6.111 Final Project Proposal: Encrypted Communications over Ethernet

Hyo Won Kim, Mark Theng

2018

1 Overview

Today's world is defined by constant surveillance, by the government and by the various companies that seek access to our data. Furthermore, methods for intercepting communications are easily accessible to any motivated entity through exploit kits commonly sold on the darknet. We seek to change this, using an AES-encrypted Ethernet scheme.

Our system will securely transmit messages from one FPGA to another. While the type of message transmitted is arbitrary, we will demonstrate this through images: An image would be downloaded into one FPGA via the USB UART interface, encrypted, encoded into Ethernet packets, transferred over an Ethernet cable to another FPGA, decrypted, and finally displayed on a VGA monitor.

Ethernet is the most common physical layer networking standard for Internet communications, so using Ethernet as the base communication medium for our cryptosystem would allow our system to communicate with a large range of devices. The Nexys 4 DDR comes with an Ethernet port behind a SMSC 10/100 LAN8720A Ethernet PHY, which exposes up to 100Mpbs Full Duplex Ethernet over an RMII interface. In particular, we can interface with Ethernet without worrying about the analog intricacies of digital communication. This ensures that information would be transported in a reliable and noise-resistant fashion.

AES (Advanced Encryption Standard) is a secure, time-tested symmetric key encryption algorithm widely used in modern digital systems. Our project will use an insecure variant of AES – AES-128 in ECB mode without authentication – but could be extended to more secure algorithms.

2 Goals

2.1 Baseline

- One round of AES-128 algorithm working in testbench, for both encryption and decryption.
- Receive messages from a laptop over USB UART
- Transmit messages to a laptop over Ethernet

2.2 Expected

- Encrypt and decrypt 64x64 color images in AES-128 ECB mode
- Send ciphertexts from one FPGA to another over an Ethernet cable
- Display decrypted images on a VGA monitor

2.3 Stretch

- Encrypt and decrypt plaintexts in AES-128 CBC mode
- Securely negotiate encryption keys using a Diffie-Hellman algorithm or similar
- Transmit 800x600 color images and display them from a buffer in DDR2 RAM
- Make the system resilient to dropped or out-of-order packets, which would be important in AES CBC-mode encryption

3 Required Materials

- An Ethernet cable

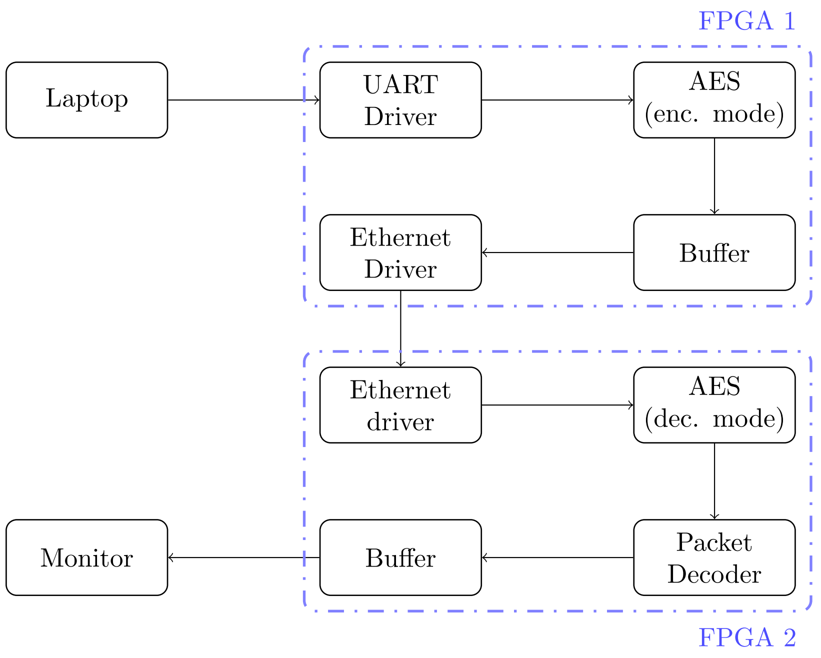
4 External Interfaces

Our project will only need to interface with components on the Nexys 4 DDR board.

- FTDI FT2232HQ USB-UART bridge (RS232 interface, max 12Mbps for USB 2.0 Full Speed)
- SMSC 10/100 LAN8720A Ethernet PHY (RMII interface, max 100Mbps Full Duplex)
- VGA monitor (800x600, 72Hz mode)

5 Block Diagrams

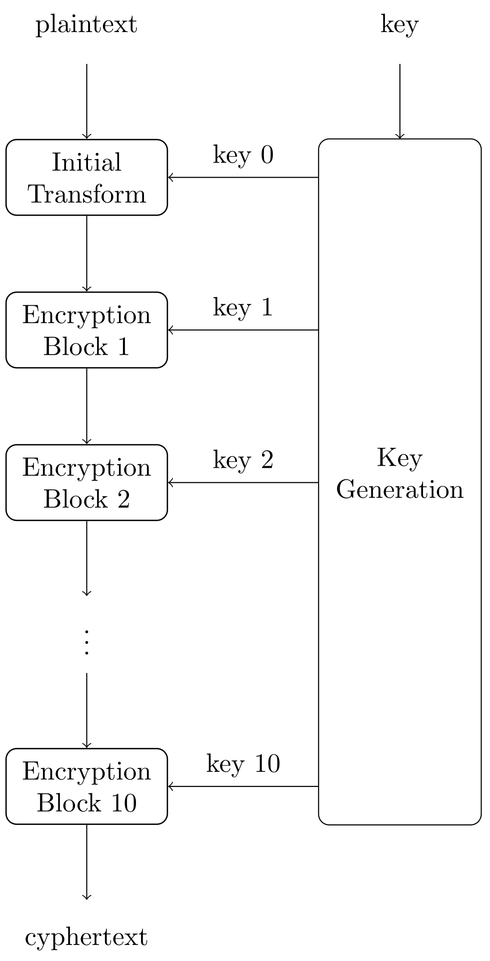
5.1 Flow of Information



Not shown in this diagram are the controller modules that coordinate data flow and respond to errors.

5.2 AES Cryptosystem

The AES encryption system uses a chain of 10 rounds of a single encryption block, which looks like this.



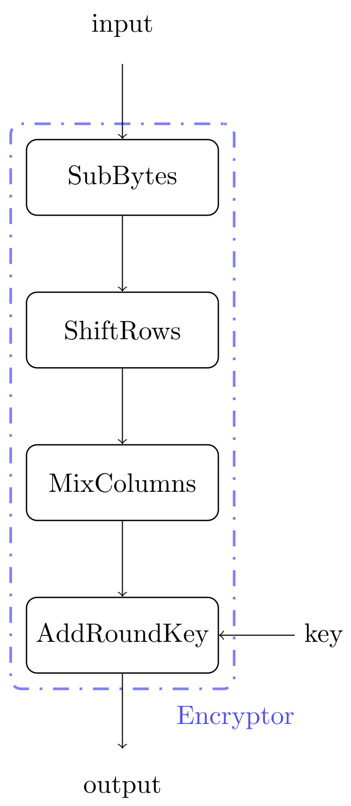
The decryptor works similarly, with the encryption blocks replaced with decryption blocks, and the flow of information reversed. Each submodule in the decryptor block is equivalent to a submodule in the encryption block, with different parameters selected.

6 Modules

All modules implicitly take in a 50MHz clock and reset input.

6.1 AES Encryption/Decryption (Ashley)

A single encryptor module is composed of several submodules, as follows.



Each encryptor module in the chain looks like the above, with the exception of the last round, which skips the mixcolumns step.

6.1.1 SubBytes

Using a lookup table, each of the 16 bytes of the input is replaced with a substitution byte from a pre-programmed lookup table (a 16x16 grid of 1-byte values). This requires storing 2kbits of memory for both both encryption and decryption. We have enough BRAM to support storing this table, but if any unexpected roadblocks occur, we can compute the substitution using an affine transformation.

6.1.2 ShiftRows

We separate the block of 16 bytes that we got as the input into four rows of four bytes, and cyclically shift each column to the right by its index.

6.1.3 MixColumns

For each column, we left multiply by the matrix

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

to derive a linear transformation. Then, we recombine the columns and reassemble the 128 bits in row-major order, taking the first four bytes of the result to be the first byte of each of the four columns, and analogously for the next twelve bytes.

6.1.4 AddRoundKey

Using the key that was derived using our key generation module, we xor the input with the key we received, and output that result.

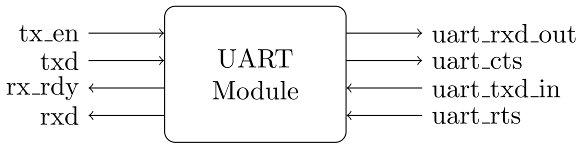
6.2 AES Round Key Generation (Ashley)

Using the initial 16-byte key, we derive the key using a recursive algorithm We split the initial key into four 32-bit blocks. Then, we derive an expanded key, which ends up being 11 128-bit keys concatenated together, as follows.

We define $W_0, ..., W_{4R-1}$ as the 32-bit blocks that make up the expanded key. Then, we utilize two submodules, $RotWord$, which cyclically shifts the input left by a byte, and $SubBytes$, which was a submodule used in AES for substitution. Each block W_i can be derives as follows.

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{RotWord}(\text{SubWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N,\ N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

6.3 UART (Mark)

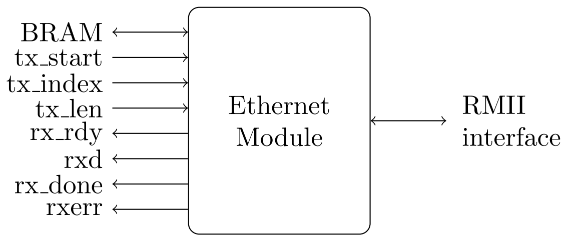


The UART module will communicate with a laptop to receive plaintext images and dump debug information. This module will (at least) operate at 115200 baud. Since this is significantly slower than Ethernet, a BRAM buffer will be required somewhere in the pipeline to collect the data before initiating Ethernet frame construction. We arbitrarily choose to place this buffer after the

AES encryption step. Note that this is necessary even at the maximum baud rate of 12Mbps, since Ethernet operates at 100Mbps.

The control signals tx_en (transmit enable) and rx_rdy (data ready, asserted for a single clock cycle when a byte is ready on rxd) allow information flow to be synchronized with the downstream module.

6.4 Ethernet (Mark)



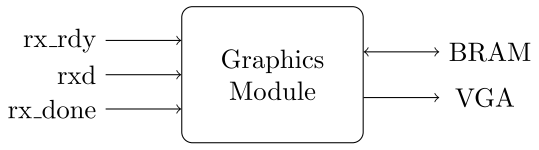
The Ethernet module communicates with the on-board Ethernet PHY to transmit and receive data through the Ethernet port. On the physical layer, it will need to take care of initializing and configuring the Ethernet PHY chip and decoding the CRSDV signal.

When transmitting, the Ethernet module will take care of adding the necessary frame headers, appending the CRC checksum, and streaming the bytestream as dibits through the RMII interface. It will also include the offset and length of the image data being transmitted in the Ethernet payload. Since this module will be reading the payload from RAM, it can process the bytestream at its own pace and there is no need for advanced synchronization.

When receiving, the raw dibit stream will be fed into a state machine that strips the preamble and checks the CRC. The MAC address will be checked and the dibit stream will be converted into a bytestream for downstream processing. If any errors are encountered (such as a de-assertion of DV or a wrong CRC), rxerr is asserted for one clock cycle. Otherwise, at the end of a packet, rx_done is asserted for one clock cycle.

Initially, errors will be ignored since false carriers will be detected at the very least by the MAC filtering stage. This would obviate the need for an additional packet buffer before packet decoding. However, this means that our system will need to be robust to incomplete packets.

6.5 Graphics (Mark)



The graphics module receives packet data essentially the same as was transmitted from the laptop over UART, except divided into packets that include the offset and length of the data. This module will parse the packets and copy the data to the required locations in BRAM. If rx_done is asserted earlier than expected, it aborts the operation. Writing an image to a buffer in this manner allows our system to be resilient to dropped or incomplete packets.

At the same time, this module will generate the required VGA signals, and obtain pixel colors from the BRAM buffer. The buffer is necessary because the image has to persist even when no data is being transferred. VGA will tentatively be driven at 800x600 72Hz since the required 50MHz clock period would be the same as the one already being used for the rest of the system, so we do not need to deal with clock boundaries.

6.6 Coordinator (Mark)

The coordinator module communicates with all the modules described to orchestrate the entire pipeline. It takes care of miscellaneous tasks such as dividing the bytestream into Ethernet-sized packets at the transmitting end, and propagating errors at the receiving end. No data passes through this module – even when dividing the bytestream into packets, this module will only need to count bytes, since the data is buffered before reaching the Ethernet driver.

7 Projected Challenges

This project requires interfacing with a new component – the Ethernet PHY. Unexpected problems may arise trying to get it to work, especially between two FPGAs. Additionally, while the overall system is designed to be robust to network problems, unforeseen constraints may arise that could require the design of additional modules like flow control.

8 Timeline

- Week of Oct 29:
  - AES encryption/decryption blocks
  - Receive Ethernet packets from laptop over PHY
- Week of Nov 5:
  - AES encryption/decryption blocks, test bench
  - Send Ethernet packets from laptop over PHY
- Week of Nov 12:
  - AES chain encryption/decryption, get working ECB
  - Send data from one FPGA to another
- Week of Nov 19:

  - AES chain encryption/decryption, get working ECB
  - Display images from BRAM over VGA

- Week of Nov 26:

  - Integrate Ethernet and AES cryptosystem

- Week of Dec 3/Dec 10:

  - CBC mode and/or key exchange
  - Stretch goals and buffer