C: Approved Random Number Generators for FIPS PUB 1402, Security Requirements for Cryptographic Modules," Randall J. Easter and Carolyn French, <http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf>
7. "Compromising Industrial Facilities from 40 miles away," IOActive, Lucas Apa and Carlos Holmman, Blackhat 2013.

KRIS PISTER is the founder and chief technologist of the Dust Networks Product Group, Linear Technology Corp. The inventor of Smart Dust, he cofounded Dust Net-



works in 2002 to deliver his vision of a commercially robust wireless sensor networking platform. The company was acquired by Linear Technology in 2011. He also is the chief architect of Dust Networks' patent pending Dust SmartMesh technology. He is a frequent invited speaker and lecturer on wireless sensor networking and related core technologies. He is also a professor of electrical engineering and computer sciences at the University of California, Berkeley. He holds a PhD and MS in electrical engineering and computer sciences from UC Berkeley and a BS from UC San Diego.

JONATHAN SIMON is the systems engineering lead of Dust Networks Products, Linear Technology Corp. He has over a decade of experience making the lowest power,



highest reliability wireless sensor networks in the world. Previously, he worked at Agilent research labs focusing on opto-mechanical design and thermal management for high-speed optical communications modules, at LLNL on counter proliferation technologies, and briefly as a special effects artist. He has a PhD in mechanical engineering (MEMS) from the University of California, Los Angeles.