How Much Security is Good Enough?

Introduction: StarDust™ Power

Certainly 2 thousand, one hundred, trillion, trillion, trillion, trillion, trillion years is a bit of an overkill to protect even your favorite IoT device, but wouldn't you feel safer knowing that the odds were against your privacy being invaded through the countless cameras or microphones in the home, or the automated systems upon which you rely so much such as temperature control or electronic locks which have been taken over by some impish pre-teen? How about something a bit more serious such as granddad's pacemaker? From slight annoyance, to safety, to privacy invasion – all need to be dealt with if the IoT era is to take off.

What comprises "Good" IoT Security?

The failure of the IoT market to resolve even this rudimentary security question is due to the conflicting requirements for a reliable IoT encryption solution. Cost, size, features, all go into a real world functional solution. This can be expressed in "You can have, small, cheap, or good – pick any 2". This is in effect the IoT encryption conundrum today. So what are the 3 choices that our IoT marketing person has to select from?

	Description
Low Complexity Encryption	Must be easily implemented, providing excellent security.
Low Design Impact	Solution should not change the original IoT device design and system implementation.
Low Battery Drain	Cannot increase battery drain.

Low Complexity Encryption

Encryption of IoT transmissions is a complicated subject, but encryption solutions generally rely upon three different methods by which devices can share encryption keys, since it is impossible for communicating devices to read a message unless they both possess the same decrypting key to open a message. The three most popular methods are generally classified as: Public Key Infrastructure (e.g. PKI), Symmetric Key Cryptography and Public-Key Cryptosystems (e.g. RSA).

Encryption systems used by businesses fall into two broad categories:

> **Private key** (or secret key) systems use the same key to encrypt and decrypt data. Because of this, you need to keep your key hidden so that no one else can access it.
>
> **Public key** systems use two keys: a private key, which you keep hidden, decrypts the data, and a public key used to encrypt the data. Because the public key only encrypts your information, you can safely share it with anyone, in fact you have to otherwise this scheme will not work. Public key encryption works well in situations where you can't securely share a key, like over the Internet, but it has some real disadvantages.

- **Public-Key-Infrastructure**

Overview

PKI, or public key infrastructure is a framework for services that provide for the generation, distribution, control and accounting of public key certificates. This public key system ensures secure user authentication, network traffic encryption, data integrity and non-repudiation. PGP meanwhile is an application actually derived from the IETF open standard OpenPGP. Like PKI systems, OpenPGP uses both public-key cryptography and symmetric key cryptography, but the program differs in how it vets and binds public keys to user identities. Unlike PKI arrangements, OpenPGP is based on a web of trust rather than certificate authorities (CA). OpenPGP allows users to choose who they trust, whereas users in a PKI system defer to a trusted CA. Commercial CAs, however, need to ensure that their own certificate is incorporated into the major browsers and messaging applications in order to provide this chain of trust. Finally the definition of logistics is the activity of supplying or providing something, and in the case of OpenPGP and PKI, this would be considered the efficient management, distribution and validation of a public key contained within a user's certificate.

So what are the weaknesses of these two systems in terms of managing, distributing and validating digital certificates? Well, while PKI can identify Web servers and allow transactions over SSL, it lacks ease of scalability for an organization because the cost and registration process involved with "supplying and providing" client-side certificates is onerous. Additionally, the management and revocation of certificates requires a highly complicated structure, not to mention scalability brings additional costs of computer resources and help desk support. On the other hand, PGP has flourished for many years without the need to establish a centralized

CA because OpenPGP makes use of the concept of trusted introducers, allowing anyone to sign anyone else's public key. This decentralized approach removes the cost of CAs from the delivery process, but still requires key servers to act as public repositories so that everyone can reference users' public keys.

Most modern applications easily cope with X.509 digital certificates used by PKI systems, even when it comes to the less experienced user. There are plug-ins implementing PGP functionality for the more popular email applications, such as Microsoft Outlook, but a plug-in is always susceptible to implementation errors or even a backdoor path to the data directly – who do you trust?

Although neither PKI nor OpenPGP are perfect, (neither arrangement has economically solved the problem of user certificate mobility and security, for example), the programs provide defense for original Internet protocols that don't have built-in security.

Speed

Public key encryption works very well and is extremely secure, but it's based on complicated mathematics. Because of this, your computer has to work very hard to both encrypt and decrypt data using the system. In applications where you need to work with large quantities of encrypted data on a regular basis, the computational overhead means that public key systems can be very slow.

Certification Problems

Many public key systems use a third party to certify the reliability of public keys. For instance, if you were to encrypt sensitive corporate data to send to your attorney's computer, you'd want to be sure that the computer you were sending it to was really tied to the law firm. The third party, called a certification authority, digitally signs their public key, turning it into a digital certificate, so that you can be sure it's safe to use. However, if the certification authority gets compromised, the criminal that did it could issue false certificates and fool people into sending data to the wrong place. This has already happened.

Direct Compromise

There are two ways to crack data encrypted with a public key system. The first is to find a hole in the underlying mathematics that can be used to break the cipher. As of the date of publication, no such hole is publicly known but is most likely known to the NSA (or other Secret Services). The other way to crack the encryption is to guess the correct key. Since public key encryption works on the basis of having an extremely large number that is derived from multiplying a large number hidden in the public key with a large number hidden in the private key, if you could factor that extremely large number, you could break the encryption. As computers become more powerful and as quantum computing, which uses light to create even faster speeds than traditional supercomputers, becomes a reality, brute force attacks on public key encrypted data become practical. Quantum computing, if it reaches the promised hype, will obsolete all the current techniques.

To read further:

http://smallbusiness.chron.com/disadvantages-public-key-encryption-68149.html

https://searchsecurity.techtarget.com/answer/The-strengths-and-weaknesses-of-PKI-and-PGP-systems

- **Symmetric Key Cryptography**

Overview

Secret-key cryptography, also known as symmetric-key cryptography, employs identical private keys for users, while they also hold unique public keys. “Symmetric- key” refers to the identical private keys shared by users. Users employ public keys for the encryption of data, while the private keys serve a necessary purpose in the decryption of data. People wishing to engage in a secure exchange of information will swap public keys and use some method to ensure the existence of identical private keys. In theory, private keys would be brought into the transaction through either the duplication of an existing key or the creation of two identical keys. In modern practice, users utilize key generators to create both keys, but the private keys must still be distributed in a confidential mode.

Strengths

The private keys used in symmetric-key cryptography are robustly resistant to brute force attacks. While only the one-time pad, which combines plaintext with a random key, holds secure in the face of any attacker regardless of time and computing power, symmetric-key algorithms are generally more difficult to crack than their public- key counterparts. Additionally, secret-key algorithms require less computing power to be created than equivalent private keys in public-key cryptography.

Weaknesses

The biggest obstacle in successfully deploying a symmetric-key algorithm is the necessity for a proper exchange of private keys. This transaction must be completed in a secure manner. In the past, this would often have to be done through some type of face-to-face meeting, which proves quite impractical in many circumstances when taking distance and time into account. If one assumes that security is a risk to begin with due to the desire for a secret exchange of data in the first place, the exchange of keys becomes further complicated.

Another problem concerns the compromise of a private key. In symmetric-key cryptography, every participant has an identical private key. As the number of participants in a transaction increases, both the risk of compromise and the consequences of such a compromise increase dramatically. Each additional user adds another potential point of weakness that an attacker could exploit. If such an attacker succeeds in gaining control of just one of the private keys in this clandestine world, every user, whether there are hundreds of users or only a few, is completely compromised.

To read further:

http://www.csc.villanova.edu/~mdamian/Past/csc3990fa08/csrs2007/01-pp1-7-MattBlumenthal.pdf

- **Public-Key CryptoSystems (e.g. RSA)**

Overview

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the so-called "factoring problem". The acronym RSA is formed from the initial letters of the surnames of Ron Rivest, Adi Sahamir, and Leonard Adleman who first publicly described the algorithm in 1978. Clifford Cocks, an English mathematician working for the British intelligence agency Government Communications Headquarters (GCHQ), had developed an equivalent system in 1973, but this was not declassified until 1997.

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message in practice.

RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography, which in turn can perform bulk encryption-decryption operations at much higher speed.

Currently, RSA is used in security protocols such as:

- TLS/SSL - transport data security (web)
- PGP - email security
- IPSEC/IKE - IP data security
- SILC - conferencing service security
- SSH - terminal connection security

The public-key method consists of separate encryption and decryption keys, with users only being able to decrypt an encrypted message if they have the appropriate decryption key. Users will exchange public keys employing a process such as Diffie-Hellman; this transaction does not need to be done in a secure manner because the release of public keys does not threaten the security of any private information. After this swap, someone who wishes to send private information to another user will encrypt the data with the intended recipient’s public key and then pass along the encrypted message. The recipient, who always keeps their private key secure, can use the private key to decrypt the encoded message.

Strengths

The asymmetric nature of public-key cryptography confers a sizable advantage over symmetric-key algorithms. The unique private and public keys provided to each user allow them to conduct secure exchanges of information without first needing to devise some way to secretly swap keys. This glaring weakness of secret-key cryptography becomes a crucial strength of public-key encryption.

Weaknesses

Keys in public-key cryptography, due to their unique nature, are more computationally costly than their counterparts in secret-key cryptography. Asymmetric keys must be many times longer than keys in secret-cryptography in order to claim equivalent security. Keys in asymmetric cryptography are also more vulnerable to brute force attacks than in secret-key cryptography. There exist algorithms for public-key cryptography that allow attackers to crack private keys faster than a brute force method would require. The widely used and pioneering RSA algorithm has such an algorithm that leaves it susceptible to attacks in less than brute force time. While generating longer keys in other algorithms will usually prevent a brute force attack from succeeding in any meaningful length of time, implementing these more complex algorithms for an IoT device becomes more computationally intensive. The longer keys can still vary in effectiveness depending on the computing power available to an attacker. Therefore, the increasingly frequent appearance of sophisticated new algorithms which can factor big numbers as if mere child’s-play (need we remind the reader of China, Russia, Israel and even the NSA?), along with the brute strength of supercomputers (US NSA, China 3PLA, Japan PSIA) renders public-key cryptography a juicy target for government funded hacking or just plain fun.

Finally, Public-key cryptography also has vulnerabilities to attacks such as the man in the middle attack. In this situation, a malicious third party (perhaps your precocious teen) can lay in wait to intercept a public key on its way to your bank account. Your teen then passes along his or her own public key with a message claiming to be from you. Your bored teen can use this process to successfully impersonate each member of the conversation without any other parties having knowledge of this deception. Now onto mom’s checking account. That’s much more fun than homework.

To read further:

https://www.ijcsi.org/papers/IJCSI-9-1-3-175-178.pdf https://en.wikipedia.org/wiki/RSA_(cryptosystem))

http://www.csc.villanova.edu/~mdamian/Past/csc3990fa08/csrs2007/01-pp1-7-MattBlumenthal.pdf

• Side Channel Attacks

One relatively new area of hacking concern is the hotly debated and discussed "side-channel" attack, which can hack all the above encryption techniques. In computer security terms, a side-channel attack is any attack based on information gained from the physical implementation of an encryption system, rather than weaknesses in the implemented algorithm itself, (e.g. crypto-analysis and software bugs). For example, information that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process.

Side channel analysis techniques are of concern because the attacks can be mounted quickly and can sometimes be implemented relatively cheaply using off the shelf hardware. The amount of time required for the attack and analysis depends on the type of attack e.g. Differential Power Analysis, Simple Power Analysis, Timing, etc. Simple Power Channel attacks on smartcards typically take merely a few seconds per card – and since some IoT devices may be of similar complexity to a smartcard, it is easy to see the reason for concern.

A simple TIMING ATTACK is based on measuring the time it takes for a unit to perform operations. This information can lead to information about the secret keys. For example: By carefully measuring the amount of time required to perform private key operations, an attacker might find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. If a unit is vulnerable, the attack is computationally simple and often requires only known ciphertext. PKI, Symmetric Keys and Public Cryptographic systems are all designed to exchange multiple messages in order to authenticate and secure communications, hence are by definition vulnerable to characterization by side-channel attacks. Once a particular design is "side-channel-classified", then the entire system has been broken.

By comparison StarDust™ method is deterministic so it avoids analysis of completion time and power signatures as used by side-channel hackers.

To read further:

https://en.wikipedia.org/wiki/Side-channel_attack

http://gauss.ececs.uc.edu/Courses/C653/lectures/SideC/intro.pdf

https://conference.hitb.org/hitbsecconf2016ams/materials/D2T1%20-%20Anders%20Fogh%20-%20Cache%20Side%20Channel%20Attacks.pdf

Low Design Impact

As can be seen from the above high-level review, encryption is an exceptionally complex feature to implement in low cost IoT devices, let alone understand. Where to start? IoT device designers are left struggling to sort out the best way to add encryption to their devices while minimizing impact. IoT devices by definition are small, inexpensive, and perform scant little data processing - how much computing power and number crunching is needed to send a sensor chip's numeric output of temperature, heart rate, or moisture? Answer – none, so likely the IoT device will not have much reserve horsepower (headroom) to perform any additional chores; it will have just enough to deal with its single dedicated task.

Quite likely the IoT designer is neither an encryption expert, nor would the designer be very pleased to add 100's of kilobytes of new software; even a few bytes of extra data could be a major undertaking. The designer would quickly find that the increased burden placed on their CPU to fulfill the intensive data manipulation and messaging required to support a secure transmission would grow their cute little sensor into something resembling a beached whale.

IoT devices thus are highly constrained and efficient designs, and have very little room for additional processing burdens. Short of a wholesale redesign to accommodate security, it is impractical to simply upgrade the IoT unit with a more powerful CPU, add memory, or possibly switch to a different wireless mode of communication; these would all increase costs and destabilize the cost model for IoT deployment. Congratulations, you just quadrupled the size of your IoT design team and likely increased the IoT BOM cost 10x. Adding in a 32-bit ARM processor to deal with your device's encryption is overkill of gargantuan proportions, especially when an 8-bit processor would be overkill on a galactic scale in the first place.

Neither can the solution force the receiving site of these now encrypted IoT messages completely re-engineer the network and communications front-end. The final format of the message, must remain the same, nor should the network server side be forced to change to manage the encryption. This sort of drastic network redesign would result in a significant and unpleasant increase in company expense if it was necessary to purchase and implement a new infrastructure to support a Public Key Infrastructure (PKI) with X.509 certificate servers, not to mention all the software to manage and verify certificate management on IoT end-devices. On top of which there will be billions of IoT devices, so how will this entire new infrastructure really scale?

Low Battery Drain

It is impossible for any battery-powered IoT device to increase its battery drain exponentially in order to tack on an endless array of new features. Batteries are after all a finite resource and device designers purposely manage the battery power chain with this in mind – in fact the frequency of device transmissions is directly related to battery longevity. Now considering the recent IoT hair pulling over the unpredicted need for data encryption and decryption – what is the impact on battery budget for IoT security?

The simple expedient of merely sending extra messages or even extending the message size to provide security is not an option. Sensors or monitoring devices powered by battery are expected to last for at least 10 years or more. Just replacing a battery in a remote sensor could cost upwards of $10,000 (site visit) for a $2 sensor, which makes little sense since IoT sensors are designed to be in effect disposable. Even identifying/locating a deployed sensor can be a problem of nightmarish proportions; ask any utility company. Adding in one extra message per transaction might halve the battery life so you now have a 5-year sensor. Adding on extra encryption bits would have a similar ratio effect on the battery life. Thus any security that is added must piggyback onto messages that are already being sent, perhaps stealing some extra bits, nevertheless chewing up valuable battery, although minimized as much as possible.

Ok, so what is the impact of encryption processing on a CPU – generally? It is an undeniable fact that encryption/decryption burdens any CPU, so imagine the impact of CPU punishing encryption/decryption processes on a sensor that was never designed to manipulate complex mathematics. Even the mighty Cisco struggles with the impact of encryption/decryption on its equipment – never mind a lowly sensor.

To read further:

https://supportforums.cisco.com/t5/wan-routing-and-switching/high-cpu-usage-due-encrypt-proc/td-p/2343246

https://www.semanticscholar.org/paper/Power-Supply-Issues-in-Battery-Reliant-Wireless-%3A-A-Guo-Healy/5ab8897df511f1487445bfcf18e2fdad3898b283

So there we have it. Battery management is unpredictable and highly suspect to failure regardless of its published specifications. Batteries are an unreliable power source that IoT manufacturers in their desire to meet market demand for secure/encrypted devices, are attempting to task with power draining CPU cycles.

It is thus quite impossible to simply add complicated and mathematically intensive software code to an already battery challenged IoT sensor.

StarDust™ – you can indeed have it all!

Low Complexity Encryption + Low Design Impact + Low Battery Drain, the stuff of IoT marketing dreams. It is indeed possible to have all 3 engineering features – as long as you select Stardust™. However, before stepping through each of the design choices, it would be helpful to provide some of the StarDust™ design philosophy:

StarDust™ Design Philosophy

We started with the old axiom “Keep It Simple & Secure” (KISS).

The innovation was to allow designers of IoT devices free range to pick and choose their preferred embedded solution and then add the required security into their design with very little impact, secure in the knowledge that it then confers upon their design the required security level.

The IoT device end of the solution should be kept very simple, leaving the complex and heavy processing to the network side of the solution. The impact should be as low as simply adding in a piece of C-source code and compiling the design. The required code should only add very few extra bytes to the IoT end device. Naturally space must be reserved for keys and it is recommended that at least 128bit keys be used – anything less would be susceptible to brute force hacking attacks as seen from the overview earlier. Space for the keys can be reserved in the design and could be Flash memory rather than valuable RAM. Then a simple call by the IoT software would encrypt or decrypt the data. Both the sent and received packets would thus be encrypted and decrypted when transmitted on either a wireless link or hardwired link.

The encryption must also be radio agnostic so long as it can send the required data: BT LE, ZigBee, Z-Wave, WiFi, LoRa, Cellular IoT, etc. The point being that the IoT device designer should not need to be a radio expert in order to provide device security. The method must not impact the preferred communication technology; it must work regardless of the means of transmission.

The receiving end of the encrypted message should be in the fixed network and with the addition of some small additions to the network, the secured IoT data can be decrypted for the IoT application, which also serves to simultaneously validate and authenticate the IoT device. That's it - you are all done. No need for heavy, complicated math intensive processors, or hard to manage certificate servers.

StarDust™ Low Complexity Encryption

The encryption method used in the current implementation is SPECK for the processor or SIMON if a hardware-based design. These were chosen because they are very simple to implement and they can exist in software or hardware. Other low complexity encryption solutions exist (or you can roll your own) and they can be substituted as the requirements demand. Any alternative of course needs to support a minimum of 128-bit keys.

StarDust™ encryption/decryption complexity is almost entirely eliminated when compared to an encryption method such as AES-128, which requires a large pad of 1,280 prime values, with repeated multiplication/division of these prime values. Talk about CPU loading!.

Solutions such as today’s AES tend to fall short of what is required for today’s most constrained environments, and surely won’t meet tomorrow’s needs. For example, the consensus has long been that a budget of 2400 gates is the smallest chip area that might reasonably be allocated for security on constrained IoT sensors, and this is well out of reach for AES implementations. How would this impact the software or CPU needs of an IoT device? On microcontrollers, AES implementations can be very fast but they also tend to be large and complex. Implementations that decrease size or complexity certainly exist, but small implementations tend to be complex (and slow), while simple implementations tend to be large (and slow).

Instead the use of SIMON/SPECK requires very little software code – in fact StarDust™ encryption can be done in few hundreds bytes! Very simply, StarDust™ can accomplish security through much simpler math and limited to the following list:

- bitwise XOR,
- modular addition and subtraction, + and −
- bitwise AND, &
- left circular shift, S^j, by j bits
- right circular shift, S^{-j}, by j bits

To read further:

https://eprint.iacr.org/2015/585.pdf

StarDust™ Low Design Impact

Remember we mentioned above that StarDust™ can be implemented in a few hundreds bytes? StarDust™ thus has very low overhead on any design and is suitable for both software and hardware implementations. The data sent is simply encrypted without the addition of any extra bytes of data, although some bits in the data stream need to be reserved for the authentication and validation part of the security; however this could be as simple as a common CRC, which is likely already part of the data stream.

So sadly (or happily for StarDust™), there is not much more to say about StarDust™ impact on IoT devices. StarDust™ is small, light and fast. Verdict: virtually no impact.

StarDust™ Low Battery Drain

As can be seen the energy requirements needed to implement the solution are trivial. The biggest battery drain is always caused by wireless communications, which in this case is not impacted at all by StarDust™. In fact the IoT device is not sending any more data than is absolutely required, and in most cases it is sending no more data than it normally sends in its current implementation.

StarDust™ wins as the only wireless friendly technology that does not require multiple radio messages just to acquire and set the encryption keys or gain a certificate for authentication and data protection. Any of the aforementioned security schemes such as TLS/SSL would require at least 29 messages between the IoT end-device and the central server just to confirm each other's identity – this is before the tiny little sensor can even consider sending its data. What about NB-IoT now being advertised by cellular companies in Europe and US? How does a minimum of 44 authentication messages sound? Any existing solution or NB-IoT would kill any battery in a matter of days if the sensor needs to report 2-3x daily. Batteries simply cannot sustain this kind of abuse. For example, if a sensor needs to report 3x daily, current security technologies would require 87-132 battery crippling messages daily prior to sending the data and would go flat in no time at all, versus StarDust™'s mere 3 messages which includes the data. It is clear to any IoT designer attempting to include security that using existing encryption technologies is the proverbial "banging a square peg into a round hole".

Compared to StarDust™'s 1x message to both authenticate as well as deliver data – StarDust™ wins again.

StarDust™ Infinity Option

Infinity is forever, and if this option is turned on, your device would be safe from hacking forever. The IoT implementer may option StarDust™ to replace keys with every transmission. In effect never using the same key twice, thus forcing a hacker to restart their hacking process with each try as they never gain the opportunity to "eliminate" future guesses. A StarDust™ key "once used, is forever gone". This adds yet another level of security to thwart hacking attacks that rely on sniffing used keys, or "replaying" messages through man in the middle attacks.

SUMMARY: How safe is safe?

So how safe is your device - really? If any of the conventional security solutions PKI, Symmetric Keys, or Public-Key Cryptography were implemented, the increase in size and cost of your IoT sensor does not relate to how safe your device will be. The question really is, how good is the encryption technology on your device?

At the heart of the problem is that all of the above security solutions rely on either factoring very large prime numbers, or simply brute force guessing very large sized numbers. Ultimately, even the process of factoring very large prime numbers can be ignored if the hacker has enough resources to simply brute force guess as quickly as possible. Then once it has guessed the shared secret key, or public key, then it will have successfully hacked your device. Ignore for the moment that all of the above methods (except StarDust™) have already been hacked – so in a sense your device is already quite hackable since hackers already know how to repeatedly attack your device. Assuming your device is a low complexity endpoint device, do you really want it to store security certificates just to do simple tasks?

The Chinese supercomputer TaihuLight, one of the most powerful in the world, can hurtle along at 94 petaflops or 1015 operations/second trying to hack your device by brute force guessing every single possible key. The table below summarizes the results of this hypothetical hack. To make it fair, we have “frozen” StarDust™ keys so that they do not change – but in fact StarDust™ keys change with every transmission if the “Infinity Option” is turned on, so in the real world TaihuLight would not in fact gain any advantage building a database of StarDust™ keys; TaihuLight would need to guess to infinity (hence the term Infinity Option):

	DEVICE KEY SIZES	TOTAL NUMBER OF OPTIONS	TOTAL SECONDS TO BRUTE FORCE		
1	128bit key (STD)	3.40E+38	1.16E+23	3.67E+15	Years
2	70bit (Optimized)	1.18E+21	4.02E+05	111.64	Mins
3	StarDust™	1.94E+84	6.61E+68	2.10E+61	Years
4	256bit key (STD)	1.16E+77	3.94E+61	1.25E+54	Years

5	128bit (Optimized)	**3.40E+38**	**1.16E+23**	**3.67E+15**	**Years**
6	StarDust™	**1.34E+154**	**4.56E+138**	**1.45E+131**	**Years**

The table provides an analysis of "brute-force" hacking by programming TaihuLight to guess keys as quickly as possible:

- Rows 1 & 4 analyze a brute force attack on standard 128 and 256 bit keys, with the simple assumption that TaihuLight needs to simply guess every possible solution.
- Rows 2 & 5 analyze the same 128 and 256 bit keys, however this time assuming that mathematically it has been proven that TaihuLight does not need to brute force guess every single key. It has been proven that applying Grover's algorithm to break a symmetric (secret key) algorithm by brute force requires time equal to roughly 2n/2 invocations of the underlying cryptographic algorithm, compared with roughly 2n in the classical case, meaning that symmetric key lengths are effectively halved: AES-256 would have the same security against an attack using Grover's algorithm that AES-128 has against classical brute-force search (see key size).

To read further:

https://en.wikipedia.org/wiki/Quantum_computing

- Finally Rows 3 & 6 reveal the StarDust™ analysis, clear and away the winner since it looks like it would take the world's faster supercomputer TaihuLight 2,100,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 years to brute force guess your StarDust™ device keys (2 thousands, one hundred, trillion, trillion, trillion, trillion, trillion years). Super Safe!

How does StarDust™ fare against today's standard 128-bit encryption technique? It looks like AES-128 would be hacked in 111 minutes – so sorry but your device could get hacked roughly every 2 hours. StarDust™ wins even if we push the key length up to 256 bits, since the challenge for a poor underpowered IoT sensor to store a table of primes sufficient to generate a 256bit key rather defeats the purpose of a low-cost, low-powered sensor.

Our best recommendation: cover your design with StarDust™ …

Before it's too late!

Follow us at www.iotaBEAM.com or look for iotaBEAM on your social media (LinkedIn, Twitter, Instagram or Facebook) to learn more about how we have been able to resolve these issues with StarDust™, the enduring security solution for low complexity IoT sensors and devices.