Blurry Box Encryption Scheme and Why It Matters to Industrial IoT

Authors:

Dipl. Ing. Oliver Winzenried
CEO and founder
Wibu-Systems
oliver.winzenried@wibu.com

Prof. Dr. Jörn Müller-Quade
Karlsruhe Institute of Technology (KIT) and IT Research Center (FZI)
muellerq@ira.uka.de

Patrik Scheidecker
IT Research Center (FZI)

Brandon Broadnax
Karlsruhe Institute of Technology (KIT)

Bernhard Loewe
Karlsruhe Institute of Technology (KIT)

Dr. Matthias Huber
IT Research Center (FZI)

1. INTRODUCTION

Ever since software was first introduced, developers and crackers, product inventors and pirates, solid infrastructure builders and hackers have been engaged in an arms race. Regardless of where one stands in the debate between homegrown solutions and specialized technologies, conventional software protection often relies on the principle of "security through obscurity". According to this principle, the security of a system is fundamentally tied to the secrecy of the protection mechanisms that are shielding it from attacks.

A related principle, first postulated in the nineteenth century, in which the security of a cryptosystem relies on the secrecy of just the cryptographic keys, could be the game changer of the digital age. With the rise of software as the primary resource for the evolution of technology, cyberattacks to its availability, confidentiality, and integrity are reaching new levels of sophistication. Manufacturing plants, devices, and end products require new protections. While the Industrial Internet of Things (IIoT) offers new avenues for monetizing software, unprotected software puts the underlying system at risk, whether it is an industrial computer, an embedded system, a mobile device, a Programmable Logic Controller (PLC), or a microcontroller.

2. METHODOLOGY OF MODERN CRYPTOGRAPHY

From ancient history until a few decades ago, cryptography meant the art of using codes and ciphers to keep the contents of messages private. Since then, the mindset of cryptographers has changed dramatically and cryptography has evolved into a science. One prerequisite for this change is Kerckhoffs' Principle, which made it possible to discuss cryptographic methods openly.

2.1 KERCHKHOFFS' PRINCIPLE

Auguste Kerckhoffs, best-known for the principle named after him

In 1883, Auguste Kerckhoffs published two articles in the *Journal des sciences militaires*[1], in which he surveyed the military ciphers of the time and proposed six principles for the design of new ciphers. Some of those principles were dependent on the technology available at the time, but one principle, known today as Kerckhoffs' Principle, is still valid today and fundamentally shaped the mindset of modern cryptography (translated from French):

The system should not require secrecy, and it must not be a problem if it falls into enemy hands

More than a century later, Claude Shannon put it more pithily in *Communication Theory of Secrecy Systems*[2]*:* "the enemy knows the system being used". In essence, the security of encryption schemes depends only on the cryptographic key, and the enemy knows everything else. This is a very pessimistic view, but one which enables us to conduct public research on cryptography. Progress stalls when Kerckhoff's Principle is disregarded.

Usually, the term *security through obscurity* is used to describe systems where security relies on keeping the system secret. This practice is discouraged by security experts, not only for encryption schemes, because undisclosed security mechanisms are typically not well analyzed.

While it is strongly discouraged to build a system that relies only on security through obscurity, there is value in keeping secret the protection mechanisms that protect cryptographic hardware against tampering. Taking a closer look at Kerckhoffs' original statements, he did not demand the publication of the algorithm itself. He stated that security should not break down completely if the enemy knows the system. However, without knowing the system, it is impossible to check whether Kerckhoffs' Principle is satisfied. Therefore, it is common practice in cryptography to publish new schemes, so that anyone can verify their claims.

Kerckhoffs' Principle provides a revolutionary approach to cryptography, yet little work has been done to demonstrate its efficacy or practical business applications. Proof of its correctness and viability would pave the way for a totally new conversation and public evaluation within the security expert and hacker communities. If successful, it would raise the bar for security

[1] A. Kerckhoffs, "La cryptographie militaire," Journal des sciences militaires, vol. IX, pp. 5–83, Jan. 1883, pp. 161–191, Feb. 1883 (http://petitcolas.net/fabien/kerckhoffs/).

[2] C. Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal 28: 662, 4. October 1949.

standards while safeguarding digital intellectual property against counterfeiting and reverse engineering.

The impact on smart factories would be particularly dramatic. Sensitive data can include production data in many forms and sizes, such as 3D blueprints or punch schemes for embroidery machines, or the technology data or configurations used in manufacturing processes. This invaluable data needs to be protected against know-how theft, counterfeiting, and tampering, otherwise software-as-a-service will easily degrade into piracy-as-a-service. Applying Kerckhoffs' Principle would provide encryption methods associated with hardware anchors of trust and ensure IP confidentiality and the integrity and authenticity of digital signatures.

2.2 MODERN CRYPTOGRAPHY

Today, cryptography is based on a systematic approach of defining and proving security properties of systems. Two steps are needed to define a system's security:

1. The security property has to be described precisely, and
2. There must be as few restrictions placed on the attacker as possible.

For example, it is not safe to assume that the attacker only has access to encrypted ciphertext. Instead, he might have pairs of ciphertext and plaintext. This is called a known-plaintext attack. Even less restrictive is the assumption that the attacker can choose either ciphertexts or plaintexts and learn the corresponding counterpart by, for example, interacting with its sender or receiver. A well-known example is the Enigma machine, which violated Kerckhoffs' Principle. The internal wiring of the rotors was the secret. Only few rotors were used, and breaking Enigma became feasible after capturing one machine. Furthermore, the cryptanalysts at Bletchley Park broke Enigma with a known-plaintext attack because they were able to guess message parts correctly. This was not disclosed to the public until 1974. Throughout the war, the Germans were confident that Enigma had not been broken.

Today's cryptography is not only about keeping messages private. The field has broadened its purview. A major step in this direction was the introduction of public key cryptography. This is a different kind of message encryption, where different keys are used for encryption and decryption. Several other primitive approaches such as digital signatures, key exchanges, and commitment schemes have been designed and are used in more complex systems, such as voting schemes, online banking, crypto currencies or general secure multi-party computation.

2.3 PROVABLE SECURITY

For many schemes using algebraic structures, such as public key encryption and digital signatures, security is not mere conjecture, but proven mathematically. This is not absolute proof of security, but it is based on a problem that is considered hard to solve. Finding the prime factors of a large number is one such problem, used for example in the Rabin and the well-known, closely related

RSA cryptosystem. These problems usually are stated with a parameter for their scale, used mostly to indicate the key length.

The term "hard to solve" has a specific meaning in cryptography: The problem is not only difficult, but its difficulty grows exponentially as the scale increases. For state-of-the-art cryptosystems, for example, RSA encryption with 2048 bit keys[3], these scales are high enough to make finding solutions practically impossible, even given all the computation power in the world.

Additionally, cryptography has evolved from statements like "this looks random" into a science with exact definitions and rigorous proofs based on mathematical concepts, mostly from algebra, number theory, and probability theory. Such proofs always use the abstract concept of an *attacker*, without defining one specific strategy. Similar to the concept of known or chosen plaintext attacks, defining the attacker usually results in a stronger notion of security – assuming security can still be proven and the proof is correct.

Therefore, it is common practice to publish a new cryptographic system with a proof and exact description of security properties. Abiding by Kerckhoffs' Principle, this does not compromise the security of the system, because the security of each instance depends on the cryptographic keys. Publication is considered very important, because only it can establish that Kerckhoffs' Principle applies, and everyone can validate the proof.

3. COPY PROTECTION

Companies are interested in preventing the unauthorized reproduction of software and intellectual property. Over the years, various methods have been designed, but a hidden conflict remains: Software should be able to operate unhindered by the protection methods. At the same time, the protection methods should make it difficult to reproduce or recreate the software.

3.1 COMMON PRACTICE AND THE ARMS RACE

In the past, most copy protection relied on security through obscurity. The reason for this is that to run an algorithm, it has to be available to regular users. At best, these mechanisms can make it more difficult to analyze a program, but not on the scale required for a cryptographically hard problem. An example of this is code obfuscation, which increases the effort to analyze and reverse-engineer the algorithm, but does not make it practically impossible.

Therefore, copy protection schemes and the closely related digital rights management are usually only secure for some time, until an attacker determines the method, and security through obscurity fails. Once an attack is published, developers go on to release the next version of their schemes. However, this results in an arms race between the developers of software protection and the people trying to break it.

[3] Recommendation for Key Management, Special Publication 800-57 Part 1 Rev. 4, NIST, 01/2016

3.2 THE NEW PARADIGM

In an ideal world, a copy protection scheme would be able to turn arbitrary software into a black box: That is, attackers could observe input and output in regular use, but nothing else. Particularly, the inner workings of the algorithms are hidden. Unfortunately, this has proven impossible to achieve[4]. Even with such a black-box, it would be impossible to prevent an attacker from executing the software regularly, analyzing all observations, and reconstructing the software.

From a theoretical perspective, within the purview of secure multi-party computation, there are methods to evaluate a function securely, so that only its result is revealed to anyone. However, these solutions often provide only basic functionalities and are limited in that they are only secure on their own and cannot be used within a larger context. Moreover, while some functionalities, such as either addition or multiplication, can be realized efficiently in practice, they cannot be used together with the same constructions. There are constructions that do so, but they are of only theoretical interest for the time being, because of performance constraints of their implementations. An example of this is fully homomorphic encryption and various constructions using it[5].

In this article, we present a new paradigm for copy protection, known as the "Blurry Box® Scheme," which is based on an assumption about the complexity of software. To our knowledge, this is the first scheme that has a provable security property and is useable in practice. Informally speaking, it proves that the attacker cannot learn more about the software than if he tried to iterate every possible path through the program.

4. THE BLURRY BOX SCHEME

4.1 MECHANISMS

At its core, Blurry Box is based on the assumption that a hacker lacks the *domain knowledge* necessary to create a software product. For instance, a hacker may not be familiar with the underlying mathematics of a computer algebra system. The rationale behind this is that a hacker who knows the relevant domain knowledge could write the software himself, bypassing the protection altogether. One cannot protect software against such a hacker. This assumption is therefore justified and necessary.

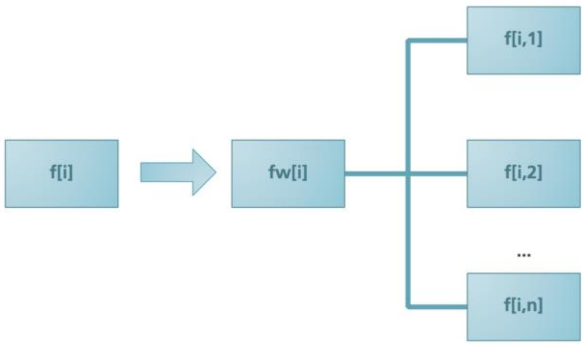
This lack of domain knowledge can be exploited to achieve secure protection. The main idea is to split the program code into small pieces to make it practically infeasible to retrieve all pieces

[4] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K, "On the (im) possibility of obfuscating programs," Annual International Cryptology Conference (pp. 1-18), Springer Berlin Heidelberg, August 2001.

[5] Craig Gentry, "Fully homomorphic encryption using ideal lattices," Symposium on the Theory of Computing (STOC), pp. 169-178, 2009

by running the code. The hacker's lack of domain knowledge prevents him from creating additional pieces on his own. The following sequence describes the protection mechanisms of the Blurry Box scheme:

- Assume that program code consists of several function blocks.
- Each function block is copied multiple times.
- Each copy is modified in such a way that it yields the correct values only for a *restricted set of inputs*. These modified copies are called *variants* of the function block.
- All variants together cover the entire input range of the original function block.
- Variants may be created by, e.g., deleting operations that are not necessary for a specific interval or by using approximation techniques.
- The software developer may aid the variant creation process through the domain knowledge used to build the software.
- A wrapper function that maps inputs to the address of the corresponding next variant is created. These wrapper functions are moved into a dongle, which can be done since these functions are sufficiently lightweight to run on restricted hardware

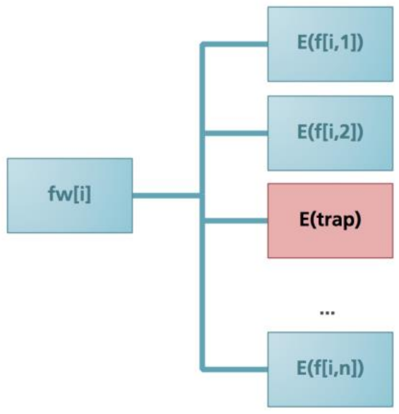


Each function block f[i] is split into several variants f[i,j] which yield the same values as f[i] only on restricted input sets. A wrapper function fw[i] maps inputs to the address of the corresponding variant.

Each variant is encrypted with a different key, known as the variant key, using the *Advanced Encryption Standard* (AES)[6]. Each variant key is encrypted with a secret key stored on the dongle. During each program execution, only the variants that correspond to the current set of input are decrypted. The hacker can only see the parts of the program code that correspond to previous input values.

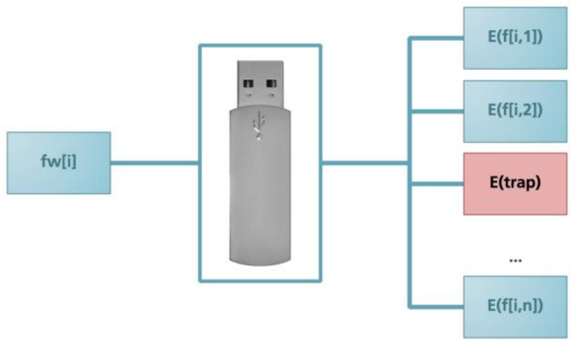
Up to now, the scheme described above can be trivially broken by simply decrypting one variant after another using the dongle. In order to prevent such a trivial attack, *traps* are introduced. Traps contain special variant keys that, when decrypted, force the dongle to lock itself, invalidating the license. Of course, during normal program execution, traps are never decrypted.

[6] Daemen, Joan, and Vincent Rijmen, "The design of Rijndael: AES-the advanced encryption standard," Springer Science & Business Media, 2013



Each variant is encrypted with a different key. In order to prevent an adversary from decrypting each variant, traps are introduced. Traps contain code that, when decrypted, forces the dongle to lock itself.

Note that the dongle is used only for determining the address of the next variant and for decrypting variant keys. The dongle has a state storage for detecting illegal sequences of variants which, like traps, causes the dongle to lock itself. This prevents replay attacks, since the hacker cannot go back to any previous point of the program without running it all over again.



The wrapper functions are moved into the dongle. In addition, the dongle has a state storage for detecting illegal sequences of variants.

4.2 SECURITY GUARANTEE AND ASSUMPTIONS

The security goal is to prevent a hacker from constructing a copy of a protected program that can be executed *without a dongle*. The following will reveal how the Blurry Box scheme meets this security goal.

One attack strategy would run the program and store all variants that are decrypted during the program's execution. This type of attack is called a *Copy-and-Paste attack*. If the program is sufficiently complex, it is not feasible to retrieve all variants with this strategy, because a hacker using only a Copy-and-Paste strategy would have to run the program for an impractically large set of input values.

Hackers therefore need to find a way to go beyond Copy-and-Paste attacks. If a hacker does come up with additional variants given already decrypted ones, he could completely bypass the protection. This is where the assumption becomes important that a hacker has almost none of the domain knowledge necessary to create the program code. Neither should the hacker be able to learn the inner workings of the software. The latter is called the *assumption of the inherent complexity of program code*, which can be formalized as follows: *Given a subset of variants, a hacker should not be able to create additional ones.*

Not all programs allow for creating variants with the above-mentioned security property.

For instance, a simple program such as "Hello World" does not have this property and cannot be protected by the Blurry Box scheme. Only programs that are sufficiently complex can be protected effectively. Note that not the whole program has to fulfill this requirement; it suffices if only a part of it is sufficiently complex. Typically, this complex part is exactly the critical part that needs to be protected. Programs that meet this requirement include video games, raster graphics editors, and feedback control systems.

In accordance with the principles of modern cryptography, the security of the Blurry Box scheme can be proven rigorously based on falsifiable assumptions. To this end, a mathematical security model was conceived that is tailored to this setting. In this model, security is not an absolute factor, but *defined by comparison*. More specifically, security is defined by comparing every conceivable attack on the Blurry Box scheme with the mentioned Copy-and-Paste attack. This definition is reasonable, since Copy-and-Paste attacks cannot be prevented, but become practically infeasible and therefore not a security concern. It can be proven that no attack strategy can do better than the Copy-and-Paste strategy.

Formally, this means that, for every attack strategy, there exists a Copy-and-Paste attack that retrieves the same number of variants with the same number of dongle calls. The proof is based on three assumptions:

1) The security of the encryption scheme (IND-CCA2 security)[7],
2) The security of the dongle (the key is not extractable) and
3) The assumption on the inherent complexity of the program code.

IND-CCA2 security is a well-established security notion for encryption schemes. Intuitively, an encryption scheme is IND-CCA2 secure if no adversary can gain any information about a plaintext by analyzing a corresponding ciphertext, even if he can decrypt other arbitrary ciphertexts. The encryption scheme in the Blurry Box scheme has to be IND-CCA2 secure, because a hacker – analyzing the (encrypted) program code – has access to the dongle that contains the secret key.

4.3 CORRECTNESS

The protected program has the same functionality as the original, as each mechanism preserves the functionality of the original program. This holds for variant encryption: By design, each variant yields the same values as the original function block for restricted input sets. All variants

[7] Katz, Jonathan, and Yehuda Lindell, "Introduction to modern cryptography," CRC press, 2014

together cover the entire input range for the original function block, retaining the functionality of the original function block in the protected program. Furthermore, moving the wrapper functions into the dongle does not alter the functionality if inputs are mapped to correct variants, i.e. the variants that compute the correct outputs for the respective input values. If a variant is constructed using an approximation technique, it has to be shown that this approximation is good enough. The approximation error has to be smaller than a given value ε that depends on the application. Due to limited machine accuracy, the original program may already have an approximation error.

An implementation of these mechanisms in protection solutions requires experience with tools to create robust protected applications while avoiding false positives in traps. Decrypted program code is checked with hashes and executed only if these are correct. Restarting and using specified functions needs no preset state.

5. MARKET RELEVANCE, USE CASES, AND INDUSTRIAL APPLICATIONS

Know-how protection is key for keeping Unique Selling Points (USPs) and retaining competitive advantages. Protection against counterfeiting secures revenue streams. Integrity protection reduces liability risks and improves safety for society, human life, and the environment. Innovative business models using software licensing enable new earning and post-sales opportunities.

The protection mechanisms described in this article exceed the standard required by the market, but give manufacturers a clear sense for the effectiveness of a protection solution prior to implementing it. In times of increasing cyberattacks and espionage, as well as smart grids in critical infrastructure and industrial automation systems, preventive protection systems have become more important than ever before.

5.1 PROTECTION AGAINST REVERSE ENGINEERING

Reverse engineering can be used to reveal the algorithms and mechanisms realized through software. Typical tools are disassemblers or de-compilers, including sophisticated tools like IDApro for native code, Reflector with plugins for .NET, and Java de-compilers like JD. In combination with memory dumps of executable code, these tools can analyze executable code. Competitors can use this knowledge to implement the mechanisms in their own software without having to invest major efforts in R&D.

When unscrupulous attackers know the program code, there is also a risk for the data processed by it. Hackers use this knowledge to develop malware and other attacks by detecting weak implementations and security holes.

Mechanisms that can make reverse engineering very difficult include:

1. Code encryption

2. Code duplication
3. Code modification
4. Insertion of traps, i.e., program code which is not functionally required, but can lock the keys
5. Selection of code variants in a secure element hardware
6. State behavior of decryption hardware

5.2 PROTECTION AGAINST COUNTERFEITING

Using the mechanisms above – and storing cryptographic keys in unclonable, secure hardware – provides protection against counterfeiting. According to a survey by German engineering federation VDMA published in 2016[8], 90% of industrial machine vendors are affected by product piracy – half of them by the counterfeiting of entire machines. Dr. Festge, president of VDMA (Verband Deutscher Maschinen- und Anlagenbau, Mechanical Engineering Industry Association), said *"As data is becoming the lifeblood of commercial value creation, counterfeiters and product pirates will be taking the same route. Simply copying the nuts and bolts or discrete circuitry will not be enough for them. They will be targeting digital designs, the software running on our machines, and the data stored in our databases."*

5.3 PROTECTION AGAINST TAMPERING

Pieces of machinery are increasingly brought to life by software. It is the software operating the device that enables functions and features, making it essential that only genuine, not manipulated program code from authorized parties can be executed. Protected code cannot be tampered with and can execute secure mechanisms for software updates and upgrades.

5.4 BENEFITS OF SOFTWARE LICENSING

By shifting the added value from hardware to software, vendors can benefit from cheaper logistics and production. Devices, machines, and software are deployed in identical versions for all users. Only the individual licensing decides how the vendor's product can be used in practice. License deployment is unique for each product or user, but also highly automated through integration in ERP systems like SAP or ecommerce platforms.

More important than cost reduction are the opportunities this creates for expanding one's target group by tailoring solutions to each customer's needs and configuring product features via licensing. Furthermore, an app-store-like concept can be introduced to seize new post-sales opportunities or recurring revenue streams in the form of pay-per-use or subscription models and more. This helps vendors increase their revenues over their products' lifetime, gives users more flexibility, and reduces upfront investments.

[8] Steffen Zimmermann, "Study on Product Piracy 2016," VDMA, April 2016 (http://pks.vdma.org/article/-/articleview/13069313)

The security goals and possible benefits of these solutions are illustrated in these use cases.

5.5 USE CASE: INDUSTRIE 4.0

Industrie 4.0[9] can only take off if holistic and sustainable security architectures with software protections are implemented. The increasing number of networked devices is a magnet for cyber-attackers ready to exploit vulnerabilities and manipulate production processes to cause massive damage or seize technical know-how. The opportunity to test preventative measures at SmartFactory^{KL}, a leading competence center and manufacturer-independent demonstration and research platform, delivered great results for all manufacturing vendors.

The flexible production line designed according to Industrie 4.0 principles at SmartFactory^{KL} has been upgraded with added security. Software protections have been applied to the connected IoT and cyber-physical systems that form part of the facilities. Cryptographic keys for secure communication with open standard OPC UA are stored securely in secure elements. The product data used in the production process and stored in RFID tags is secured by digitally signing the data on the tags. Its integrity is protected from the tag to the cloud or vice versa and verified with tablet computers on the shop floor. This ensures that the production line only accepts valid production data from authorized parties.

5.6 USE CASE: IOT INDUSTRY

Microcontrollers are the heart of some of today's IoT devices: Small, energy-efficient systems that implement the algorithms and features for sensors, actuators, motor controls, and more. They communicate via open networks and need appropriate protection. The chapter, Protecting Endpoints, of the IIC's Industrial Internet Security Framework (IISF) describes these requirements: Software protection, which is a must, is the basic vehicle for truly secure IoT devices. To facilitate this, makers of microcontroller and semiconductor integrate protection mechanisms in their development tools and offer secure boot strap loaders. This allows production volume controls, secure, tamper-proof firmware downloads and updates as well as flexible feature licensing. IoT/IIoT devices using such features include different sensors with intelligent preprocessing and network interfaces, intelligent RFID-readers with OPC UA communication, pumps, actuators, valves, or intelligent grippers. Manufacturers benefit from lower logistics costs due to unified production and feature configuration during final tests or even at the customer. After-sales business will be created with the app-store concept known from the consumer market. Licensing is key to new monetization mechanisms in today's digitization.

[9] http://www.plattform-i40.de/I40/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html

5.7 USE CASE: AUTOMATION INDUSTRY

Modern machines communicate hierarchically with Manufacturing Execution Systems (MES) and Enterprise Resource Planning (ERP) systems. Integrated robots contain PLCs and multiple intelligent servo mechanisms. A milling machine has a PLC and a set of sensors and actuators. These and other components need to be configured, calibrated, or replaced during initial commissioning or later maintenance, e.g., calibrating an arm, the quality data of a spindle, or motor parameters. The process is time-consuming and requires highly trained staff.

It would be an immense benefit for all parties involved if components could be installed and replaced in a simple plug-and-play process, not unlike that known from USB devices. This creates a new challenge: plug-and-play processes must be secure to guarantee the safety of the machine and the quality of its output. It must also comply with all open standards and be interoperable to enable vendor independence. This can be achieved by using OPC UA for secure communication and standardized data with protected software that ensures integrity and protects the vendor’s know-how and USPs.

5.8 USE CASE: AUTOMOTIVE INDUSTRY

In modern cars, embedded software has become ubiquitous. One typical application for software protection would be the car diagnostic system. It needs to be updated regularly, as new models are brought to market and new part numbers or repair instructions are added. The diagnostic mechanisms contain a lot of AI (artificial intelligence) and know-how of the system’s vendor. Integrity is key, as are licensing and software protection.

Vendors of diagnostic systems use software protection to safeguard the know-how in their software and data and offer subscription-based licensing to car maintenance stations.

5.9 USE CASE: BANKING INDUSTRY

Automatic Teller Machines (ATMs) have to handle substantial amounts of cash and are exposed to severe threats: Gas is inserted to blow up the ATMs, banknotes are forged, and ATM software is manipulated with maintenance functions originally meant for authorized service staff only. The answer is software protection with strong authentication and mobile security devices like dongles. This offers strong protections for the integrated software, licensing for additional service functions, two-factor authentication with dongle and password, or time-limited authorizations that can be regularly renewed for authorized staff.

5.10 USE CASE: MEDICAL INDUSTRY

Software protection is used for protection against product piracy and integrity protection for embedded software to fulfill the regulatory standards for medical equipment (such as FDA, Medical Products Act, etc.). Flexible feature licensing expands existing business models with pay-per-use schemes for dental, computed radiography, or medical diagnostics systems. Dental labs or dentists can buy machines at a reduced price and pay separately for the actual fillings produced by pay-per-use billing. Small laboratories, orthopedic doctors, and other healthcare facilities, who often cannot afford large upfront investments, can rent top-of-the-breed equipment, offer highly professional service, and stay competitive.

5.11 USE CASE: MICROGRID CONTROLS

Intelligent microgrids require the intelligent sharing of data for energy-efficient and reliable power distribution. Their controls depend on software-enabled functions and form part of a critical infrastructure that is highly exposed to cyber-attacks. Countermeasures include protection against reverse engineering, integrity protection from secure boot to secure firmware updates, and flexible licensing to enable features and measure their use reliably.

5.12 USE CASE: TEXTILE INDUSTRY

In the textile industry, embroidery machines are a common sight in the production of clothes, shoes, high-tech wearables, or industrial textiles for seat heating or antennas. Factories often use multiple machines in one location.

The challenge for machine vendors is to prevent the pirating of complete machines. Their control software contains the know-how needed for high-quality, flexible embroidering at high efficiency. The threat is mostly due to the extremely competitive textile industry itself. Machines are made with a set of features; their vendors offer a basic model at competitive low prices and sell more sophisticated functions – often software-realized – on top. This makes it important that factories do not, for example, buy several basic machines and only one with the full feature set that they then copy to the others.

Buyers of textiles face additional challenges: they provide production data – so called punch data – to the factories producing their fabrics on these machines. Unscrupulous factory owners could run extra shifts and produce additional wares for the grey market. The purchasers' data needs to be controlled, from the original CAD design to the embroidery machine, which also includes controlling how many products made from the designs.

This can be seen as a general "digitization" or "Industrie 4.0" use case. It shows the great importance of protecting production data, which is essential for other areas as well, for example, additive manufacturing (3D printing) or technology data, in the form of specific parameters for machines.

With software protection, flexible feature licensing, production data encryption, and secure storage of encryption keys and options, all of these challenges can be solved.

There are many more use cases. The chosen examples reveal the commercial relevance of the topic and the benefits of software protection. The mechanisms according to Kerckhoffs' Principle published here allow developers and manufacturers to understand how much protection they need and why they should not put all of their trust into "security through obscurity."

- Return to IIC Journal of Innovation 3nd Edition landing page for more articles

- Download the IIC Journal of Innovation 3nd Edition