Annotating, Tracking, and Protecting Cryptographic Secrets with CryptoMPK

Xuancheng Jin, Xuangan Xiao, Songlin Jia, Wang Gao, Dawu Gu,

Hang Zhang, Siqi Ma, Zhiyun Qian, and Juanru Li

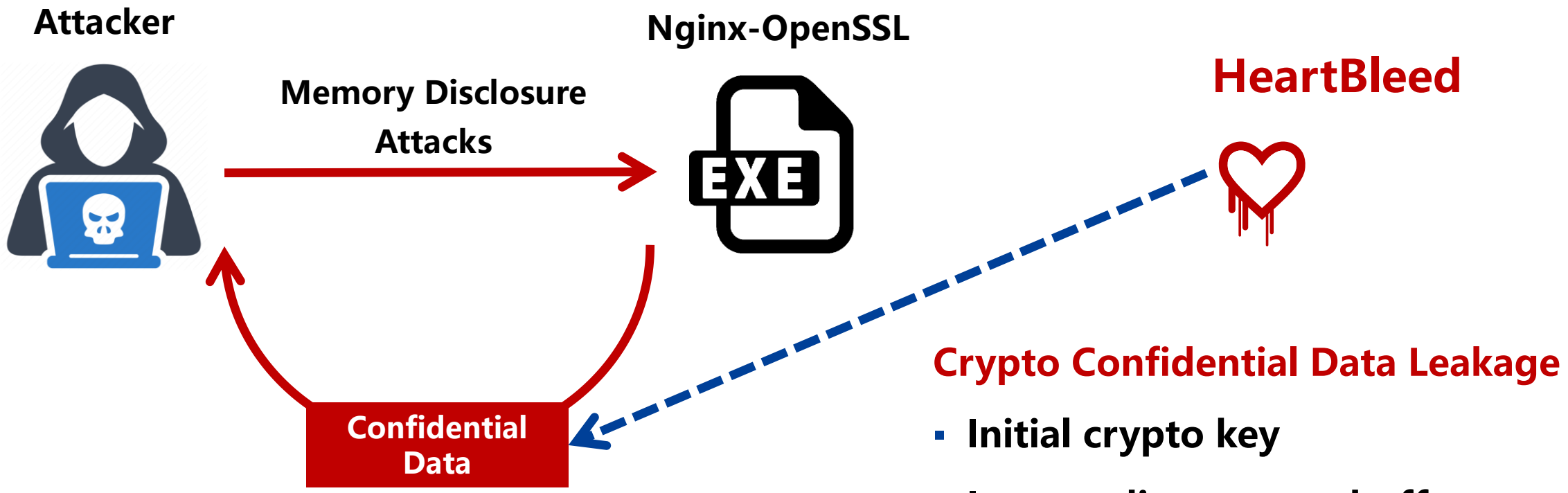
IEEE S&P 2022

SHANGHAI JIAO TONG UNIVERSITY

UC RIVERSIDE

UNSW AUSTRALIA

Memory Disclosure Attacks



Requirement for an Accurate Protection

```
1   void encrypt(void *ciphertext, void *plaintext,
2                char *passwd) {
3       roundkey rkk;
4       hashstring(passwd, keyblock);
5       xrijndaelKeySched(keyblock, 256, 256, rkk);
6       xrijndaelEncrypt(ciphertext, plaintext, rkk);
7   }
```

Secret Non-secret

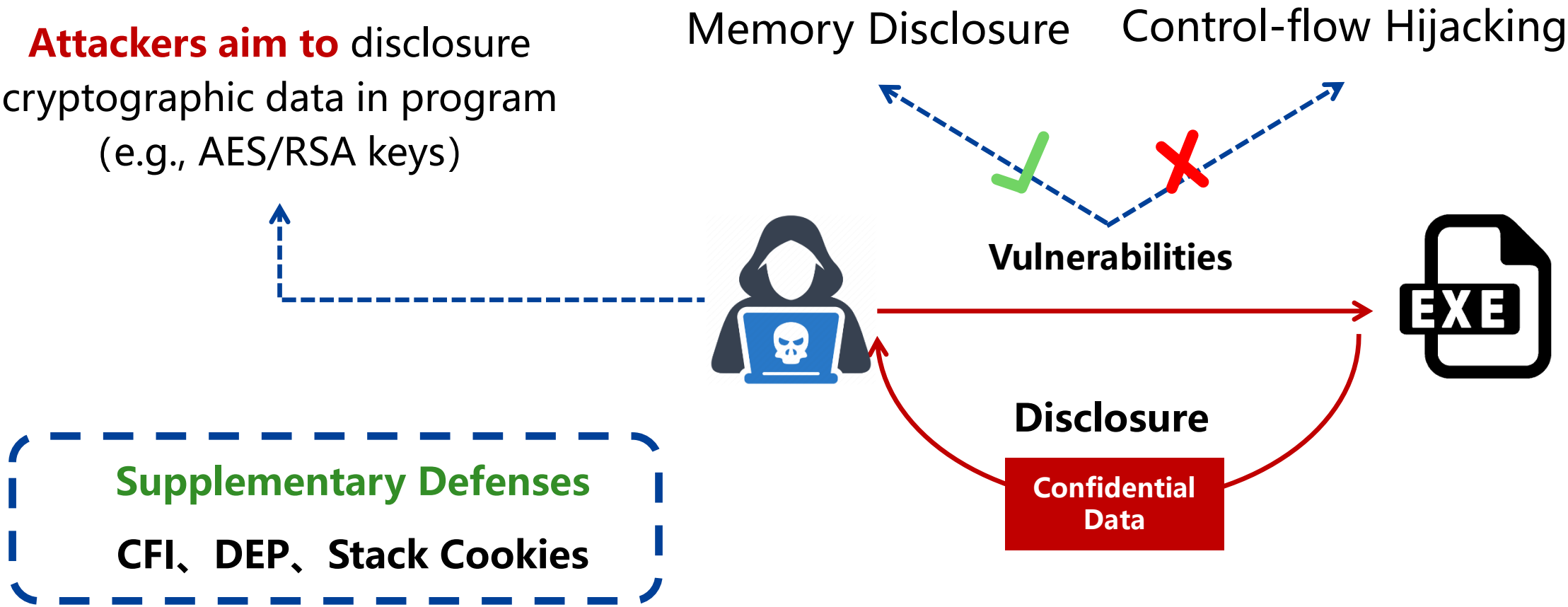
Listing 1: An example of secret propagation in ccrypt

Find all crypto secrets

Exclude non-secrets

Threat Model

Attackers aim to disclosure cryptographic data in program (e.g., AES/RSA keys)



Supplementary Defenses

CFI、DEP、Stack Cookies

Solution

Challenges

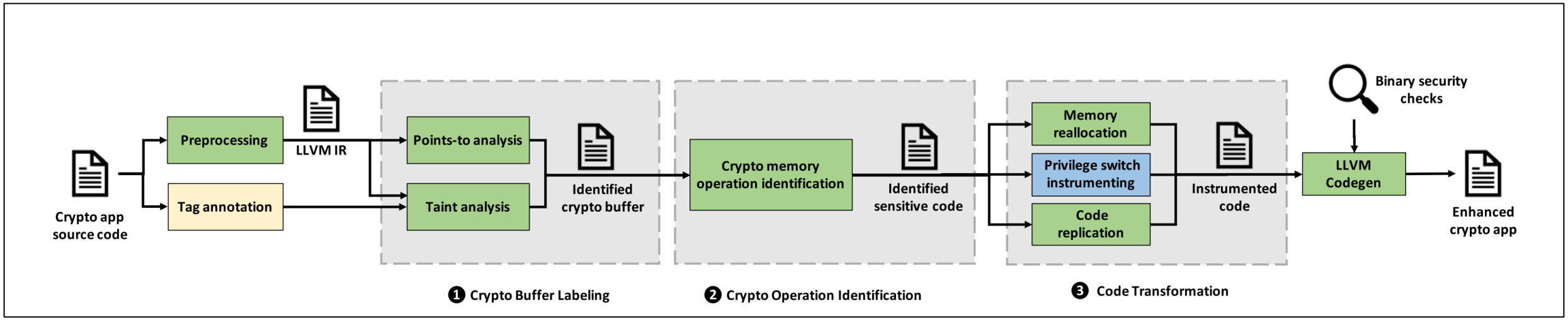
- Accurately label crypto memory buffers and related operations
- Appropriately isolate crypto data from unauthorized memory accesses
- Implement the isolation without changing the execution model

CryptoMPK

- Analysis-driven Sensitive Data Labeling
- Hardware-supported Domain Isolation
- Compiler-assisted Code Transformation

Workflow of CryptoMPK

1. Preprocessing code and manually annotating initial crypto secrets
2. Automatically labeling crypto buffers
3. Automatically identifying crypto operations
4. Automatically transforming source code into MPK-protected, secure binaries



Crypto Buffer Labeling

Manual Annotation

```
1   void do_encrypt() {
2     #pragma tainter cryptotag(passwd)
3     __cryptotag_fdf0f8a65855a52b(passwd);
4     #pragma tainter mxortag(plaintext)
5     __mxortag_fdf0f8a65855a52b(plaintext);
6     encrypt(ciphertext, plaintext, passwd);
7   }
```

Listing 3: An example of initial tag annotating

mxor tag

crypto tag

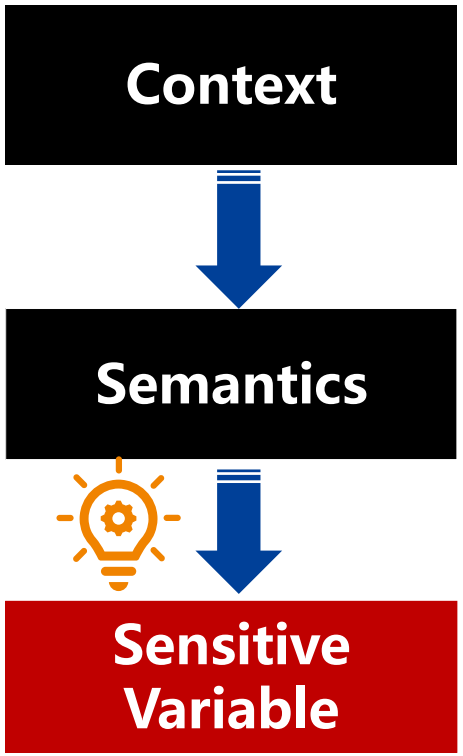
Automated Labeling

```
1   void encrypt(void *ciphertext, void *plaintext,
2                char *passwd) {
3       roundkey rkk;
4       hashstring(passwd, keyblock);
5       xrijndaelKeySched(keyblock, 256, 256, rkk);
6       xrijndaelEncrypt(ciphertext, plaintext, rkk);
7   }
```

Secret Non-secret

Listing 1: An example of secret propagation in ccrypt

Context-sensitive Analysis



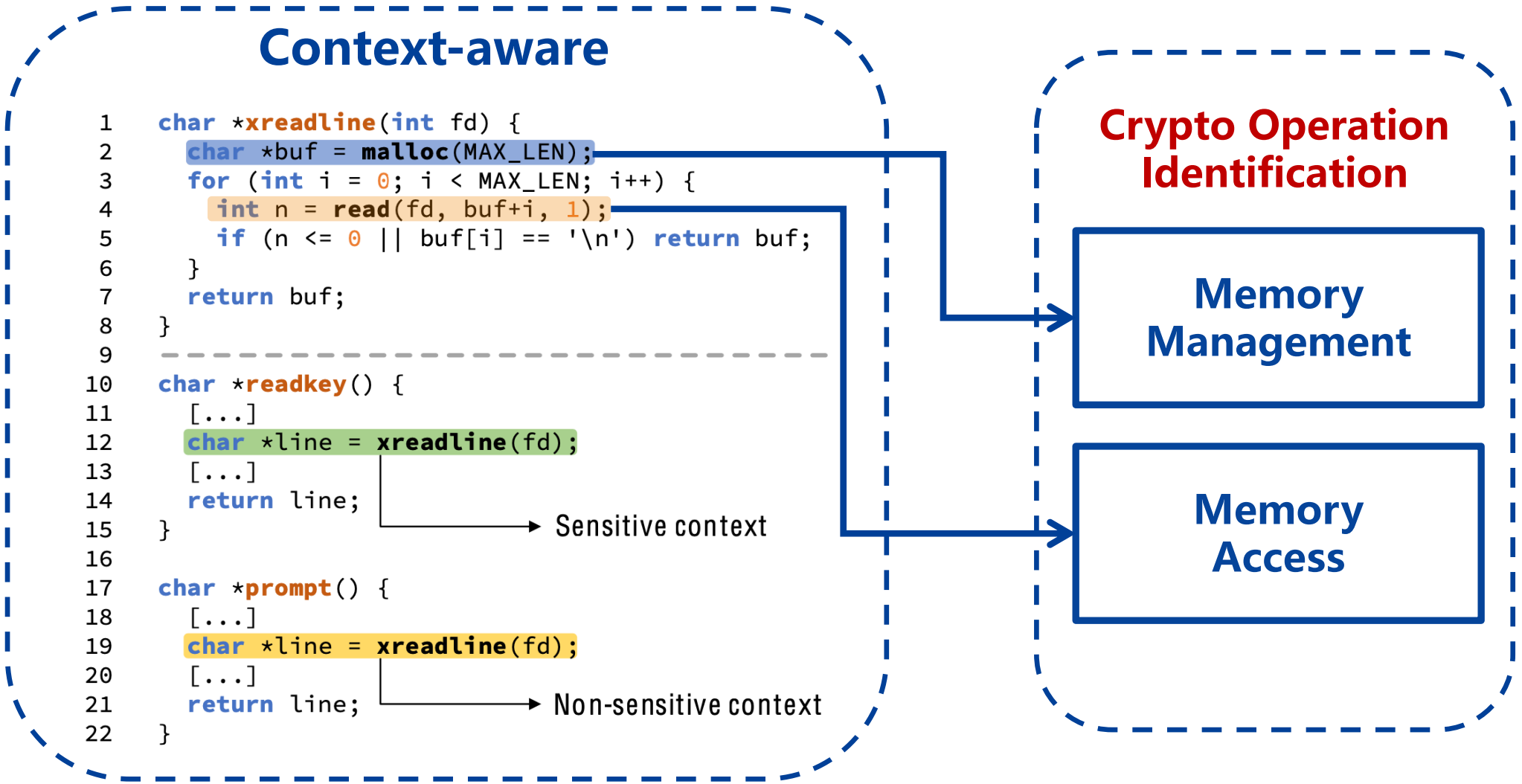
Context-aware

```
1   char *xreadline(int fd) {
2     char *buf = malloc(MAX_LEN);
3     for (int i = 0; i < MAX_LEN; i++) {
4       int n = read(fd, buf+i, 1);
5       if (n <= 0 || buf[i] == '\n') return buf;
6     }
7     return buf;
8   }
9   ----------------------------------------
10  char *readkey() {
11    [...]
12    char *line = xreadline(fd);
13    [...]
14    return line;
15  }
16
17  char *prompt() {
18    [...]
19    char *line = xreadline(fd);
20    [...]
21    return line;
22  }
```

Sensitive context

Non-sensitive context

Crypto Operation Identification



Code Transformation

Memory Relocation

Privilege Switch Instrumentation

Code Replication

```
1   char *xreadline(int fd) {
2     char *buf = malloc(MAX_LEN);
3     for (int i = 0; i < MAX_LEN; i++) {
4       int n = read(fd, buf+i, 1);
5       if (n <= 0 || buf[i] == '\n') return buf;
6     }
7     return buf;
8   }
9   ----------------------------------
10  char *readkey() {
11    [...]
12    char *line = xreadline(fd);
13    [...]
14    return line;
15  }
16
17  char *prompt() {
18    [...]
19    char *line = xreadline(fd);
20    [...]
21    return line;
22  }
```

crypto domain

non-crypto domain

Listing 2: An example of context-dependent memory allocation

Code Transformation

Memory Relocation

Privilege Switch Instrumentation

Code Replication

Intel MPK

```
[Non-crypto Operations]
xor ecx, ecx                                  Prologue
xor edx, edx
mov eax, PKRU_CRYPTO_DOMAIN
WRPKRU
[Crypto Operations]
xor ecx, ecx                                  Epilogue
xor edx, edx
mov eax, PKRU_NORMAL_DOMAIN
WRPKRU
[Non-crypto Operations]
```

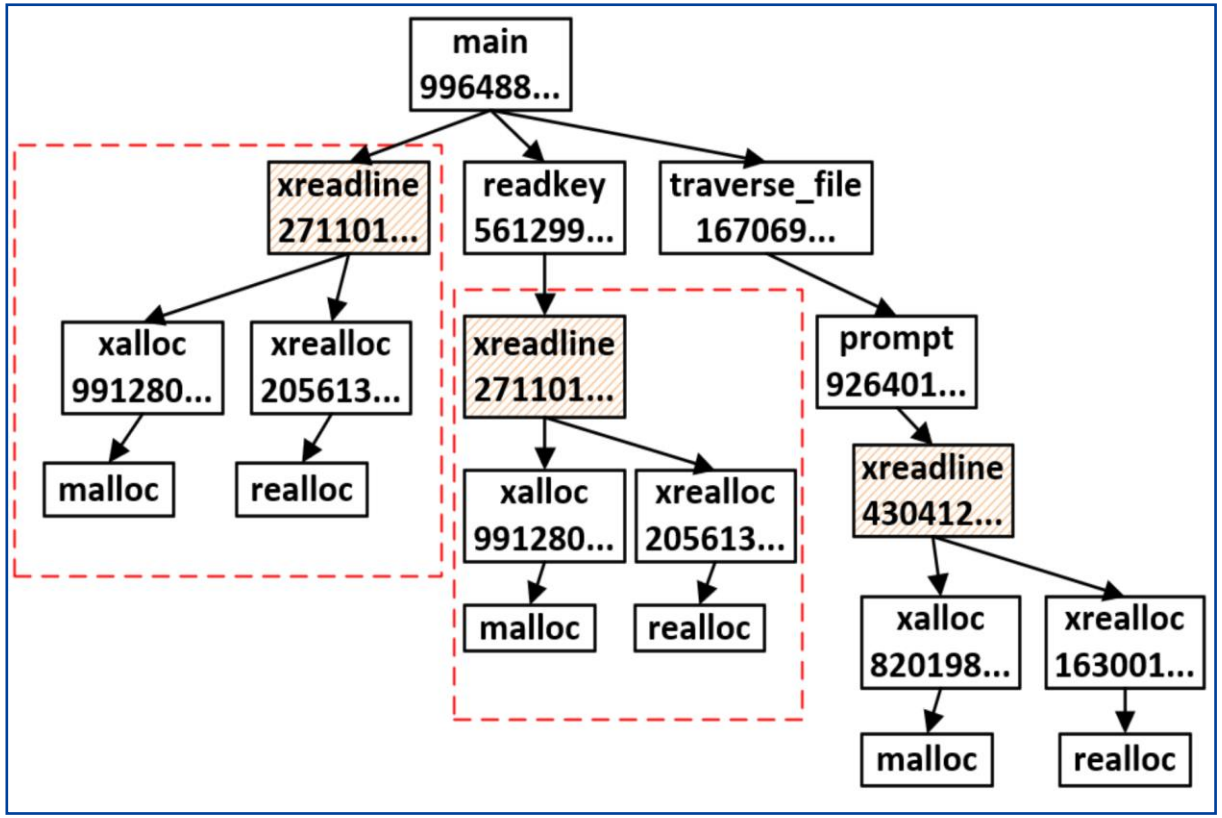
Fig. 2: MPK privilege switching at binary code level

Code Transformation

Memory Relocation

Privilege Switch Instrumentation

Code Replication

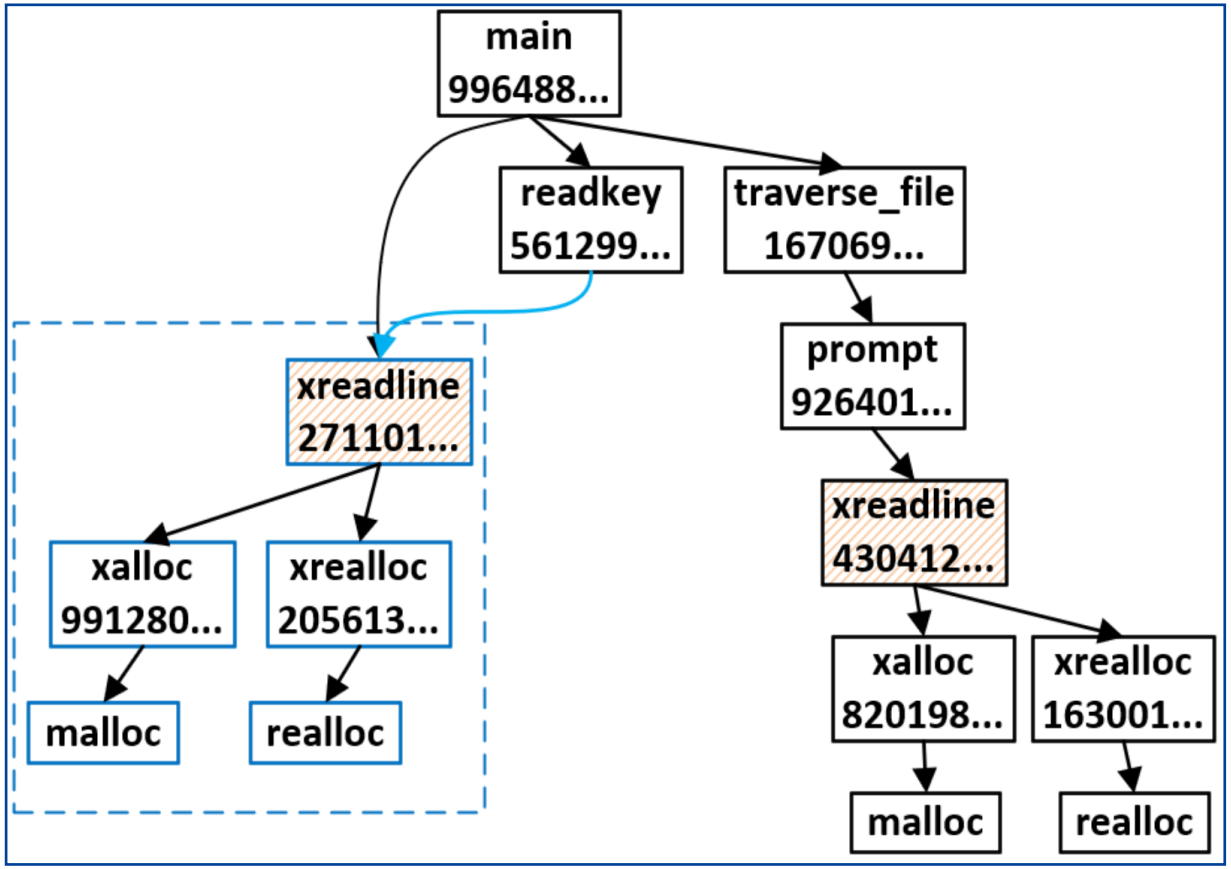


Code Transformation

Memory Relocation

Privilege Switch Instrumentation

Code Replication



Experiment Setup

Library	Version	Line of Code
libcrypt	2.27	3,016
libhydrogen	0.2.0	2,597
libsodium	1.0.18	32,279
OpenSSL	1.0.2u	163,734
Application	**Version**	**Line of Code**
Apache	2.4.43	235,956
ccrypt	1.11	21,558
OpenSMTPD	6.0.3p1	39,713
Nginx	1.17.10	137,092
vsftpd	3.0.3	15,052

TABLE IV: The evaluated programs and crypto libraries.

Labeling Accuracy

TABLE I: Results of buffer labeling, operation identifying, and code protection

Program	Tags	Total MemOps	Tainted MemOps (declassification)	Tainted MemOps	Total Functions	Crypto Functions	# of Protected Functions				
							Total (HF)	AP	OP	UP	RF
ccrypt	4+3	1049	829	838	70	61	22(19)	19	2	1	30
libsodium	1+1	6513	140	151	930	71	26(21)	25	0	1	31
libhydrogen	2+3	277	57	162	116	62	11(8)	11	0	0	17
Apache+OpenSSL	7+2	92779	0+2149	2528+2223	4879+11082	34+246	0+75(0+66)	75	0	0	177
Nginx+OpenSSL	7+2	70126	0+2149	4439+2223	1347+11082	24+246	0+75(0+66)	75	0	0	177
OpenSMTPD+crypt	2+1	14170	4+492	7+566	892+40	22+33	4+18(0+16)	22	0	0	19
vsftpd+crypt	3+1	9122	9+492	10+566	596+40	42+33	9+18(0+16)	27	0	0	19
Total	**39**	**194,036**	**6,321**	**13,713**	**31,074**	**874**	**296(212)**	**254**	**2**	**2**	**470**

HF: Hotspot functions; **AP**: accurately protected; **OP**: over protected; **UP**: under protected; **RF**: replicated functions for different contexts

Protection Effectiveness

Additionally Protected Buffers

- **SSL/TLS:** $\mathrm{RR} \equiv 2^{\mathrm{n}} \ (mod\ p)$
  - intermediate BIGNUM structs
- **AES:**
  - round key buffers (recovering main key)
- **Chacha20-Poly1305:**
  - intermediate array (guess nonce offset)
- **Crypt-SHA512:**
  - Intermediate buffer (recovering password)

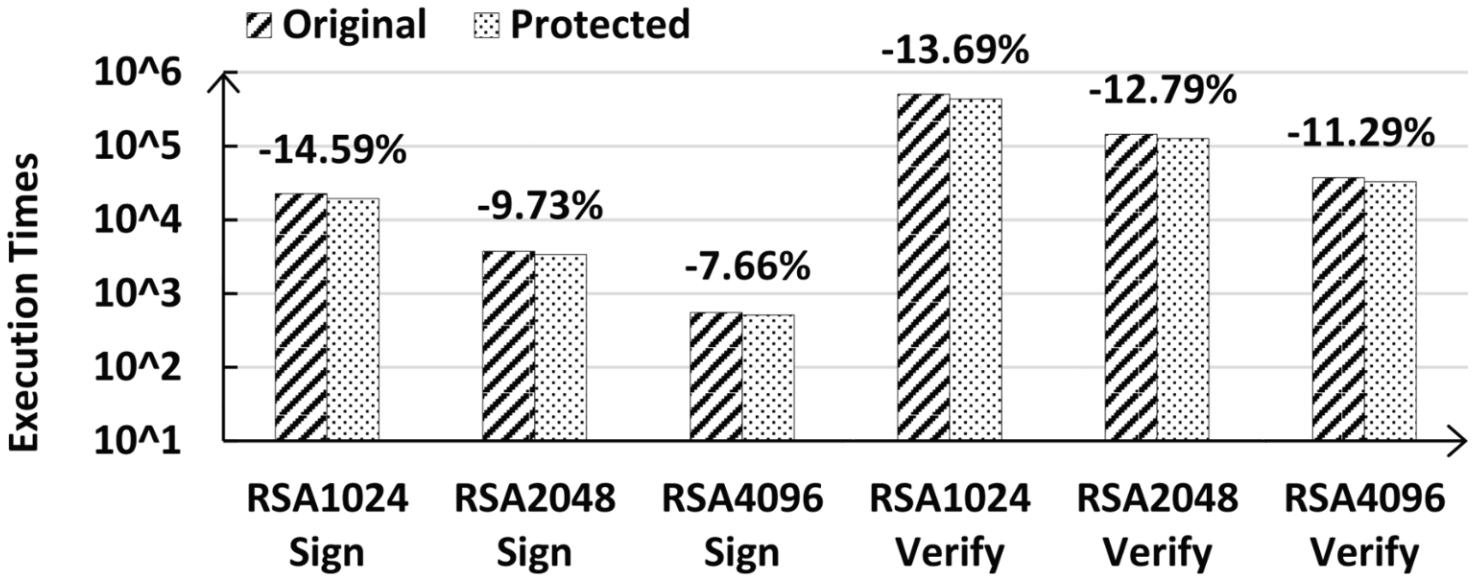
Protection against Memory Disclosures:

- **CVE-2011-4576:**
  - uninitialized variable vulnerability
- **CVE-2014-0160:**
  - HeartBleed
- **CVE-2016-2176:**
  - out-of-bounds reads
- **CVE-2017-9798:**
  - use-after-free vulnerability
- **CVE-2018-16845:**
  - out-of-bounds reads

Performance

- **OpenSSL Speed (RSA)**
- OpenSSL Speed (AES-128-GCM)
- Server
- Code Bloating
- Analysis Costs

Benchmark slow down: from 7.13% to 13.69%



(a) OpenSSL speed test: RSA sign/verify in 10s

Performance

OpenSSL Speed (RSA)

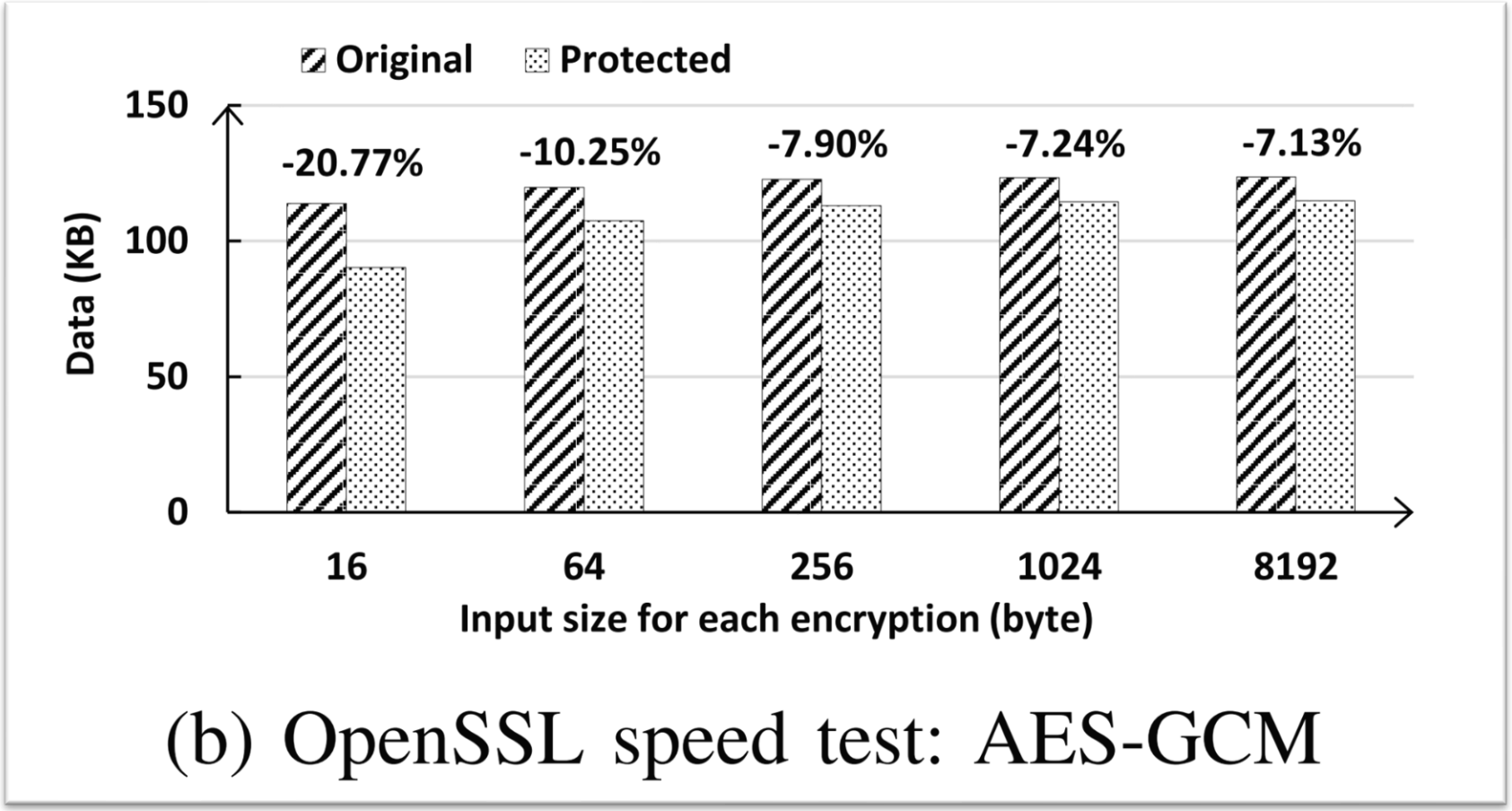
OpenSSL Speed (AES-128-GCM)

Server

Code Bloating

Analysis Costs

Benchmark slow down: from 7.13% to 12.79%



(b) OpenSSL speed test: AES-GCM

Performance

OpenSSL Speed (RSA)

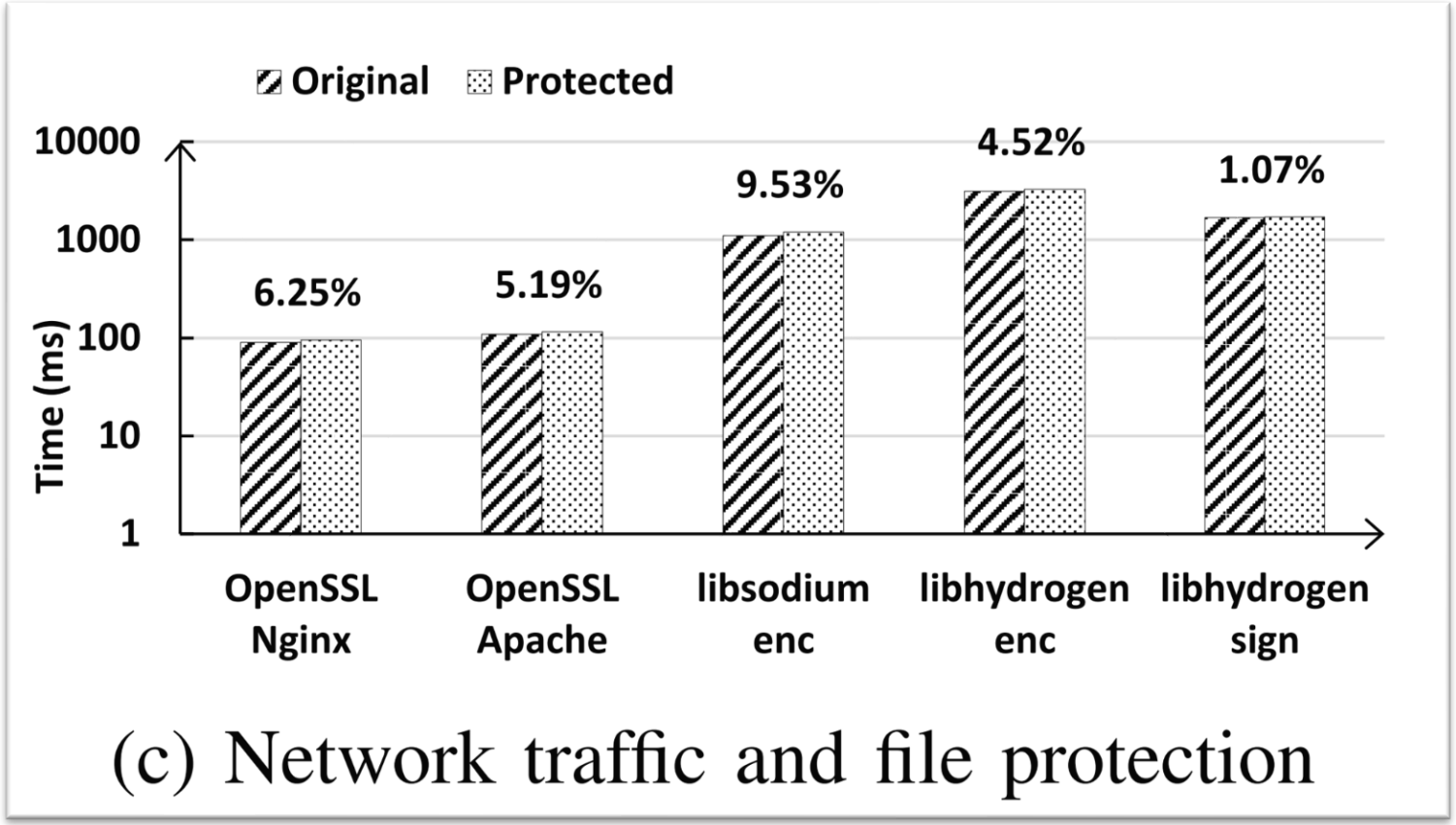
OpenSSL Speed (AES-128-GCM)

Server

Code Bloating

Analysis Costs

Runtime overhead: less than 10%



(c) Network traffic and file protection

Performance

OpenSSL Speed (RSA)

OpenSSL Speed (AES-128-GCM)

Server

Code Bloating

Analysis Costs

Large Targets : bloating ratios are less than 20%

Small code base : increased size is less than 50KB

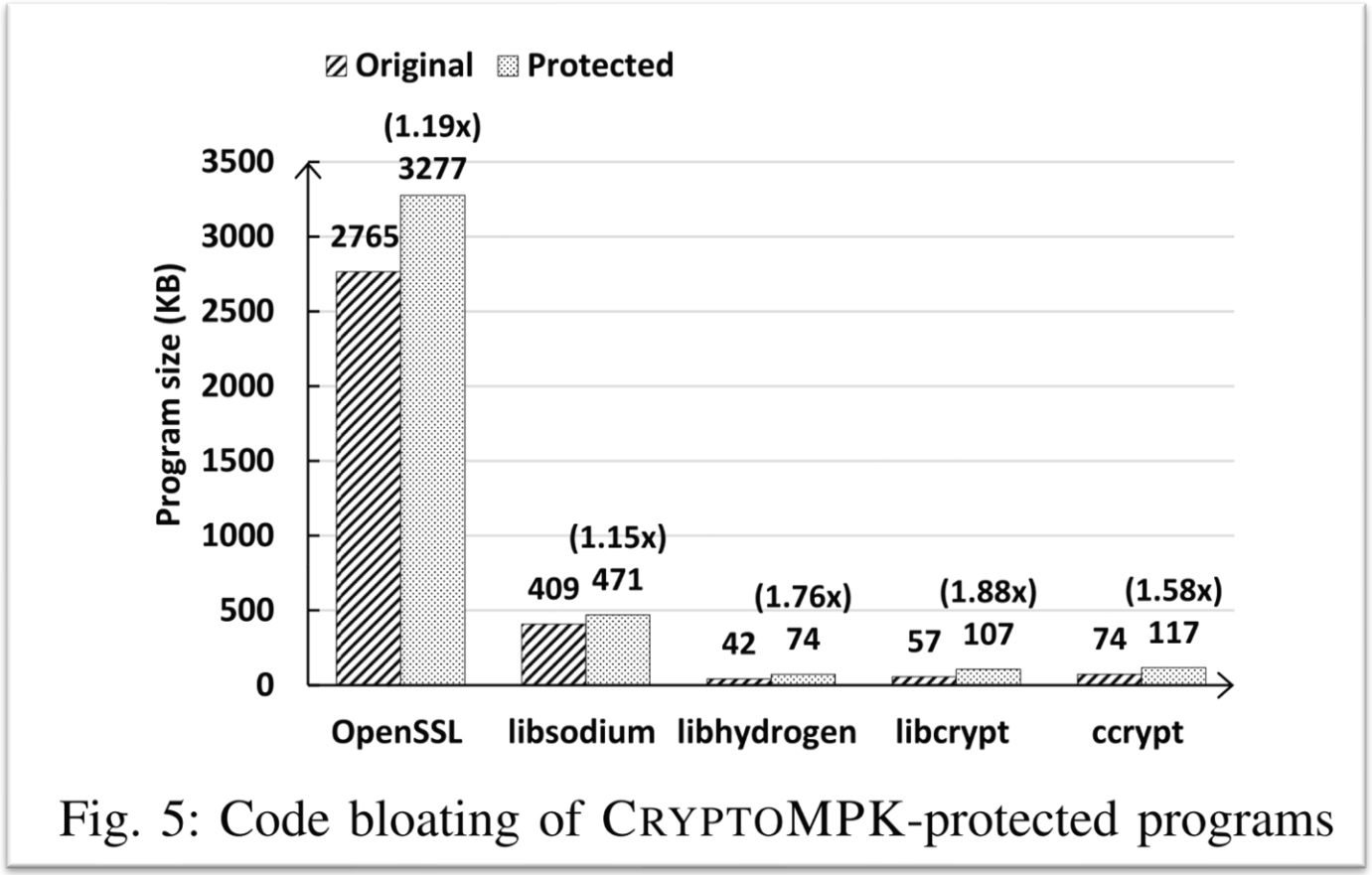


Fig. 5: Code bloating of CRYPTOMPK-protected programs

Performance

OpenSSL Speed (RSA)

OpenSSL Speed (AES-128-GCM)

Server

Code Bloating

Analysis Costs

analysis time: less than 20 minutes

TABLE II: Analysis performance for each experiment target

Case	SBL(s)	SMOI(s)	Total(s)
libhydrogen	0.31	0.41	0.72
ccrypt	0.97	1.02	1.99
OpenSMTPD+libcrypt	1.41	1.02	2.43
libsodium	2.04	1.81	3.85
vsftpd+libcrypt	244.39	52.91	297.30
{Apache, Nginx}+OpenSSL	638.29	540.94	1179.23

SBL : time of crypto buffer labeling;
SMOI : time of sensitive memory operation identification

Conclusion

- **Accurate Protection**
  - Automated and comprehensive crypto secrets tracking (crypto-aware, context sensitive)
- **Efficient Isolation**
  - Intel MPK enhanced fine-grained in-process isolation
- **Easy-to-use**
  - build and deploy protected binary executables with widely used toolchain (LLVM) for non-expert developers
- **Low overhead**
  - a maximum 9.53% runtime overhead for widely used crypto applications

More details at: https://cryptompk.code-analysis.org/

Q & A