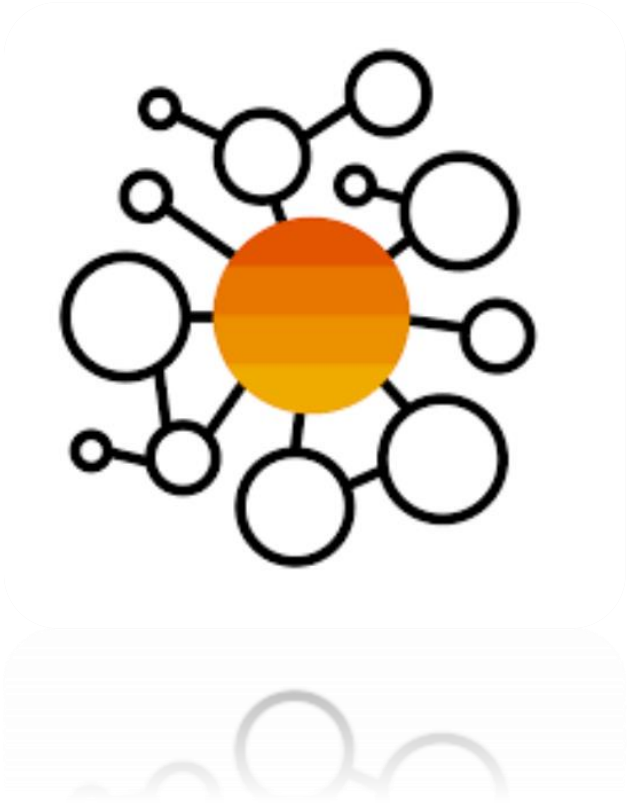
Static Analysis in encrypted domain

*Tae Yoon Hwang, *Ji Won Yoon

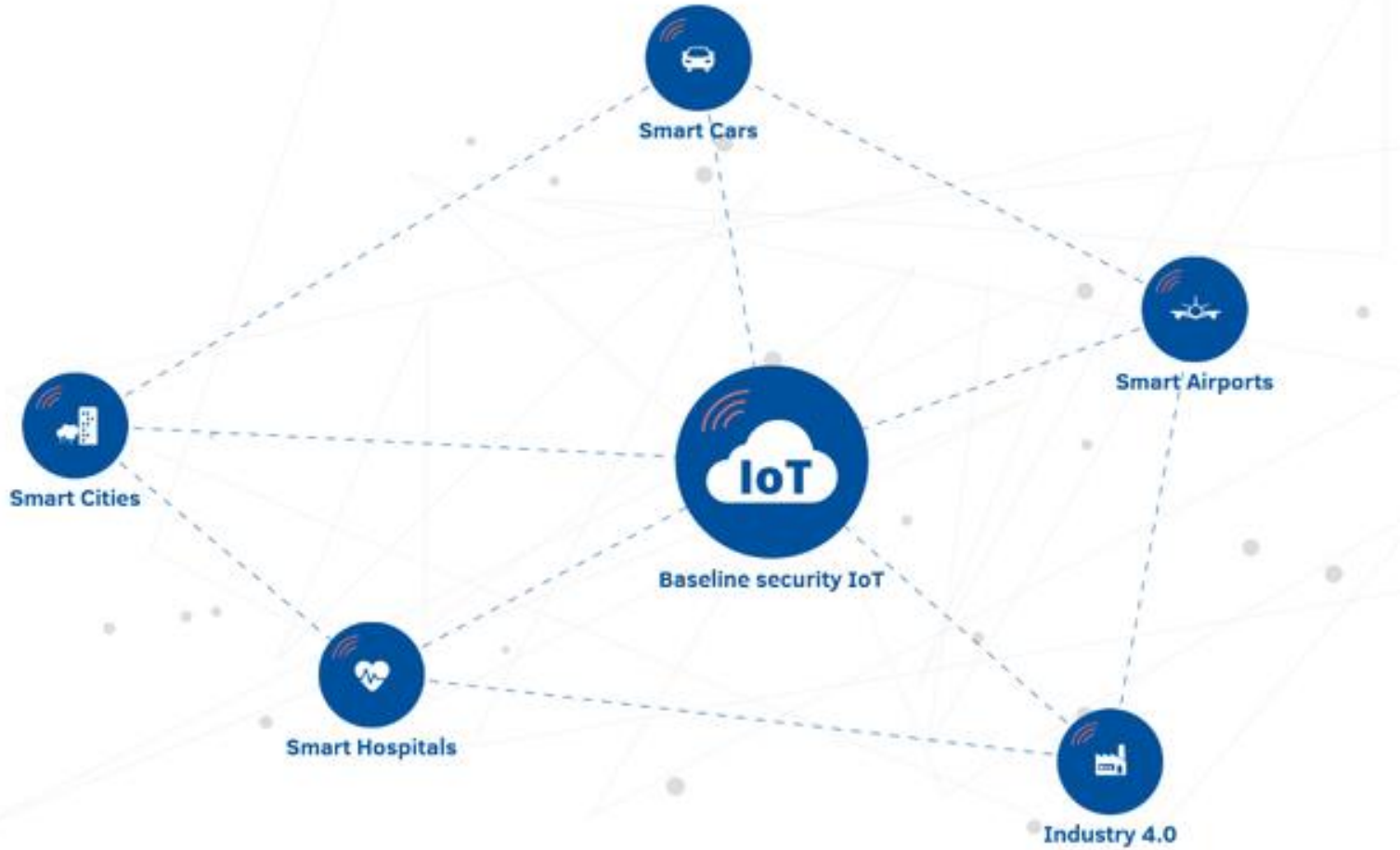
*Korea University, Seoul, Republic of Korea

Master degree



- IOT Tech
- Cloud computing
- Homomorphic encryption
- Static Analysis for malware detection
- Proposed Detection system
- Homomorphic logic
- Experiment
- Future work

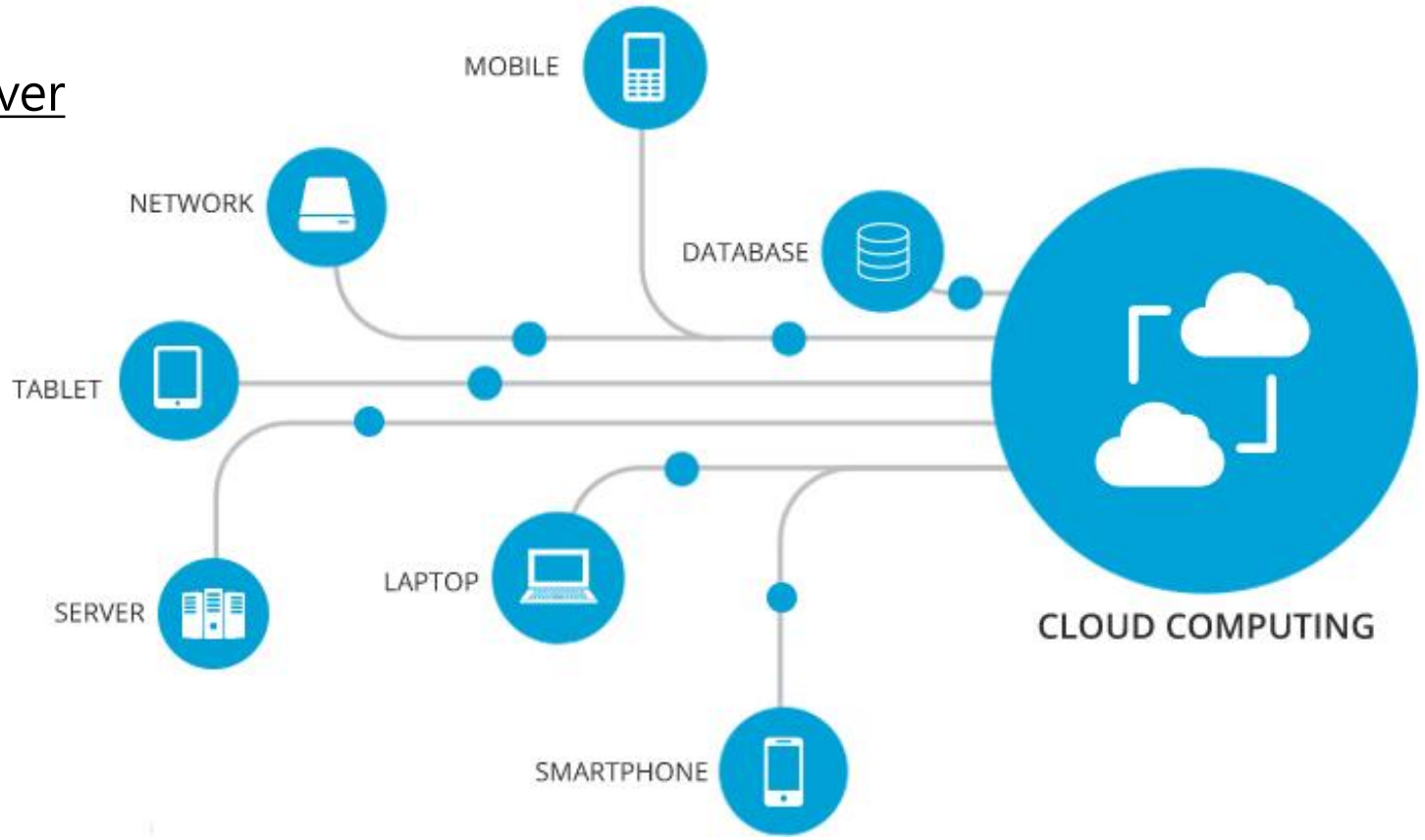
IOT tech


Smart Cars
Smart Airports
Smart Cities
IoT
Baseline security IoT
Smart Hospitals
Industry 4.0

Cloud computing

- Server can handle a lot of data

- Many users upload data to the server and receive the results



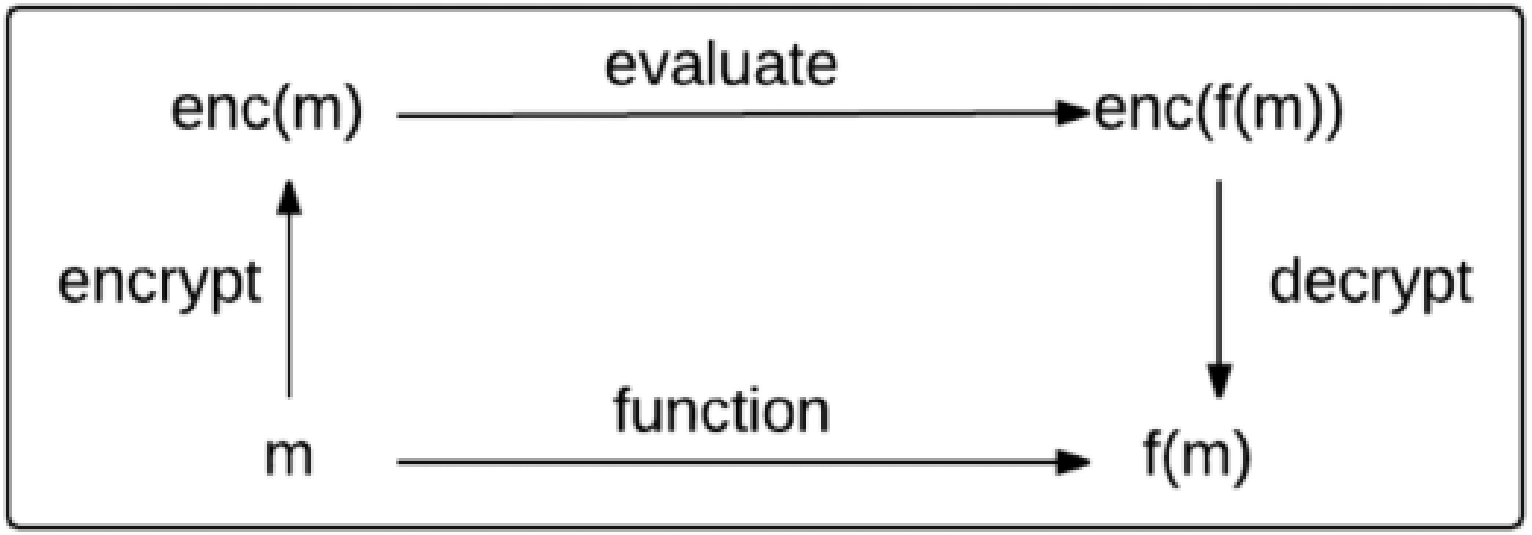
Cloud computing

- Huge increase of computation and storage in Internet an **IOT environment.**
- Very useful in managing complicated data
- Data transferred from users to the server are **not secure**

Homomorphic Encryption

- The user/client provides the data encrypted.

- Server can computes the encrypted data with the cloud key.



Homomorphic Encryption

- **TFHE** library
- Fast Fully Homomorphic Encryption over the Torus
- Library made by **Chillotti et al.**

Static Analysis for malware detection

- **YARA** system

- Check ASCII range **strings** OR **Binary** Sequence

Static Analysis for malware detection

TABLE I
MIRAI MALWARE YARA RULESET.

```
rule Mirai
{
    meta:
        description = "Mirai Variant 3"
        author = "Joan Soriano / @joanbtl"
        date = "2017-04-16"
        version = "1.0"
        MD5 = "bb22b1c921ad8fa358d985ff1e51a5b8"
        SHA1 = "432ef83c7692e304c621924bc961d95c4aea0c00"

    strings:
        $dir1 = "/dev/watchdog"
        $dir2 = "/dev/misc/watchdog"
        $s1 = "PMMV"
        $s2 = "ZOJFKRA"
        $s3 = "FGDCWNV"
        $s4 = "OMVJGP"
        $ssl = "ssl3_ctrl"

    condition:
        $dir1 and $dir2 and $s1 and $s2 and $s3 and $s4 and not $ssl
}
```

Proposed Detection system

- How personal information can be leaked when using malware detection.
- Server owner can see all the information about the file.
- If we can encrypt the internal data when scanning the file, we can protect the information about files.

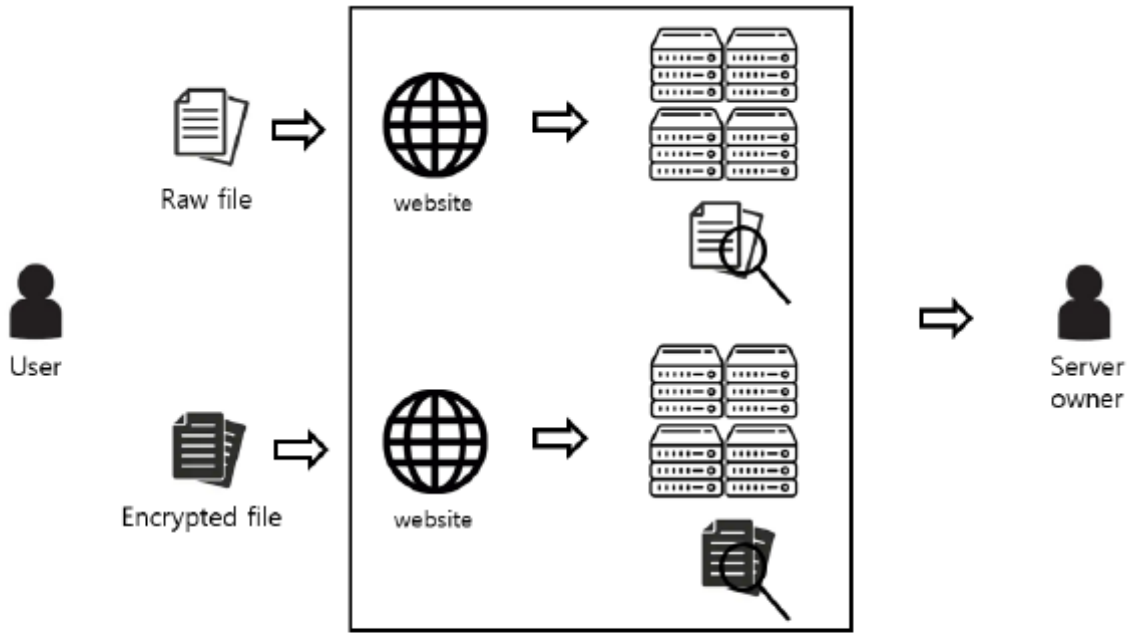
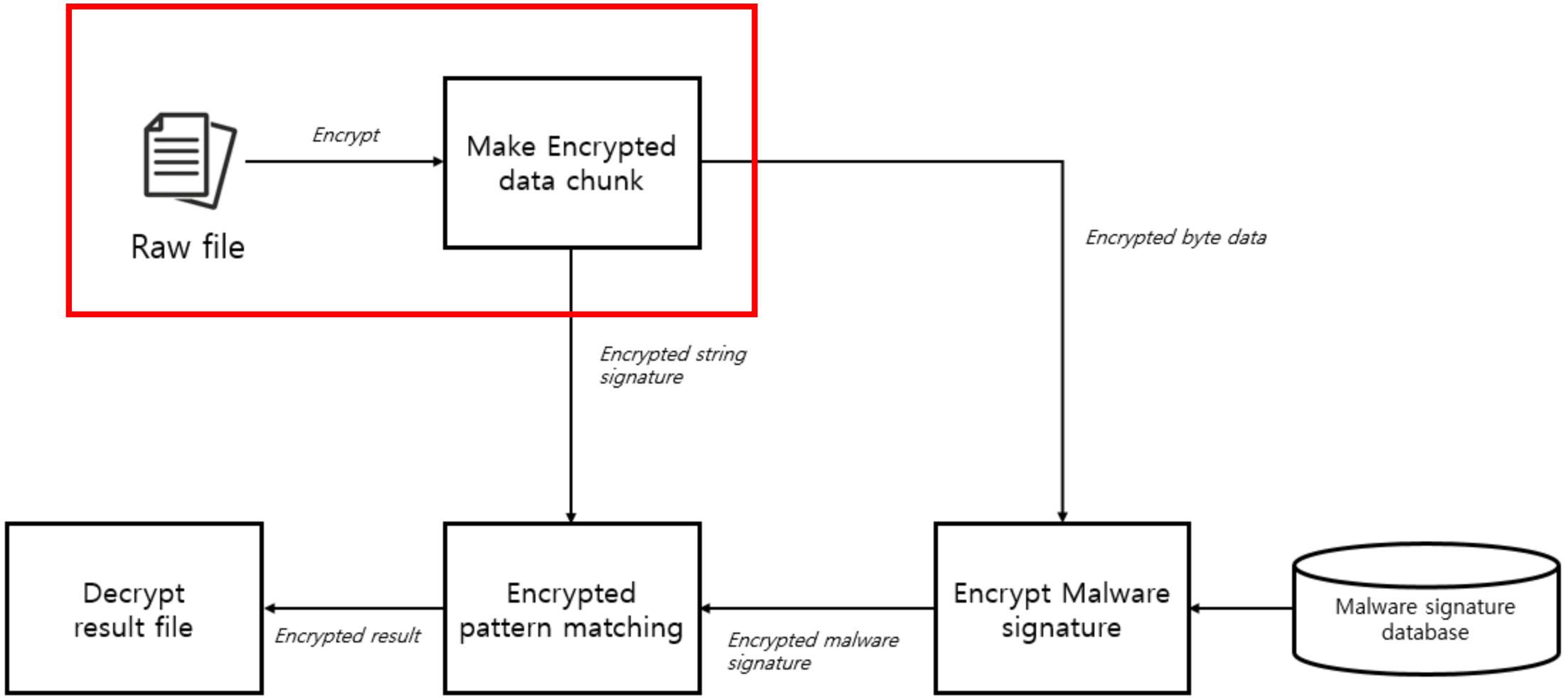


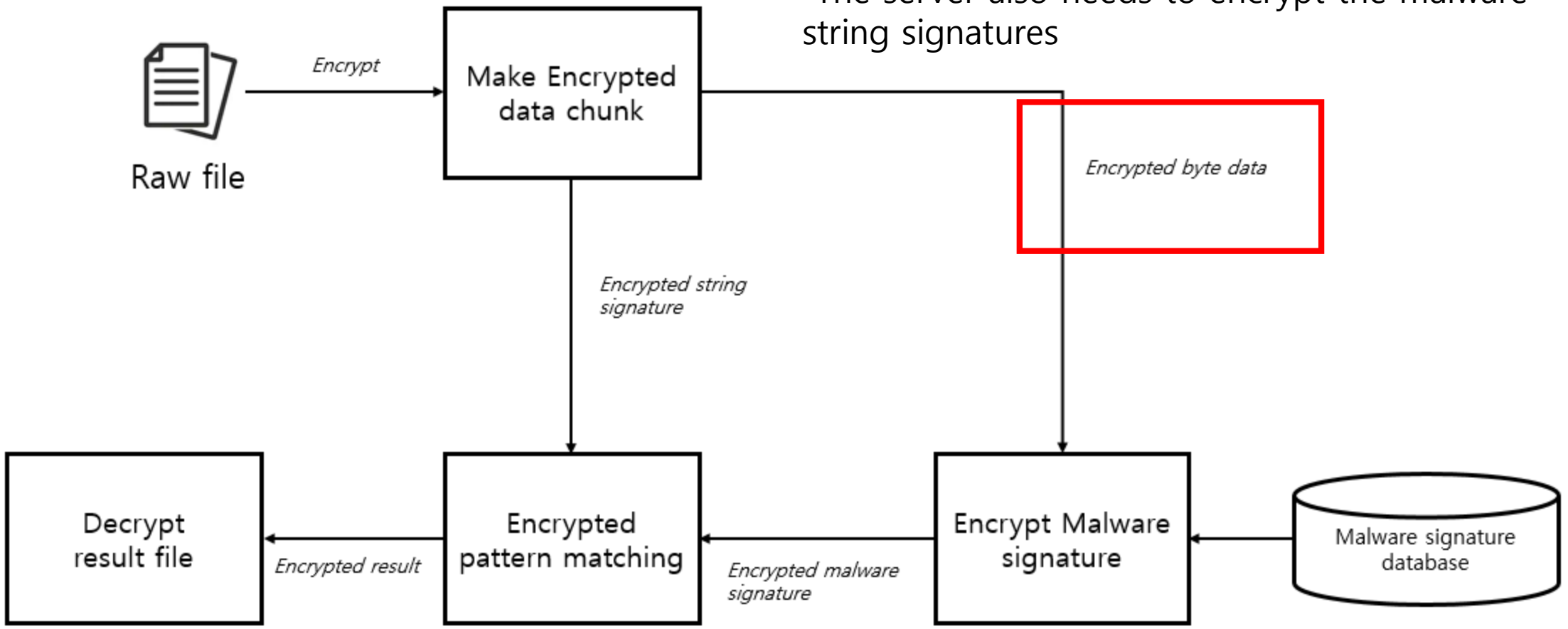
Fig. 1. Malware detection process

Proposed Detection system

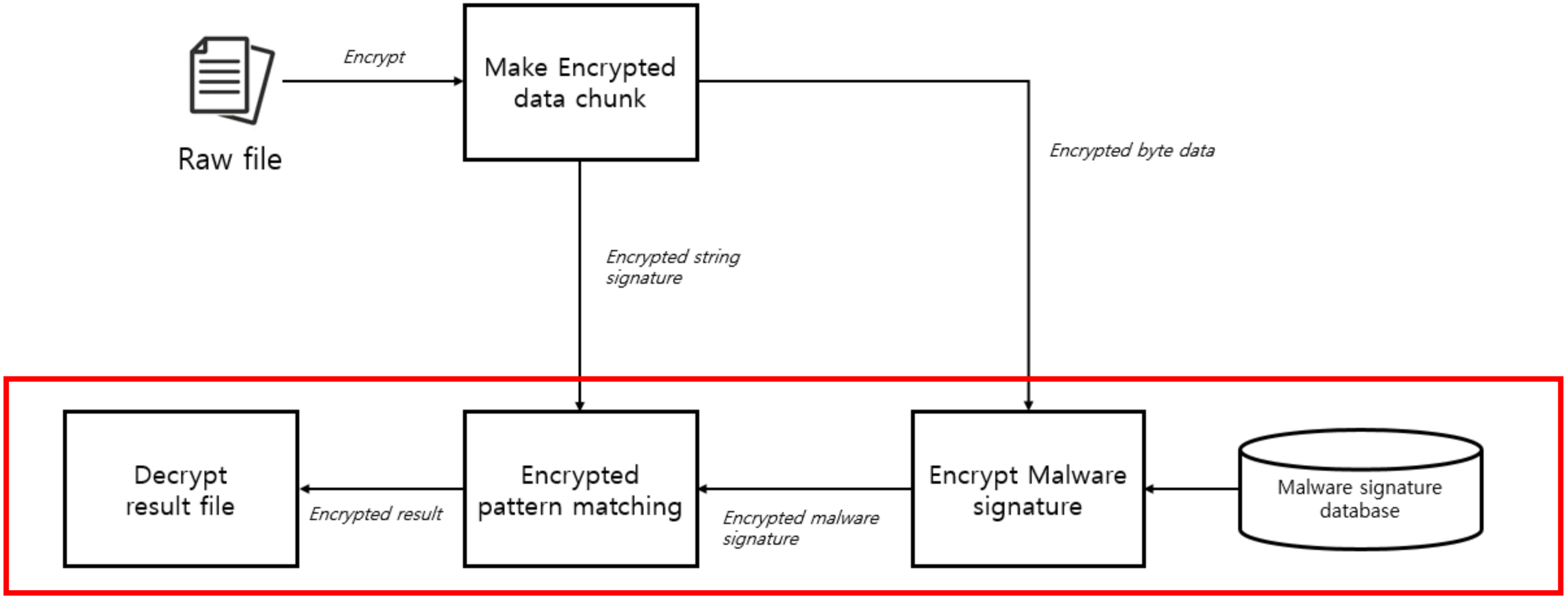


Proposed Detection system

The server also needs to encrypt the malware string signatures



Proposed Detection system



Homomorphic logic

- In the "Encrypt" string, "r" indicates 0x72 when converted to hexadecimal.
- Bit-wise calculations are performed each bit is encrypted.

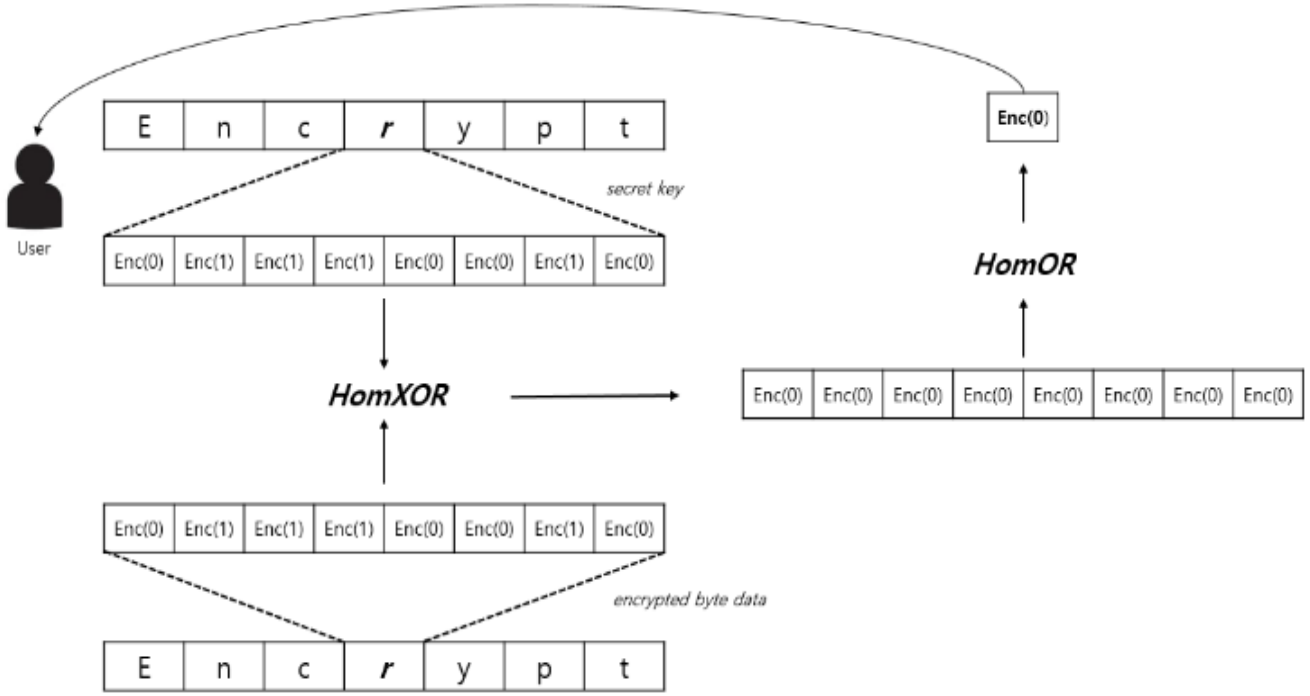


Fig. 4. Detection process

Homomorphic logic

- TFHE library
- supports APIs for logical operation

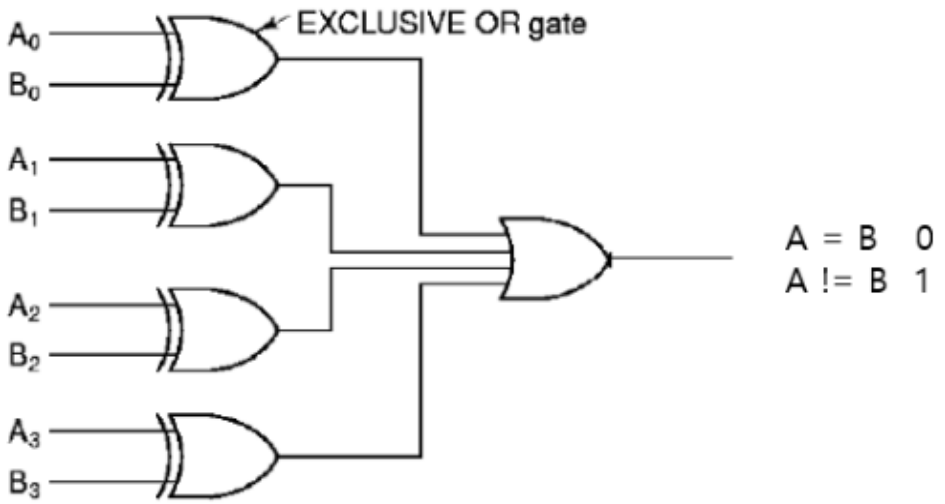


Fig. 3. Compare gate using XOR, OR gate

$$Enc_{sk}(A_1) \oplus Enc_{sk}(B_1) + Enc_{sk}(A_2) \oplus Enc_{sk}(B_2) + \cdots + Enc_{sk}(A_n) \oplus Enc_{sk}(B_n) \quad (4)$$

Homomorphic logic

Algorithm 1 Data encryption using TFHE library

- **Input:** String signature **S**, length of signature n
- **Output:** Encrypted signature file

1: Create $arr[8]$, size of 1bytes(8bits)
2: iter $\leftarrow n$
3: **for** $i = 0$ to $iter$ **do**
4: **for** $j = 0$ to len(arr) **do**
5: idx $= (\text{len}(arr) * i) + j$
6: $arr[j] \leftarrow Enc_{sk}(\mathbf{S}[idx])$
7: **end for**
8: Write arr
9: **end for**

Homomorphic logic

Algorithm 2 Comparing Encrypted Data

- **Input:** Encrypted string signature chunk **S**, Encrypted string signature chunk **M**, length of signature n
- **Output:** Compare result file

1: Create an array *temp* of length 2.
2: Initialize $temp[0]$ to $Enc_{ck}(0)$
3: **for** $i = 0$ to n **do**
4: **for** $j = 0$ to 8 **do**
5: $temp[1] \leftarrow HomXOR_{ck}(\mathbf{S}[8 * i + j], \mathbf{M}[8 * i + j])$
6: $temp[0] \leftarrow HomOR_{ck}(temp[0], temp[1])$
7: **end for**
8: Return $temp[0]$
9: **end for**

Experiment

- 2 malicious code data sets: **DarkComet, LostDoor**
- Ubuntu 16.04bit, Intel Core i7-7700 CPU 3.60GHz, 16GB RAM
- Yara ruleset: Strings signature -> ASCII range

Experiment

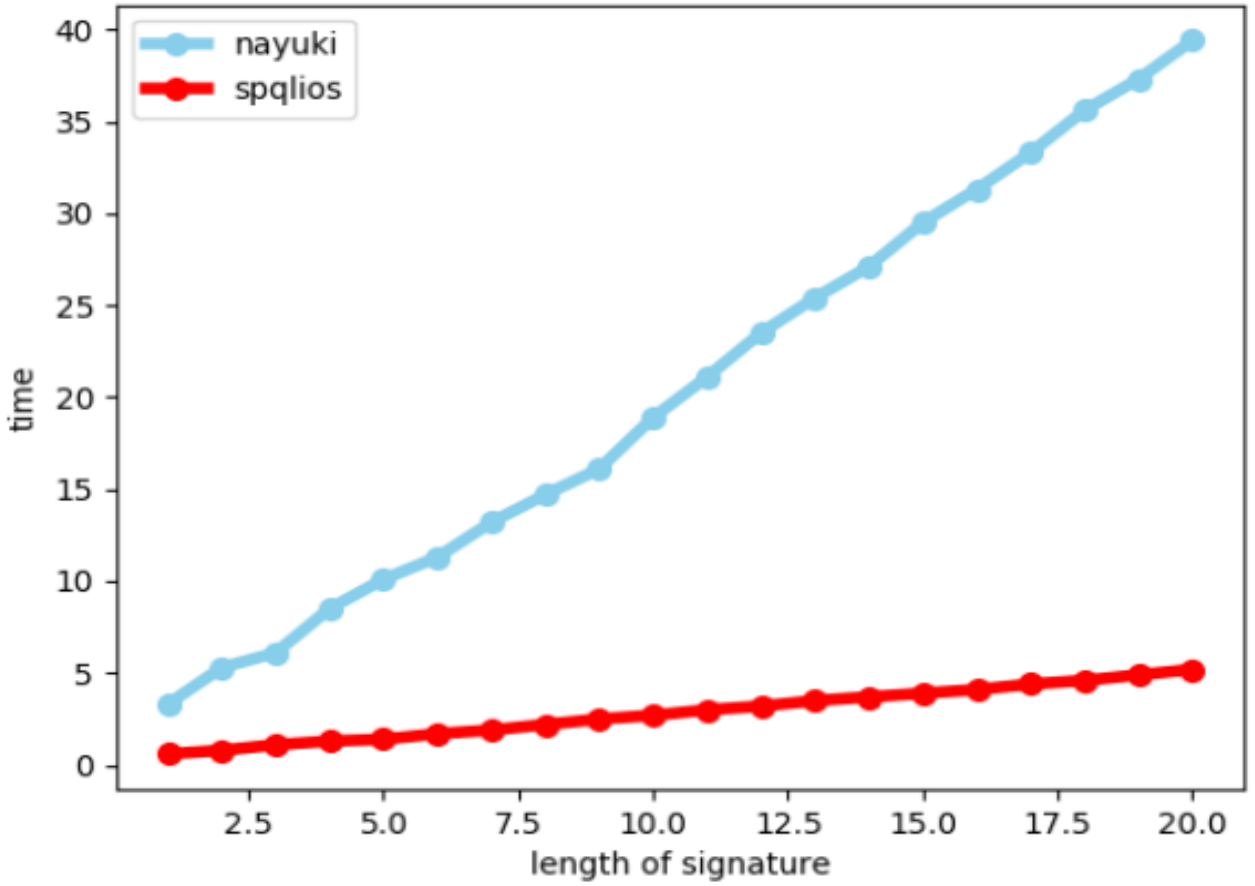


Fig. 5. Speed comparison by length

Experiment

Table 2. Malware signature search speed

	size(byte)	*Pattern length*	*nayuki(sec)*	*spqlios(sec)*
LostDoor	72,268	(9, 8, 10, 13)	3,850	476
	72,284		3,762	477
	72,355		3,862	505
	163,022		39,352	7,877
	225,592		2,601	530
	237,159		3,374	423
DarkComet	238,080	(13, 15, 22, 36)	2,508	321
	266,240		2,441	246
	314,368		1,936	329
	333,312		2,347	330
	370,176		2,238	286

Experiment

Table 2. Malware signature search speed

	size(byte)	*Pattern length*	*nayuki(sec)*	*spqlios(sec)*
LostDoor	72,268	(9, 8, 10, 13)	3,850	476
	72,284		3,762	477
	72,355		3,862	505
	163,022		39,352	7,877
	225,592		2,601	530
	237,159		3,374	423
DarkComet	238,080	(13, 15, 22, 36)	2,508	321
	266,240		2,441	246
	314,368		1,936	329
	333,312		2,347	330
	370,176		2,238	286

Experiment

Table 2. Malware signature search speed

	size(byte)	*Pattern length*	*nayuki(sec)*	*spqlios(sec)*
LostDoor	72,268	(9, 8, 10, 13)	3,850	476
	72,284		3,762	477
	72,355		3,862	505
	163,022		39,352	7,877
	225,592		2,601	530
	237,159		3,374	423
DarkComet	238,080	(13, 15, 22, 36)	2,508	321
	266,240		2,441	246
	314,368		1,936	329
	333,312		2,347	330
	370,176		2,238	286

Experiment

Table 2. Malware signature search speed

	size(byte)	*Pattern length*	*nayuki(sec)*	*spqlios(sec)*
LostDoor	72,268	(9, 8, 10, 13)	3,850	476
	72,284		3,762	477
	72,355		3,862	505
	163,022		39,352	7,877
	225,592		2,601	530
	237,159		3,374	423
DarkComet	238,080	(13, 15, 22, 36)	2,508	321
	266,240		2,441	246
	314,368		1,936	329
	333,312		2,347	330
	370,176		2,238	286

Future work

- Detection method by pattern matching -> Find all string but simple.
- Because of the use of very simple techniques, the malware that can be detected is still limited.
- Need a detection method applicable to various binaries.

Future work

- Bootstrapping process takes a very long time.

- Increase the number of threads to operate on or use the GPU to perform fast calculations

Q & A

- Feel free for any questions! Thank you