sensors

MDPI

Article

UHF RFID Prototyping Platform for ISO 29167 Decryption Based on an SDR

Georg Saxl *, Manuel Ferdik, Moritz Fischer, Martin Maderboeck and Thomas Ussmueller

Microelectronics and Implantable Systems Group, Department of Mechatronics, University of Innsbruck, Innsbruck 6020, Austria; manuel.ferdik@uibk.ac.at (M.F.); moritz.fischer@uibk.ac.at (M.F.); martin.maderboeck@student.uibk.ac.at (M.M.); thomas.ussmueller@uibk.ac.at (T.U.)

* Correspondence: georg.saxl@uibk.ac.at

Received: 1 March 2019; Accepted: 11 May 2019; Published: 14 May 2019

Abstract: Ultra high frequency radio frequency identification (UHF RFID) is becoming a key technology in the Internet of Things. It allows the implementation of batteryless and wireless nodes, including sensors and actuators. Due to its possible transmission range of >10 m and potential to carry critical information, security is a highly important topic. For this reason, the International Organization for Standardization has published several crypto suites for UHF RFID within the ISO-29167 standard in 2014. Recently, research has focused on implementing those encryption features on the transponder side. However, currently no crypto enabled UHF RFID readers are available. In order to cope with the rapid development in this field, 'open' and flexible readers based on software defined radios are needed. They make it possible to quickly adapt the protocol and to test new functionalities such as encryption. This paper deals with the first implementation of the ISO 29167-19 standardized RAMON decryption on a software defined radio. The programming of this hardware is done in LabVIEW which allows for controlling the built-in transceiver modules. However, first measurements show that the decryption takes 51 s. This is because LabVIEW is not suitable for handling very large numbers like they are utilized in cryptography. Because such a long processing time is not feasible in experiments nor in a real-life scenarios, this method is not suitable for a prototyping platform. Thus, a different approach is chosen to optimize the decryption processing time. LabVIEW still provides the framework for handling the protocol and controlling the transceivers, but the decryption is performed in a Java application. In that way, the entire decryption process takes only about 2.2 ms, which is 23,318 times faster than the implementation in LabVIEW. Thus, this new approach meets the necessary timing requirements and is suitable for realistic application scenarios. The shown method allows development and testing of new functionalities in UHF RFID systems but may also be employed in any application that require long processing times in LabVIEW. Furthermore, the implementation of decryption features is the first necessary step towards a fully compliant, crypto enabled interrogator for UHF RFID, featuring a high adaptability.

Keywords: cryptography; data security; radio frequency identification; software defined radio; UHF communication

1. Introduction

The Internet of Things (IoT) is a vision of the future that is currently attracting a lot of attention. It describes a network of billions of devices, sensors and gadgets interconnected via embedded, digital communication interfaces. This allows quick and decentralized information processing, execution, sharing and broadcasting. Typically, wireless connections between nodes are preferred in IoT, due to reduced installation efforts, higher flexibility and often reduced costs [1]. However, wireless systems

have the disadvantage that the power supply cannot be ensured via a cable. Therefore, either batteries are needed, which are to be avoided from an environmental and maintenance point of view, or the energy is generated locally on the wireless device itself. Thus, batteryless and wireless systems are a very interesting option. One of the best known representatives is "Ultra High Frequency Radio Frequency Identification (UHF RFID)".

The original idea of this technology is contactless identification of goods or persons, which is demonstrated in Ref. [2]. However, recent research also deals with sensors based on UHF RFID, which can be used to determine physical parameters of the environment. This is an extremely exciting development with regard to the advancing digitization in the course of Industry 4.0 as well as in home automation and many other areas [3]. Due to battery-free wireless operation, the sensors are maintenance-free and can be used almost anywhere, even in inaccessible places such as inside components. Another recently discussed application of UHF RFID is traffic management and intelligent parking, where cars can be labeled with passive UHF RFID tags to allow, for example, direct billing of parking costs or better planning of traffic flows [4,5].

Due to the advantages of UHF RFID, the technology has become widespread in recent years. This is accompanied by a growing need for security [6,7]. Especially due to the fact that UHF RFID allows ranges of more than 10 meters, the security of the transmitted data is even more crucial than with near field communication systems. Possible implementations of encryption for RFID have already been shown several times [8–10]. The problem with UHF RFID is the low energy available at the passive transponder [11]. Therefore, special attention must be paid to low complexity of the encryption method to reduce hardware and energy consumption. Suitable approaches for tag side encryption such as the use of Elliptic Curve Cryptography have already been presented in Refs. [12,13]. In Ref. [14], a tag prototype is presented, which demonstrates RAMON encryption on a Field Programmable Gate Array (FPGA). However, the authors postpone detailed measurements and a silicon implementation until compatible readers are available. Until 2014, there was no standardized encryption procedure for UHF RFID. With the ISO 29167 standard, this has changed. It provides nine different Crypto Suites including AES-128, PRESENT-80 and RAMON.

To cope with the rapid development of UHF RFID both in cryptography and other areas such as passive sensors and actuators [15], "open" (easy adaptable) readers in the form of prototyping platforms are required. To keep up with rapid development, it is important that both hardware and software changes can be made quick and easily. Software defined radios (SDR) are one way to meet this requirement as shown in Refs. [16,17].

With this paper, we present the first SDR based prototyping platform with a time-optimized implementation of the ISO 29167-19 standardized RAMON decryption. An NI PXIe-1075 chassis is chosen as the hardware platform, which is equipped with a transceiver module PXIe-5644R. The implementation of a basic reader on this platform has already been demonstrated [16]. The RAMON procedure is well suited for UHF RFID because the encryption on the transponder side is resource-saving and the main processing work occurs on the reader side. However, this leads to the fact that the reader takes a very long time to decrypt if not properly implemented. Therefore, the time efficiency is taken into account in the following procedure.

2. RAMON Crypto System

The RAMON crypto system is a mixture of the "RAbin crypto system" [18] and the "MONtgomery multiplication" [19]. The Rabin crypto system is an asymmetric encryption technique whose security is based on the factorization problem. In combination with the Montgomery multiplication, it is used to achieve a low power consumption for encryption, but with a higher effort needed for decryption. Therefore, this method is particularly suitable for UHF RFID systems, since little energy and thus little computing power are available on the passive node (transponder), but, in proportion, a lot more energy and computing power is available on the reader side. This paper focuses on the reader side decryption

and implementation on an SDR. For a better understanding, the encryption [20] is described briefly in the following section.

2.1. Encryption

The plain message x shown in Figure 1 has a total length of 128 bytes and consists of 16 bytes CHI (challenge interrogator), 16 bytes RNT (random number tag), 95 bytes TLV (tag length value) and 1 byte zero padding. This message will be processed as shown in the flow chart in Figure 2. First, the message is mixed with a specific pattern to further enhance security.

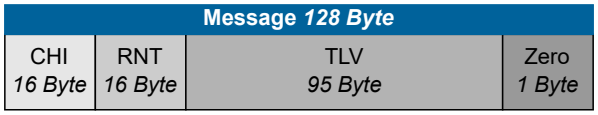

Figure 1. Format of the plain message x which will be processed by the RAMON crypto system.

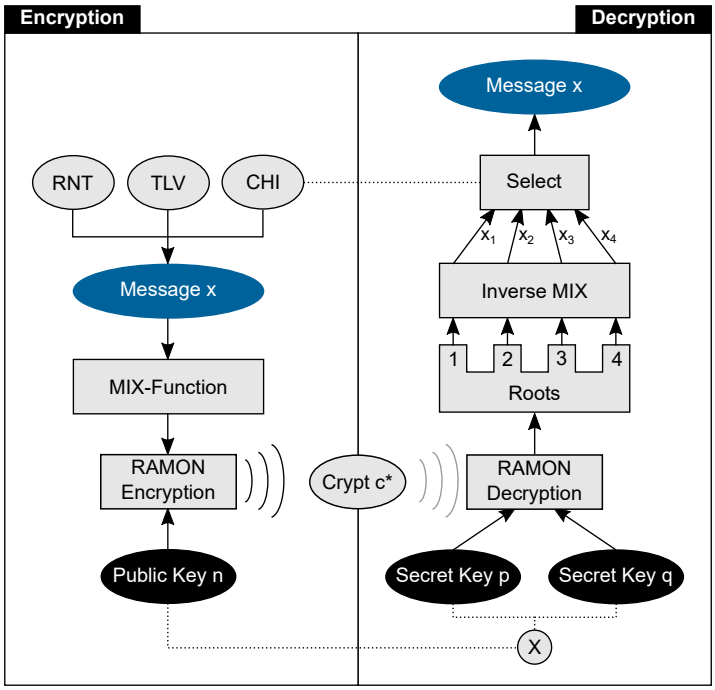

Figure 2. Schematic representation of the RAMON crypto system process. The plain message x is mixed and then RAMON encrypted. This requires the public key n, which has already been pre-transmitted from the reader to the tag. The Crypt c^* is transmitted to the reader via the air interface and decrypted with the help of the secret keys p and q. There are four possible solutions to choose from. Since the reader already knows CHI, it can be used to identify the actual plain message and to select the correct x.

To encrypt the given message, the public key n, which is the product of the secret keys p and q, has to be pre-transmitted and already known at the transponder. The Rabin crypto system calculates the crypt c, which is the result of encryption performed on the mixed message x with the public key n:

$$c = x^2 \bmod n.$$

In order to avoid the power-intensive modulo operation the Montgomery multiplication is implemented within the RAMON crypto system. Its purpose is to calculate c^*, which is defined as

$$c^* = x^2 \, R^{-1} \bmod n. \tag{1}$$

The Residuum R is defined as well chosen power of two (in case of RAMON $R = 1088$). Equation (1) is calculated by the efficient Montgomery multiplication algorithm, which contains only simple arithmetic operations and is therefore very power-efficient.

2.2. Decryption

The simplicity of encryption via the efficient Montgomery multiplication has to be compensated by a more complex decryption. Basically, the reader has to calculate the square roots of the problem to receive the decrypted message x. Thus, it is necessary to transform the crypt c^* back to c by

$$\begin{aligned} c &= c^* R \bmod n \\ &= x^2 R^{-1} R \bmod n \\ &= x^2 \bmod n. \end{aligned} \tag{2}$$

Next, the inverse elements of the secret keys p_i an q_i have to be determined by the "Extended Euclidean Algorithm" [21] so the following equation is valid:

$$p_\mathrm{i} \cdot p + q_\mathrm{i} \cdot q = 1. \tag{3}$$

To calculate the square root of c an intermediate step is needed, which first calculates the square root of $c \bmod p$ and $c \bmod q$ resulting in

$$a_\mathrm{p} = c^{\frac{p+1}{4}} \bmod p, \tag{4}$$

$$a_\mathrm{q} = c^{\frac{q+1}{4}} \bmod q. \tag{5}$$

With a_p and a_q, it is possible to determine the four square roots modulo n via the "Chinese Remainder Theorem" (CRT) [22]. One of those solutions is the correct decrypted message x:

$$x_1 = (p_\mathrm{i} \cdot p \cdot a_\mathrm{q} + q_\mathrm{i} \cdot q \cdot a_\mathrm{p}) \bmod n, \tag{6}$$

$$x_2 = n - x_1, \tag{7}$$

$$x_3 = (p_\mathrm{i} \cdot p \cdot a_\mathrm{q} - q_\mathrm{i} \cdot q \cdot a_\mathrm{p}) \bmod n, \tag{8}$$

$$x_4 = n - x_3. \tag{9}$$

After inverse mixing all four solutions, the correct plain message is determined by the CHI which is part of the message and is known by the reader.

3. Basic Implementation in LabVIEW

As a hardware platform the software defined radio from NI was chosen. It consists of a PXIe-1075 chassis, a PXIe-8135 controller and a PXIe-5644R transceiver module. The implementation of a UHF RFID reader using this platform and the graphical programming in LabVIEW has already been shown in [16]. In order to integrate the decryption seamlessly into the existing reader protocol, LabVIEW is also utilized for the implementation of the RAMON procedure. Since LabVIEW is only able to handle numbers with a maximum of 64 bit, the standard calculations (add, multiply, divide, ...) are not suitable for cryptography where numbers of 1024 bits and more are used. Due to this fact and the lack of a data type that can handle numbers of this size in LabVIEW, it is necessary to develop an own library which contains all needed operations:

- Big Binary operations: Basic arithmetic operations,
- RAMON crypto system: Equations for decryption,
- Array conversion: Conversion of test vectors and results.

With these basic operations, it is possible to rebuild the equations given in Section 2.2 and thus construct a LabVIEW program for the RAMON decryption as shown in Figure 3. The correlations of the functions (a) to (f) from Figure 3 and the Equations (3)–(9) are shown in Table 1.

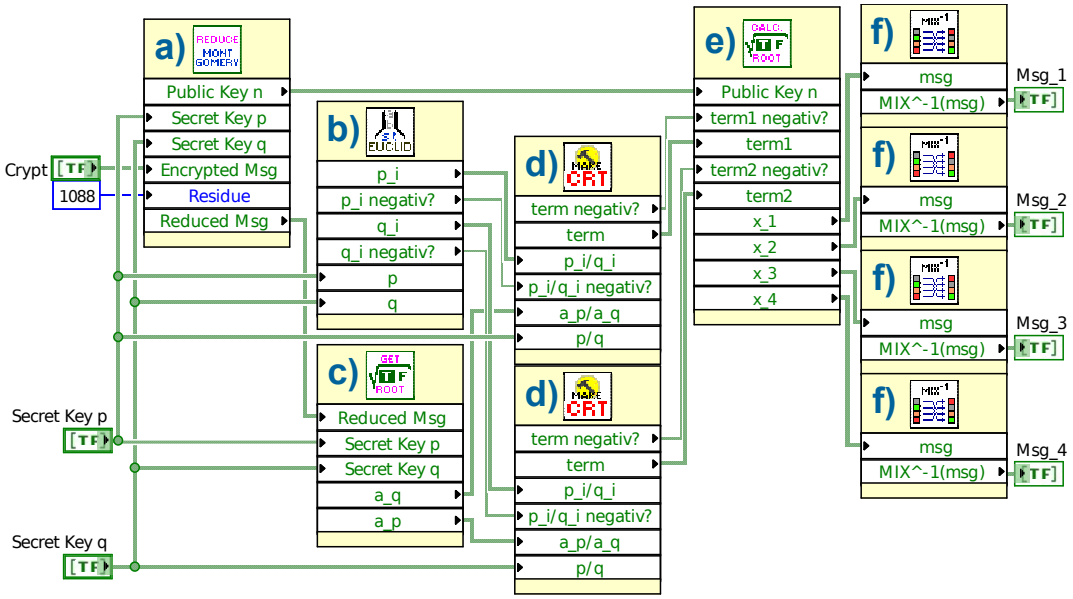

Figure 3. Decryption process implemented in LabVIEW: The crypt c^* and both secret keys p and q are inputs. The four possible solutions are computed by the LabVIEW-SubVI Blocks: (**a**) Reduce Montgomery; (**b**) Extended Euclidean Algorithm; (**c**) Get-Root; (**d**) make CRT-Term; (**e**) Calculate Root; (**f**) Inverse MIX-Function.

Table 1. Correlation between the LabVIEW SubVI-Blocks and RAMON decryption equations.

SubVI	Equation	Description
(a)	(3)	Calculates real cyphertext c from given c^*, p, q and R
(b)	(3)	Computation of inverse elements p_i and q_i from secret keys p and q
(c)	(4)–(5)	Intermediate step to calculate square roots depending on p and q
(d)	(6)–(9)	Computing $term1$ and $term2$ of the root-calculation: $term1 = p_\mathrm{i} \cdot p \cdot a_\mathrm{q}$ $term2 = q_\mathrm{i} \cdot q \cdot a_\mathrm{p}$
(e)	(6)–(9)	Calculating the four roots of the problem
(f)	(-)	Inverse MIX function for demasking to plain messages x_1,...x_4

3.1. Runtime Optimization of Arithmetic-Algorithms

After basic tests with test vectors taken from the ISO 29167-19 standard [20], very long computation times have been measured. To achieve an efficient decryption, the operation time has to be cut down. Therefore, faster algorithms have to be used for basic binary arithmetic which are described in the following.

3.1.1. Ripple Carry Adder

The most used arithmetic block by all hierarchical higher SubVIs (RAMON-Blocks) is by far the "ADD" Algorithm. To enhance the performance of all other blocks, the add algorithm has to be optimized. This is achieved with the use of a "Ripple Carry Adder" [23]. It is based on an XOR operation on both addends $a[\,]$ and $b[\,]$ as well as on a third array $c[\,]$ which contains carry bits. The result $s[\,]$ is defined as

$$s[i] = a[i] \oplus b[i] \oplus c[i].$$

The carry array $c[\,]$ is initialized with $c[0] = 0$ and all further entry's are calculated with

$$c[i+1] = \big(c[i] \wedge (a[i] \oplus b[i])\big) \vee (a[i] \wedge b[i]).$$

3.1.2. Multiply

The multiplication of two large numbers is implemented by using the "Shift-and-Add" procedure mentioned in Ref. [24]. The flow diagram of the algorithm is shown in Figure 4. It calculates the product of the two n-bit long numbers X and Y. Therefore, the multiplicand X is added to itself Y times. As a result, the algorithm delivers the $2n$-bit long product register A.

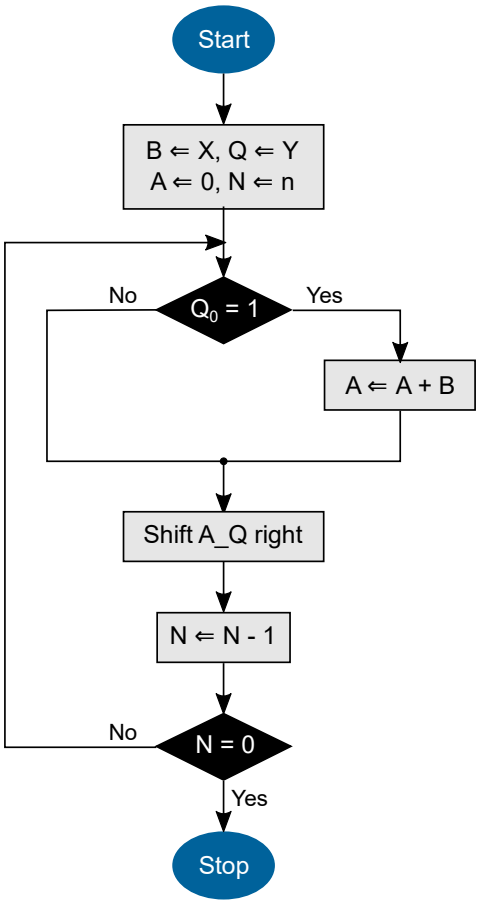

Figure 4. "Shift-and-Add" multiplication algorithm based on Ref. [24]. The aim is to multiply the n-bit long numbers X and Y. The product register A is initialized with 0 and the counter N with n.

3.1.3. Quotient and Remainder—Modulo

Another important arithmetic operation for cryptography is the modulo operation. The LabVIEW implementation is based on the "Quotient and Remainder" theorem shown in Algorithm 1. The pseudo-code is similar to the "Multiple-precision division" algorithm from Ref. [25]. To calculate quotient q and remainder r of a division x/n, it is possible to subtract $x - n$ exactly q-times until

$$r = x - n \cdot q < n.$$

The algorithm shown in Algorithm 1 basically manipulates the divisor n by left-shifting (multiplication with power of two) until the most significant bit (MSB) of the boolean array is at the same position as the MSB of x. Therefore, it is possible to reduce x by power-of-two times at once, which results in a faster solution for the problem than step-wise subtraction. How often x has been shifted is noted and transferred to q.

Algorithm 1: Pseudo Code of the 'Quotient and Remainder' algorithm to calculate the modulo of two numbers.

```
Input: BigNumbers x[], n[] in Boolean-Array format
Output: q[], r[] ← x = d · n + r
i ← 0; q ← 0; m ← 0; r ← x;
while n < r do
    i = length(r) - length(n);
    m = n ≪ i;
    if r < m then
        m = m ≫ 1;
        i = i - 1;
        r = r - m;
    else
        r = r - m;
    end
    q = q + i;
end
```

3.1.4. Modular Exponentiation—ModPower

The "Modular Exponentiation" or "Mod-Power" calculation is primarily used by Block (c) "Get Root" from Figure 3. It computes

$$r = x^e \bmod n.$$

When x^e becomes a huge number, it is necessary to split up the calculation. By increasing the x exponential and dividing it by n, an overly large number can be avoided as an intermediate result. To calculate the result of x partially exponential, an algorithm called "Exponentiation by squaring" [26] is used. It considers e as binary number and for each 0 (starting from MSB) x gets squared ('S') and for every 1 in e x gets squared and multiplied ('SM') by x. Equation (10) shows an example for $x^{11_{10}} = x^{1011_2}$. Therefore, the exponent $e = 11_{10} = 1011_2$ causes $1011 \rightarrow SM\ S\ SM\ SM$. The first SM only means $1^2 \cdot x$ and thus can be canceled and started with x:

$$x \xrightarrow[S]{\exp(2)} x^2 \xrightarrow[S]{\exp(2)} x^4 \xrightarrow[M]{\cdot x} x^5 \xrightarrow[S]{\exp(2)} x^{10} \xrightarrow[M]{\cdot x} x^{11},$$

$$x^{11_{10}} = x^{1011_2} = \left((x^2)^2 \cdot x\right)^2 \cdot x. \tag{10}$$

After each intermediate step ('S' and 'SM'), it will be divided by n and continued with the remainder to keep the result a small number.

3.2. Simulation Results

In order to test the proposed implementation, 1024 bit test vectors taken from [20] are employed. The encrypted message and secret keys are passed to the routine for decryption. To determine the performance, the calculation time is evaluated.

With the mentioned algorithms in Section 3.1, it is possible to decrypt c^* using LabVIEW correctly. However, it takes a very long time to calculate. The average duration of ten decryption processes is 51.3 s. Since such a long processing time does not allow a proper experiment in a real-life scenario, this method is not suitable for a prototyping platform of a UHF RFID reader.

The LabVIEW built-in resource monitor allows for determining the execution times and numbers of calls of each SubVI. Based on this, the usage in percent of each basic operation called by the decryption blocks (Figure 3) is calculated and listed in Table 2. Since it is called more than 2×10^6 times, the add operation consumes the most processing time with a total time of 28.4 s. In order to

achieve an acceptable total processing time of a few milliseconds, which means an improvement of several orders of magnitude, only an optimization of the algorithms is not sufficient. For example, the ripple carry adder algorithm would have to be accelerated by a factor of about 10^4. Because this is not possible even with the fastest algorithms, the decryption cannot be implemented in this way in LabVIEW.

Table 2. Breakdown of the calculation duration to the individual blocks from Figure 3. As one can see, the adder consumes the most time due to the high number of calls.

		Number of	Block Usage [%]					
Operation	Time [ms]	executions	(a)	(b)	(c)	(d)	(e)	(f)
Adder	28454	2029269	0.07	26.82	72.96	0.04	0.11	0.00
Shift	6817	1940217	0.06	30.84	68.95	0.09	0.07	0.00
Modulo	3556	1327	0.08	23.06	76.71	0.00	0.15	0.00
Multiply	3588	2420	0.04	37.93	61.86	0.17	0.00	0.00
Subtract	2277	508301	0.08	23.18	76.53	0.00	0.15	0.00
Resize	5413	5088584	5.03	24.99	64.84	0.00	5.11	0.00
Compare	1170	1015246	0.08	23.11	76.67	0.00	0.15	0.00

4. Time-Optimized Implementation Using External Calculations

Due to the long processing time of the decryption with LabVIEW, a different approach had to be found. However, since the transceiver modules of the PXIe chassis can only be controlled with LabVIEW, and the fully functional protocol of the UHF RFID reader is already available, the use of LabVIEW cannot be avoided. Therefore, the idea is to outsource the decryption to an additional, independent application. The decision for the platform was made in favor of Java, based on its ability to easily handle large numbers with the BigInteger class and previous experience in Java programming. To benefit from the speed advantage of the Java code, the idea is to run a Java application for the calculations in parallel with the LabVIEW program. In order to exchange data between the programs, a TCP/IP connection is employed. After this connection is established, LabVIEW sends the crypt c^*, the secret keys p and q as well as CHI and receives only the correct plain message x from the Java application. In the proposed implementation, LabVIEW is configured as a Client, while the Java applications provide the TCP/IP server. The flowchart in Figure 5 shows the decryption process executed by both LabVIEW and the Java application, which are communicating via the given TCP/IP connection. The introduction of a TCP/IP connection does lead to a potential point of attack, since the decrypted data is ultimately transmitted unencrypted, even if internally. However, this is the implementation of a prototyping platform, which is used for the development of UHF RFID systems. Thus, the security of this implementation is not in focus, unlike in a commercial RFID reader.

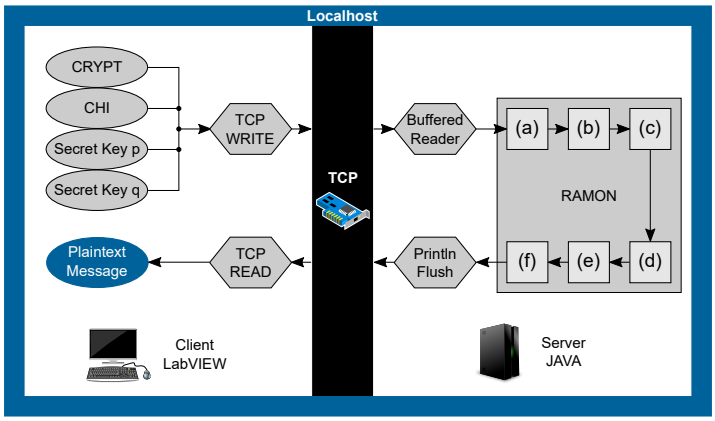

Figure 5. Flow chart of the time-optimized decryption using a TCP/IP connection to interact between LabVIEW and the Java application. CHI, the secret keys p and q and the crypt c^* are transferred from LabVIEW to the Java application via a TCP/IP connection. The decryption process is performed in the Java application. It sends back the plain message x to LabVIEW when completed.

4.1. Implementation

4.1.1. LabVIEW

Figure 6 shows all necessary steps in LabVIEW to initialize a TCP/IP connection and sends all data to the Java application. Since Java interprets input from a socket as a string, all needed variables like the crypt c^*, CHI and both secret keys p and q have to be converted from boolean arrays to strings. This conversion is implemented by simply checking for "true" or "false" in the array and putting together a string of ASCII coded ones and zeros. To send only one message to the Java application for simplicity, all strings are separated by an "X" and merged together, so that Java can later identify all separate variables. The "End-of-Line" added to the end of the message in Figure 6 is required; otherwise, the "Buffered Reader" in Java can not detect the end of the message.

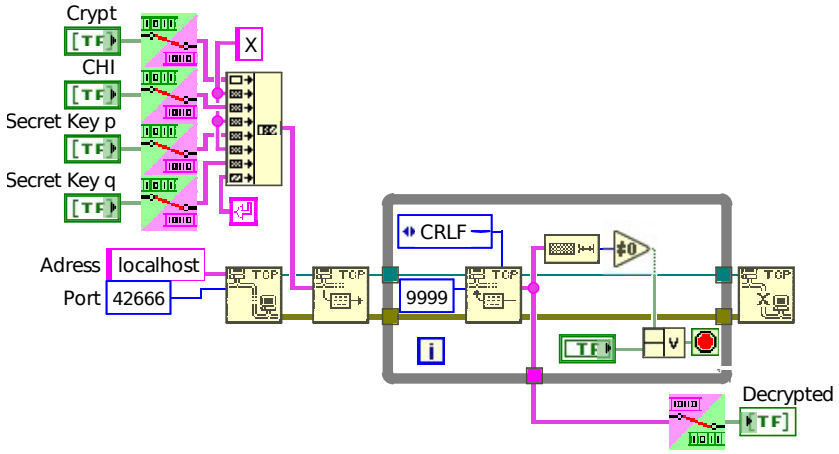

Figure 6. LabVIEW part of the time-optimized decryption process. The crypt, CHI and secret keys are prepared and then transferred via the established TCP/IP connection to the Java application on the localhost. Afterwards, the program waits for a response from the Java application. The plain message is shown at the frontend after reception and the TCP/IP connection is terminated.

The TCP/IP connection in LabVIEW is easily established by linking the address "localhost" as string (alternatively IP: 127.0.0.1) as well as a port number. With the standard block "TCP write", the message is then sent to the Java application. To receive a message back, the "TCP read" block is needed. To ensure the receipt of an incoming message, this block is placed inside a loop, which is stopped after a message is detected by checking its length. After receiving the message, LabVIEW

closes the connection and finally needs to convert the received string to a boolean array. It is important to mention that this conversion block also deletes all 'End-of-Line' symbols attached by Java; otherwise, the result is false. Finally, the boolean array is the correct plain message x and can be processed further.

4.1.2. Java

The decryption in the Java application is working like described in Figure 5. First, the Server is initialized with the local host IP address 127.0.0.1 as well as a port number. After successfully opening the server socket, Java enters a "Buffered Read" mode, meaning the application is listening on the port for incoming messages until it receives an "End-of-Line" statement. The Java application then reads the merged message sent by LabVIEW, splits on every "X" and converts the substrings to BigInteger variables. These variables are now given to the RAMON decryption algorithm used from [20], which calculates four, still mixed, possible solutions as BigIntegers.

The "Inverse Mix" function is only described as pseudo code in Ref. [20], so it had to be implemented in Java code. Since this function performs bitwise XOR operations on the individual bytes of the BigInteger, the given BigInteger in base two is converted to an integer array, where each integer represents one byte. Care has to be taken because, in this conversion, the individual bytes of the BigInteger could be interpreted as signed integers, which would lead to an incorrect result. After completing the inverse mix function, CHI is parsed from each of the four possible solutions and compared to the known CHI. Thus, the correct plane message x can be selected.

Finally, the correct plain message is sent back to LabVIEW via the TCP/IP connection. Subsequently, the Java application returns to an endless loop while waiting for a new connection to restart the procedure.

4.2. Simulation Results

After testing the RAMON Crypto Suite in LabVIEW combined with the Java application using the same test vectors taken from [20], the correct decryption is confirmed. By implementing time measuring commands before and after the computation in the Java application, it is proven that the calculation by the Java server only takes 1.3 ms. The average total execution time of 1000 calculations in LabVIEW combined with the Java application including data transmission via TCP/IP is 2.2 ms. Compared to the implementation in LABView only, the processing time is now in an acceptable range to perform realistic experiments with this prototyping platform.

These findings are similar to the work of [27] where plain LabVIEW implementations of functions for large number arithmetics are compared to C++ implementations loaded into LabVIEW as shared libraries. They find the external C++ libraries to perform calculations significantly faster.

5. Conclusions

In order to cope with the rapid developments in the UHF RFID field, easy adaptable readers based on software defined radios are indispensable. This paper shows a way to implement the ISO 29167-19 RAMON decryption on a prototyping reader platform based on National Instruments PXIe chassis. The programming of the PXIe is done with LabVIEW. The first implementation in LABView only has shown that the decryption takes 51.3 s. This would mean that only one tag per minute can be processed. Thus, it is not suitable for a prototyping platform intended to perform realistic experiments in the development of new functionalities in UHF RFID systems. Therefore, a different approach is chosen. LabVIEW still serves as a framework and controls the built-in transceivers. However, it also establishes a TCP/IP connection to a Java application. This Java application performs the decryption. The decrypted plain message is then returned to LabVIEW via a TCP/IP connection and can be further processed. The overall time consumed using LabVIEW combined with the Java application is 2.2 ms. This makes the time-optimized approach 23,318 times faster than the basic implementation in LabVIEW only. The main problem in LabVIEW only is the difficulty of dealing with large numbers as they are commonly used in encryption. Solving this problem by outsourcing the computation to

a Java application is not limited to the implementation of the RAMON procedure. The approach can be used for all encryption methods as well as for other procedures handling large numbers in LabVIEW. This way, it is possible to integrate the RAMON encryption into the already existing reader protocol [16] and get a fully functional, easy adaptable reader. It creates completely new possibilities in rapid development especially for UHF RFID systems.

Within IoT, the development of secure batteryless wireless sensor nodes is very interesting. In future studies, RFID transponders that provide encryption will be implemented either in silicon or by using Field Programmable Arrays together with an analog frontend. The presented modular UHF RFID reader then serves as an interface to access such new security features of the tag and test them via the wireless RF link. Additionally, further implementation of security and decryption features by using the same techniques can be performed. Thereby, different approaches for cryptography can be studied and performance in real RFID applications can be investigated.

Author Contributions: Conceptualization, G.S. and M.F. (Manuel Ferdik); Funding acquisition, M.F. (Manuel Ferdik); Investigation, G.S., M.F. (Manuel Ferdik), M.F. (Moritz Fischer) and M.M.; Methodology, G.S. and M.F. (Manuel Ferdik); Project administration, M.F. (Manuel Ferdik); Supervision, G.S. and T.U.; Writing—original draft, G.S.; Writing—review and editing, M.F. (Manuel Ferdik) and T.U.

Funding: This work was supported by the Tyrolean Science Fund (TWF) within the projects 'LoPoCS' (Grant No. UNI-0404-1987).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Rao, K.V.S.; Nikitin, P.V.; Lam, S.F. Antenna design for UHF RFID tags: A review and a practical application. *IEEE Trans. Antennas Propag.* **2005**, *53*, 3870–3876. [CrossRef]
3. Reinisch, H.; Wiessflecker, M.; Gruber, S.; Unterassinger, H.; Hofer, G.; Klamminger, M.; Pribyl, W.; Holweg, G. A Multifrequency Passive Sensing Tag With On-Chip Temperature Sensor and Off-Chip Sensor Interface Using EPC HF and UHF RFID Technology. *IEEE J. Solid-State Circuits* **2011**, *46*, 3075–3088. [CrossRef]
4. Mainetti, L.; Palano, L.; Patrono, L.; Stefanizzi, M.L.; Vergallo, R. Integration of RFID and WSN technologies in a Smart Parking System. In Proceedings of the 2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 17–19 September 2014, pp. 104–110. [CrossRef]
5. Tsiropoulou, E.E.; Baras, J.S.; Papavassiliou, S.; Sinha, S. RFID-based smart parking management system. *Cyber-Physical Syst.* **2017**, *3*, 22–41. [CrossRef]
6. Tan, H.; Gui, Z.; Chung, I. A Secure and Efficient Certificateless Authentication Scheme With Unsupervised Anomaly Detection in VANETs. *IEEE Access* **2018**, *6*, 74260–74276. [CrossRef]
7. He, D.; Ma, M.; Zeadally, S.; Kumar, N.; Liang, K. Certificateless Public Key Authenticated Encryption With Keyword Search for Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3618–3627. [CrossRef]
8. Narayanaswamy, J.; Sampangi, R.V.; Sampalli, S. SCARS: Simplified cryptographic algorithm for RFID systems. In Proceedings of the 2014 IEEE RFID Technology and Applications Conference (RFID-TA 2014), Tampere, Finland, 8–9 September 2014; pp. 32–37. [CrossRef]
9. Benssalah, M.; Djeddou, M.; Drouiche, K. Design and implementation of a new active RFID authentication protocol based on elliptic curve encryption. In Proceedings of the 2016 SAI Computing Conference 2016, London, UK, 13–15 July 2016; pp. 1076–1081. [CrossRef]
10. Hongsongkiat, T.; Chongstitvatana, P. AES implementation for RFID Tags: The hardware and software approaches. In Proceedings of the 2014 International Computer Science and Engineering Conference (ICSEC), Khon Kaen, Thailand, 30 July–1 August 2014; pp. 118–123. [CrossRef]
11. Dang, Y.; Zheng, A.; Zhou, Z.; Yi, Y. Study on UHF RFID Authentication Protocol. In Proceedings of the 5th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, China, 26–27 August 2013; pp. 386–388. [CrossRef]

12. Pendl, C.; Pelnar, M.; Hutter, M. Elliptic Curve Cryptography on the WISP UHF RFID Tag. In *RFID Security and Privacy*; Juels, A., Paar, C., Eds.; Springer: Berlin, Germany, 2012; Volume 7055, *Lecture Notes in Computer Science*, pp. 32–47. [CrossRef]
13. Chae, H.J.; Salajegheh, M.; Yeager, D.J.; Smith, J.R.; Fu, K. Maximalist Cryptography and Computation on the WISP UHF RFID Tag. In *Wirelessly Powered Sensor Networks and Computational RFID*; Smith, J.R., Ed.; Springer: New York, NY, USA, 2013; pp. 175–187. [CrossRef]
14. Hinz, W.; Finkenzeller, K.; Seysen, M. Secure UHF Tags with Strong Cryptography - Development of ISO/IEC 18000-63 Compatible Secure RFID Tags and Presentation of First Results. Available online: https://pdfs.semanticscholar.org/3a3a/72a268915d15353cf5774cc2eaf56cb6339a.pdf?_ga=2.13390897.721935367.1557739034-1389995397.1553694531 (accessed on 13 May 2019).
15. Ferdik, M.; Saxl, G.; Ussmueller, T. Battery-less UHF RFID controlled transistor switch for Internet of Things applications — A feasibility study. In Proceeding of the 2018 IEEE Topical Conference on Wireless Sensors and Sensor Networks, Anaheim, CA, USA, 14–17 January 2018; pp. 96–98. [CrossRef]
16. Ferdik, M.; Hesche, M.S.; Rack, L.; Saxl, G.; Ussmueller, T. NI PXIe based UHF RFID reader. In *Riding the Green Waves*; IEEE: Piscataway, NJ, USA, 2018; pp. 303–306. [CrossRef]
17. Yuechun, W.; Man, K.L.; Maunder, R.G.; Lee, J.K.; Kim, K.K. A flexible software defined radio-based UHF RFID reader based on the USRP and LabView. In *Smart SoC for Intelligent Things*; IEEE: Piscataway, NJ, USA, 2016; pp. 217–218. [CrossRef]
18. Rabin, M.O. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Available online: https://apps.dtic.mil/dtic/tr/fulltext/u2/a078415.pdf (accessed on 13 May 2019).
19. Montgomery, P.L. Modular multiplication without trial division. *Math. Comput.* **1985**, *44*, 519. [CrossRef]
20. International Organization for Standardization. ISO/IEC 29167-1:2014: Information technology - Automatic identification and data capture techniques - Part 1: Security services for RFID air interfaces. Available online: https://www.iso.org/standard/61128.html (accessed on 13 May 2019).
21. Cormen, T.H. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA; McGraw-Hill: New York, NY, USA, 2005.
22. Ding, C.; Pei, D.; Salomaa, A. *Chinese Remainder Theorem*; WORLD SCIENTIFIC: Singapore, 1996. [CrossRef]
23. Kastner, R.; Hosangadi, A.; Fallah, F. *Arithmetic Optimization Techniques for Hardware and Software Design*; Cambridge University Press: Cambridge, UK, 2010.
24. Baruch, Z.F. *Structure of Computer Systems*; Editura U.T. PRESS: Cluj-Napoca, Romania, 2002.
25. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*, rev. reprint with updates, 5. printing ed.; CRC Press Series on Discrete Mathematics and Its Applications; CRC Press: Boca Raton, FL, USA, 2001.
26. Catalano, D.; Cramer, R.; Crescenzo, G.; Darmgård, I.; Pointcheval, D.; Takagi, T. *Contemporary Cryptology*; Advanced Courses in Mathematics—CRM Barcelona, Centre de Recerca Matemàtica; Birkhäuser Verlag: Basel, Spain, 2005. [CrossRef]
27. Bobiński, P.; Winiecki, W. Large Number Library - The new LabVIEW Tool for Secure Measurement Systems. In Proceedings of the 19th IMEKO World Congress, Lisbon, Portugal, 6–11 September 2009; p. 6.