FPGA-based encryption for peer-to-peer industrial network links

Florian Sprang and Tiberiu Seceleanu

Mälardalen University, Västerås, Sweden
{Florian.Sprang, Tiberiu.Seceleanu}@mdu.se

Abstract. Securing company networks has become a critical aspect of modern industrial environments. With the recent rise of Industry 4.0 concepts, it became essential to extend IT security across increasingly connected factories. However, in the highly specialised field of operations technology and embedded systems, not every device can run additional security measures, as they are old or designed with sparse resources. We introduce here the concept of a "universal" encryption device that enables the securing of communication links in a direct peer-to-peer industrial setting by using the AES-128 encryption standard. We propose a design of such an encryption device by developing a modular system architecture with decoupled communication and cryptography. The resulting architecture is implemented as a proof of concept for Ethernet communication and tested through simulation as well as on an FPGA device. The impact of the encryption device is briefly investigated in a lab setup, followed by conclusions on system stability and performance.

Keywords: AES-128, FPGA, encryption, industrial communication

1 Introduction

Cryptography has always been embedded into society. Encryption also brought along the first efforts to gain access to data, leading to approaches to forcefully decrypt messages and gain an advantage over the “adversary". In parallel with the development of newer technologies, more potent ciphers were developed.

In modern manufacturing environments, embedded devices are often used to control, monitor and supervise industrial processes. These enabled factories to become connected, enabling machines to be remotely controlled or monitored. Industrial processes no longer have to be supervised at the machine itself, instead this can be done from anywhere in the world. The current *Industry 4.0* concepts yield many benefits for companies, such as predictive maintenance or increased throughput. Yet from an economic standpoint, it is inefficient to rebuild and redesign all factories from ground up, due to a mix of technologies in production lines, with old control units and machines being connected to the modern manufacturing network. Legacy devices are notoriously known to have security vulnerabilities and thus being a weak point in many systems [12][17].

Modern industrial cyber security standards such as *IEC-62443* or *ISO-27001* call for different measures to protect those devices. One of them is to deploy encryption on important network links. However, these requirements cannot be met by all components, especially when it comes to protocols or legacy devices. The latter might lack the computing capabilities to incorporate new requirements, leaving security vulnerabilities opened to possible attack vectors [17].

This work investigates how communication of legacy devices can be secured in order to fulfill cryptography requirements of standards. We propose to add a signal encryption device, by deploying a field prgrammable gate array (FPGA) integrated circuit (IC), on a connection link, to provide the encryption/decryption of the respective data.

We address the feasibility of a generic approach to secure communication by applying cryptography shortly after the physical network layer. This idea's application is briefly considered for a small part of the Ethernet (IEEE 802.3) [10] standards family. To reach our goal, two ICs are introduced in front of each SCADA[1] device and are responsible for the encryption and decryption of the data link. Fig. 1 depicts a high level of the proposed architecture, with each IC holding the key for data encryption and decryption.

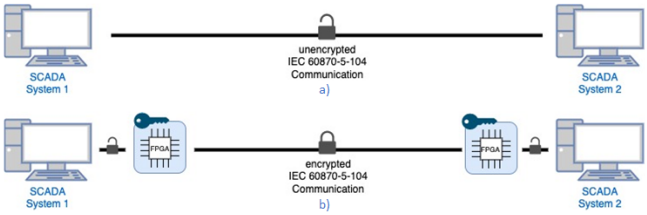

Fig. 1. High level representation of the proposed system architecture

2 Background

The ISO/OSI Model. The official ISO OSI specifications (1994), mostly known as ISO/OSI model - Fig. 2, describe how connections between two or more systems are handled. Typically, this model is used as a representation of internet connections, but it is a good generic representation of how connections work and it can be adapted to various kinds of transmission standards.

Encryption and decryption in the OSI model are handled at higher layers. Custom encryption can be implemented in layers 6 or 7, or existing protocols such as HTTPS or SFTP can be used at the session layer. Layer 3 can also introduce encryption in the IPsec context if communication is handled through standard internet protocols and both endpoints support IPsec. The model can be adapted. This often means layers are left out or combined for simpler transmission protocols, possibly removing capabilities of high-level encryption [2].

[1] Supervisory Control And Data Acquisition

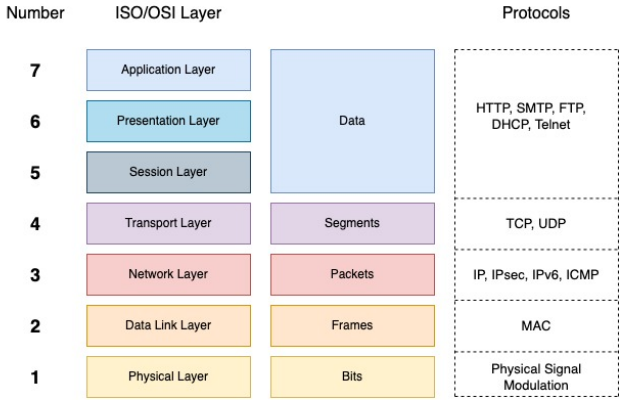

Fig. 2. ISO/OSI Model with associated protocols.

A digital signal between two compatible devices can be transmitted over an unshielded twisted pair (UTP) Ethernet cable, part of the physical layer. The modulation of the signal is handled by the so-called Ethernet Physical Layer (PHY) Chip. It is responsible for demodulating and modulating the signal and builds the bridge between the external electrical signal and a digital circuit. The PHY is further responsible for negotiating the link speed between two Ethernet devices and handling link establishment.

The media access control (MAC) controls the communication, defining the Ethernet frame: *header* - includes the destination and the sending MAC-address and the Ether Type, which either defines the size of the payload or the next layer protocol; *payload* - data; *checksum* - verifies the sent data integrity [1].

Fig. 3 displays a model of a cryptosystem. The sender uses an encryption key to turn his original message/plaintext into an encrypted cipher text. A 3^{rd} party that intercepts or copies the message can read the cipher text but cannot infer the original message, as it does not know the decryption key and the algorithm. The receiver knows those two components and can decipher the encrypted message.

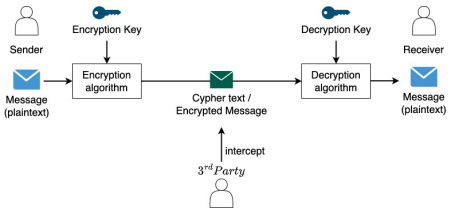

Fig. 3. Model of a cryptosystem

The advanced encryption standard (AES) [15] is a block cypher algorithm following the *symmetric encryption* approach. This means that it takes n-bits, forms them into a block, and encrypts them together. AES uses varying amounts of key lengths, most common being 128bit, 192bit or 256bit, with respective naming conventions of AES-128, AES-192, or AES-256. The algorithm can be

divided into five steps (Fig. 4), to be applied multiple times based on the length of the keys: 10 iterations for AES-128, 12 for AES-192, and 14 for AES-256.

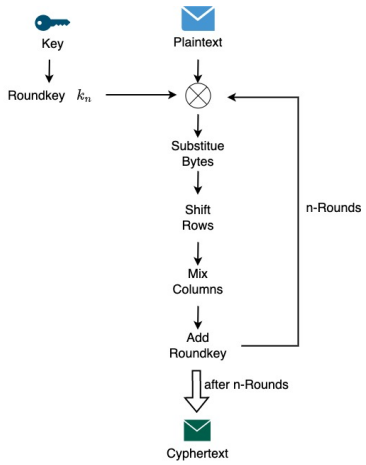

Fig. 4. Steps of the AES-Encryption Algorithm

After arranging the plaintext into the matrix format, a key is determined. The initial key serving as the pre-shared secret is created based on a true random generator that outputs a key in the desired length based on the algorithm. For each round, a different key is created by deriving it from the original pre-shared secret, with a technique called *key expansion*. Depending on the amount of performed rounds, n+1 keys with a length of 128bit are needed. With each key containing four words, a total of $4 * (n + 1)$ words are needed [13] - Fig. 5.

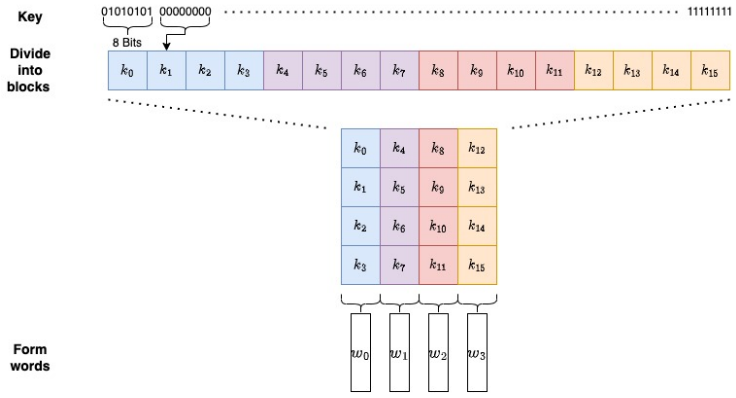

Fig. 5. Key-expansion: original key split into words.

3 Related Work

Currently, the problem of connecting old devices to the existing production network is ideally solved by deploying a firewall to protect them from malicious

traffic. Two firewalls can be configured in a way that resembles the original problem solution: creating an encrypted site-to-site VPN, for sending the SCADA traffic. This works by taking the original protocol and packaging it into a standard TCP/IP package, encrypting it and sending it to the receiver. This task can be performed by small devices, such as a Raspberry Pi. Both firewall and VPN tunneling devices need to understand the original protocol [6]. This shows that securing industrial network connections and links is possible with lower-cost devices, yet encryption is applied on a higher layer. For Layer 2, encryption firewalls are necessary, coming with a premium in cost. Both solutions need to be maintained, patched, and licensed, bringing in additional costs.

Xue et al. are proposing a FPGA combined with a two-core ARM processor solution. Aiming to achieve a throughput of 1 GBit/s, a maximum of 700 Mbit/s was measured. The final design includes an additional chip that acts as a networking card accessing the Ethernet signal [18]. Ideally, the prototype that we propose here would not transfer the Ethernet frames back to a CPU, but only re-creates the digital signal in such a way that it is processable by the FPGA.

The approach of combining an FPGA with an Ethernet controller chip, such as the WIZnet W5500, is often used in designing Ethernet solutions for FPGA [5][16]. Such a design is proposed by Herrmann et al., focusing on UDP traffic [8]. It shows that Gigabit-Ethernet speeds can be achieved, by including RAM on both the transmitter and receiving side to buffer packets. The overall architecture and inclusion of the buffer RAM are relevant to us here. It would solve problems related to AES cypher rotation, as well as to clocking.

There is a relatively rich body of results on implementing AES on FPGAs. Naidu et al. are implementing FPGA by applying a fully pipelined architecture. They find a speed advantage over normal sequential encryption and further concluded that using low energy registers can also yield a lower power consumption [3]. This pipelining approach could be applied by us, as it allows the possibility to encrypt multiple (2) data lines at the same time.

Trang et al. [9] limited their approach to 128-bit length and focused on a low latency design, which was shown to depend on the used FPGA chip. Their longest encryption or decryption duration was done within 25 clock cycles, achieving an overall encryption throughput of 1GBit/s and 615Mbit/s for decryption. The throughput related research is interesting, as it shows that lower-speed Ethernet standards 10Base-T, and 100Base-T are theoretically not limited by the encryption, resulting in no loss of network speed. Unfortunately, they do not provide any insights on the maximum usable clocking frequency.

Harb et al. [7] further show that, in a parallelized application, high speeds can be observed for hardware implementations of AES. In a video stream decoding implementation, they achieved 63 GBit/s, due to optimization techniques. In difference, though, they do not address networked system encryption, where data comes in continuous streams.

Caldas-Calle et. al [4] looked into the QoS, which can be achieved by small embedded devices that use site-to-site VPNs themselves. Applying encryption decreases the computational capacity of these devices, as the load can exceed the

available resources, causing a slowdown of the overall system. Hence, not every device is capable of performing encryption with its provided resources, leading back to the solution of an additional device.

Park et al. [14] showed that in case of a continuous large data stream for video transmission, the deployment of VPNs increases the time needed to transmit the video between 20% of up to 60%. Measurements show that a significant latency is added to transmissions when encrypted.

The MAC security standard (*MACsec*) is defined by the IEEE 802.1AE standard, published in 2008. Despite its age, adoption has been slow with its addition to the Linux Kernel in 2016. The strength of MAC security is its application on all Layer 2 protocols and is independent of other higher-level security mechanisms [11]. At this moment, MACsec is the solution that implements important security mechanisms and uses key exchanges and common cryptography mechanisms. Nevertheless, its slow adoption will mean that legacy devices might not be able to support this standard, especially as two devices have to support it.

4 Implementation

An FPGA is ideal for prototyping and testing IC designs. As the design, construction, and validation of a deployment-ready custom PCB board cannot be performed in the limited available time, then *development boards* or *evaluation kits* are preferred. These contain the FPGA chip integrated into the necessary power circuitry, additional storage and communication interfaces, making them an ideal solution for evaluating the proposed implementation.

4.1 System architecture

The overall idea of the implementation is that neither of the original systems is aware of the additional devices. The theory behind this is that it enables easier adaptability to other protocols in the future. Overall, the proposed architecture imitates the principle of a VPN. However, instead of using the entire TCP\IP stack, it uses only the MAC-Layer to communicate between the two crypto devices. In order to achieve this, the communication side towards the systems has to only implement the retrieval of the data stream of the original bus system. This is done in the PHY-chip, which has to be connected properly to the FPGA. The overall idea of the individual connection parts can be seen in Fig. 6.

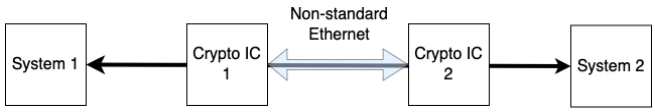

Fig. 6. MAC-tunneling between the crypto devices

The disadvantage of using a standard Ethernet connection between the two crypto devices is that it will add latency to the transmission. As the FPGAs do not come with a preset MAC address, it might be interesting to adapt the

existing MAC core to address agnosticism. This would be a cost-saving measure with little impact on performance, as the area of deployment is directly connected peer to peer links. It is necessary to consider the usage of a non-standardized frame size, as adding an MMAC header might exceed the maximum standard size. Those frames are also called jumbo frames and cannot be routed easily through traditional network switches, hinting towards the direct network link limitation. The design of the crypto implementation, seen in Fig. 7, includes an abstract overview of the elements that must be implemented.

The interface consists of two data connections to the PHY chips, for receiving (RX) and for sending/transceiving (TX). On the open traffic side the FPGA needs to interface with the corresponding protocol PHY-chip but only directly forwards the bit-stream to the AES_encoder. After the message is encrypted, it is wrapped into a standard Ethernet frame and sent out through the other PHY chip. If that chip receives an encrypted message, it will unwrap it by removing the MAC header and then run it through the AES_decoder to retrieve the original signal. This is then forwarded to the PHY-chip to re-modulate the original signal.

The AES_encryptor and AES_decryptor will need the correct key for each round. Further, they will have to perform the key-expansion rounds and counting for the CTR operation mode of AES. The time within the FPGA should be decreased as much as possible to decrease the impact on latency.

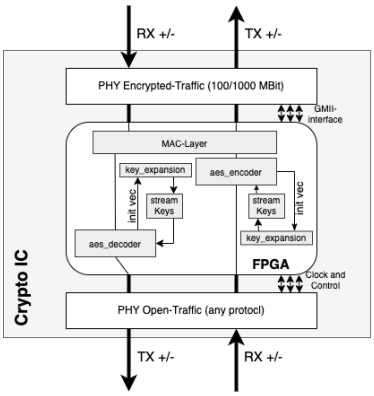

Fig. 7. The crypto IC with the dataflow and the sections in the FPGA.

4.2 System implementation

A limitation of this project is that only a few protocols can be implemented, given the short time frame. Therefore the project will focus on providing an interface for Ethernet only, whose physical layer is utilized by many bus systems. As different physical mediums use different clock speeds it is essential to decouple the encryption from the communication side, providing more flexibility in terms of protocols and usable physical mediums.

For this, an approach is chosen which utilizes three different clocking zones. One for the open physical layer chip communication, one for encryption/decryp-

tion, and the last for the encrypted Ethernet communication. As proposed by the high-level system architecture in Fig. 7, the device can be divided into two independent data streams for de-/encryption. In order to connect the different clocking zones, two FIFO queues are used per data stream.

The encryption data stream consists of a minimal receiving interface and the "chunk generator" that combines the received data to be stored in chunks of 128 bits in the FIFO, alongside status information representing if it is the last chunk in the transmission and how many bits it contains. From there, the encryption controller takes out blocks of 128 bits and performs the encryption, after which it stores the cyphertext, with the status information, in the second FIFO of the data stream. This FIFO builds the bridge between clocking zones 2 and 3. From here, the FIFO is emptied continuously and its content is stored in a block of RAM until the original data frame is complete. This frame is then transmitted through the Ethernet PHY to the other crypto-IC.

The decryption data stream works equivalent to the encryption stream, with the significant difference being that the output interface does not write any protocol-specific data, but only the decrypted data from memory which already includes the necessary control bits, such as the start of frame delimiter, in case of a standard Ethernet transmission.

Fig. 8, shows the previously described concept of the streams. Each stream consists of the four major implemented design blocks, shown on the right of the picture. The actual schematic overview of the pipeline is available in Fig. 9.

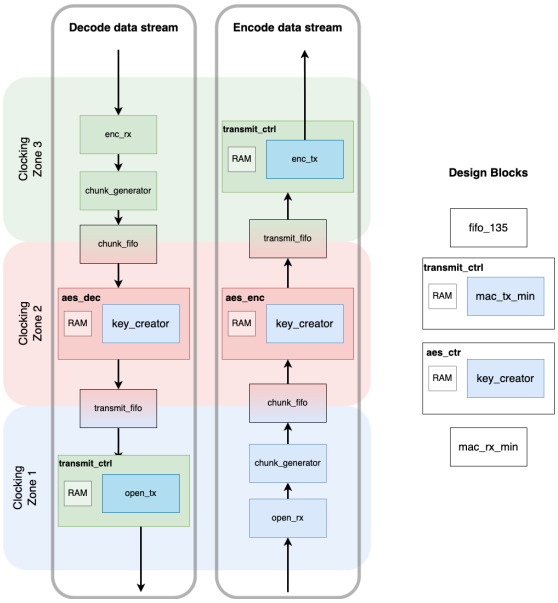

Fig. 8. Data streams combined with clocking zones and actual implemented modules

In the following, each implemented module's pin assignments and programmed logic, will be discussed in detail. In many cases, the module will be implemented

based on state machines. Finally, the actual flow of the program and implementation reasoning will be provided.

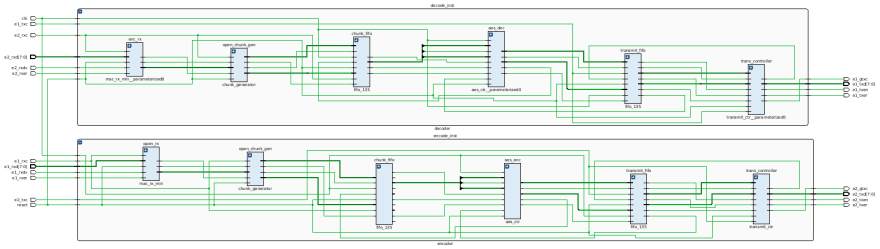

Fig. 9. Schematic drawing of the two data streams in a crypto device

4.3 Hardware implementation

The resulting system was implemented in System Verilog, due to its benefits in data structures and lower complexity compared to VHDL. The used board is a custom communication board - Fig. 10. It provides the required network interfaces. The FPGA-chip is a Spartan 6 (xa6slx45) automotive chip that interfaces with two Ethernet PHYs through the MMII protocol. The Ethernet PHYs are Marvell 88E3018 chips. It further provides a Serial port through USB-A and JTAG to interface with the FPGA.

4.4 Test Methodology

The testing of the system is done in two steps, as follows.
A simulation approach that will test for system functionality. The simulation produces exact timings for signal propagation and models an ideal scenario with no delay within the developed circuit.

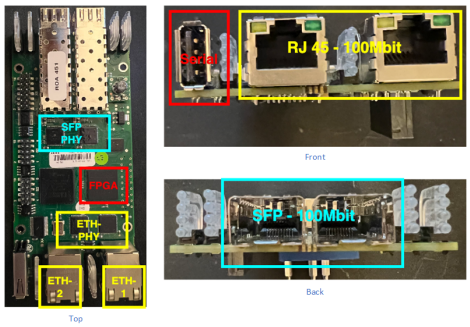

Fig. 10. Communication board.

In order to evaluate the functionality of all developed modules, different test benches have been created, as unit tests for the **Chunk Gen Tester**, the **FIFO 135 Tester**, the **Transceiver Test**, the **Key Creator Tester** and the **AES**

Test. A **Main test** testbench was conceived to test *all* the components together, followed by a **Two device test**, where the **Main test** was extended to simulate communication on both endpoints.

The testbench structure is illustrated by Algorithm 1. It initializes one crypto core, and wires the output of the second Ethernet interface to its input. On the first interface, it injects a payload that resembles a real Ethernet package, consisting of the preamble, header, and payload. After encryption and decryption, the same data is written to the data out array. If the data matches, the simulation will halt, otherwise it will run indefinitely. With a clock of 25 MHz, it takes about $11\mu s$ to complete for one frame.

Algorithm 1: Testbench FPGA simulation (*test_ bench.v*)

```
clk = clk125MHz(); rst = resetSignal(); e1 = Eth.Port(); e2 = Eth.Port();
topMod testSub (.clk(clk), .reset(rst), .e1_rxc(e1.rxc), ..submodules..,
 .e1_txer(e1.txer));                                  //Rewired eth2 to itself

1  dataIN = [h'5555 5555 55d5 abcd effe dcba fffe fdfc fbfa 0800]
2  + [h'656c 6c6f 2057 6f72 6c64 2054 6869 7320 6973 206d 65ff];
3  dataOUT = [];
4  begin
5      e1.rxdv = 1                                          //Start transmission
6      foreach byte in dataIN do
7          @(posedgeclk); e1.rxd = byte[3 : 0];
8          @(posedgeclk); e1.rxd = byte[7 : 4];
9      while e1.txen == 1 do
10         @(posedgeclk); dataOUT = e1.txd;
11         if dataOUT == dataIN then
12             exit();
```

An actual real-world deployment in a test environment. For testing the latency, a commonly used technique is the Internet Control Message Protocol (ICMP) echo, more commonly known as *ping*. An ICMP package containing a sequence number is sent to a device upon which it replies with the same sequence number back. The sender measures the time it takes from sending the package until an answer is returned. The package counts as lost if no reply is returned within two seconds. For establishing a baseline, 400 pings are sent without the encryption device. Later the encryption device is added, and the same amount of pings is repeated.

Fig. 11 shows a picture of the test setup. The Pi connected to Ethernet interface 2 (left) and the MacBook connected to Ethernet interface 1 (right), via USB. Wireshark[2], a networking monitoring utility, and ping run on the laptop, to check and induce network traffic.

A ping command measures the round trip time of a network link. The times are aggregated and stored for evaluation. At 25MHz frequency, the expected

[2] https://www.wireshark.org/

Fig. 11. Hardware test setup

throughput of the cryptography part is around 63 MBit/s. A frequency of 40MHz is also tested, to see if the device can still function correctly. Testing the crypto core is done until 400 valid time samples have been gathered.

5 Evaluation and Results

We first perform a functional evaluation of all major components via simulation, and we follow up with tests run on the actual hardware.

5.1 Simulation

The introduction of different clocking zones is used to evaluate the functionality of the system based on simulation. As the FIFO does not contain important logic and works straightforwardly with data storing and reading, its functionality can be evaluated based on the output and input to the major modules. Further, we decided to go through one whole transmission of the main test bench, rather than looking into each module test.

Simulation Level. The simulated testbench runs a full execution until the output data matches the input. The duration of the entire transmission with full encryption and decryption is $8.88\mu s$. Sending of the open data takes $1.76\mu s$, and the transmission of the encrypted data takes $3.32\mu s$. This increase in time is an additional overhead of 88% to the original time. In total, the actual sending of the data makes up $6.84\mu s$, or 77% of the added latency. Therefore the encryption, decryption, and additional control flows are $2.04\mu s$ long. These simulation timings reflect the sending of a 45 byte message (Fig. 12), where the last byte (ff) is for determining the data-streams end.

Frame part	Preamble	Mac-Header	Data
Data in Hex	5555 5555 5555 55d5	abcd effe dcba fffe fdfc fbfa 0800	4865 6c6c 6f20 576f 726c 6420 5468 6973 2069 7320 6d65 ff
Data in ASCII	-	-	"Hello World This is meÿ"

Fig. 12. Simulation test data.

One can test if the sent frame is correctly stored in the individual chunks: **Chunk 1:** 5555 5555 5555 55$d5$ | $abcd\ effe\ dcba\ fffe$; **Chunk 2:** $fdfc\ fbfa$ | 0800 | 4865 6c6c 6f20 576f 72 6c; **Chunk 3:** 6420 5468 6973 2069 7320 6d65 ff | 00 0000.

Note that in the third stored chunk, the final three bytes are left blank, as they do not correspond to any message. This correlates to the previously established $size\_last\_frame$ value. Thus the message is fully stored in the queue.

Encryption / Decryption. The encryption controller uses these 128 bit chunk data blocks for encryption. For the first message, the keys are not yet available as none has been set yet. The key generator starts generating keys once the new_iv signal is triggered, at approximately $3.5\mu s$. As of the end of the simulation, the write address of the RAM is located at $7a8_{16} = 1960_{10}$. The first generated 119 bit initialization vector is $f056638484d609c0895e8112153524_{16}$ with the static key being $2b7e151628aed2a6abf7158809cf4f3c_{16}$.

Once the controller has the data to encrypt and the keys, it creates the ciphertext blocks for the transmission. The stored header is: $00, 0, 01f0$ $56638484d609$ $c0895e8112153524_{16}$, where the first byte of data $000000\ 01_2$ is the protocol flag for a new initialization vector. After that, the controller remains in an idle state until the first keys are created. Then, a XOR function is applied to the original message block and the key instance. This produces the output in Fig. 13 a), based on the created key instances and plain message blocks.

a) Encryption per input and the resulting ciphertext

Data input	00,0, 5555 5555 5555 55d5 abcd effe dcba fffe	00,0, fdfc fbfa 0800 4865 6c6c 6f20 576f 726c	68,1, 6420 5468 6973 2069 7320 6d65 ff00 0000	...
Key instance	71d7 3c8e 03f0 1117 3dbe 55a1 a9d9 030b	ea63 cede 5428 426b 8a3e 5b14 9b2b 9b6d	a931 7854 1207 31ea 2eb2 cdf8 924b c1fb	...
Output	00,0, 2482 69db 56a5 44c2 9673 ba5f 7563 fcf5	00,0, 179f 3524 5c28 0a0e e652 3434 cc44 e901	68,1, cd11 2c3c 7b74 1183 5d92 a09d 6d4b c1fb	...

b) Decryption per input and the resulting message

Data input	00,0, 01f0 5663 8484 d609 c089 5e81 1215 3524	00,0, 2482 69db 56a5 44c2 9673 ba5f 7563 fcf5	00,0, 179f 3524 5c28 0a0e e652 3434 cc44 e901	70,1, cd11 2c3c 7b74 1183 5d92 a09d 6d4b c1fb
Key instance	-	71d7 3c8e 03f0 1117 3dbe 55a1 a9d9 030b	ea63 cede 5428 426b 8a3e 5b14 9b2b 9b6d	a931 7854 1207 31ea 2eb2 cdf8 924b c1fb
Output	00,0, 5555 5555 5555 55d5 abcd effe dcba fffe	00,0, fdfc fbfa 0800 4865 6c6c 6f20 576f 726c	70,1, 6420 5468 6973 2069 7320 6d65 ff4b c1fb	...

Fig. 13. a) Encryption per input and the resulting ciphertext; b) Decryption per input and the resulting message.

The input and output blocks are combined with the status flags of size and last frame, denoted by the comma-separated values at the beginning. After the plaintext is converted to the cyphertext, it is stored for transmission in the next FIFO-queue. The basic flow is the same when considering the decryption mode. It only differs upon reading the first data block from the queue, which is taken out right away, as it represents the protocol header. The decryption of each read data block, with the keys read from RAM are listed in Fig. 13 b).

Looking at the encrypted data output, it can be seen that it is equivalent to the data that is now inserted into the decryption. The read keys from the RAM are equivalent to the keys used for encryption, showing that the key generators create the same keys in both modules. The final output data to the transmit queue of the decryptor is also equivalent to the previously encrypted data blocks.

The transceiver. The transmit controller contains the control logic, the IP-RAM implementation, and the transmitter itself. The transmitter and the RAM are embedded within the controller. The controller is connected to the transmit FIFO. Therefore it needs inputs to the data, the status flag if a frame is complete, and the size of the last chunk. It also uses the status flags of the FIFO.

The block memory is able to store 2048 bytes, larger than the maximum transmission size of Ethernet. For this, it uses the state machine displayed in Fig. 14. The controller takes every data chunk from the FIFO until the last frame is detected. Each data frame can be fully stored in RAM. Upon detecting the last frame, it enables the transmitter to send out the full data frame.

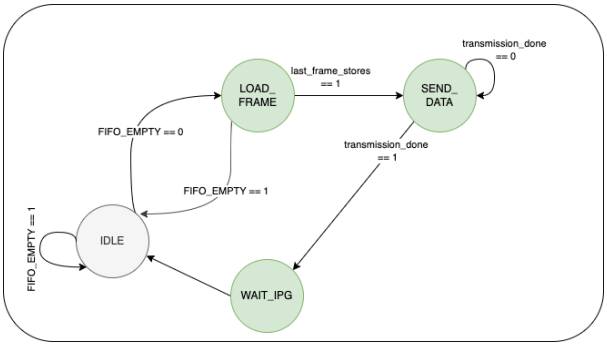

Fig. 14. State machine of the transmit controller

Upon a negative edge of a signal (*fifo_ empty*), the controller changes from IDLE to LOAD_FRAME (state machine of Fig. 14), where it starts to write the data blocks of 128 bits to the RAM. It changes back and forth between these two states until another signal (*is_ last_ frame*) is triggered. After storing the last data block, the controller transitions to the SEND_DATA state, where the transmitter starts to work. Upon the transmitter finishing, the WAIT_IPG state is entered for waiting for the defined inter-packet gap and resetting all variables. After the waiting period, the controller moves back to IDLE and waits for the next frame. The transmit controller works for both operation modes the same.

Expected signal impact. It is possible to infer signal delays based on the simulation time from the first signal received to the last byte sent. A delay of approximately $8\ \mu s + 2 * modulation\_time_{encrypted} + 2 * modulation\_time_{plaintext}$ is added for the pipelined implementation. Looking at the slower low area implementation, the delay is significantly larger, with $\approx 242\mu s + 2 * modulation\_time_{encrypted} + 2 * modulation\_time_{plaintext}$. Both values represent a worst case, where no keys are stored in the key ram yet. Therefore, the modulation time has to be added and can be calculated based on the following formula for the encrypted packet and Fast Ethernet (100 MBit/s):

$$modulation\_time_{encrypted} = \frac{8 * (23 + plaintextLength)}{25 * 10^6 Bits/s}$$

For the plaintext packet, the used length shortens by the added 23 bytes. These 23 bytes represent the additional preamble, mac-header, size, and encryption header. If keys are already available, the logic's added base delay for the low-area implementation equates to $127\mu s$ instead of the $242\mu s$.

It takes $\approx 20.4\mu s$ for a key to be generated in the low area implementation. Therefore a theoretical maximum throughput of $\frac{128bit}{20.4\mu s} = 62.74 MBit/s$ is pos-

sible if the key generator is clocked at 25MHz. For a 40 MHz clock this time is reduced to $\approx 12.6\mu s$, equating to a theoretical throughput of 101 MBit/s.

5.2 Hardware

Synthesis Results. The exact result of the synthesis is shown in Fig. 15, where the top result shows the non-fitting pipelined AES-core and the bottom one is the low area implementation for a single crypto device. With a more modern chip selection, a possible placement was reachable. Yet the amount of used LUTs, for the pipeline core, is significantly higher than the documentation-estimated 4000.

only AES-core — Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	5536	54576	10%
Number of Slice LUTs	10832	27288	39%
Number of fully used LUT-FF pairs	3535	12833	27%
Number of bonded IOBs	385	316	121%
Number of Block RAM/FIFO	29	116	25%
Number of BUFG/BUFGCTRLs	1	16	6%

a) Pipelined AES-Crypto Core (not mapable)

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	6224	54576	11%
Number of Slice LUTs	6317	27288	23%
Number of fully used LUT-FF pairs	4916	7625	64%
Number of bonded IOBs	28	316	8%
Number of Block RAM/FIFO	45	116	38%
Number of BUFG/BUFGCTRLs	5	16	31%

b) Synthesis results for hardware test

Fig. 15. Synthesis results.

Performance test. For performance evaluation, we develop an extensive testing routine and establish a performance baseline. For the link latency, we conducted 400 individual pings at different times, while no load is applied to either link, with a median latency of 700 μs (Fig. 16). The highest measured latency is at $982\mu s$, and the quickest transmission was $302\mu s$. These values represent the round trip time between the devices, expecting half of them for one-way communication.

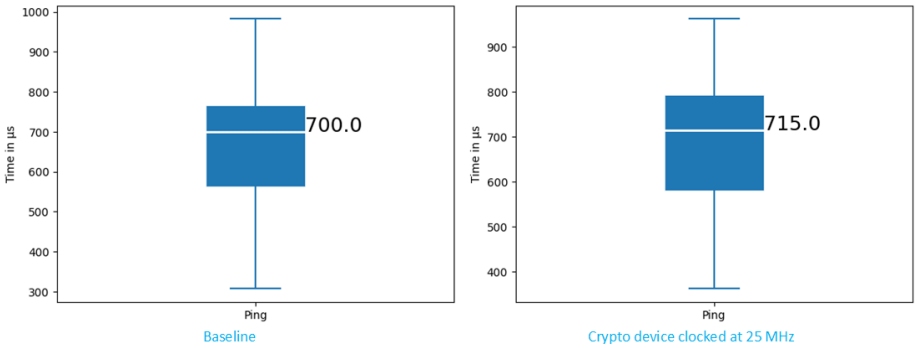

Fig. 16. Ping comparison in μs

The same approach with the crypto device clocked at 25 Mhz, resulted in a median round trip time of $715\mu s$. The highest measured round trip time is 1.97s,

compared to the $362\mu s$ of the lowest time. For the 40 MHz test, no data could be gathered. The increase of the clock speed seems to violate timing constraints within the process resulting in failure of correct decryption. Data is still received by the Pi but is omitted by the MAC layer.

The package loss due to instabilities is significant. In order to get 400 data points for the 25Mhz encryption, 2800 ICMP echo requests were sent, with 475 succeeded and another 38 slower than the required 2 seconds. This results in a package loss of 83%. In comparison, the median round trip time was increased by $100\mu s$, yet the overall spread increased. Also, more extreme highs were observed.

When running a quick throughput test, the measured transmission instance resulted in a measured throughput of 217 kbit/s.

6 Discussion

The overall functionality of the architecture was tested within a controlled simulation environment, in which the transfer of a fixed amount of Ethernet frames was successfully conducted. The simulation proved that both the synchronization as well as en-/de-cryption function according to the specifications. Using a stream cypher approach and the preparation of keys increases the system throughput and should theoretically support high bandwidth protocols such as Gigabit Ethernet. This could not be tested in a real-world scenario.

A downside of the conducted simulation is that it only looked into one-way communication and did not take simultaneous send and receive into account. Due to the separation of the encryption and decryption pipeline, the behaviour should be explored in additional simulations.

Depending on the choice of the FPGA, either a pipeline or a slower single-key generation approach can be chosen. The pipeline approach trades FPGA resources for speed and should be used for high throughput protocols. The low area single key implementation will significantly reduce the used resources, leaving more space for additional communication interfaces or other custom logic.

We also tested the impact of a crypto device on an actual network link, following a developed testing routine. The routine was first used without the cryptography device to establish a baseline. Given the amount of processed data, the used protocol proved highly stable and robust, with almost no re-transmissions.

The conducted tests provide a proof of concept. It is possible to encrypt a link's data independent of the protocol and maintain the original communication between the devices. This comes at the cost of latency, yet not a significant one. A more elaborate testing on the latency impact has to be pursued to get more definitive results, though.

In order to satisfy modern interfaces such as Gigabit Ethernet, a frequency of 25MHz is insufficient. The performance should be increased, and the testing devices should be configured not to insert any additional traffic data.

Security considerations. A lot has yet to be determined, when it comes to security. The devices themselves were not tested for implementation vulnerabili-

ties. Due to the large volume of conducted work, errors and oversights are easily made that can lead to problematic outcomes.

One finding that came up, though, was that a potential leak of the key can be achieved by observing the data stream, and gaining the initialization vector while sending prepared payloads of continuous zeros. With a brute-force approach, it might be possible to identify the key. Another downside is that the message length can be inferred. This is especially important in industrial applications because messages and control signals are repeated multiple times. Depending on the protocol and deployment architecture, an attacker could potentially infer what message has been sent based on length.

Lastly, the remaining unencrypted link pieces have to also be secured physically. However, this is easier done for small sections of a long cable than physically securing the whole link.

7 Conclusions

We proposed here an architecture that takes extendability and flexibility into consideration, based on an FPGA design, for evaluation purposes. The architecture separates the communication from the actual cryptography part, by introducing FIFO queues to connect the communication and crypto core. We introduce a custom protocol header to synchronize the devices to the same initialization vector for key generation. We use state machines, resulting in simplified logic flows, increased readability, and code maintainability. In addition, dividing the architecture into different zones provides enhanced adaptability to multiple protocols, proven by the implementation of GMII and MII.

The implementation shows that encrypting any traffic on the network link and regaining the original signal can be a viable alternative to existing technologies. The result is a proof of concept to be seen as a complementary technology in case of non-accessibility to higher-layer solutions, with increased stability. It further indicates that a universal cryptography device interfacing with different physical layer chips can solve encryption requirements in industrial links.

Future work. We present here a list of possible actions that will improve the quality and utility of the presented efforts.

• The approach should be scaled to cover different protocols (such as CAN) to enable the greater scheme of the protocol-independent encryption.

• The current design deals with direct network links. It is possible to extend the implementation to a one-hop distributed network by using the MAC addresses, assuming that the network is able to support larger frame sizes.

• Removing the buffering of the full frame in the transceivers and the FIFO queues. This alternative would only use the stream cypher approach, where the encryptor would no longer encrypt 128-bit blocks but rather apply a portion of the key on either the 4 bits of RGMII or the 8 bits of the GMII, to be sent out without waiting to rebuild a whole message. This may lead to time savings, but communication between the cryptography devices has to be solved differently, while also losing modularity.

References

1. IEEE Standard for Ethernet. Tech. rep., IEEE. https://doi.org/10.1109/IEEESTD.2022.9844436, iSBN: 9781504487252
2. OSI model (Jan 2023), https://en.wikipedia.org/w/index.php?title=OSI_model&oldid=1134681638, page Version ID: 1134681638
3. Anusha Naidu, A.P., Joshi, P.K.: FPGA implementation of fully pipelined advanced encryption standard. In: 2015 international conference on communications and signal processing (ICCSP). pp. 0649–0653 (2015). https://doi.org/10.1109/ICCSP.2015.7322568
4. Caldas-Calle, L., Jara, J., Huerta, M., Gallegos, P.: QoS evaluation of VPN in a Raspberry Pi devices over wireless network. In: 2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS). pp. 125–128 (Jun 2017). https://doi.org/10.1109/ICCDCS.2017.7959718, iSSN: 2165-3550
5. Choudhary, A., Porwal, D., Parmar, A.: FPGA Based Solution For Ethernet Controller As Alternative For TCP/UDP Software Stack. In: 2018 6th Edition of International Conference on Wireless Networks & Embedded Systems (WECON). pp. 63–66 (Nov 2018). https://doi.org/10.1109/WECON.2018.8782050
6. Fattahi, A.: IoT product design and development: best practices for industrial, consumer, and business applications. Wiley, Hoboken, NJ (2022)
7. Harb, S., Ahmad, M.O., Swamy, M.N.S.: A High-Speed FPGA Implementation of AES for Large Scale Embedded Systems and its Applications. In: 2022 13th International Conference on Information and Communication Systems (ICICS). pp. 59–64 (Jun 2022). https://doi.org/10.1109/ICICS55353.2022.9811140, iSSN: 2573-3346
8. Herrmann, F.L., Perin, G., de Freitas, J.P.J., Bertagnolli, R., dos Santos Martins, J.B.: A Gigabit UDP/IP network stack in FPGA. In: 2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009). pp. 836–839 (Dec 2009). https://doi.org/10.1109/ICECS.2009.5410757
9. Hoang, T., Nguyen, V.L.: An efficient FPGA implementation of the advanced encryption standard algorithm. In: 2012 IEEE RIVF international conference on computing & communication technologies, research, innovation, and vision for the future. pp. 1–4 (2012). https://doi.org/10.1109/rivf.2012.6169845
10. IEEE: IEEE 802.3 ETHERNET, https://www.ieee802.org/3/
11. Luber, S.: Was ist MACsec? (Jun 2022), https://www.security-insider.de/was-ist-macsec-a-e945e21bc26faeed7999ee600aa61d78/
12. National Cyber Security Centre: Obsolete products, https://www.ncsc.gov.uk/collection/device-security-guidance/managing-deployed-devices/obsolete-products
13. National Technical Information Service (NTIS): FIPS 197, Advanced Encryption Standard (AES). FIPS 197 (Nov 2001)
14. Park, S., Matthews, B., D'Amours, D., McIver Jr., W.J.: Characterizing the Impacts of VPN Security Models on Streaming Video. In: 2010 8th Annual Communication Networks and Services Research Conference. pp. 152–159 (May 2010). https://doi.org/10.1109/CNSR.2010.60
15. Prof. Dr. Hanno Lefmann: AES - Advanced Encryption Standard (Rijndael) (2005), https://www.tu-chemnitz.de/informatik/ThIS/vlzits/aes.html
16. Shi, Y., Jin, C., Gao, F.: The solution of ethernet based on hardware protocol stack W5300 and FPGA. In: Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology. vol. 3, pp. 1328–1331 (Aug 2011). https://doi.org/10.1109/EMEIT.2011.6023339

17. Stouffer, K., Pease, M., Tang, C., Zimmerman, T., Pillitteri, V., Lightman, S.: Guide to Operational Technology (OT) Security: Initial Public Draft. preprint (Apr 2022). https://doi.org/10.6028/NIST.SP.800-82r3.ipd, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.ipd.pdf
18. Xue, T., Pan, W., Gong, G., Zeng, M., Gong, H., Li, J.: Design of Giga bit Ethernet readout module based on ZYNQ for HPGe. In: 2014 19th IEEE-NPSS Real Time Conference. pp. 1–4 (May 2014). https://doi.org/10.1109/RTC.2014.7097556