Cryptographic Hardware and Embedded Systems
21st September, 2022

One Truth Prevails: A Deep-learning Based Single-Trace Power Analysis on RSA–CRT with Windowed Exponentiation

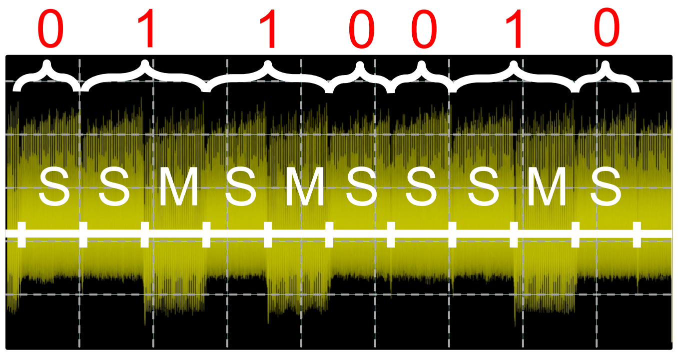
Kotaro Saito, Akira Ito, Rei Ueno, and Naofumi Homma
Tohoku University

Side-channel attack (SCA) on RSA

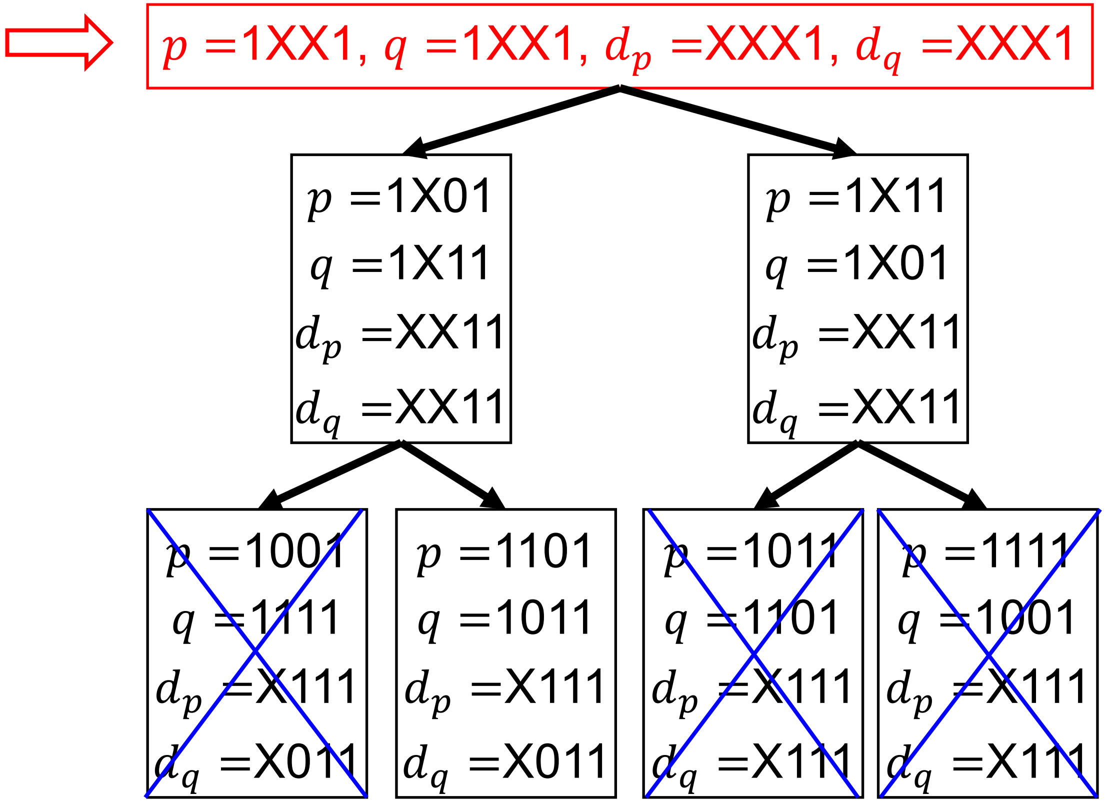
- SCA on modular exponentiation to estimate secret exponent
  - Traditional attacks distinguish squaring and multiplication to estimate exponent
  - Many studies have been devoted to how to accurately estimate exponent

Secret key leakage

Power/EM trace from RSA module



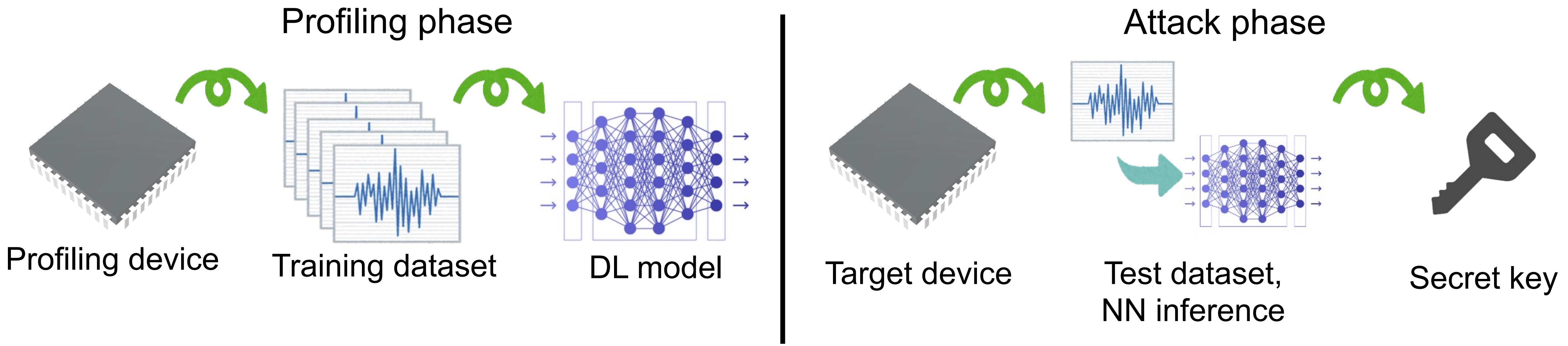
S: Squaring, M: Multiplication



- Partial key exposure attack
  - Secret keys estimated by SCA is not always correct/complete
  - Estimate full RSA–CRT secret key from partial/noisy leakage

Deep-learning based SCA (DL-SCA)

- Strongest profiled SCA which requires detailed assumption about leakage (compared to, for example, template attack)



- DL is very strong tool for SCAs, but researchers should still consider "what-to-learn" for key recovery
  - For symmetric cipher, it would be well established
  - But for public key decryption/signing, it varies depending on PKE

This work

- We present new deep-learning based single-trace power/EM analysis on state-of-the-art RSA–CRT implementations
  - New attack methodology for windowed exponentiation with dummy load
  - Leverage DL technique to estimate window values accurately
  - New partial key exposure attack algorithm designed for our situation
- Proposed attack achieves full-key recovery of 1,024-bit and 2,048 RSA–CRT implementations
  - Experimentally demonstrated on GMP implementation
    - Major multiprecision arithmetic library, used in cryptographic libraries
      - OpenSSL has option to adopt it in back-end
    - Can be used on embedded microcontroller
  - Applicable to (stand-alone) OpenSSL, Botan, and ligcrypt

RSA cryptosystem

Plaintext: m, Ciphertext: c,
Public key: (e, N), Secret key: (p, q, d),
$N = pq,\ ed = 1 \bmod N$

- Encryption:

$$c = m^e \bmod N$$

- Decryption:

$$m = c^d \bmod N$$

Nice math!

But how to implement it efficiently and securely?

Open-source RSA implementations

- Chinese remainder theorem (CRT) is used in decryption/signing

$$m_p = c^{d_p} \bmod p, m_q = c^{d_q} \bmod q, m = p^{-1}(m_q - m_p) \bmod q$$

Secret key: (p, q, d_p, d_q, p^{-1}),
$N = pq$, $ed = 1 \bmod N$, $d_p = d \bmod p$, $d_q = d \bmod q$

  - Yields 2–4 times faster computation
- Exponentiation algorithm mainly determines the performance
  - Open-source software (OSS) usually employ windowed exponentiation

Exponentiation algorithm	Relation to S–M seq.	Execution time	Examples of OSS adoption
Left-to-right binary	Exponent-dependent, and bijective to exponent.	Non-constant, slow	None
Square–multiply always Montgomery ladder	Exponent-independent	Constant, slow	None
Fixed window	Exponent-independent	Constant, fast	GMP, OpenSSL, WolfCrypt, etc.
Sliding window	Exponent-dependent, but not bijective to exponent	Non-constant, fast	libgcrypt, Gnu TLS, Bouncy Castle, etc.

Fixed window exponentiation $m = c^d \bmod N$

- Fastest constant-time exponentiation (let w be window size)
  - Precomputation: Calculate c^i for $i = 0$ to 2^w-1 and make table where $\mathrm{table}[i]$ = c^i
  - Main loop: Perform squaring w times and then multiplication with $\mathrm{table}[i]$
    - i is temporal window value

Example of $d = (110111100111)_2$ and $w = 4$

Temporal window value	1101	1110	0111
Square-Multiply sequence	SSSSM	SSSSM	SSSSM

$m \leftarrow (((1^2)^2)^2)^2 \times c^{13}$ $\quad m \leftarrow (((m^2)^2)^2)^2 \times c^{14}$ $\quad m \leftarrow (((m^2)^2)^2)^2 \times c^{7}$

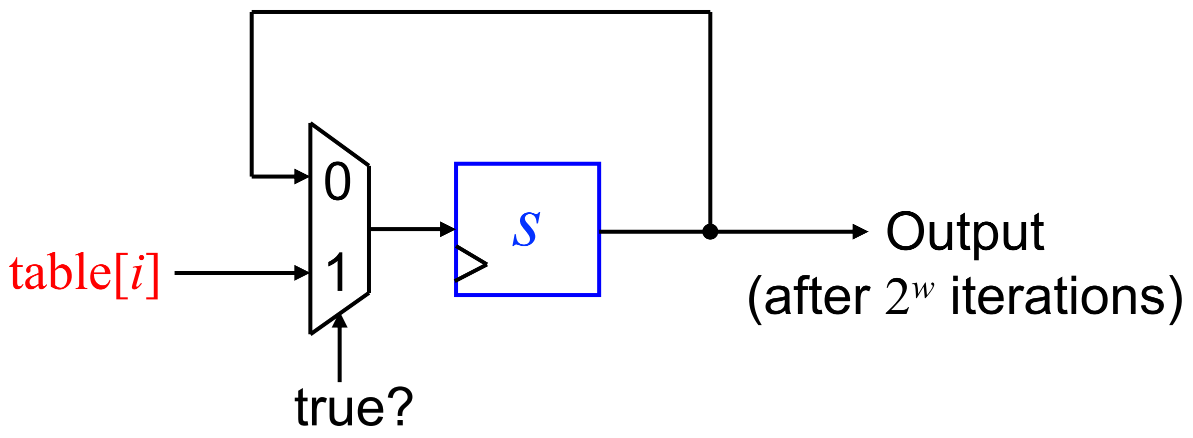
- SCA security?
  - Secure against SPA (square–multiply sequence is independent of exponent)
  - Leakage of temporal window values (loaded table address) yields key recovery
    - Prime+Probe, address bit DPA, collision analysis, etc.
    - *Leakage/security of operand loading should be considered*

Dummy load for hiding temporal window value

- Many windowed exponentiation in OSS employ dummy load
  - All operands in precomputation table are accessed in every multiplication

Operand loading in GMP
(addr is temporal window value)

Function LoadOperand(addr);
$s \leftarrow 0$;
for $i \leftarrow 0$ to $2^w - 1$ **do**
  mask $\leftarrow -(i = \text{addr})$;
  $s \leftarrow$ or(and(s, ¬mask), and(table[i], mask));
return s



Equivalent circuit representation

- Windowed exponentiation + dummy load seems sufficient to counter known remote timing/cache attacks

But how about power/EM analyses?

Overview of proposed attack

DL-SCA

- Step 1: Profiling (DL training phase)
- Step 2: Temporal window value estimation (DL attack phase)

Step 3: Partial key exposure attack

Step 1: Acquire traces for NN training and training NN

Step 2: Temporal value inference from attack traces by NN inference

・We develop very efficient methodology (specify what to learn) via in-depth analyses on implementation

Step 3：Full-key recovery via secret key leakage obtained in Step 2

・Estimated secret exponents may not be completely correct

・New partial key exposure attack dedicated to our methodology

Proposed methodology: *One truth prevails*

- An operand loading consists of one true load and $2^w - 1$ dummy loads
  - Value of register s is changed only when true load
    - Possibility of distinguishing true/dummy load by its physical side-channels
  - Order of true and dummy loads fully depends on temporal window value
    - True/dummy load sequence is one-hot coding of temporal window value

```
Function LoadOperand(addr);
s ← 0;
for i ← 0 to 2^w – 1 do
    mask ← – (i = addr);
    s ← or(and(s, ¬mask), and(table[i], mask));
return s
```

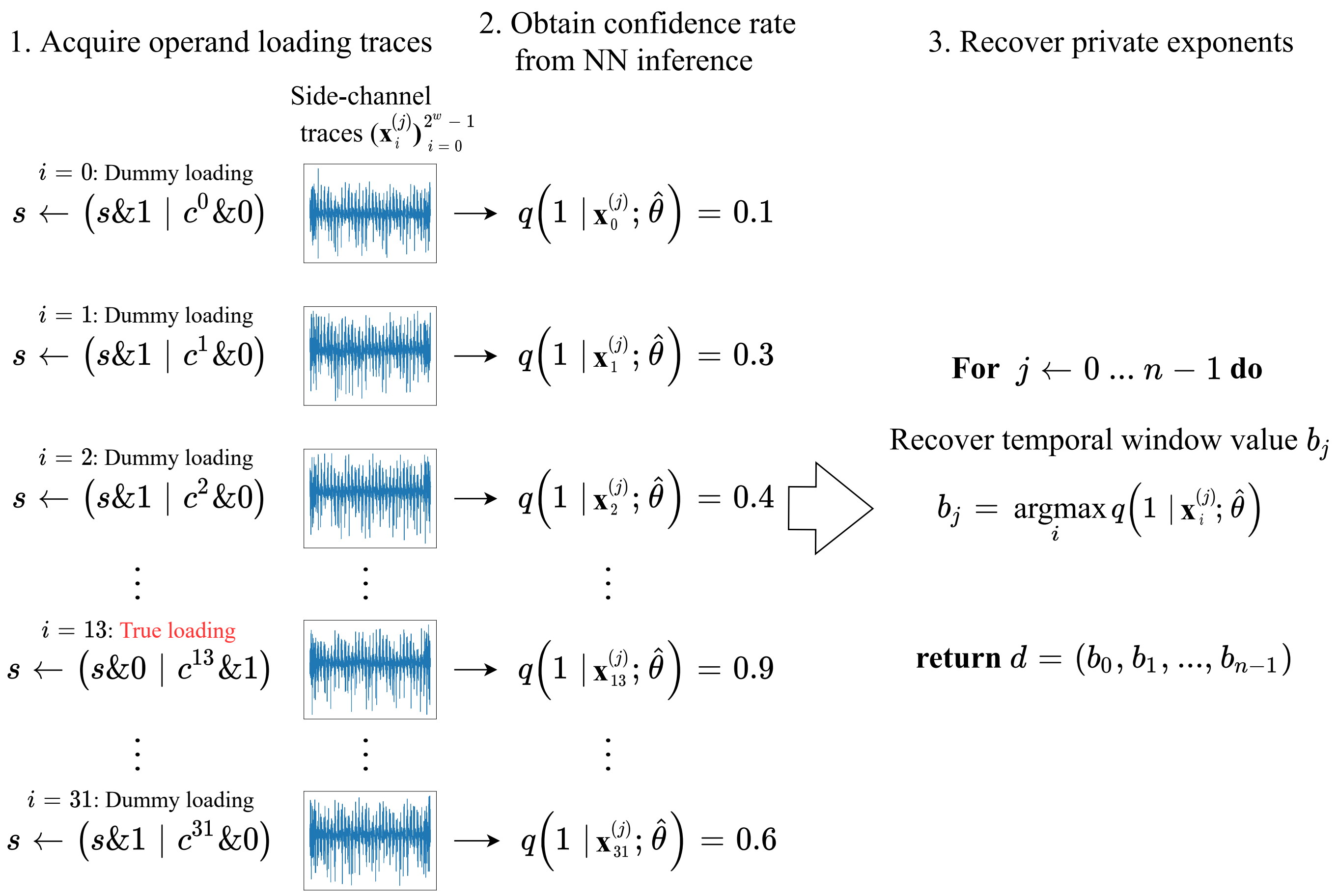
Example of $d = (110111100111)_2$ and $w = 4$

Temporal window value	1101	1110	0111
Square-multiply sequence	SSSSM	SSSSM	SSSSM
True/dummy load sequence	DDDDDDDDDDDDDTDD	DDDDDDDDDDDDDDTD	DDDDDDDTDDDD

Distinguishing true/dummy load yields temporal window value recovery

How to distinguish true/dummy load: DL-SCA

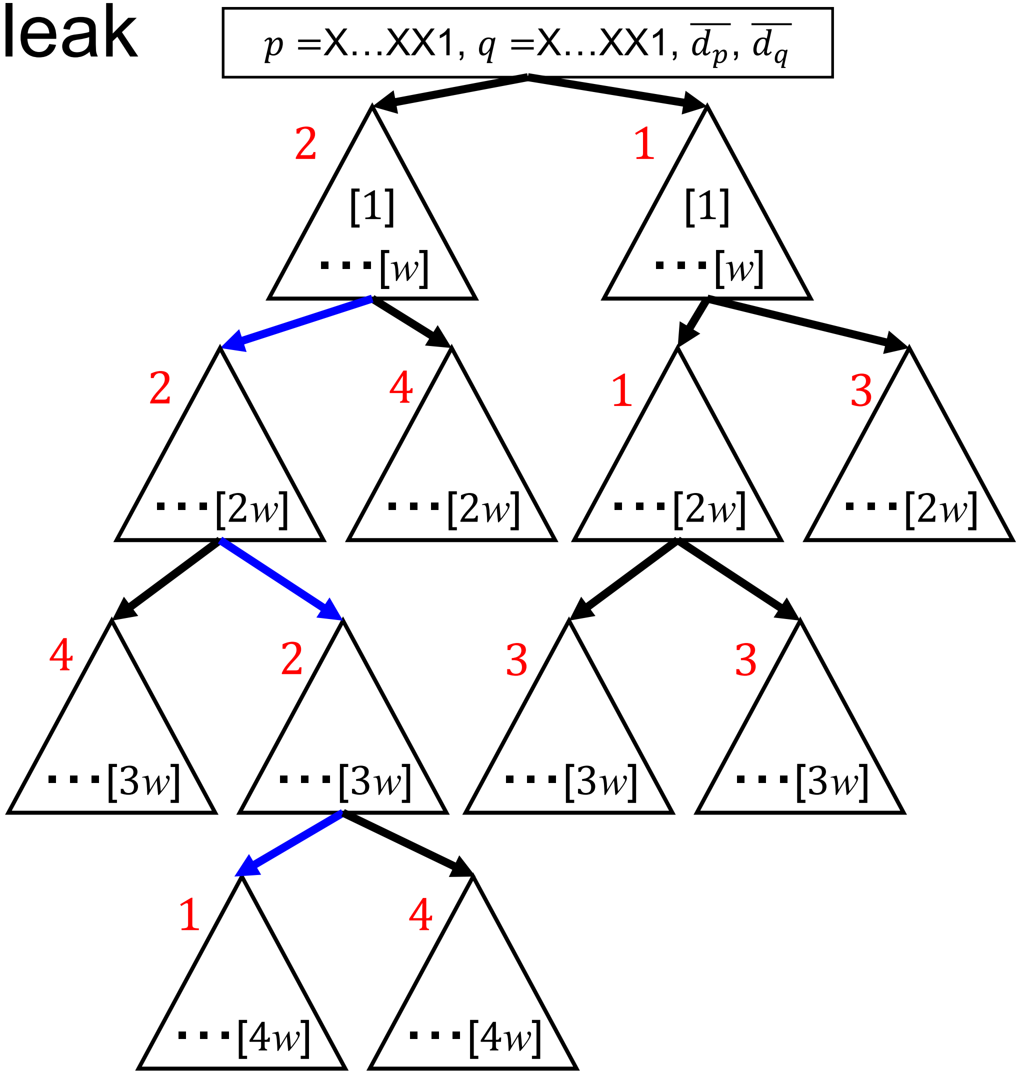
- Employ two-classification NN to distinguish true/dummy load
  - Training phase:
    - Train NN using traces labeled as true or dummy load (from profiling device)
  - Attack phase:
    - Perform 2^w two-classifications to distinguish true/dummy load
    - Estimate load operation with highest probability of true load as the true load (Take argmax of NN outputs)



- NN inference is reduced to two-classification from 2^w-classification
  - Improve NN accuracy and reduce learning cost, which yields efficient attack

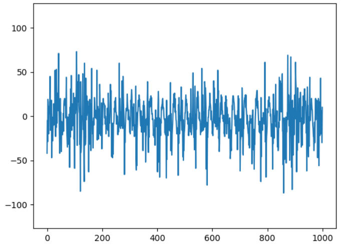
New partial key exposure attack

- Heninger–Shacham attack: Random bit leak
  - Inapplicable to our scenario
- Henecka et al.'s attack: Random bit flip
  - Computational cost grows exponentially by maximum length of consecutive bit errors
- Our attack: w-bit wise error
  - Utilize heuristics and priority deque to correct errors in w-bit wise manner
  - Heuristics determine cost of each key candidate due to inconsistent bit obtained from side-channels
    - Unlikely candidates are efficiently prone

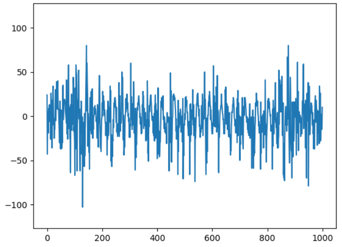


Experimental evaluation

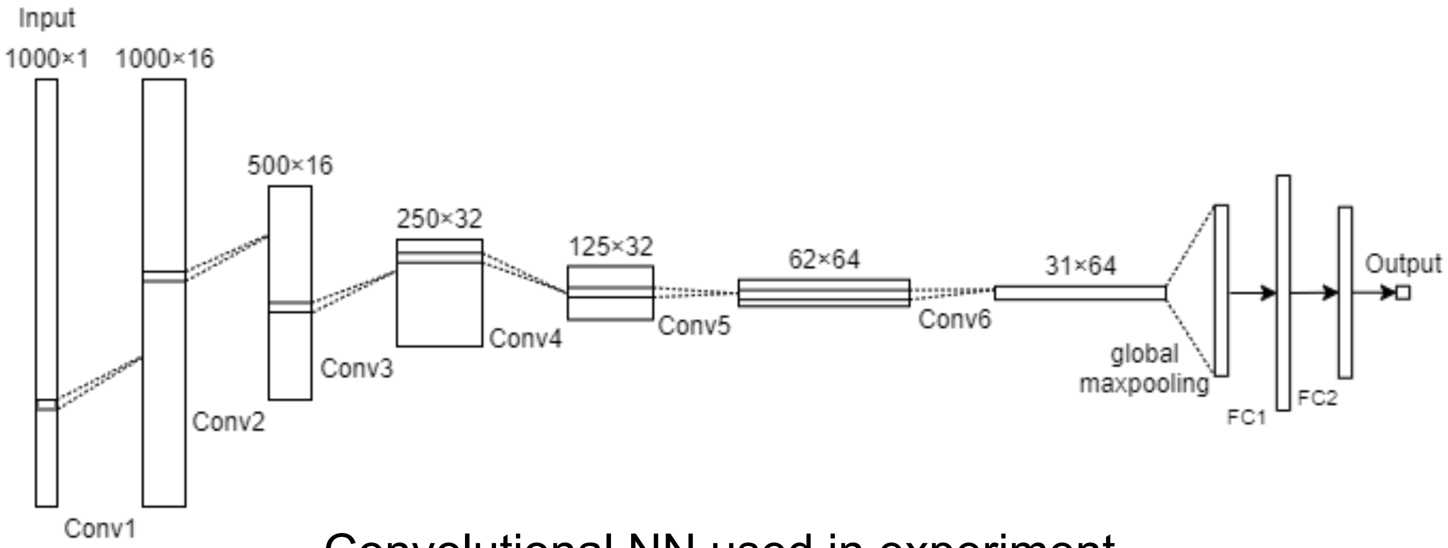
- Evaluate accuracy of temporal window value estimation on 1,024-bit RSA–CRT implementation with GMP
  - 1,024-bit RSA–CRT = 128 × 2 temporal window value estimations ($w = 4$)
  - Training trace dataset: 61,440,000 EM traces for true and dummy loads
  - Profiling and target device: ARM Cortex-M4 with 168 MHz frequency



EM trace of true load



EM trace of dummy load



Convolutional NN used in experiment
（6 convolutional layer followed by 2 fully connected layers）

Result (without partial key exposure attack)

- Evaluate test phase accuracy (attack success rate) using 24 different secret keys
  - We estimated 48 exponents, 48 × 128 temporal window values, and 48 × 128 × 16 true/dummy loads ($w = 4$)
  - Success rate is sufficient to break exponent-blinded RSA–CRT if multiple traces are available

Estimation accuracy

	True/dummy load	Temporal window value	Exponent
Proposed DL-SCA	99.94%	99.82%	79.17%
Template attack	79.17%	4.16%	0.00%
2^w-classification NN	N/A	11.53%	0.00%

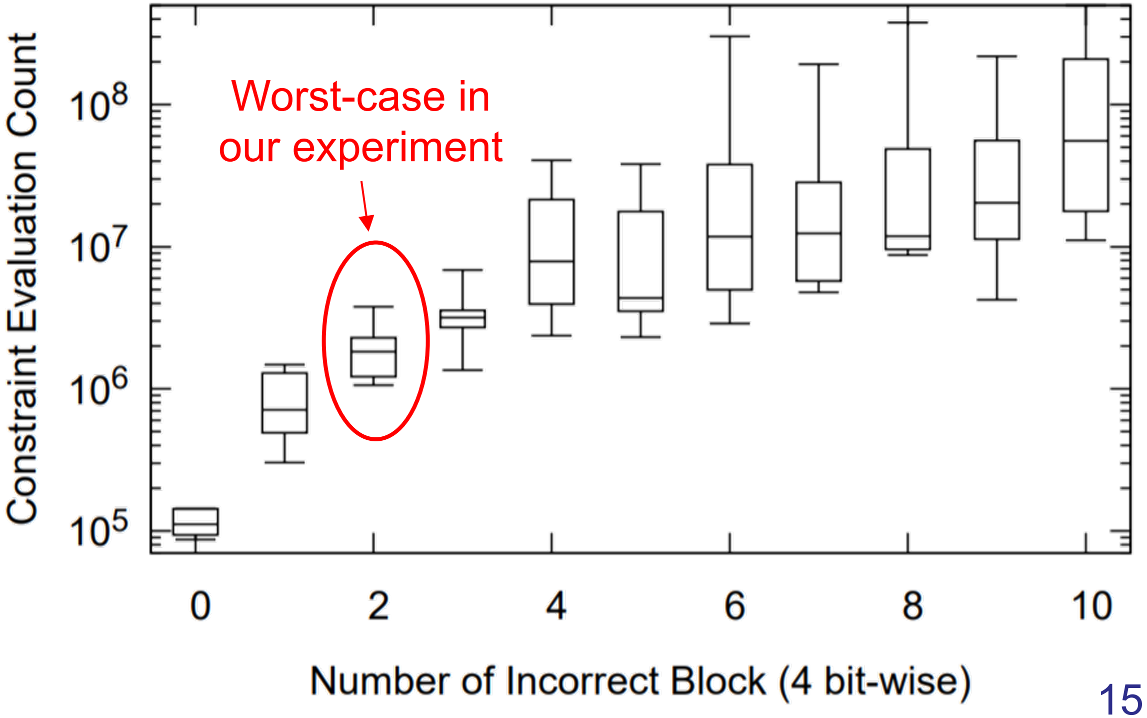
- Number of estimation errors is at most two

Frequency of # estimation errors in proposed DL-SCA

# Errors	0	1	2	3 >
Frequency	38	8	2	0

Overall success rate evaluation with partial key exposure attack

- Generate 100 random RSA–CRT secret keys with w-bit-wise errors and apply proposed partial key exposure attack
  - Simulate errors included in secret keys estimated by proposed DL-SCA
- Proposed attack can recover full key with 100% success rate
  - A few seconds when # errors is 2
  - A dozen of seconds when it is 10
  - Success rate of Henecka et al.'s attack was at most 80%
    - Our attack is well-calibrated for our DL-SCA (w-bit-wise error)



Concluding remarks

- New DL-SCA and partial key exposure attack on RSA–CRT
  - Applicable to practical implementations with windowed exponentiation and dummy load as hiding countermeasure
    - Utilized in, for example, GMP, OpenSSL, libgcrypt, and Botan
  - Experimentally confirmed full-key recovery of 1,024- and 2,048-bit RSA–CRT
  - Countermeasure: randomizations of initial register value and loading order (See our paper for concrete algorithm)
- DL can offer strong attacks even if detail of implementation is not known, but can achieve stronger attack if it is available

Existing DL-SCA on RSA/discrete log

- Many existing DL-SCAs focus on binary exponentiation
  - Left-to-right, Montgomery ladder, square–multiply always, etc.
  - Two-classification NN is used to directly estimate secret exponent
    - Its feasibility and accuracy have been studied
- Natural extension to windowed exponentiation: 2^w-classification NN
  - But its feasibility is unclear in general
    - 2^w-classification NN would be more difficult task than two-classification
    - Hiding countermeasure would make classification more difficult
      - 2^w-classification on WolfSSL EdDSA implementation in [WCBP20], but it neither employs hiding countermeasure nor protects operand loading
    - In our experiment, 2^w-classification NN achieved only 11.52% accuracy on Gnu MP implementation, which would be insufficient for key recovery