Getting Security Right in Wireless Sensor Networks

K. Pister, Chief Technologist, Dust Networks® Product Group
J. Simon, Systems Engineering Director, Dust Networks Product Group

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. WSNs are already in use in critical monitoring and control applications around the planet. 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 WSN in particular. Surprisingly, many products on the market do not embrace even the most basic concepts of system security, and many other products with well-intended security fall short of the mark. We document here some of the common mistakes, and their well-known solutions. Wireless security is not trivial, but with rigorous attention to detail, it is straightforward to build systems that are not vulnerable to wireless attack.

BASICS

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, require careful precautions against attack.

Goals

The primary goals of security in WSN are to provide:

- **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.

Confidentiality is required not only for security-related applications, but also for common everyday applications. For example, sensor information regarding production levels or equipment status may have some competitive sensitivity – e.g., the National Security Agency (NSA) doesn't publish the power consumption of their data centers because this data might be used to estimate computing resources. Sensor data should be encrypted so that only the intended recipient can use it.

Both sensing and command information needs to arrive intact. If a sensor says "the tank level is 72cm" or the controller says "turn the valve to 90 degrees," it could be very bad to lose one of the digits in either one of those numbers.

Having confidence in the source of a message is critical. Either of the two messages above could have very bad consequences if they were sent by a malicious attacker. An extreme example is a message like "here's a new program for you to run."

Consequences

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. However, imagine 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 impact on customer loyalty, let alone sales, would be dramatic. The safest course is to encrypt all data.

In the early days of ZigBee, most networks were run without any 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, which 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. With faulty process control information being delivered to the control system, an attacker could cause physical damage. For example, a sensor feeding data to a motor or valve controller saying 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 [Stuxnet].

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.

TOOLS

Fortunately, there are powerful tools for building secure, robust wireless communications networks. It takes diligence and attention to detail, but there is nothing fundamentally hard about it.

Ciphers and Nonces

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 ciphertext). Anyone with the same key can decrypt the ciphertext to get back the plaintext. Anyone without the key cannot get back the plaintext. The advanced encryption standard (AES) cipher is easy to implement in software, and is 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 plaintext without the key. Indeed, the same basic cipher was chosen by the US National Security Agency for encryption of top 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 determine which one gives a reasonable message. Trying every possible 128-bit key is a big task. If you had one billion computers, and each computer could check one billion keys every second, and you ran all of those computers for one 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 that only the destination (with the same key) can decrypt it. Of course, 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 number each message sent. Due to the nature of the cipher, any change in the message plaintext will result in a different ciphertext, and 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 not in possession of 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."

Message Integrity Check

The message integrity check, or MIC (also sometimes called a message authentication code or MAC), is a cryptographic checksum of the message. By sequentially running all of the 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. The receiver, using the same key, can then 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.

Random Number Generators

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 on. Making a computer truly random is not a trivial task, and always involves interaction with something 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.

Access Control

Even a legitimately obtained device incorrectly deployed 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 cannot disrupt a network.

MISTAKES

Lack of Understanding of the Problem

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 tout their channel hopping protocols as having 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 [DES-cracking1998].

Many people who understand that security is good in principle are concerned that it will not be practical, requiring too much computation or battery power. Fortunately, all of the technologies described in this paper can be (and are) used in wireless sensor nodes with very limited computational resources running at microwatt power levels. Others are worried about the "hassle" of security. As one concerned customer once said, "the only tools our installers know how to use are a sledgehammer and a blowtorch." Fortunately, most applications can be deployed such that all of the security mechanisms are automated, requiring no human interaction, and are completely transparent to the end user. The sledgehammer-wielding technicians now routinely install secure sensor networks, whether they know it or not.

Shared Keys and Software Reverse Engineering

If 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 recently reported vulnerability of a Bluetooth-controlled toilet/bidet combo, in which the default pairing key of all zeros was used [Trustwave]. 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 up 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 random number generator, and you control all of the hardware in your network, then this approach is fine. However, 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 an attacker from reading out the program in a microprocessor if he has 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).
- Use this information to either figure out how the key was calculated (see the Poor Quality RNG section), or to make it much quicker to get the key out of hardware captured from the actual network under attack.

Digital video disk (DVD) security has fallen victim to such an attack. Both the original DVD Content Scrambling System (CSS), and the HD-DVD/Blu-Ray Advanced Access Content System (AACS) were compromised by hackers examining player code and exposing and publishing several of the processing keys protecting that material [AACS].

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 in possession of one of the network nodes must be assumed to be able to gain complete control of that node, and 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

When 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. However, key distribution exposes vulnerabilities in some systems. It is usually inconvenient to preprogram 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 of time during network formation. Unfortunately, an attacker may well be able to set up his snooping equipment and wait patiently for a network reset, or in fact cause a network reset by power cycling the network controller or gateway, or through some other method.

A simple solution to this problem 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 that 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 [PKI].

Poor Quality RNG

Among those who take security seriously, perhaps the single most common mistake is the use of a random number generator with poor randomness. Even with all of the proper protocols and ciphers, the network is only as hard to attack as the keys are hard to guess. Common mistakes here are the use of noncryptographic random number generators, or cryptographic random number generators with seeds (initial values) that are nonrandom.

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 four 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 four 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 [NIST].

Even the best RNG algorithm is only as random as the seed it was provided. A common mistake in two WSN security systems was pointed out by [IOActive], in which they discovered by reverse engineering the software binaries of both products that they were using a very nonrandom seed. 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.

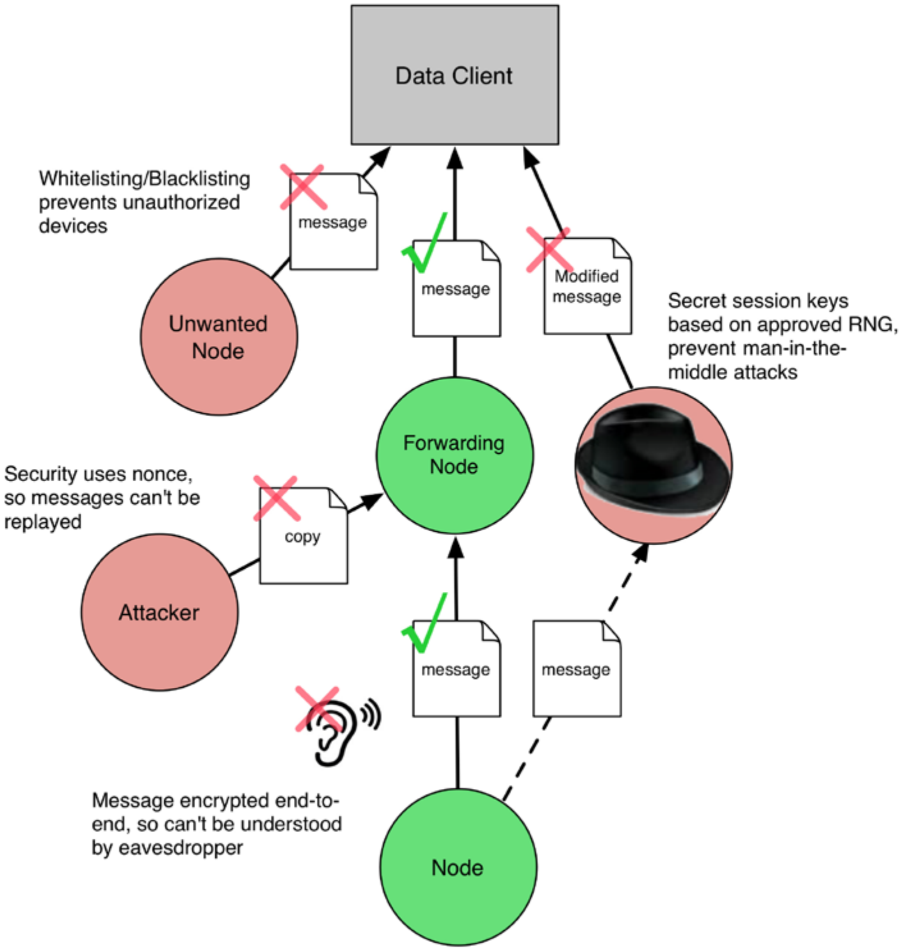
SECURE NETWORKS

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 often newsworthy. As discussed above, a secure network requires both a secure protocol, and a secure implementation.

Some examples of well-designed security protocols in WSN include the Wireless HART and ISA100.11a industrial automation protocols, and 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 WSN to supply process control information reliably and confidentially between authenticated endpoints. Customers in that industry, 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 WSN for industrial process automation, it was understood from the beginning that security is critical, and the protocols and implementations reflect that reality. As new protocols emerge, especially for the IoT, some of the hard lessons will need to be relearned in application environments where it may not be as obvious that security is critical. As examples above 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. Examples include SmartMesh IP™ from Linear Technology and several of the emerging IP standards.


Data Client
Whitelisting/Blacklisting prevents unauthorized devices
message
Unwanted Node
message
Modified message
Secret session keys based on approved RNG, prevent man-in-the-middle attacks
Forwarding Node
Security uses nonce, so messages can't be replayed
copy
Attacker
message
message
Message encrypted end-to-end, so can't be understood by eavesdropper
Node

CONCLUSION

The consequences of poor security in wireless sensor networks are severe. It is unfortunate that many products on the market today have either not made a serious attempt at being secure, or have failed in that effort. Fortunately, 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 the world is covered with secure wireless networks. Everyone in the wireless sensor networking space will benefit when all of the rest of the networks are secure, too.

BIBLIOGRAPHY

[Stuxnet] http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[DES-cracking1998] Electronic Frontier Foundation, "Cracking DES," O'Reilly Media, 1998.

[Trustwave] https://www.trustwave.com/spiderlabs/advisories/TWSL2013-020.txt

[AACS] https://en.wikipedia.org/wiki/AACS_encryption_key_controversy

[PKI] https://en.wikipedia.org/wiki/Public_key_infrastructure

[NIST] http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf

[IOActive] Lucas Apa, Carlos Hollman, "Compromising Industrial Facilities from 40 miles Away," Blackhat 2013.

Linear Technology Corporation
1630 McCarthy Blvd. Milpitas, CA 95035-7417
Tel: 408 432-1900 • FAX: (408) 434-0507 • www.linear.com

Information furnished by Linear Technology Corporation is believed to be accurate and reliable. However, no responsibility is assumed for its use. Linear Technology Corporation makes no representation that the interconnection of its circuits as described herein will not infringe on existing patent rights.

LT 1213 • PRINTED IN USA