Detecting Block Cipher Encryption for Defense against Crypto Ransomware on Low-end Internet of Things

Hyunji Kim, Jaehoon Park, Hyeokdong Kwon, Kyoungbae Jang, Seungju Choi, and Hwajeong Seo

khj1594012@gmail.com

한성대학교 HANSUNG UNIVERSITY

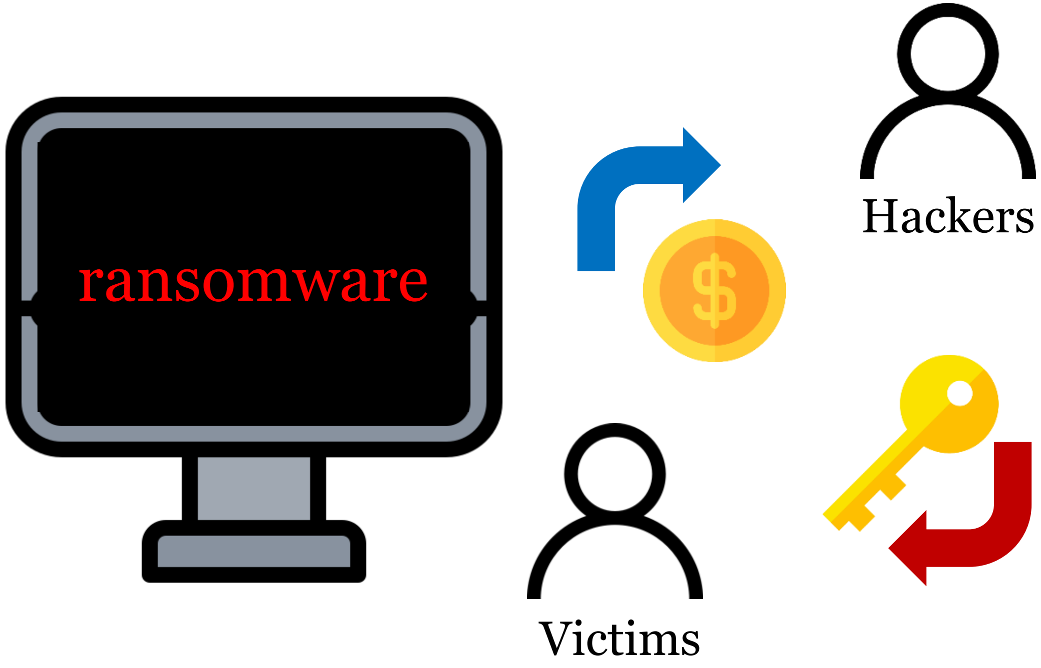
CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

Crypto Ransomware

- **A crypto ransomware encrypts files of victims using block cipher encryption.**



1) The hacker requests a ransom for encrypted files from the victim.

2) The victim pays the ransom and receives a secret key.

3) The victim recovers the encrypted file using a secret key.

- **The ransomware virus has became a massive threat of people with digital devices.**

  → It is necessary to defend against ransomware.

Previous Works

Features	Grobert et al. (2011)	Lestringant et al. (2015)	Kiraz et al. (2017)	This work (2020)
Target	Block & PKC	Block Cipher	PKC	Block cipher
Analysis	Dynamic	Static	Dynamic	Static
Method	Heuristics	Data graph flow	System monitor	Deep learning
Machine	Desktop	Desktop	Desktop	Microcontroller

Proposed Method

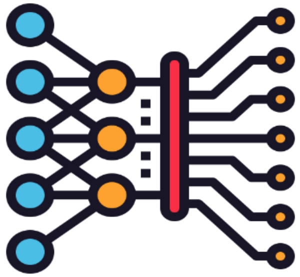
1. Crypto ransomware

- Cryptographic process

2. Static analysis

- Binary file
- Instruction, Opcode

binary file (lss, hex)

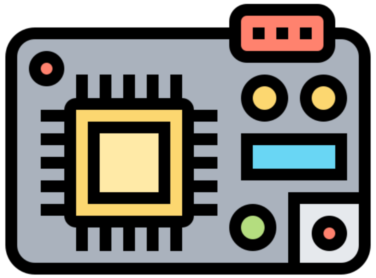


3. Deep learning based detection of block cipher encryption

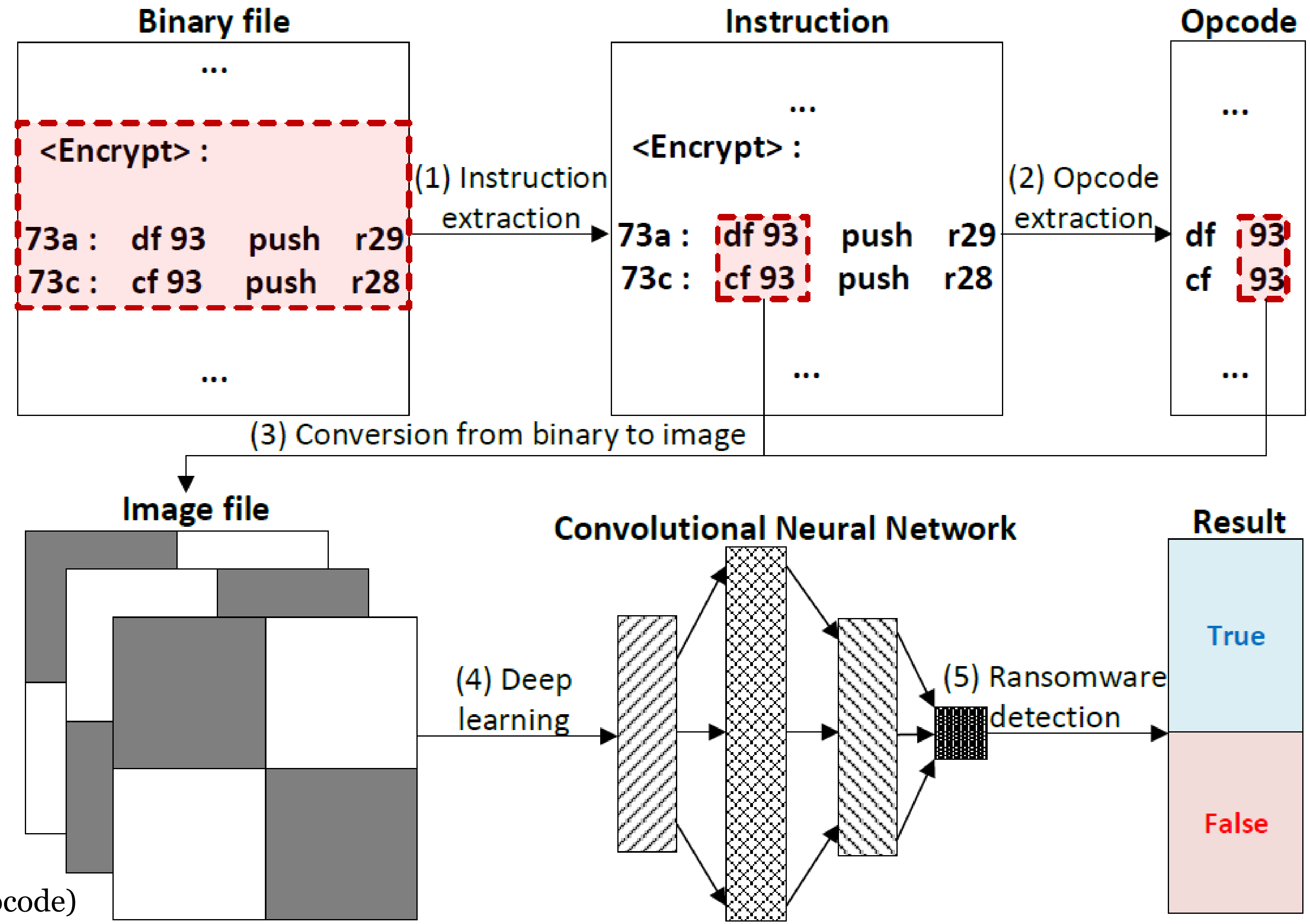
- Learning the encryption pattern through images
- Convolutional Neural Networks (CNN)

4. Low-end 8-bit AVR microcontrollers

- 8-bit AVR ATmega128
- Lightweight block cipher library and general firmware
- GNU AVR-GCC compiler

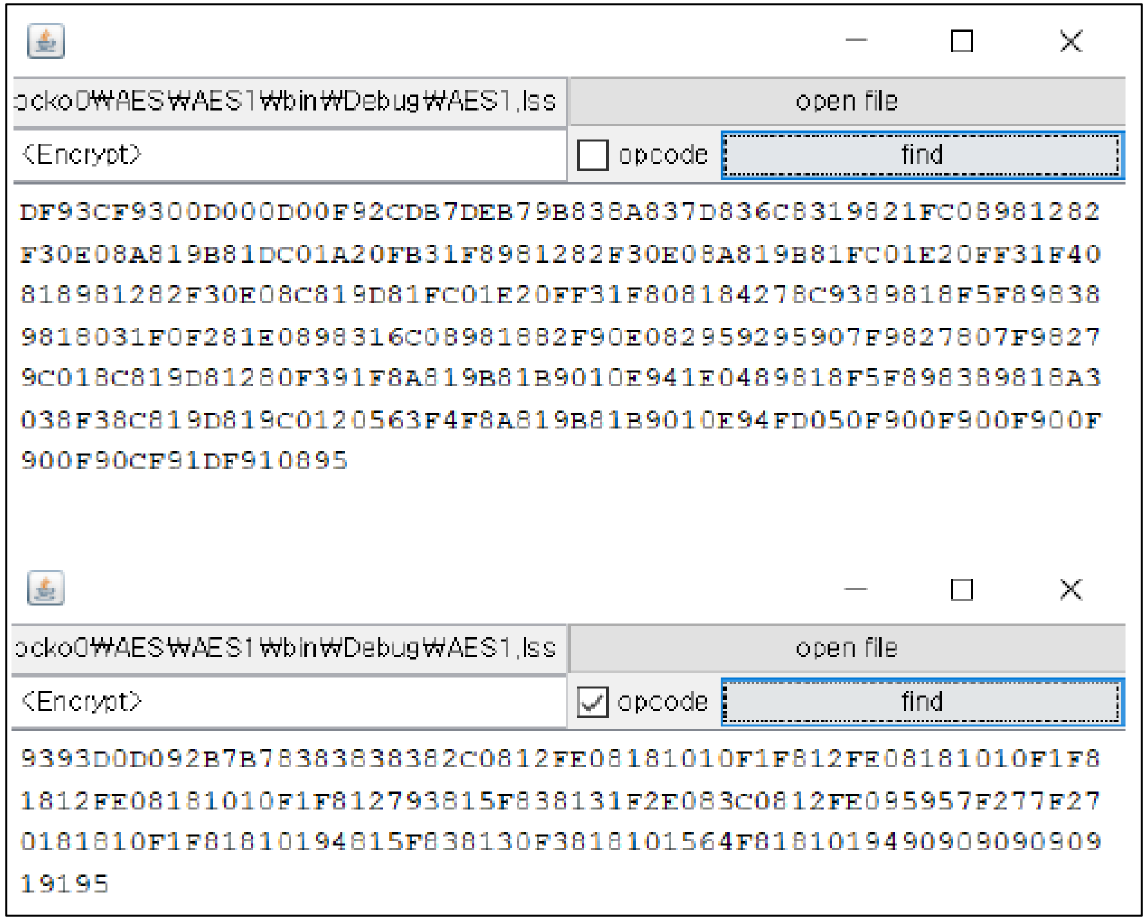


Proposed Method



*instruction (operand + opcode)

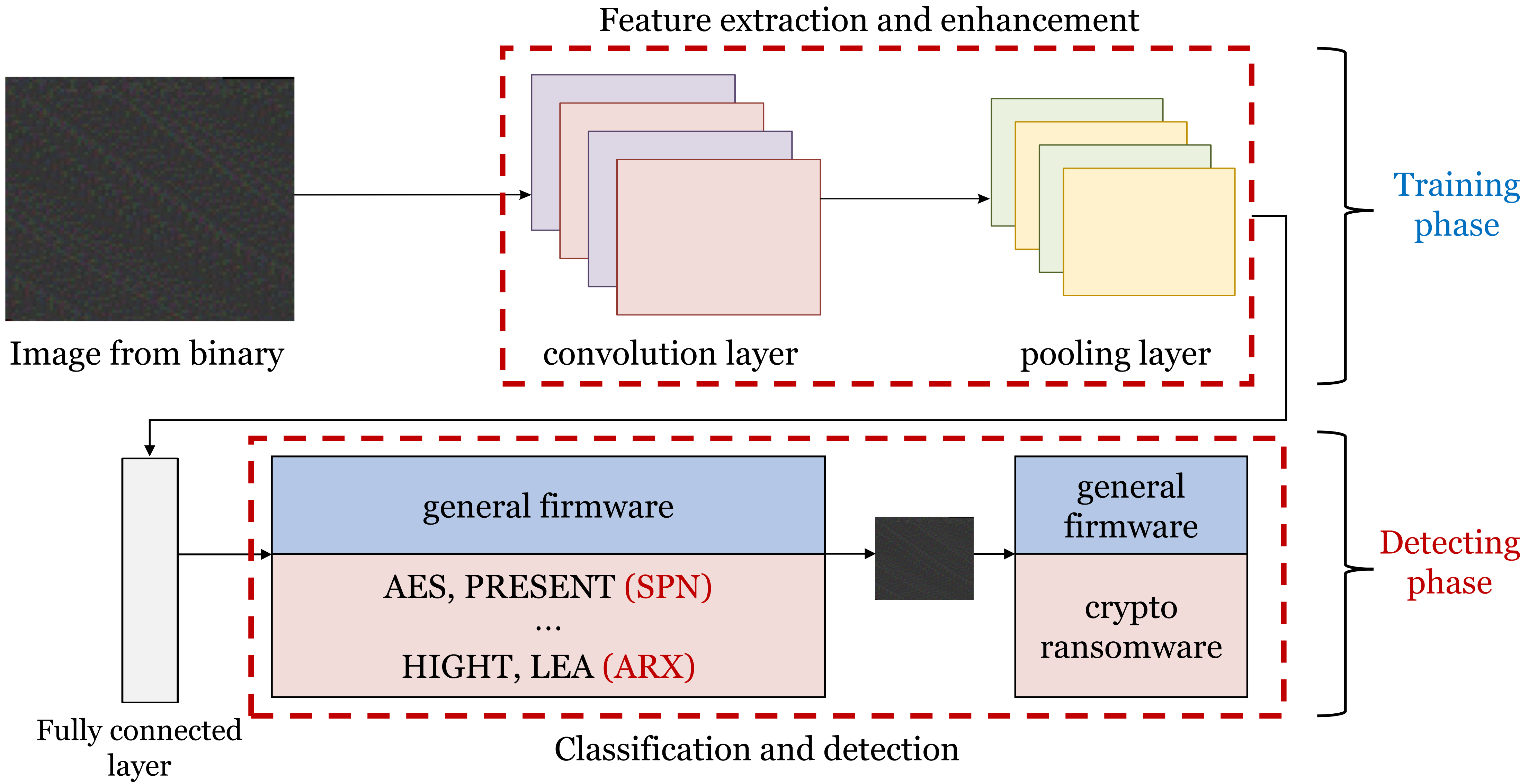
Convert instruction and opcode into image



Instruction and opcode extraction from binary file (.lss)

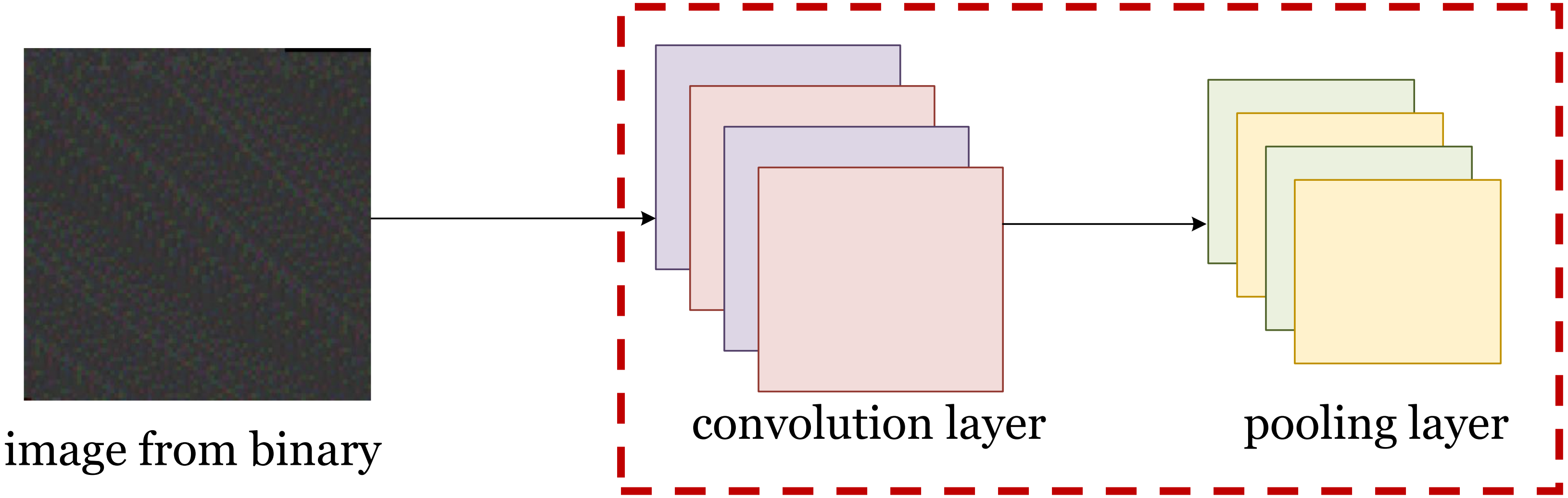
Image from binary

Deep learning



Training phase

- **Repeat convolution and max pooling operation**
  - convolution : feature extraction
  - max pooling : feature enhancement
  - ➢ Learning the operation types and patterns of the encryption process

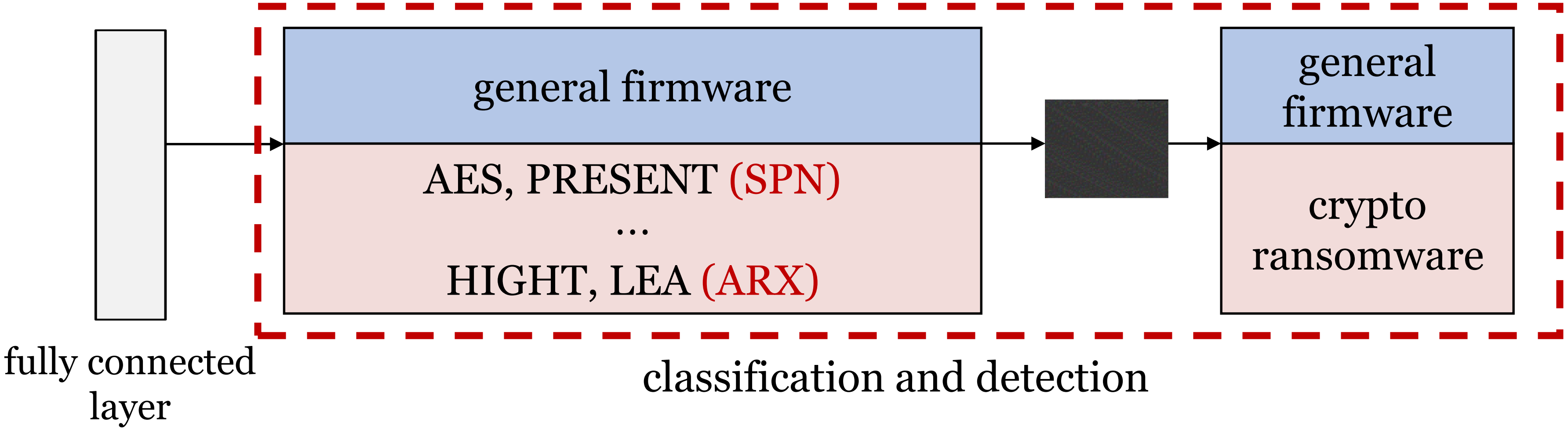


Feature extraction and enhancement

Ransomware detection phase

- **Detecting block cipher encryption for defense against crypto ransomware**
  1. algorithm : AES, PRESENT ⋯ HIGHT, LEA, general firmware
  2. architecture : SPN, ARX, general firmware

  ➢ Classified as the label with the highest probability



Model and Hyper parameters

- **Inception-v3** (pretrained weights) + **3 fully connected layers**
  → To adjust the classication problem

- **Categorical crossentropy and Softmax**
  → Multi-class classification

- Set the optimal hyper parameter through grid search.

Table 2. Deep learning training hyperparameters.

Hyperparameters	Descriptions	Hyperparameters	Descriptions
pretrained model	Inception-v3	epochs	20
loss function	categorical crossentropy	steps per epoch	10
optimizer	RMSprop(lr=0.001)	batch size	5
active function	ReLu, Softmax	train, validation and test ratio	0.7, 0.2, 0.1

Dataset

- **Block cipher**

  Cryptographic modules written in C language among implementations of FELICS

  (Fair Evaluation of Lightweight Cryptographic System)

- **General firmware**

  From AVR packages

- **On 8-bit AVR ATmega128**
- **Unbalanced dataset**
  - macro and micro average

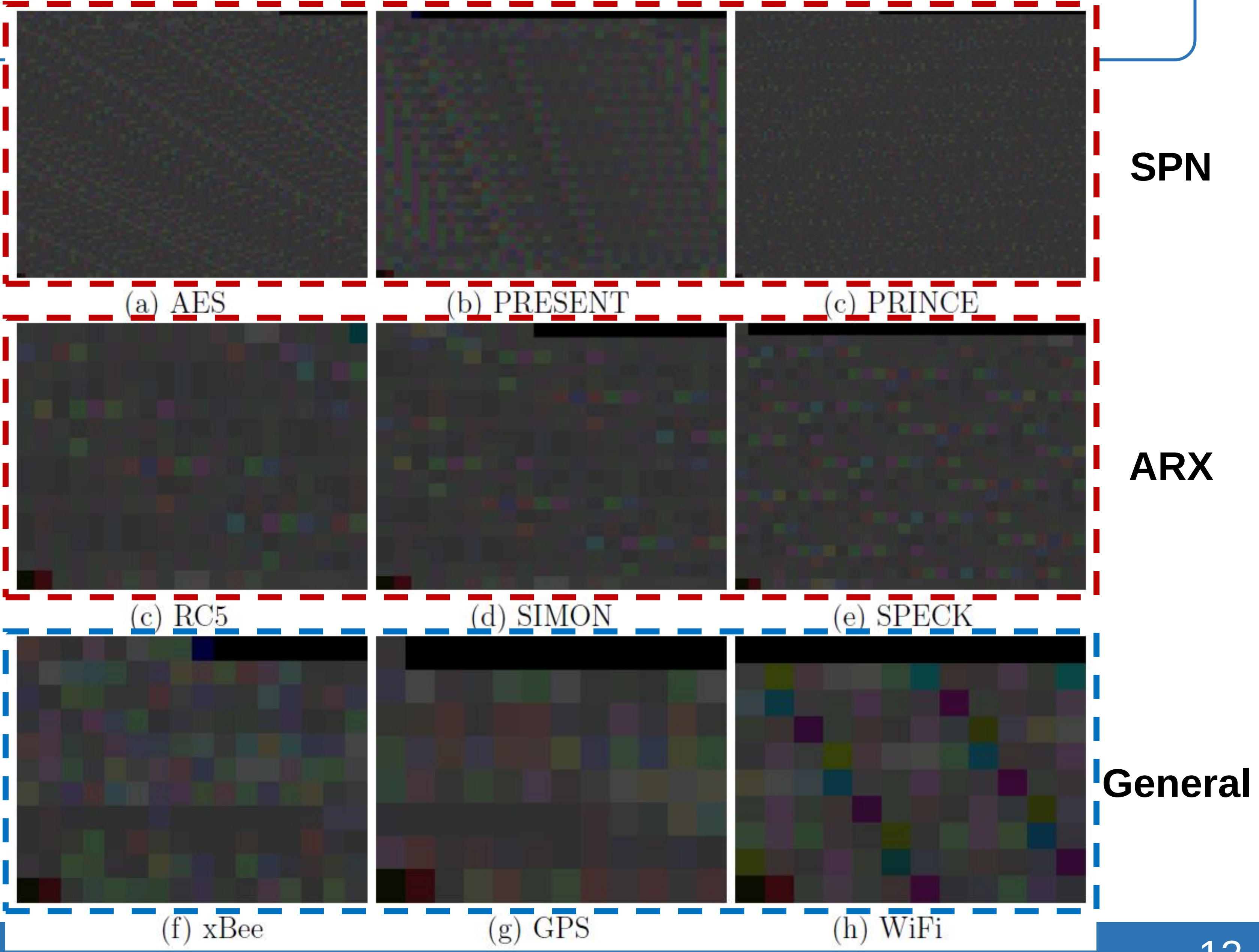
Table 3. Dataset (block ciphers and general firmware).

Architecture	Descriptions of programs
SPN	AES, RECTANGLE, PRESENT, PRIDE, PRINCE
ARX	HIGHT, LEA, RC5, SIMON, SPECK, SPARX
General	Bluetooth, GPS, WiFi, RFID, XBee, etc.

*FELICS : Fair Evaluation of Lightweight Cryptographic System

Dataset

- SPN has a more complicated structure than ARX.
- Images differ depending on the type or pattern of instructions used.



Instruction-based vs Opcode-based

The opcode is same, but the operand is slightly different.

```
89 89    ldd     r24, Y+17 ; 0x11
9a 89    ldd     r25, Y+18 ; 0x12
fc 01    movw    r30, r24
3c 96    adiw    r30, 0x0c ; 12
20 81    ld      r18, Z
31 81    ldd     r19, Z+1  ; 0x01
42 81    ldd     r20, Z+2  ; 0x02
53 81    ldd     r21, Z+3  ; 0x03
```

```
8a 89    ldd     r24, Y+18 ; 0x12
9b 89    ldd     r25, Y+19 ; 0x13
fc 01    movw    r30, r24
3c 96    adiw    r30, 0x0c ; 12
20 81    ld      r18, Z
31 81    ldd     r19, Z+1  ; 0x01
42 81    ldd     r20, Z+2  ; 0x02
53 81    ldd     r21, Z+3  ; 0x03
```

operand opcode

*instruction (operand + opcode)

Instruction-based vs Opcode-based

- **Opcode-based**

  Stable performance for untrained data

  → The standard deviation for micro and macro is 0.12.

  → The standard deviation of the metrics for each experiment is also less opcode based.

Table 4. Validation and test on instruction and opcode.

Target	Validation						Test					
	F-measure		precision		recall		F-measure		precision		recall	
	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro
Instruction	0.91	0.84	0.91	0.87	0.91	0.95	0.80	0.60	0.80	0.60	0.80	0.60
Opcode	0.96	0.93	0.96	0.92	0.96	0.94	0.77	0.58	0.77	0.59	0.77	0.60

*F-measure : a harmonic mean of precision and recall

*micro : considering the number of data belonging to each class (for each data)

*macro : considering all classes with the same weight (for label)

GCC optimization option (O0, O1, O2)

- The extracted opcode is slightly different depending on the optimization option.
- O0 : Best performance for validation dataset, but not best for test dataset
- O1 : Stable and best generalization performance for test dataset (untrained data)

Table 5. Validation and test on GCC optimization options.

Op.	validation						test					
	F-measure		precision		recall		F-measure		precision		recall	
	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro
O0	0.96	0.93	0.96	0.92	0.96	0.94	0.77	0.58	0.77	0.59	0.77	0.60
O1	0.90	0.81	0.90	0.80	0.90	0.85	0.85	0.79	0.85	0.82	0.85	0.81
O2	0.92	0.89	0.92	0.90	0.92	0.9	0.81	0.67	0.81	0.69	0.81	0.68

Frequently used instructions for each architecture

- **SPN**

  S-box operation → the pattern of LD – XOR – ST , memory access (LD, ST)

- **ARX**

  Addition, Rotation, and eXclusive-or (arithmetic and logical operations) → ADD, XOR, SUB

- **General**

  interrupt function, I/O register, branch statements → RJMP, BNE, CP

Table 6. Frequently used instructions for each architecture.

Architecture	Ordered by frequency in program							
	1	2	3	4	5	6	7	8
SPN	LD	ST	MOV	XOR	ADD	SUB	AND	SWAP
ARX	LD	ADD	XOR	MOV	ST	SUB	ROR	RJMP
General	LD	RJMP	BNE	CP	OUT	NOP	MOV	SEI

SPN vs ARX vs General

- **Cryptographic algorithms of the same architecture have similar patterns.**

  → incorrect classification between algorithms with the same architecture

- **Classification for each architecture** (SPN, ARX and General firmware, not each algorithm) **achieved better performance.**

- It accurately predicted test data in optimization option O1.

Table 7. Validation and test depending on architectures (SPN, ARX, or general).

Op.	validation						test					
	F-measure		precision		recall		F-measure		precision		recall	
	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro
O0	0.99	0.99	0.99	0.99	0.99	0.99	0.94	0.91	0.94	0.91	0.94	0.95
O1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
O2	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.98	0.99	0.98	0.99

Conclusion

- **Contribution**

  Deep learning based crypto ransomware detection for low-end microcontrollers

  Experiments with several options for high accuracy

- **Future work**

  Test and evaluate our method using the real ransomware samples.

Q & A