CSCI262 : System Security

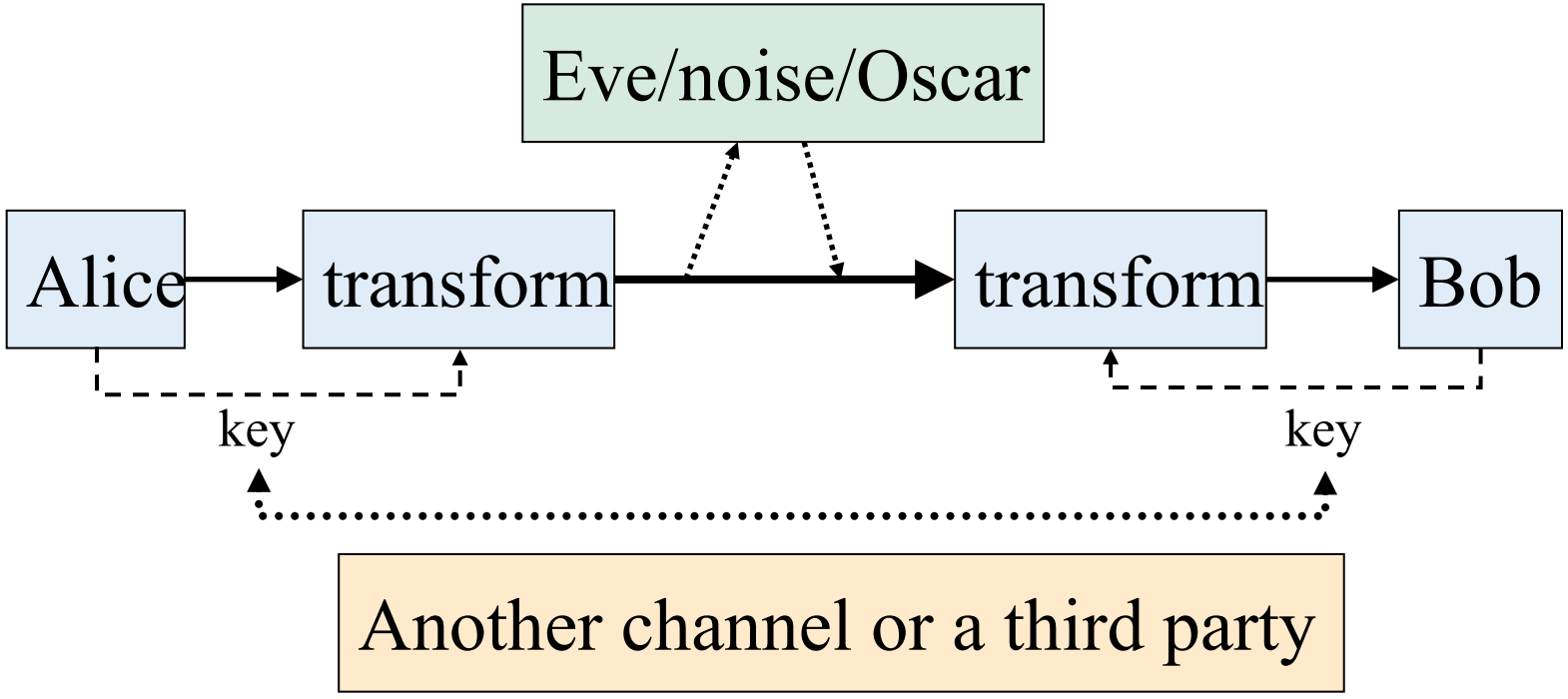
Side-Channel Attacks

Outline

- What are side channels attacks?
- Classes of side channel attacks.
- Fault Analysis.
- Protection/countermeasures.

What are side channels?

- It’s useful to know what side channels are first!
- There is a standard communication channel…
  - Side channels are another source of information, likely dependent on the implementation and likely the cryptographic device, rather than the ciphertext and plaintext/ciphertext relations.



- There is a standard abstract model of communication, we looked at it earlier.
- Side channels are really any information that is not represented by that abstract standard model.
- There has been a fair bit of research into side channels.

What are side channel attacks?

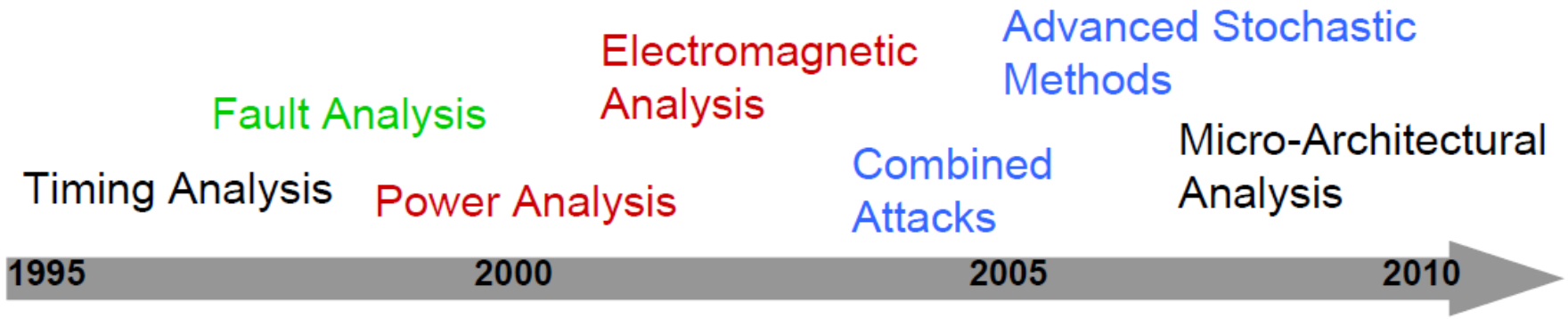
- Attacks based on the side channel information, so typically on the physical implementation!
  - They can be passive or active, or both.
- There is some documentation that suggests such attacks occurred as early as 1943.
  - Electromagnetic side channel in rotor key generator.
- However it wasn’t until the mid 1990’s that work was published demonstrating the possibility of using side channels.

History of SCA – The smart card world I

(1) P. Kocher: Timing analysis on implementations of DH, RSA, DSS, and other systems, 1995/96.
(2) D. Boneh, R. DeMilo, R. Lipton: On the importance of checking cryptographic protocols for faults, 1996/97.
(3) A. Lenstra: Memo on RSA signature generation in the presence of faults, 1996/97.
(4) E. Biham, A. Shamir: Differential fault analysis of secret key cryptosystems, 1997.
(5) P. Kocher, J. Jaffe, B. Jun: Differential power analysis, 1997/98.
(6) W. Schindler: A timing attack against RSA-CRT, 2000.
(7) J.J.Quisquater, D. Samyde: Electromagnetic anaylsis, 2001.
(8) D. Boneh, D. Brumley: Remote timing attacks are practical, 2003.
(9) S. Chari, C. Jutla, P. Rohatgi: Template attacks, 2003.
(10) E. Brier, C. Clavier, F. Olivier: Correlation power analysis with a leakage model, 2004.
(11) W. Schindler, K. Lemke, C. Paar: A stochastic model for differential side-channel cryptanalysis, 2005.
(12) F.-X. Standaert et al.: Template attacks in principal subspaces, 2006.
(13) B. Gierlichs et al.: Mutual Information Analysis, 2008.
(14) J. DiBatista et al.: When failure analysis meets side-channel analysis, 2010
(15) D.J. Bernstein: Cache-timing attacks on AES, 2004/05, D.A. Osvik et al. Cache attacks and countermeasures, 2006.
(16) O. Aciiçmez et al. On the power of simple branch prediction analysis, 2006.

From Rohde & Schwarez: Side Channel Attacks – A comparative approach, 2010.

Classes of side channel attacks



From Rohde & Schwarez: Side Channel Attacks – A comparative approach, 2010.

- The attacks can be against processors or chips, on smartcards for example, which people have access to.

Timing attacks

- Kocher described the idea in 1996, and it only took until 1998 for a practical attack to be given by Dhem et al.
- The idea is to measure how long operations take for known input.
- Knowing the algorithm used, we try to distinguish between different key values.

- For example, with target key k and input x:

```
If ( k[i] == 1 & x[i] == 1 )
  do a modulo operation.
else
  don't do a modulo operation.
Endif
```

Relatively costly...

We might need a lot of samples to get one bit, depending on the typical difference in relative cost.

MessageDigest.isEqual

- This is a Java function.
- In some versions prior to Java SE 6 Update 17, this was vulnerable to a timing attack.
- It's a fairly typical timing vulnerability problem.
- See http://codahale.com/a-lesson-in-timing-attacks/

```
public static Boolean isEqual (byte digesta[], byte digestb[]) {
      if (digesta.length != digestb.length)
            return false;

      for (int  i=0; i < digesta.length; i++) {
            if (digesta[i] != digestb[i]) {
                  return false;
            }
      }
      return true;
}
```

- Attack on Password verification
  - Assume the password verification routine is implemented as in Alg. 4 below

Algorithm 4 Password verification.

Input: $\widetilde{P} = (\widetilde{P}[0], \ldots, \widetilde{P}[7])$ (and $P = (P[0], \ldots, P[7])$)
Output: 'true' or 'false'

```
1: for j = 0 to 7 do
2:     if (P̃[j] ≠ P[j]) then return 'false'
3: end for
4: return 'true'
```

  - *P* denotes the 8-byte password proposed by the user and *P* denotes the correct password.
  - The routine returns "true" if the entered password is valid and "false" if it is not.

1. For $0 \leq n \leq 255$, the attacker proposes the 256 passwords $\widetilde{P}^{(n)} = (n, 0, 0, 0, 0, 0, 0, 0)$ and measures the corresponding running time, $\tau[n]$.
2. Next, the attacker computes the maximum running time

$$\tau[n_0] := \max_{0 \leq n \leq 255} \tau[n] \ .$$

   The correct value for the first byte of P, $P[0]$, is given by n_0.
3. Once $P[0]$ is known, the attacker reiterates the attack with

$$\widetilde{P}^{(n)} = (P[0], n, 0, 0, 0, 0, 0, 0),$$

   and so on until the whole value of P is recovered.

Example from the book “Cryptographic Engineering” by Cetin Kaya Koc edt.

Attack against RSA cryptosystems

- RSA public-key cryptosystem is widely used to secure electronic data transfer
  - Included as part of Web browsers from Microsoft and Netscape
  - Used by SSL (Secure sockets layer)
  - Invented by Rivest, Shamir and Adleman in 1978
- Its security is based on the integer factorisation (IF) problem:
  - **Given a large integer N (say 1024 or 2048 bit), find a non-trivial factor of N.**
  - IF is believe to be hard. In fact, most efficient algorithm for IF is still running in sub-exponential time.
  - IF is easy for quantum computers (due to Shor's algorithm in 1994)
  - Taken from: https://www.cs.sjsu.edu/faculty/stamp/students/article.html

RSA cryptosystem

- Let N = pq be the product of two large primes p and q
- Choose integers e and d such that ed = 1 mod (p-1)(q-1)
- The public key is (N,e), the private key is d

Example: let p = 11, q = 3, then N = 33. Choose e = 3 and d = 7, then ed = 21 = 1 mod (11-1)(3-1).
The public key is (33,3) and private key is 7

- Assume that Bob want to send a message M to Alice. Bob encrypts a message M, by computing the ciphertext C by $C = M^e mod\ N$.
- To decrypt the ciphertext C, Alice uses her secret key to compute $C^d mod\ N$.
- It is clear that if ed = 1 mod (p-1)(q-1) then

$$C^d = M^{ed} = M\ mod\ N$$

Hence Alice can recover the message M.

Example: go back to the previous example N = 33, (p-1)(q-1) = 30, e = 3 and d = 7.

- Assume message M = 19, then the ciphertext is $C = 19^3 mod\ 33 = 28$.
- To decrypt the ciphertext C, compute $28^7 mod\ 33 = 19$.

GOAL: find the secret key d

- For timing attack, the attacker needs to have the target system compute $C^d mod\ N$ for several carefully selected values of C
- Measuring the amount of time require and analysing the timing variations
- Recover d one bit at a time

- How $C^d mod\ N$ is computed?
- Using square and multiply algorithm

```
x = C
for j = 1 to n
    x = mod(x^2, N)
    if d_j == 1 then
        x = mod(xC, N)
    end if
next j
return x
```

Here $d = d_0 d_1 \ldots d_n$ in binary with d_0=1
For example 21 is represented as 10101

Example for computing $5^{21}\ mod\ 33$

Now C= 5

Write 21 as 10101

x = 5

j=1 then x = $x^2 = 25\ mod\ 33$

$d_1 = 0$

j = 2 then x = $25^2 = 31\ mod\ 33$

$d_2 = 1$ then

x = 31*5= $23\ mod\ 33$

j = 3 then x = $23^2 = 1\ mod\ 33$

$d_3 = 0$

j = 4 then x = $1^2 = 1\ mod\ 33$

$d_4 = 1$ then

x = 1*5= $5\ mod\ 33$

Hence $5^{21}\ mod\ 33 = 5\ mod\ 33$

```
x = C
for j = 1 to n
    x = mod(x^2, N)
    if d_j == 1 then
        x = mod(xC, N)
    end if
next j
return x
```

In RSA implementation, the Montgomery algorithm is used to perform multiplication and the square operation

- Multiplications take a constant amount of time, independent of the size of the factors.
- If the result of the multiplication exceeds the modulus N, an additional subtraction, called an extra reduction, is performed.
- This extra step causes the timing difference inputs and exposes information about the secret key.
- See for the Montgomery algorithm: https://en.wikipedia.org/wiki/Montgomery_modular_multiplication

Consider the mod(x,N) operation as follows

```
mod(x, N)
if x >= N
    x = x % N
end if
return x
```

```
x = C
for j = 1 to n
    x = mod(x^2, N)
    if d_j == 1 then
        x = mod(xC, N)
    end if
next j
return x
```

- If $d_j = 0$ then $x = mod(x^2, N)$
- But if $d_j = 1$ then we need to perform 2 operations $x = mod(x^2, N)$ and $x = mod(xC, N)$
  - So the computation time is longer
  - In addition, the % operation is only executed if the intermediate result of the multiplication is greater than N

Attack: recover d_j, use two set of messages

- one such that $mod(xC, N)$ requires a reduction,
- and another for which is not

Assume that we can get the system to decrypt the messages of our choice (which is often possible in real systems).

We start attacking d_1 by choosing two messages y and z such that $y^3 < N$ and $z^2 < N < z^3$

- If $d_1 = 1$ then $x = mod(x * x^2, N)$ will be performed.
- Since $y^3 < N$ then % does not occur for the message y
- But $z^2 < N < z^3$ then % occurs for z, hence take more time for z.
- If $d_1 = 0$ then $x = mod(x^2, N)$ will be performed, and it is the same for y and z that % does not occur
- Similarly for d_2 and so on.

```
x = C
for j = 1 to n
    x = mod(x², N)
    if dⱼ == 1 then
       x = mod(xC, N)
    end if
next j
return x
```

```
mod(x, N)
if x >= N
   x = x % N
end if
return x
```

How to defend?

(i) Make all private key operations not depend on the input

- Always do % even though it may not be used
- Careful to ensure that extra reduction is not optimised away by the compiler

(ii) Another way is to blind RSA.

- Before decrypting C, choose a random r and compute $X = r^e C \ mod \ N$
- Then compute $X^d = rC^d \ mod \ N$
- Then multiply with $r^{-1} mod \ N$ to recover $C^d \ mod \ N$ which is the message

Power analysis

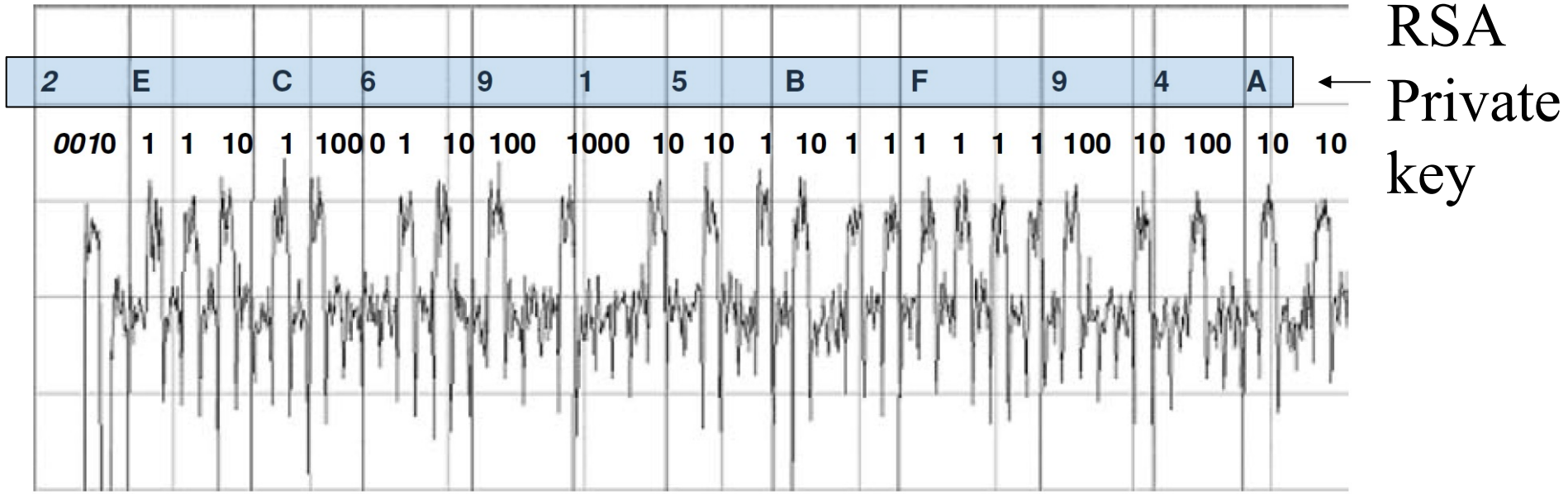
- Here we infer information about the internal behaviour of a cryptographic device, or likely a specific chip, on the basis of how much power is being used by it.
- Two types, simple power analysis and differential power analysis.

Simple Power analysis (SPA)

- These involve directly interpreting the measurements of power consumed during cryptographic operations.
  - The basic premise for this to be useful is that different operations have different power requirements.
- We define power traces associated with the use of algorithms.
- It’s possible, sometimes to distinguish between the use of different secret values.

For example: SPA on RSA

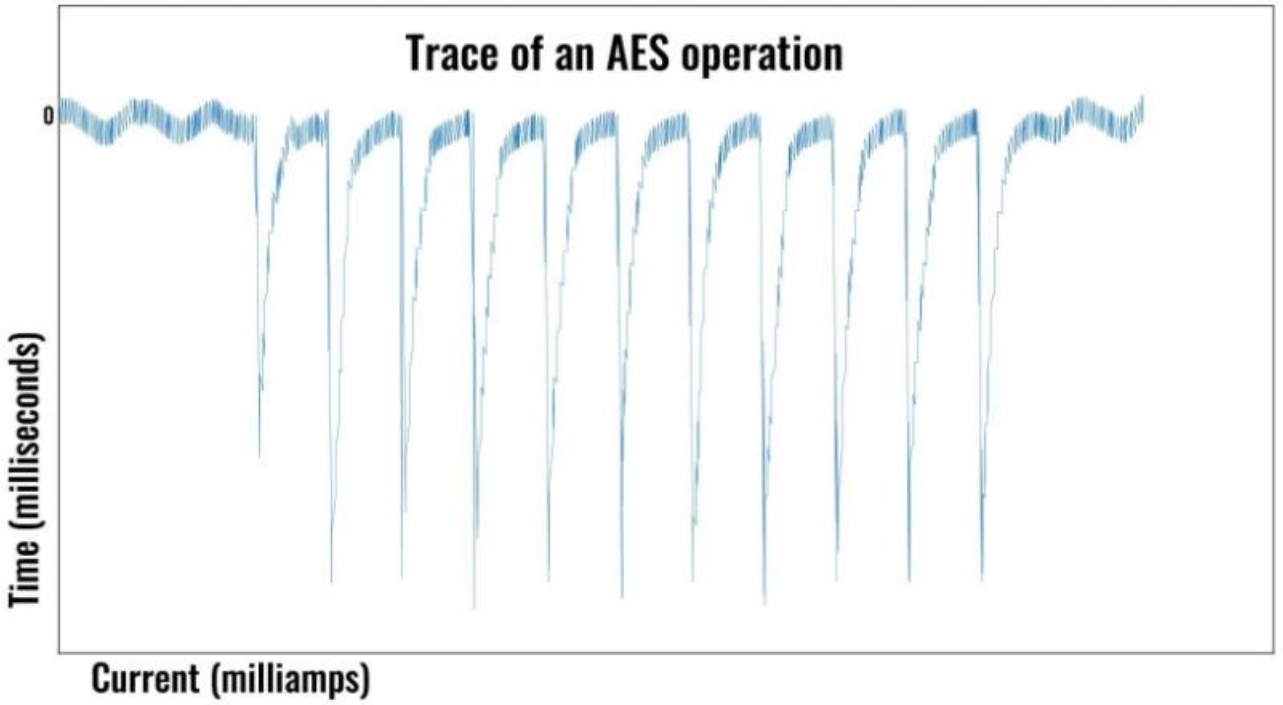
- RSA Decryption: $\mathrm{m} = c^d \ (mod\ n)$
  - Employing square and multiple to carry out the exponentiation.
  - Here goes the power trace for an RSA exponentiation using square and multiply algorithm.
  - We square at every step, but the 1's require multiply too.



From Joye, M., Olivier, F.: Side-channel analysis. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security. pp. 1198{1204. Springer US (2011)

AES

- AES: Advanced Encryption Standard
  - Symmetric encryption, established by NIST in 2001
  - https://en.wikipedia.org/wiki/Advanced_Encryption_Standard



  - From: https://www.comparitech.com/blog/information-security/side-channel-attack/

Countermeasures

- Implementations of cryptographic systems should be designed so that there are no secret values that affect conditional branches (these are essentially programmed instructions that say "*if x happens, branch to y*") within the software.
- Implementations need to be designed so that some sources of variation (e.g., CPU instructions with variable timing, microcode variations, various arithmetic operations…) do not offer information to be used to compromise the system.

Differential Power analysis (DPA)

- This uses statistical analysis to extract hidden information from a large sample of power traces.
- By using a large sample, we can identify differences that are too small to be useful in SPA.
- Kocher-Jaffe-Jun's paper: https://paulkocher.com/doc/DifferentialPowerAnalysis.pdf

- Include error correction and signal processing properties.
  - gain insights about the cryptographic operations that aren't visible through simple power analysis.
- Advanced form: **second-order differential power analysis.**
  - https://www.win.tue.nl/~berry/papers/ches05hodpa.pdf

Countermeasures

- Reducing the size of the signals
  - reduce the quality of the information that's available for the statistical analysis
- Adding noise
  - increases the number of samples an attacker would need to mount a successful attack.
- Design cryptosystems with realistic assumptions about the hardware they operate on
  - include key counters that stop attackers from being able to gather large numbers of samples
  - aggressively use exponent and modulus modification processes in public-key schemes

Electromagnetic analysis

- Measuring, possible from quite some distance, the electromagnetic radiation from a cryptographic device.
- This is applicable from a distance, which significantly changes the application domain of the attack.

https://m.tau.ac.il/~tromer/ecdh/

- Simple electromagnetic analysis
  - Attackers directly measure the electromagnetic activity
  - Attackers need a deep understanding of their targeted algorithm and the way that it has been implemented.
- Differential electromagnetic analysis
  - Useful when simple electromagnetic analysis didn’t yield enough information
  - These attacks don’t require as much knowledge about the targeted cryptosystem in order for them to be effective

Countermeasures

- Secure the location from attackers
- Physically shield the electromagnetic radiation
- Design the system so that there is no longer a correlation between the cryptographic algorithm and electromagnetic radiation

(Induced) Fault analysis

- In fault analysis attacks the attacker tries to force errors.
  - Maybe by spiking the voltage...
- It is possible that partial information can be returned.
  - For example, if a reduced round version of the ciphertext is returned a known cryptanalytic attack may be practical.
- Or the fault may cause something to be directly revealed.

Countermeasures

- Additional randomness helps.
- If the cost of the actual operations is hidden in among some random operations that take place analysis will be much trickier, if it is even possible.
- Environmental tampering sensors are implemented on most smartcards.
- In Surreptitious Software, by Collberg and Nagra, specific tamperproofing mechanisms are referred to.
- See also fips 140 (-1, -2 and -3) at http://csrc.nist.gov/publications/PubsFIPS.html

- Countermeasures should be thought about during the design phase.
- As we see in this subject we identify threats and manage them … and side channel attacks are a viable threat.
- For cache attacks, for example, they might mean we use Oblivious RAM where the pattern of access to memory is independent of the content so the access pattern doesn’t leak any information on the content.

- Countermeasures are going to cost and there are likely to be trade-offs between the efficiency and protection against side-channel attacks.
  - But this is standard for security engineering.
- The system might not run as quickly as it otherwise might, the power usage might not be so low, we might require more memory to carry out tasks.
  - But these are likely to be the costs of securing the system.