FPGA Bitstream Encryption

Ronald Gigerl

November 30, 2022

Introduction

What is the bitstream?

- Is the configuration for an FPGA

What is the bitstream?

- Is the configuration for an FPGA
- Can be seen as "binary" for the hardware

What is the bitstream?

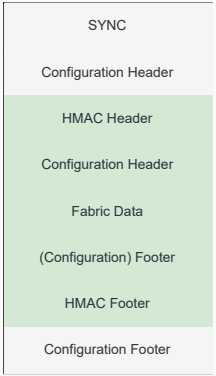

Figure 1: The structure of the bitstream (green rows are encrypted) [1] [2]

Why do we need the encryption?

- Prevents reading and reverse engineering (Confidentiality)

Why do we need the encryption?

- Prevents reading and reverse engineering (Confidentiality)
- Prevents manipulation of the design (Authenticity and integrity)

Why do we need the encryption?

- Prevents reading and reverse engineering (Confidentiality)
- Prevents manipulation of the design (Authenticity and integrity)
- Prevents hardware Trojans

Encryption on Xilinx 7 Series

- Board features a AES decryption logic [3]

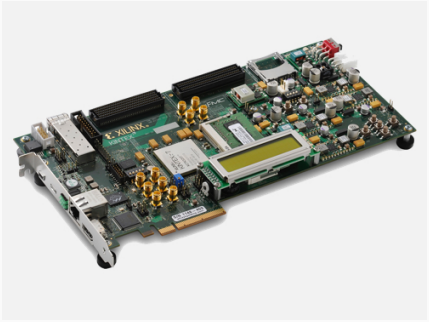

Figure 2: Xilinx Kintex-7 FPGA [5]

Encryption on Xilinx 7 Series

- Board features a AES decryption logic [3]
- Decryption logic can only be used for decrypting the bitstream

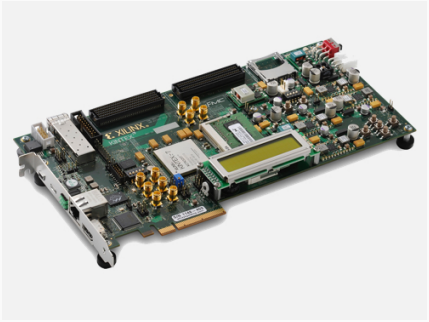

Figure 2: Xilinx Kintex-7 FPGA [5]

Encryption on Xilinx 7 Series

- Board features a AES decryption logic [3]
- Decryption logic can only be used for decrypting the bitstream
- Key can be set via JTAG

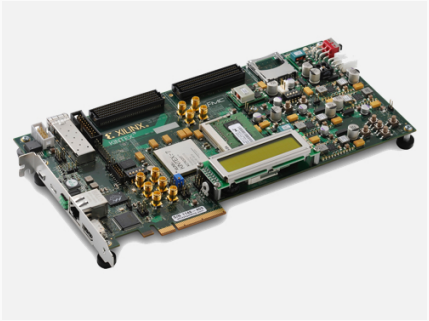

Figure 2: Xilinx Kintex-7 FPGA [5]

Encryption on Xilinx 7 Series

- Board features a AES decryption logic [3]
- Decryption logic can only be used for decrypting the bitstream
- Key can be set via JTAG
- Board uses a SHA-256 Hash Message Authentication Code (HMAC) [4] for verifying authenticity

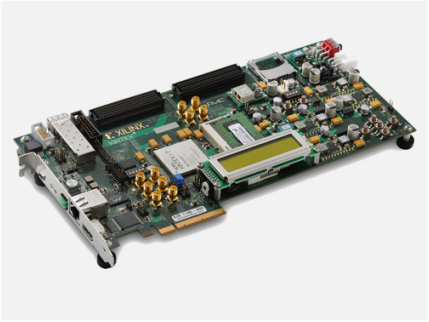

Figure 2: Xilinx Kintex-7 FPGA [5]

The encryption process

Overview

1. Selecting the key storage

Overview

1. Selecting the key storage
2. Generate bitstream and encrypt

Overview

1. Selecting the key storage
2. Generate bitstream and encrypt
3. Decrypting and interpreting the bitstream

Key storage options

- Battery backed RAM - BBRAM

Key storage options

- Battery backed RAM - BBRAM
- eFuse

Battery Backed RAM - BBRAM

- Volatile storage

Battery Backed RAM - BBRAM

- Volatile storage
- Needs continuous power to keep data

Battery Backed RAM - BBRAM

- Volatile storage
- Needs continuous power to keep data
- Key can cleared / changed

eFuse

- Nonvolatile one-time-programmable technology

eFuse

- Nonvolatile one-time-programmable technology
- Once programmed it cannot be changed anymore

eFuse

- Nonvolatile one-time-programmable technology
- Once programmed it cannot be changed anymore
- No battery required

eFuse

- Nonvolatile one-time-programmable technology
- Once programmed it cannot be changed anymore
- No battery required
- Key cannot be cleared

Setting the AES key (via JTAG)

1. Board enters special key-access mode

Setting the AES key (via JTAG)

1. Board enters special key-access mode
2. All memory inclusive key configuration gets cleared

Setting the AES key (via JTAG)

1. Board enters special key-access mode
2. All memory inclusive key configuration gets cleared
3. Then the key can be set

Setting the AES key (via JTAG)

1. Board enters special key-access mode
2. All memory inclusive key configuration gets cleared
3. Then the key can be set
4. Board exits the mode

Setting the AES key (via JTAG)

1. Board enters special key-access mode
2. All memory inclusive key configuration gets cleared
3. Then the key can be set
4. Board exits the mode
5. Key cannot be read anymore

Encrypting the bitstream

- Xilinx uses AES Cipher Block Chaining mode (CBC)

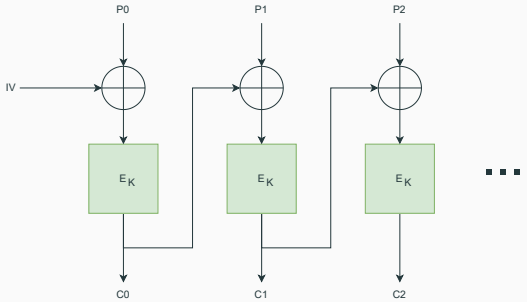

Figure 3: CBC encryption

Encrypting the bitstream

- Xilinx uses AES Cipher Block Chaining mode (CBC)
- The encryption can be done by the Vivado bitstream generator (`write_bitstream`)

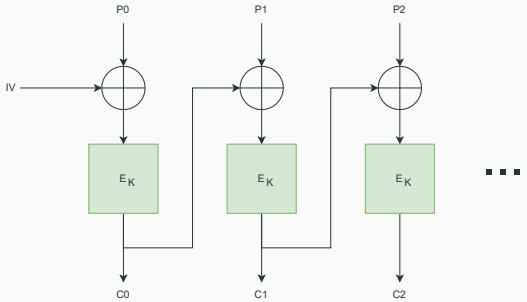

Figure 3: CBC encryption

Encrypting the bitstream

- Xilinx uses AES Cipher Block Chaining mode (CBC)
- The encryption can be done by the Vivado bitstream generator (`write_bitstream`)
- Key storage, key and HMAC key need to be configured in the constrains file

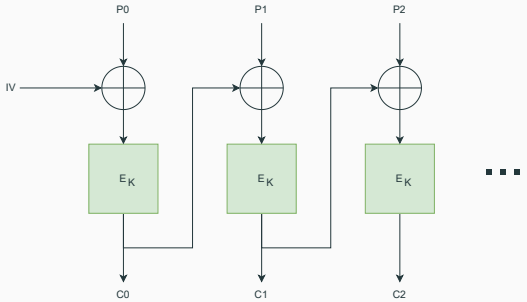

Figure 3: CBC encryption

Encrypting the bitstream

- Xilinx uses AES Cipher Block Chaining mode (CBC)
- The encryption can be done by the Vivado bitstream generator (`write_bitstream`)
- Key storage, key and HMAC key need to be configured in the constrains file
- Generated bitstream will be encrypted and written to .bit file

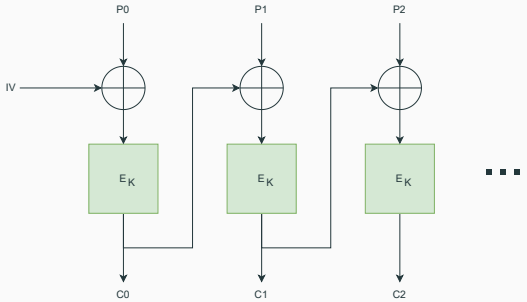

Figure 3: CBC encryption

Executing the encrypted bitstream

- Board uses a special AES decryption logic

Executing the encrypted bitstream

- Board uses a special AES decryption logic
- Debug output and readout is disabled

Executing the encrypted bitstream

- Board uses a special AES decryption logic
- Debug output and readout is disabled
- Bitstream gets authenticated

Authentication of the bitstream

- Encrypted bitstream can still be manipulated and therefore authentication is needed

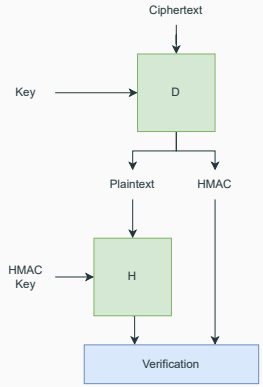

Figure 4: Verification of MAC-then-encrypt

Authentication of the bitstream

- Encrypted bitstream can still be manipulated and therefore authentication is needed
- Xilinx boards use a MAC-then-encrypt scheme [1]

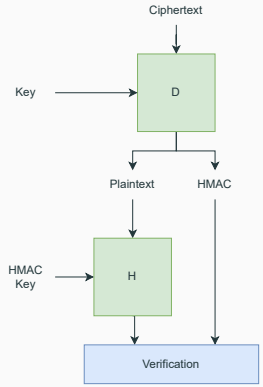

Figure 4: Verification of MAC-then-encrypt

Authentication of the bitstream

- Encrypted bitstream can still be manipulated and therefore authentication is needed
- Xilinx boards use a MAC-then-encrypt scheme [1]
- HMAC gets generated from the data and the HMAC key [6]

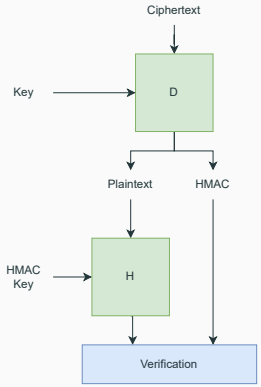

Figure 4: Verification of MAC-then-encrypt

Authentication of the bitstream

- Encrypted bitstream can still be manipulated and therefore authentication is needed
- Xilinx boards use a MAC-then-encrypt scheme [1]
- HMAC gets generated from the data and the HMAC key [6]
- Generated HMAC gets compared to stored HMAC

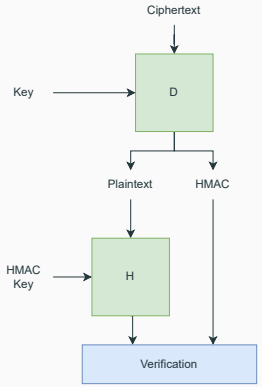

Figure 4: Verification of MAC-then-encrypt

Authentication of the bitstream

- Encrypted bitstream can still be manipulated and therefore authentication is needed
- Xilinx boards use a MAC-then-encrypt scheme [1]
- HMAC gets generated from the data and the HMAC key [6]
- Generated HMAC gets compared to stored HMAC
- If both HMACs are equal the execution will continue

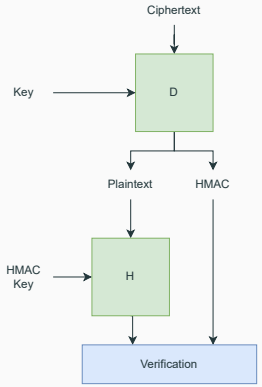

Figure 4: Verification of MAC-then-encrypt

Attacking the encryption

- The security of the FPGA is crucial

- The security of the FPGA is crucial
- Vulnerabilities in the encryption cannot be patched via update

- The security of the FPGA is crucial
- Vulnerabilities in the encryption cannot be patched via update
- FPGA are used for a long time (e.g. legacy systems)

- The security of the FPGA is crucial
- Vulnerabilities in the encryption cannot be patched via update
- FPGA are used for a long time (e.g. legacy systems)
- Successful attacks have been shown in the recent years [1]

Breaking the confidentiality

- Debugging and readout of the decrypted bitstream is prohibited

Breaking the confidentiality

- Debugging and readout of the decrypted bitstream is prohibited
- Goal is to redirect the decrypted bitstream to a register

Breaking the confidentiality

- Debugging and readout of the decrypted bitstream is prohibited
- Goal is to redirect the decrypted bitstream to a register
- Use a different bitstream to read the value from the register

Breaking the confidentiality

- A register can hold a word of 32 bits

Breaking the confidentiality

- A register can hold a word of 32 bits
- Bitstream needs to be read word by word (takes some time)

Breaking the confidentiality

- A register can hold a word of 32 bits
- Bitstream needs to be read word by word (takes some time)
- It is possible to temporarily manipulate the bitstream

The WBStar register

- In our case the WBSTAR register is useful

The WBStar register

- In our case the WBSTAR register is useful
- WBSTAR is a register of MultiBoot

The WBStar register

- In our case the WBSTAR register is useful
- WBSTAR is a register of MultiBoot
- It is used to boot from a different memory address during updating or recovery

The WBStar register

- In our case the WBSTAR register is useful
- WBSTAR is a register of MultiBoot
- It is used to boot from a different memory address during updating or recovery
- It will not be cleared upon reset

The WBStar register

- In our case the WBSTAR register is useful
- WBSTAR is a register of MultiBoot
- It is used to boot from a different memory address during updating or recovery
- It will not be cleared upon reset
- Register can still be read after tempering detection

Manipulate the bitstream

- Xilinx uses AES-CBC

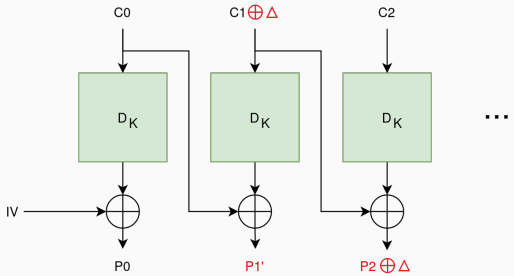

Figure 5: Malleability of CBC decryption

Manipulate the bitstream

- Xilinx uses AES-CBC
- Due to the XOR of CBC arbitrary bits can be flipped

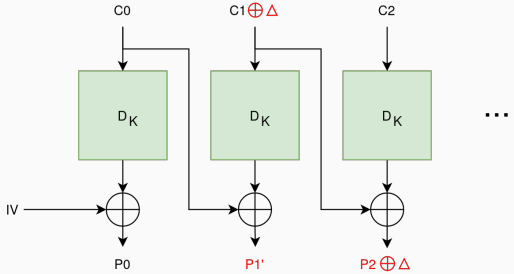

Figure 5: Malleability of CBC decryption

Manipulate the bitstream

- Xilinx uses AES-CBC
- Due to the XOR of CBC arbitrary bits can be flipped
- Vivado bitstream generation is deterministic

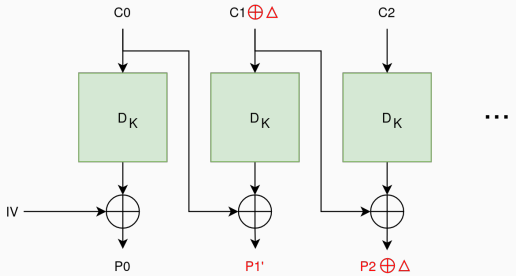

Figure 5: Malleability of CBC decryption

Manipulate the bitstream

- Xilinx uses AES-CBC
- Due to the XOR of CBC arbitrary bits can be flipped
- Vivado bitstream generation is deterministic
- Header commands are the same

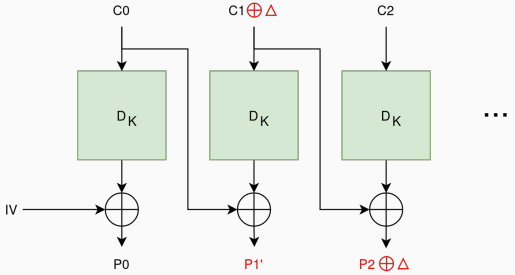

Figure 5: Malleability of CBC decryption

Manipulate the bitstream

- Xilinx uses AES-CBC
- Due to the XOR of CBC arbitrary bits can be flipped
- Vivado bitstream generation is deterministic
- Header commands are the same
- Plaintext can be assumed

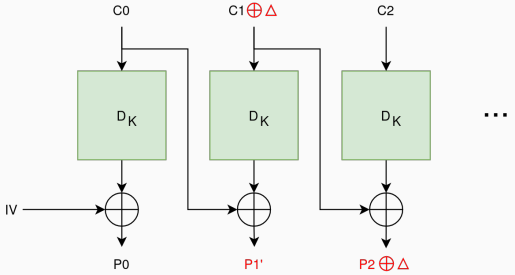

Figure 5: Malleability of CBC decryption

Manipulate the bitstream

- Xilinx uses AES-CBC
- Due to the XOR of CBC arbitrary bits can be flipped
- Vivado bitstream generation is deterministic
- Header commands are the same
- Plaintext can be assumed
- HMAC is only checked after the interpretation

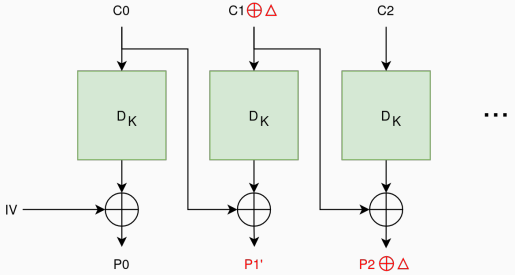

Figure 5: Malleability of CBC decryption

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P_i'

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P'_i
- We need to set: $C'_{i-1} = P_i \oplus C_{i-1} \oplus P'_i$

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P'_i
- We need to set: $C'_{i-1} = P_i \oplus C_{i-1} \oplus P'_i$
- So we have $P'_i = dec_{KAES}(C_i) \oplus C_{i-1} \oplus P_i \oplus P'_i = P_i \oplus P_i \oplus P'_i$

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P'_i
- We need to set: $C'_{i-1} = P_i \oplus C_{i-1} \oplus P'_i$
- So we have $P'_i = dec_{KAES}(C_i) \oplus C_{i-1} \oplus P_i \oplus P'_i = P_i \oplus P_i \oplus P'_i$
- We can repeat this until P'_1 and then set IV to C_0 in the unencrypted header

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P'_i
- We need to set: $C'_{i-1} = P_i \oplus C_{i-1} \oplus P'_i$
- So we have $P'_i = dec_{KAES}(C_i) \oplus C_{i-1} \oplus P_i \oplus P'_i = P_i \oplus P_i \oplus P'_i$
- We can repeat this until P'_1 and then set IV to C_0 in the unencrypted header
- HMAC and key for verification is located in the bitstream

Breaking the authenticity

- Previous attack provides the FPGA as decrypt oracle which can be use to encrypt data [7]
- The CBC function: $P_i = dec_{KAES}(C_i) \oplus C_{i-1}$
- We can manipulate the chiptertext and want a desired P'_i
- We need to set: $C'_{i-1} = P_i \oplus C_{i-1} \oplus P'_i$
- So we have $P'_i = dec_{KAES}(C_i) \oplus C_{i-1} \oplus P_i \oplus P'_i = P_i \oplus P_i \oplus P'_i$
- We can repeat this until P'_1 and then set IV to C_0 in the unencrypted header
- HMAC and key for verification is located in the bitstream
- Attacker can manipulate the HMAC this way

Issues of the implementation

Issues of the implementation

1. Interpreting the data before HMAC validation

Issues of the implementation

1. Interpreting the data before HMAC validation
2. Storing the HMAC key in the bitstream itself

Conclusion

Conclusion

1. Bitstream encryption is important for authenticity and confidentiality

Conclusion

1. Bitstream encryption is important for authenticity and confidentiality
2. It prevents adversaries from reverse engineering

Conclusion

1. Bitstream encryption is important for authenticity and confidentiality
2. It prevents adversaries from reverse engineering
3. It prevents adversaries from manipulating

Conclusion

1. Bitstream encryption is important for authenticity and confidentiality
2. It prevents adversaries from reverse engineering
3. It prevents adversaries from manipulating
4. Current implementation has flaws and can be attacked without sophisticated tools

References i

[1] M. Ender, A. Moradi, and C. Paar, The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series fpgas, in 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 1803–1819.

[2] S. M. Trimberger and J. J. Moore, Fpga security: Motivations, features, and applications, Proceedings of the IEEE, vol. 102, no. 8, pp. 1248–1265, 2014.

[3] Xilinx, Using encryption to secure a 7 series fpga bitstream,, 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp1239-fpga-bitstream-encryption.

[4] Xilinx, 7 series fpgas data sheet: Overview,, 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview.

[5] Xilinx, Xilinx kintex-7 fpga kc705 evaluation kit,, 2022. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html.

References ii

[6] S. Drimer, Authentication of fpga bitstreams: Why and how, in International Workshop on Applied Reconfigurable Computing, Springer, 2007, pp. 73–84.

[7] J. Rizzo and T. Duong, Practical padding oracle attacks, in 4th USENIX Workshop on Offensive Technologies (WOOT 10), 2010.