Securing the Next Generation of Smart Devices

By Kristopher Ardis, Executive Director; Micros, Security & Software Business Unit, Maxim Integrated

May 2018

maxim integrated™

Abstract

As we discover the value in the hidden data all around us and we begin to build businesses based on data, the devices we create to collect that data want to become more invisible. But making them more invisible involves challenges in power management and processing. However, one particular challenge could kill the entire movement of the internet of things (IoT) before it can really begin to thrive: can we design devices that we trust enough to manufacture and that people feel comfortable enough to buy, install, and use?

Introduction

Things have intelligence

Security protects connected devices while enabling new opportunities

I'm not suggesting that inanimate objects have brains. In "Powering the Next Generation of Smart Devices" and "Adding Intelligence to the Next Generation of Smart Devices," I talked about what your house's door and what street lights could tell you if they had the capability to process more information. I also talked about the challenges of coaxing that intelligence out of the devices around us: we need to be able to power things responsibly and build enough intelligence into devices to provide value now and in the future. In fact, solving the power and the processing challenges will give us devices that provide usable intelligence and can last in the field for a long time in a perfect world. Unfortunately, no discussion of adding intelligence to things should be considered relatively thorough without covering the topic of security.

The world is not perfect, and any connected device that delivers valuable intelligence is a target for attack. Motivations for attacks can be varied: from the more humorous hacking of highway signs (which still can pose a public safety threat (Figure 1)) to financially motivated data theft to government-sponsored, targeted attacks on critical infrastructure, the move to connect devices makes them more vulnerable. The increasing value of the data they provide or assets they control can help tip the reward of an attack to outweigh the risks. In today's world, solving the power and processing challenges will get you an IoT device that does something, but addressing its security can protect your investment, your brand, and even the IoT movement itself!

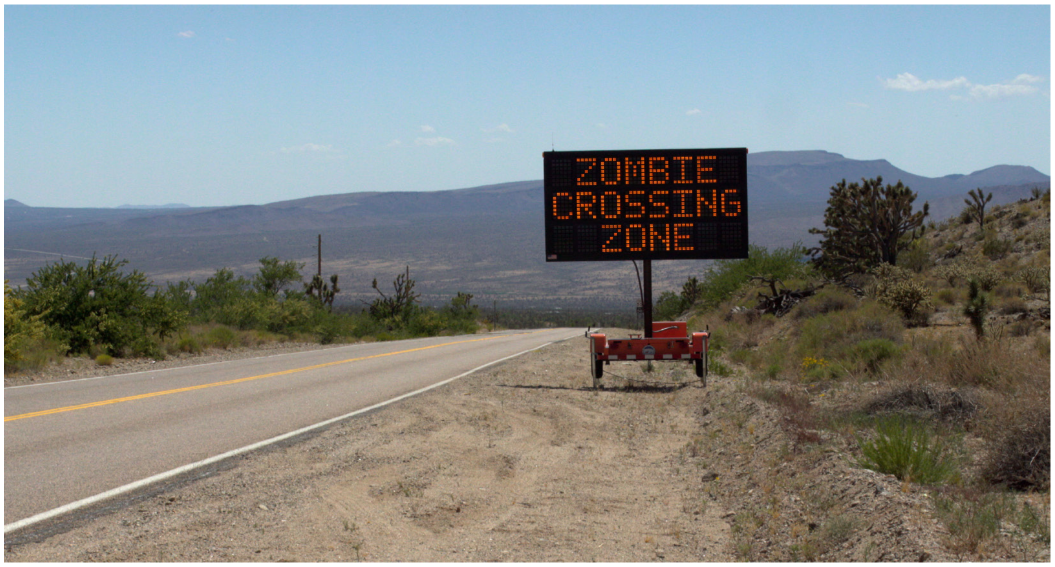


Figure 1. A hacked road sign might be funny, but, depending on what it says, can also be dangerous.

We are surrounded by an invisible intelligence, waiting to be freed. Freeing that intelligence is risky — security flaws can not only expose your IP and your valuable data to attackers for their use, but can turn public sentiment against connected devices. In my previous white paper, I discussed designing smarter sensors into street lights because you may only have one chance to deploy street lights in your city, but we may have limited opportunities to properly secure this army of smart devices before they are released into the wild and become targets for attackers. But the story isn't all fear, uncertainty, and doubt, and we need to stop talking about it as such. In fact, security is an enabler: enabling new business models, enabling innovative applications, and enabling invisibly intelligent products to become trusted parts of a new economy based on data.

What is Invisible Intelligence?

I have previously introduced the concept of 'invisible intelligence,' which in brief rests on three principles:

1. The things we interact with and have relationships with have data, and when enabled by technology they shouldn't fundamentally change the way in which we interact with them. We still set the dial on a smart thermostat, but the thermostat starts to learn our preferences based on a number of conditions. We still glance at our wrist to tell the time, but we are presented with other meaningful data that we can explore further if we wish.
2. We shouldn't notice that the smart device is fundamentally different from its 'dumb' counterpart, thus it is invisible. A window sensor that is so large it blocks part of the light from the outside defeats the purpose of the window.
3. The data from our smart device needs to have value. Invisibly intelligent things will inherently be more expensive than their dumb counterparts. A smart brick might be easy to use and won't change how you interact with or think about bricks, but is it really a good return to buy more expensive bricks that will send you a text message when they fall off your house?

Let's consider the three principles of invisible intelligence and how they pertain to our topic of discussion: how can we use security technology to build trusted and valuable IoT devices?

- Our interaction: while security is critical in the army of connected devices we will have deployed around us, the management of that security should not fall to the consumer. Devices should not need special setup to function properly or be properly secured—this is one of the faults in the Mirai attack where hundreds of thousands of online cameras had default passwords and relied on the consumer to change those passwords...very few people did this, so the cameras were subject to attack. In addition to not needing special setup, it may seem obvious, but devices should not be subject to attacks where their behavior changes such that it changes our interaction with them. Smart light bulbs that have been hacked and now turn on at 2am every day in your bedroom clearly will change your fundamental interaction with a light bulb!

- The device's appearance: integrating security technology into the devices around us won't change their appearance significantly, except importantly in the case of a compromised device (i.e., the hacked highway sign clearly has a different appearance!) Security technology can help insure the device continues to appear as designed.
- The data's value: securing the value of the data and the device is where security technology is critical. We need to be able to trust the data we are making decisions on: are we going to remotely allow the UPS delivery person into the house based on the camera image? What if the image was compromised to show us our familiar delivery person instead of who is really there? In addition to trusting the data, we need to trust that the intellectual property built into the device is secure: can someone steal the software you worked hard to develop? Is your manufacturing chain secure enough to allow you to load your software without it being altered or stolen?

(Embedded) Security Technology Isn't New!

We have all seen lots of talk about security in the IoT, and much of that talk might lead you to the conclusion that securing the devices around us is a new idea. However, there are plenty of examples all around us of security technology being successfully deployed. Unfortunately there are only two reasons why security technology gets deployed: (1) it is the price of entry to participate in the market, or (2) you've had experience losing money/value and have decided to do something about it.

Let's look at an example from the first category, where security is the price of entry to participate in a market. More simply stated, someone said you had to do it. Financial terminals fall into this category. For years, the Payment Card Industry (which represents card brands like Visa and Mastercard in addition to banks and merchants) has demanded that credit card machines and ATM pin pads implement a certain threshold for resisting attacks. These devices must secure the code that runs on them such that an attacker cannot take control of the device, and they also must be physically secure, so an attacker can't take the device apart to try to inspect it further and extract secret password material. This industry requires the devices to have a high degree of security because it realizes financial transactions are based on trust, and breaches of that trust make electronic commerce far less attractive.

We may not think much about the financial terminals we use at the store as they tend to stay invisible, a necessary interaction when we pay for things. However, examples of the second category can strike closer to home for many of us. Consumable products such as printer cartridges and phone replacement batteries can represent a significant revenue stream to the manufacturer. In the case of printers, the printer itself may be sold at cost or a loss based on the idea that printer cartridge revenue can help to make the profit. But ink cartridges can be relatively easy to manufacture, so what prevents competitors from coming in with lower cost printer cartridges without investing in the manufacturing of the printer itself? Security technology can help to make sure the correct, quality cartridges are required to run with the printer. The same technology is used in medical consumables as well to help insure the right products are being used with the right devices—here it is not only a question of revenue stream, but of safety as well.

Sadly, for every example where security technology has been adopted to enable an application, there are countless examples of places where security technology was NOT adopted to catastrophic effect. Default passwords in an army of connected cameras enabled a bot attack that brought down Netflix, Twitter, and a host of other popular websites in the Mirai attack. A lack of validation of firmware in industrial control systems allowed Stuxnet to ruin an Iranian nuclear material processing facility. Wide open systems have been demonstrated to allow an external actor to drive a car off the road from a distance. Remote, unprotected control of insulin pumps has been demonstrated multiple times, showing how an attacker could kill their target with an overdose. And those are just a few of the attacks that have gotten the most attention.

We all agree that 'something needs to be done about security,' but we still see the industry slow to adopt security technology. The reasons why not tend to fall into a couple categories:

- It's too expensive. It is true there can be an added cost to the device itself to integrate security features. Hardware that can efficiently generate and process digital signatures isn't free. However, the per unit cost for security technology isn't unreasonable…the technology is available integrated into microcontrollers or in standalone coprocessor ICs, giving designers flexibility on cost, size, and integration. The costs must be weighed against the risk—what would the cost be on a product recall of 1,000,000 compromised devices?
- It's too complicated. Security is a complex topic—the algorithms are difficult to implement, the math is hard to understand, and there is no perfect answer. Perfect security does not exist, and the closer you get, the closer your costs approach infinity. However, with a reasonable analysis of the risks and threats your application may encounter, you can begin developing a plan for mitigating those risks. Semiconductor products can assist with the math and implementation—you don't need to be a number theory expert to use elliptic curve cryptography to secure your application.
- We'll deal with it later (also…we're not big enough for it to happen to us…yet). This mindset is common, but not surprisingly so. In the rush

Security is inexpensive compared to the costs of a massive recall to get products to market (especially for IoT startups), there is little time to worry about security ,which may not be seen as an enabling feature to your core functionality. In addition, startups tend to think that they'll fix things once there is some volume, and security threats aren't worrisome until they have manufactured a significant number of devices. Note the dilemma—once a significant number of devices are manufactured, the security gaps have already been deployed in the world! What is needed is a way to integrate security from the beginning without getting in the way of rapid application development.

But perhaps there is another reason security as a basic technology isn't being adopted widely...voices in the industry continue to talk about all the bad things that can happen without security and, like invincible teenagers, designers move forward with their design. Perhaps the problem is partially mindset though—we tend to articulate the bad things rather than talk about security as an enabling technology, something that can let your application do things you didn't imagine before. Perhaps rather than just spreading fear, we should spread opportunity.

Security as an Enabling Technology

It may seem odd to think about security as an enabling technology, other than perhaps it being a technology that enables you to combat threats. But there are instances where security technology can help create new opportunities.

A simple example is the use of security to enable lower cost manufacturing. In some applications we see companies using more expensive but local manufacturing because they believe they have more control over their IP and manufacturing runs. Using microcontrollers that contain secure bootload capabilities can enable these companies to move manufacturing to the most cost-effective and efficient locations available. A secure bootloader can be customized for a particular customer, and it would allow the designers to send encrypted firmware to the manufacturer. That encrypted firmware couldn't be copied or reverse engineered. To prevent manufacturing overruns, unique identification numbers in the microcontrollers could be used to ensure only the intended number of devices are manufactured, and no extra ones find their way into the gray market.

Security technology can also enable a more flexible supply chain, particularly where similar product variants are manufactured. When products offer multiple variants with minor hardware differences (such as different memory sizes or measurement accuracy) or multiple software options (that don't require hardware changes), security technology can be used to greatly reduce the number of variants manufactured, or enable late configuration. Instead of manufacturing six different hardware variants (for example, imagine a music player with different memory capacities) and maintaining inventory and build plans for six different devices, it is possible to only manufacture one device with the maximum amount of memory. Then, you can use strong security technology late in the manufacturing flow to lock the device into behaving at a chosen memory configuration (even if it is smaller than the maximum memory available). Strong security technology would make sure customers cannot easily 'upgrade' their device when they buy the less expensive and smaller memory product. The same technique can be used for software as well—if your product supports six different kinds of analysis features which are all implemented in software, you might include libraries for all six features in the end product, but only enable certain ones late in the manufacturing flow. Again, this could help reduce inventory and buffering, but could also help in certain applications that require certification: rather than certify multiple end products that offer feature variants, you may be able to only certify one end product since there is really only one manufactured device.

Let's take this idea a step further: instead of enabling software features late in the manufacturing flow, what if you allowed your customers to upgrade their own devices in the field? For a fee, customers could enable advanced features, and security technology could allow you to make sure it was only enabled once you have been paid, limit the amount of time the more advanced feature is available, or even offer feature sets on a rental or subscription basis. You could sell the lowest cost version available through a retail channel to build an install base, but stand to gain future subscription revenue.

It is true that security is a necessary technology to combat threats that our applications may face, but we should also see it as a liberating technology that frees applications to be manufactured anywhere and run anywhere, maximizing their value without risking their value.

Some Security Basics: Threats and What to Do About Them

The start of any discussion around security should be about threats: what are the threats you are worried about with regard to your assets? Let's take a look at some common threats we think about, and how they relate to two very different applications: a surgical robot and a printer cartridge.

There are plenty more threats we can think about: for example we might worry about IP theft, sending unauthorized commands to our devices, and overproduction in the supply chain. If we are developing devices for critical infrastructure we might also worry about advanced physical attacks that will try to get access to the secret key information: attacks such as microprobing of memories and differential power analysis. We should also worry about social threats—can your people (developers, testers, production line) be compromised in such a way that they can affect how your devices function or to get access to the data they deliver?

Once the threats are identified, we can begin to think about techniques for combatting those threats.

Identify your threats, then fight them

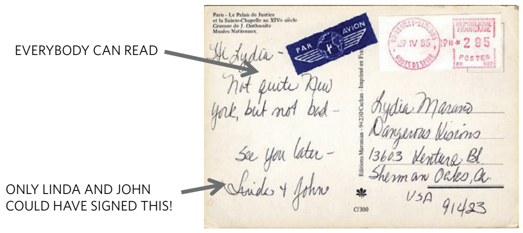
Common Threat	Surgical Robot	Printer Cartridge
Eavesdropping — are you worried someone can listen to your data?	Probably yes. Sensor data gathered by the robot during surgery may be considered private, sensitive health information.	Probably not a concern. Are you worried about the cartridge telling the printer it is 43% full?
Malware — are you worried about unknown software taking over the device?	Yes. We need to trust the surgical robot to do its job properly.	Probably not a concern. It isn't likely the printer cartridge has a micro that can be reprogrammed.
Unauthorized re-use — are you worried about someone using the device more than intended?	Yes. The robot may need cleaning or maintenance after some usage period.	Yes. We don't want someone refilling a used printer cartridge with non-authorized ink.
Counterfeiting — are you worried about someone selling a device that looks like your device?	Probably not. A surgical robot is probably too expensive to manufacture for the risk of counterfeit to be high.	Yes. We need to protect our revenue stream from printer cartridges that go along with our printer.

Table 1: Concern levels of some common threats in two example applications, surgical robots and printer cartridges.

Some Security Basics: Authenticity, Confidentiality, and Integrity

When most people think about security for embedded devices, they think about encryption—hiding data to make it a secret. But this isn't the only technique we need to secure systems. Authenticity, confidentiality, and integrity are some of the key techniques you can use to combat threats. While these all sound similar on the surface, their individual meanings are quite different when applied to security:

Authenticity: Proves the origin of a message. A command is authentic if you can prove that it came from a trusted source. Sensor data is authentic if you know it came from one of the sensors you manufactured. In the real world, we 'prove' authenticity with a signature on a document (Figure 2). Of course, in the real world, your signature can be forged by someone with time and training—in the embedded world your signature can be forged by someone who has stolen your secret keys.



Paris - Le Palais de Justice
et la Sainte-Chapelle au XIVe siècle
Gravure de J. Outhwaite
Musées Nationaux

PAR AVION

92-NEUILLY-SABLONS 29 IV 85 HAUTS DE SEINE 19H ★ 2 95 REPUBLIQUE FRANÇAISE POSTES

Hi Lydia -
Not quite New
York, but not bad -
See you later -
Linda + John

Editions Maronian - 94230 Cachan - Imprimé en France

C/300

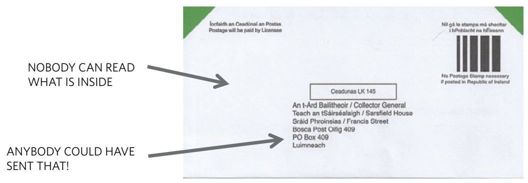
Lydia Marano
Dangerous Visions
13603 Ventura Bl
Sherman Oaks, Ca.
USA 91423

Figure 2. Authenticity proves that a command came from a trusted source.

Note that authenticity does not speak to data hiding…the sensor data does not need to be encrypted or hidden to prove its authenticity! Think about our real-world example being a postcard…anyone can read the message, but the signature gives you some trust about who actually sent the message.

Confidentiality: Protects data from access by those not authorized to see or use it. Our traditional thoughts about secret codes and encryption fall into this category. If we go back to our postcard example, you can achieve confidentiality by putting your message inside a lined envelope—the idea being that no one can see the message while it is in transit (Figure 3) In the real world you unlock the message by opening the envelope; in the digital world you unlock the message by reversing the encryption process with the same password or secret key that the sender used.

Note that confidentiality does not speak to authentication. Anyone could have sent the encrypted message, just like anyone could have sent the letter in our example: there is no signature to prove that the 'From' address is legitimate.



Íocfaidh an Ceadúnaí an Postas
Postage will be paid by Licensee

Níl gá le stampa má sheoltar i bPoblacht na hÉireann
No Postage Stamp necessary if posted in Republic of Ireland

Ceadunas LK 145

An t-Árd Bailitheoir / Collector General
Teach an tSáirséalaigh / Sarsfield House
Sráid Phroinsias / Francis Street
Bosca Post Oifig 409
PO Box 409
Luimneach

Figure 3. Confidentiality allows you to protect data against unauthorized access.

Integrity: Proves that data is well formed and has not been altered since it was sent. In our real-world example, we might think about a seal on the letter showing that the letter hasn't been altered (Figure 4). Or for a more modern example, perhaps think about tamper-evident tape rather than wax.

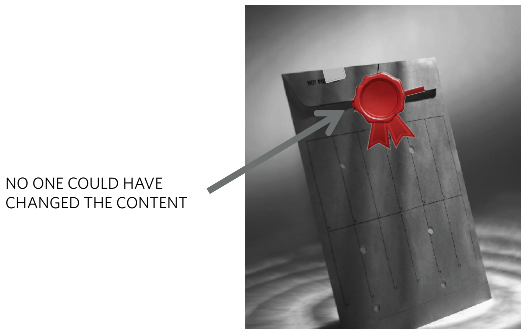


Figure 4. Integrity proves that content has not been modified.

Note that this only tells us that whoever sent the message (even if it was an attacker in the middle) sent us a well-formed message that hasn't been altered since they touched it.

It's clear that each of these techniques is only one tool, and you need some combination to secure systems.

Some Security Basics: Encryption, Hashing, and Signatures

Let's look a little deeper into these security techniques and how you might use real cryptographic tools to secure sensor data. Suppose our sensor is delivering a fair amount of data on the occupants of a building for us to make better decisions about lighting and HVAC controls. To make the challenge a bit more interesting, the building houses some laboratories that are sensitive to temperature variations.

The first thing we might worry about is whether or not the data we receive is valid—if we're going to make sensitive environmental control decisions we should make sure we're not acting on a data error. This calls for Integrity—we could use a cryptographic algorithm called a hash function, such as SHA-256 (the 256-bit Secure Hash Algorithm). Think of a hash as a one-way function—very difficult to reverse, and a small change on the input data results in a drastic and seemingly random-looking change in the output data. A CRC or checksum might be considered a very simplistic hash function. If you run the hash algorithm on your data, you have a strong way of validating the integrity of the data:

[INPUT DATA] → SHA-256 → [HASH]

You send both the input data and the hash to the recipient, who can run the same operation. If the hash they compute is equal to the one you sent, you know it was the intended message.

So now our data has integrity, but we are lacking in security: anyone could have sent a message with fake data, potentially causing us to make bad decisions. To make sure we know where the message came from, we need authentication. One simple way to try to validate that the message came from the right source is to expand our earlier hash into a signature by also including the value of a secret password or key into the computation:

[INPUT DATA] + [SECRET KEY] → SHA-256 → [SIGNATURE]

In this case the sensor sends the INPUT DATA and the SIGNATURE to the master. The master then does the same operation, also including the value of the password or secret key. If the computed signature matches, then the recipient knows that the sender shares the same secret key and presumably is a trusted member of their system. There are flaws with this approach: it relies on a shared secret key which must be distributed to lots of sensors (which increases the risk it could be discovered), and it does nothing to prevent a replay attack, where an attacker simply watches the message go by and replays it over and over again. Adding in sequence numbers or random padding (especially a random challenge from the master) can help with the replay attack, and certificate-based algorithms like DSA and Elliptic Curve DSA can be used to make sure each sensor has its own unique key.

Back to our example: we still have a potential security risk...we are sending the sensor data in the clear. Anyone listening on the network could get that data and understand what is being reported. This may not be a big deal—is it sensitive information to know that 10 people are in a room instead of 5 or none? It could be sensitive, and if you want to hide the data you would encrypt it. You would use an algorithm like AES to encrypt the data before sending it…

[INPUT DATA] → AES(SECRET KEY) → [CIPHER DATA]

[INPUT DATA] + [SECRET KEY] → SHA-256 → [SIGNATURE]

In this case the sensor would send CIPHER DATA and the SIGNATURE to the master for verification. In reality, the SECRET KEY used by the AES algorithm would probably be a temporary key agreed to between the master and the slave and changed occasionally, or data would be transferred over a well-known protocol such as SSL (Secure Socket Layer).

What Kinds of Products are Available?

The key cryptographic tools engineers should look for are standards-based algorithms such as AES, SHA, DSA, RSA, and Elliptic Curve DSA. All aren't necessary to achieve your security goals, but a broad toolbox gives engineers the flexibility to design what is needed. In addition to cryptographic tools, designers should look for a true random number generator as well.

All of these cryptographic algorithms can be implemented in software, and a random number generator could even be implemented in software as well, though you would want to seed it with some kind of real-world entropy sources such as noisy bits of an ADC, lower bits of a system timer or RTC, or timing of some external IO events (or all of the above). However, hardware implementations have some significant benefits:

- Cryptography implemented in hardware will take less code and data space, freeing that space up to be used by the application
- Hardware implementations will execute much faster than software implementations, usually by an order of magnitude
- Hardware implementations will consume less power than software implementations. The instantaneous power may be a bit higher than in the software implementation, but since the hardware completes much faster, the overall consumption will still be significantly lower
- Hardware implementations will generally do a better job resisting timing and power-analysis attacks
- A hardware random number generator can be designed to rely on true sources of entropy such as silicon-level variances in timing, voltage and temperature, rather than external sources of seeding which could be guessed more easily by an attacker

There are a wide variety of products available that integrate different sets of security features:

- Standalone authentication products like the DS28E38 with ChipDNA™ physically unclonable function (PUF) technology are typically designed to perform authentication and possibly integrity checks for sensors, consumable, and other applications.

Hardware security is stronger and lower power than software versions

They can be low cost, but have limited flexibility for functions outside authentication.

- Standalone coprocessors like Maxim's MAXQ1061 provide a tool for easily adding a complete cryptographic toolbox to existing designs, and even allowing them to easily make use of protocols like TLS, the same protocol used to secure online commerce
- Many microcontrollers integrate symmetric cryptographic engines like AES-128, but a few implement the more complex mathematical routines like modular exponentiation that are needed to run algorithms like RSA, DSA, and Elliptic Curve DSA efficiently. Maxim's MAX32631, for example, includes a modular arithmetic accelerator that helps implement those algorithms.
- Advanced high-security microcontrollers typically integrate a complete set of cryptographic algorithms, plus integrate more advanced countermeasures to resist physical attacks like clock, voltage and temperature variations. They can also zero-ize sensitive information when they detect that they've come under attack. Products like the MAX32552 implement these kinds of features to meet the stringent demands of financial terminals and critical infrastructure.

Designing an IoT that Will Stick Around

The opportunity that we have to free the invisible intelligence around us is as broad as are the ways we have to free that intelligence. Connected devices that report their data can create new businesses and innovations. But there is one common denominator to freeing this invisible intelligence: will we as a society trust these devices enough to buy them, install them, and rely on them? It has been long thought that embedded security technology doesn't exist to secure these kinds of devices, but that is not the case: the technology is available and cost effective and can be easily integrated into any application. In order to free our invisible intelligence, we will need to trust it.

More Resources

Learn more about a new breed of low-power microcontrollers built for the IoT:
Meet DARWIN

Also, read our related white papers:
Powering the Next Generation of Smart Devices

Adding Intelligence to the Next Generation of Smart Devices

For more information, visit:
www.maximintegrated.com