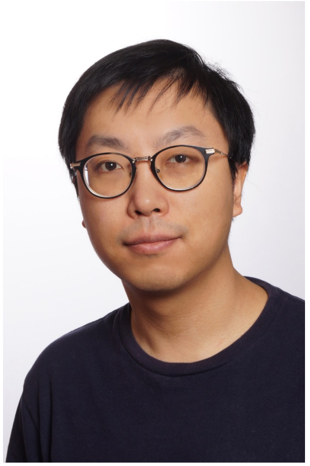
# CacheQL: Quantifying and Localizing Cache Side-Channel Vulnerabilities in Production Software

**Yuanyuan Yuan**, Zhibo Liu, Shuai Wang
*The Hong Kong University of Science and Technology*

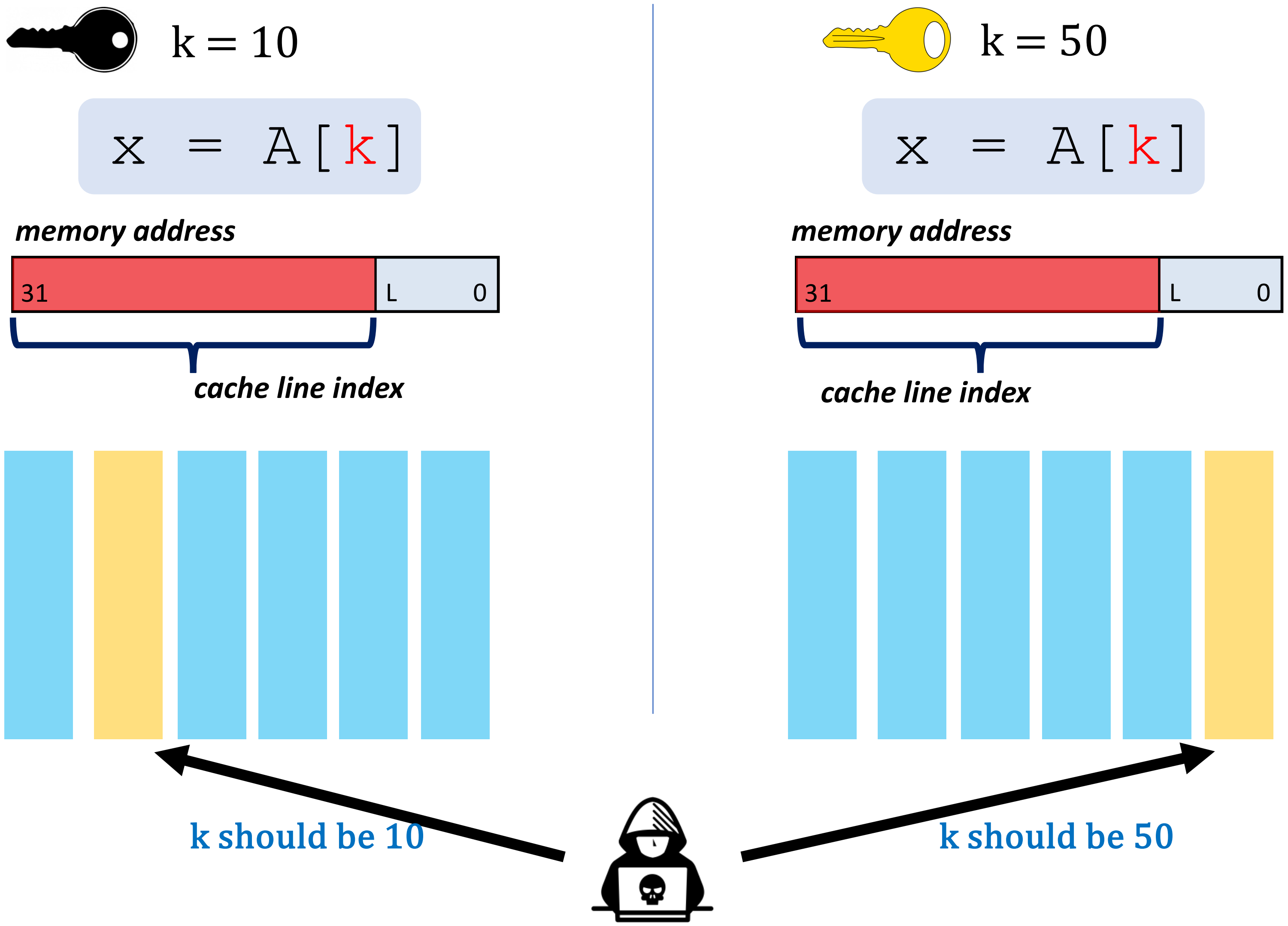




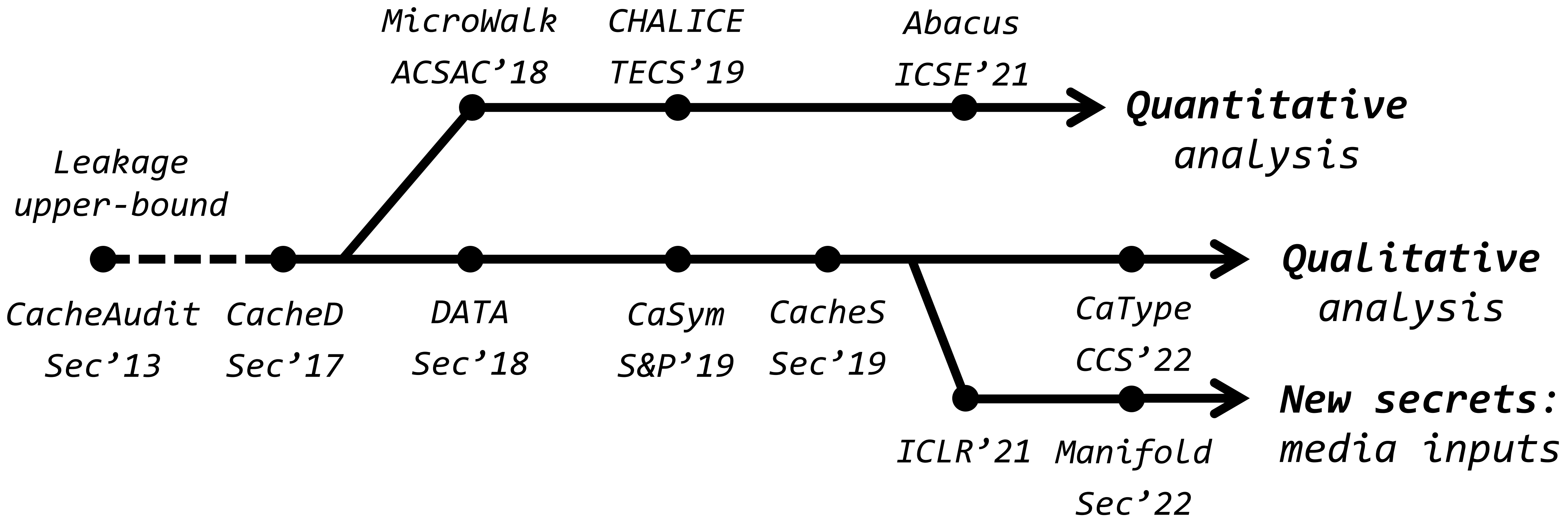
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

# Cache Side Channel Leakage




**Secret dependence in *data access* or *control branch*.**




*How to design a fully-fledged side channel detector?*

❶ Execution trace & Real-world attack logs

❷ Deterministic & Non-deterministic observations

❸ Analyze executables

❹ Qualitative vs. Quantitative analysis

❺ Localize leakage sites

❻ Different secrets: key & media data

❼ Scalability: whole-program analysis

❽ Explicit & Implicit information flow

❶ Execution trace & Real-world attack logs

❷ Deterministic & Non-deterministic observations

❸ Analyze executables

❹ Qualitative vs. Quantitative analysis

❺ Localize leakage sites

❻ Different secrets: key & media data

❼ Scalability: whole-program analysis

❽ Explicit & Implicit information flow

*CacheQL is designed to fulfill all eight requirements!*

# Quantification

Mutual Information (MI)

$$I(K;O) = H(K) - H(K|O)$$

Secrets

Side channel Observations

# Quantification: MI

Mutual Information (MI)

$$I(K;O) = H(K) - H(K|O)$$

$K$: secrets;

$O$: side channel observations

```
//s: evenly {0,1,2,3}
int s;
array a[1], b[1];
// leak log4 = 2 bits
if(s == 0)
   a[0] = 1;
// leak log(4/3) bits
else
   b[0] = 1;
```

$$H(K) = -\log\frac{1}{4} = 2 \text{ bits}$$

$o = a[0]$

$$H(K|o = a[0]) = -\log 1 = 0$$

$$I(K|o = a[0]) = 2 \text{ bits}$$

# Quantification: MI → Conditional Probability

Estimate MI is challenging:

*1) Computing Cost; 2) Estimation Error; 3) Coverage Issue*

Observe $o^*$ when the program is taking $k^*$

$T$: $o^*$ and $k^*$ co-occur

($o^*$ can be observed given another $k$)

$F$: $o^*$ and $k^*$ occur independently

$$\text{MI} = \frac{p(F)}{p(T)} \frac{p(T|k^*, o^*)}{p(F|k^*, o^*)}$$

$p(T|k^*, o^*)$ ← Conditional probability (CP)

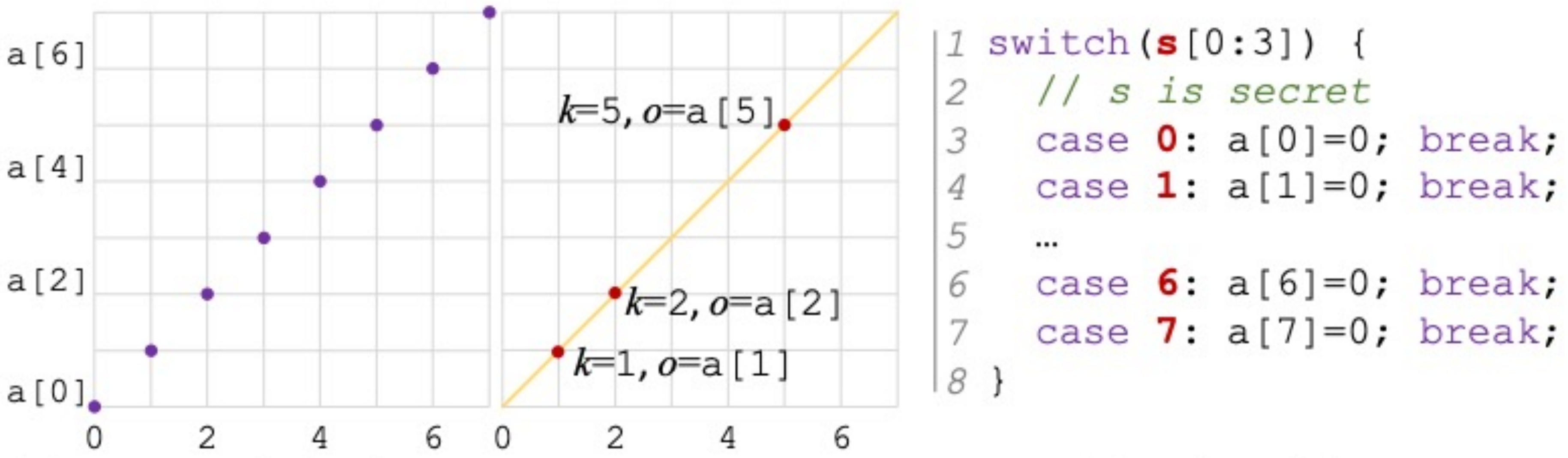
$\frac{p(F)}{p(T)}$ → Constant

$p(F|k^*, o^*)$ → $1 - p(T|k^*, o^*)$

# Quantification: Conditional Prob. (CP)

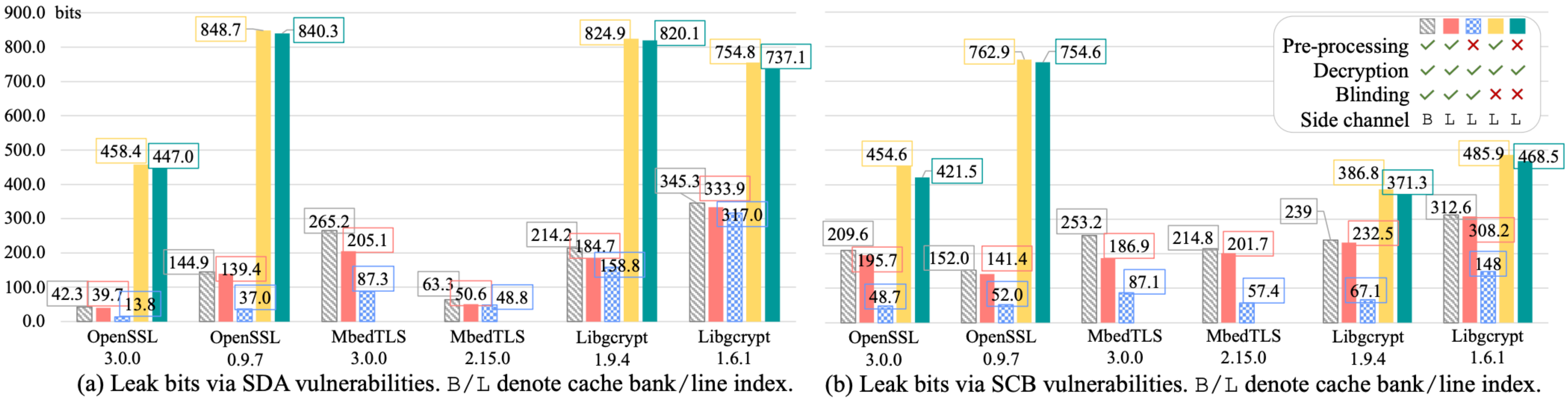
- Estimating CP is a one-time effort.
- CP reflects:

1) How many records in $o^*$ are affected by $k^*$
2) To what extent $k^*$ affects each record in $o^*$



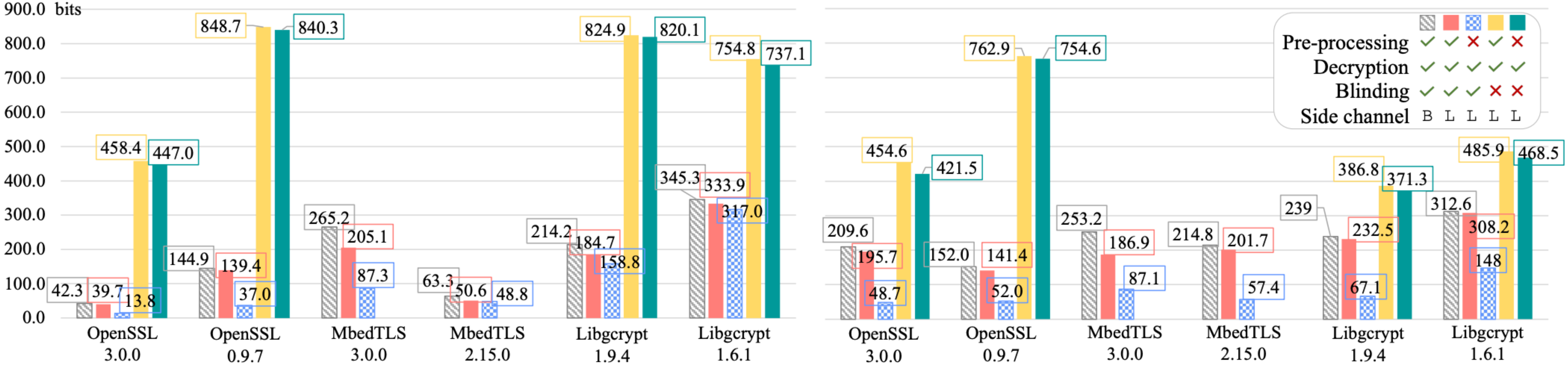

(a) program behaviors. (b) inferred behaviors by CP. (c) vulnerable program.

# Quantification Results: RSA




- Different granularities of observation
- Different stages: pre-processing vs. decryption
- Enabling vs. Disabling crypto blinding
- Old vs. New versions

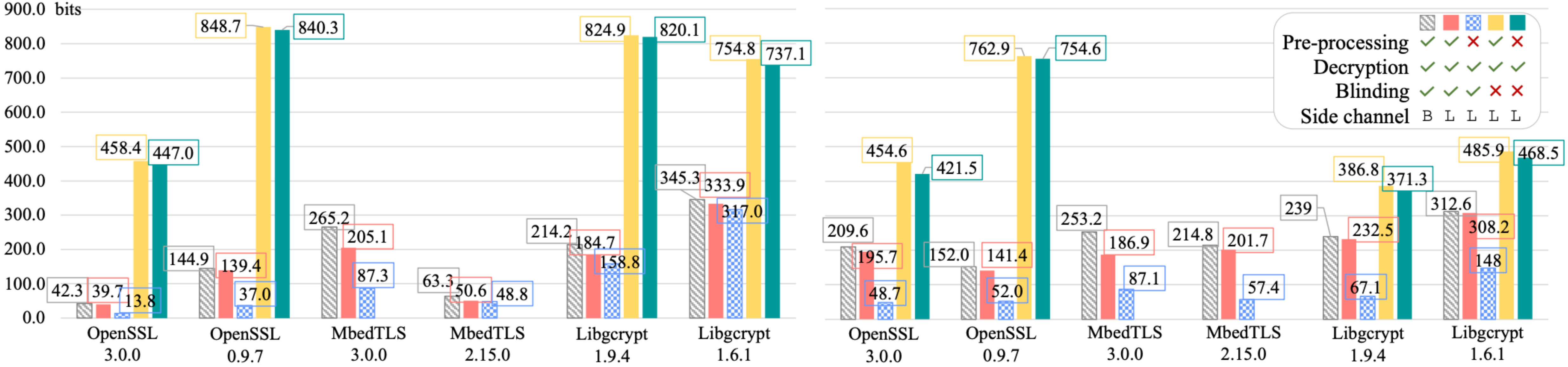
# Quantification Results: RSA




(a) Leak bits via SDA vulnerabilities. B/L denote cache bank/line index. (b) Leak bits via SCB vulnerabilities. B/L denote cache bank/line index.

- More fine-grained observation → More leaks
- Blinding can significantly reduce the leaks
- New versions usually have less leaks

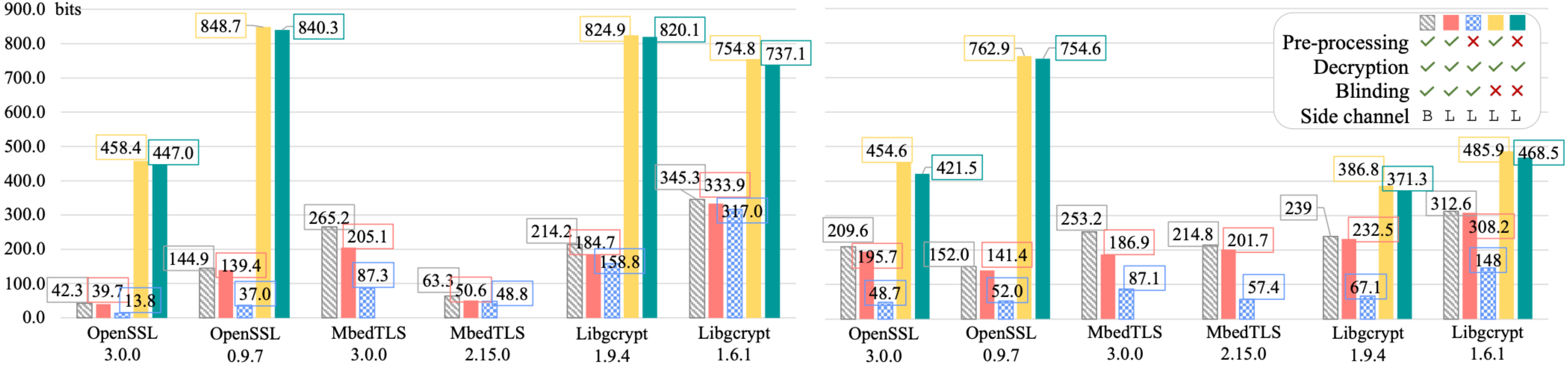
# Quantification Results: RSA




(a) Leak bits via SDA vulnerabilities. B/L denote cache bank/line index. (b) Leak bits via SCB vulnerabilities. B/L denote cache bank/line index.

- Considerable leaks in pre-processing modules
  1) encode/decode the read keys
  2) BIGNUM initialization

# Quantification Results: RSA




(a) Leak bits via SDA vulnerabilities. B/L denote cache bank/line index.

(b) Leak bits via SCB vulnerabilities. B/L denote cache bank/line index.

New vs. Old versions (reduced):

1) more constant-time impl.; 2) different computation routines.

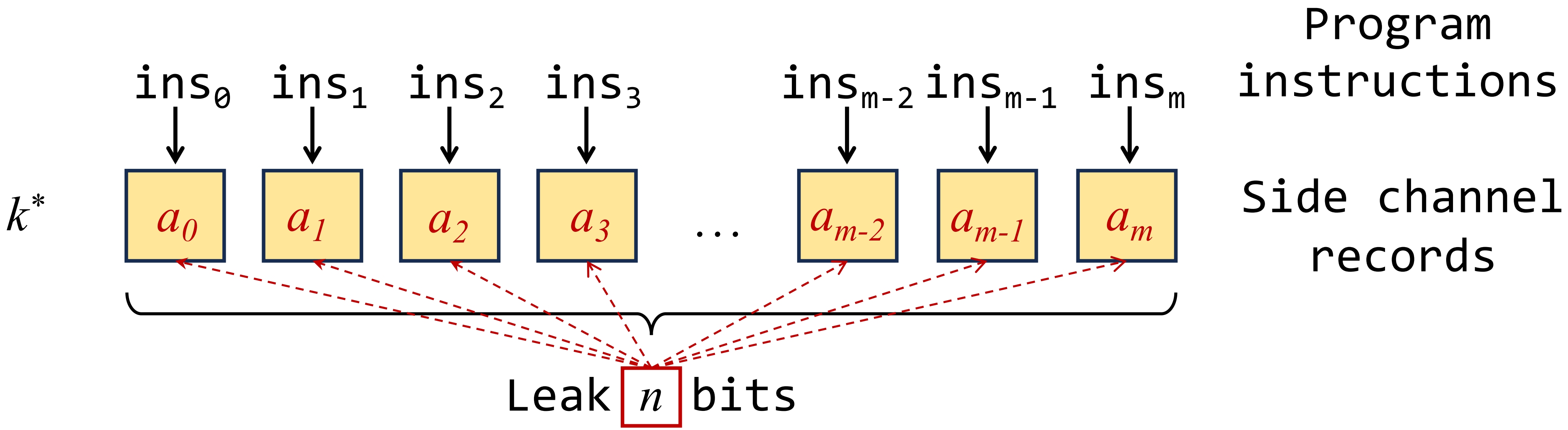
New vs. Old versions (increased):

1) different computation routines

# Localization: Shapley Value

## Apportion the quantified leaks:

**Definition 1** (Leakage Apportionment). *Given total n bits of leaked information and m program points covered on the Pin-logged trace, an apportionment scheme allocates each program point $a_i$ bits such that $\sum_{i=1}^{m} a_i = n$.*



Localization: Shapley value

Shapley value computation:

- **Exponential Cost:** $\mathcal{O}(2^{|o|})$; $|o|$: *100K~1M*

Reduced to ~hundreds magnitude

1) Not all records are correlated;

2) Many records do not contribute to the leaks.

# Localization Results: RSA

The latest versions (by the time of writing)
of OpenSSL 3.0, MbedTLS 1.9, Libgcrypt 2.1.

***A few Hundreds of new leakage sites.***

Many of them are in the pre-processing modules

↓

The ***first*** time being analyzed due to our ***high scalability***

# Localization Results: RSA

*Full list: sites.google.com/view/cache-ql*

Five categories

Ⓐ Leaking secrets in **Pre-processing**
Ⓑ Leaking secrets in **Decryption**
Ⓒ Leaking **leading zeros**
Ⓓ Leaking secrets via **explicit** information flow
Ⓔ Leaking secrets via **implicit** information flow

# Localization Results: RSA

## Pre-processing: *decode the read key*

```
int hextonibble(char s) {
  if(s >= '0' && s <= '9')
    return s - '0';
  if(s >= 'A' && s <= 'F')
    return 10 + s - 'A';
  if(s >= 'a' && s <= 'f')
    return 10 + s - 'a';
}
```

```
static gpg_err_code_t
do_vsexp_sscan(gcry_sexp_t *ret,
          char *buf, size_t len) {
  struct make_space_ctx c;
  for(char *s=buf; len; len--) {
    *c.p++ = hextonibble(*(s++));
  }
}
```

Ⓐ Leaking secrets in **Pre-processing**

Ⓓ Leaking secrets via **explicit** information flow

# Localization Results: RSA

## Decryption:

*BIGNUM computation*

```
int BN_mod_exp_mont(BIGNUM *rr, BIGNUM *a,
                    BIGNUM *p, BIGNUM *m, ) {
   // table of variables obtained from 'ctx'
  BIGNUM *val[TABLE_SIZE];
  int bits = BN_num_bits(p);
  int w = BN_window_bits_for_exponent_size(bits);
  int wstart = bits - 1;
  for(;;) {
    int wvalue = 1;
    int wend = 0;
    for(int i = 1; i < w; i++)
      if(BN_is_bit_set(p, wstart - i)) {
        wvalue <<= (i - wend);
        wvalue |= 1;
        wend = i;
      }
    bn_mul_mont_fixed_top(r, r, val[wvalue >> 1]);
  }
}
```

implicit information flow “taints” **wvalue** and **wend**

Ⓑ Ⓒ Ⓓ Ⓔ

## Decryption:

*BIGNUM computation*

```
#define BN_window_bits_for_exponent_size(b)\
                ((b) > 671 ? 6 : \
                 (b) > 239 ? 5 : \
                 (b) >  79 ? 4 : \
                 (b) >  23 ? 3 : 1)

int bn_mul_mont_fixed_top(BIGNUM *r,
                          BIGNUM *a, BIGNUM *b) {
  if(a == b)
    bn_sqr_fixed_top(tmp, a)
  else
    bn_mul_fixed_top(tmp, a, b)
}
int BN_is_bit_set(BIGNUM *a, int n) {
  int i = n / BN_BITS2;
  int j = n % BN_BITS2;
  if(a->top <= i) return 0;
  return (int)(((a->d[i]) >> j) & 1;
}
```

Ⓑ Ⓒ Ⓓ

## Pre-processing:

*BIGNUM initialization*

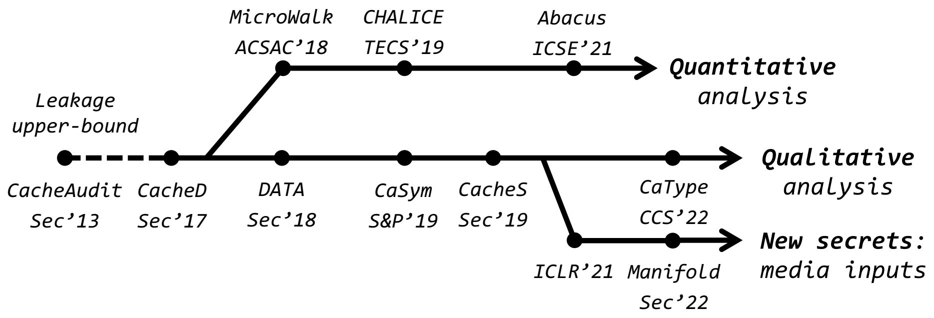
```
BIGNUM *BN_bin2bn(int len,
  char *s, BIGNUM *ret) {
  // s is secret
  for ( ; len && *s == 0; s++) {
    // skip leading zeros
    len --;
  }

  n = len;
  if (n == 0) {
    ret->top = 0;
    return ret;
  }
  i = ((n - 1) / BN_BYTES) + 1;
  ret->top = i;
  /* top is the "size" of a
  BIGNUM in later computing */
  return ret;
}
```

**len** is tainted via implicit information flow here.

Ⓐ Ⓒ Ⓓ Ⓔ

# Summary




*How to design a full-fledged side channel detector?* 🤔

2

## Quantification: MI

Mutual Information (MI)

$I(K;O) = H(K) - H(K|O)$

$K$: secrets;

$O$: side channel observations

```
//s: evenly {0,1,2,3}
int s;
array a[1], b[1];
// leak log4 = 2 bits
if(s == 0)
  a[0] = 1;
// leak log(4/3) bits
else
  b[0] = 1;
```

$H(K) = -\log \frac{1}{4} = 2$ bits

$o = a[0]$

$H(K|o = a[0]) = -\log 1 = 0$

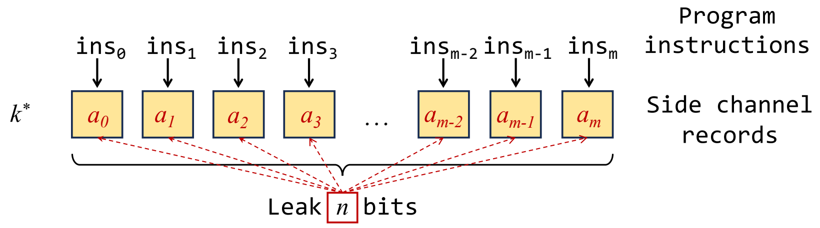
$I(K|o = a[0]) = 2$ bits

6

## Localization: Shapley Value

Apportion the quantified leaks:

**Definition 1** (Leakage Apportionment). *Given total $n$ bits of leaked information and $m$ program points covered on the Pin-logged trace, an apportionment scheme allocates each program point $a_i$ bits such that $\sum_{i=1}^{m} a_i = n$.*




13

## Localization Results: RSA

*Full list: [sites.google.com/view/cache-ql](sites.google.com/view/cache-ql)*

Five categories

Ⓐ Leaking secrets in **Pre-processing**
Ⓑ Leaking secrets in **Decryption**
Ⓒ Leaking **leading zeros**
Ⓓ Leaking secrets via **explicit** information flow
Ⓔ Leaking secrets via **implicit** information flow

16

# Thanks!

*Contact Yuanyuan for more information.*

https://yuanyuan-yuan.github.io

Paper

arxiv.org/pdf/2209.14952.pdf

Code

github.com/Yuanyuan-Yuan/CacheQL